# Use the Docker command line

To list available commands, either run docker with no parameters or execute docker help:

$ docker

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

docker [ --help | -v | --version ]

A self-sufficient runtime for containers.

Options:

--config string Location of client config files (default "/root/.docker")

-c, --context string Name of the context to use to connect to the daemon (overrides DOCKER\_HOST env var and default context set with "docker context use")

-D, --debug Enable debug mode

--help Print usage

-H, --host value Daemon socket(s) to connect to (default [])

-l, --log-level string Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")

--tls Use TLS; implied by --tlsverify

--tlscacert string Trust certs signed only by this CA (default "/root/.docker/ca.pem")

--tlscert string Path to TLS certificate file (default "/root/.docker/cert.pem")

--tlskey string Path to TLS key file (default "/root/.docker/key.pem")

--tlsverify Use TLS and verify the remote

-v, --version Print version information and quit

Commands:

attach Attach to a running container

# […]

Depending on your Docker system configuration, you may be required to preface each docker command with sudo. To avoid having to use sudo with the docker command, your system administrator can create a Unix group called docker and add users to it.

For more information about installing Docker or sudo configuration, refer to the [installation](https://docs.docker.com/install/) instructions for your operating system.

## Environment variables

For easy reference, the following list of environment variables are supported by the docker command line:

* DOCKER\_API\_VERSION The API version to use (e.g. 1.19)
* DOCKER\_CONFIG The location of your client configuration files.
* DOCKER\_CERT\_PATH The location of your authentication keys.
* DOCKER\_CLI\_EXPERIMENTAL Enable experimental features for the cli (e.g. enabled or disabled)
* DOCKER\_DRIVER The graph driver to use.
* DOCKER\_HOST Daemon socket to connect to.
* DOCKER\_NOWARN\_KERNEL\_VERSION Prevent warnings that your Linux kernel is unsuitable for Docker.
* DOCKER\_RAMDISK If set this will disable ‘pivot\_root’.
* DOCKER\_STACK\_ORCHESTRATOR Configure the default orchestrator to use when using docker stack management commands.
* DOCKER\_TLS When set Docker uses TLS.
* DOCKER\_TLS\_VERIFY When set Docker uses TLS and verifies the remote.
* DOCKER\_CONTENT\_TRUST When set Docker uses notary to sign and verify images. Equates to --disable-content-trust=false for build, create, pull, push, run.
* DOCKER\_CONTENT\_TRUST\_SERVER The URL of the Notary server to use. This defaults to the same URL as the registry.
* DOCKER\_HIDE\_LEGACY\_COMMANDS When set, Docker hides “legacy” top-level commands (such as docker rm, and docker pull) in docker help output, and only Management commands per object-type (e.g., docker container) are printed. This may become the default in a future release, at which point this environment-variable is removed.
* DOCKER\_TMPDIR Location for temporary Docker files.
* DOCKER\_CONTEXT Specify the context to use (overrides DOCKER\_HOST env var and default context set with “docker context use”)
* DOCKER\_DEFAULT\_PLATFORM Specify the default platform for the commands that take the --platform flag.

Because Docker is developed using Go, you can also use any environment variables used by the Go runtime. In particular, you may find these useful:

* HTTP\_PROXY
* HTTPS\_PROXY
* NO\_PROXY

These Go environment variables are case-insensitive. See the [Go specification](http://golang.org/pkg/net/http/) for details on these variables.

**Configuration files**

By default, the Docker command line stores its configuration files in a directory called .docker within your $HOME directory.

Docker manages most of the files in the configuration directory and you should not modify them. However, you *can* modify the config.json file to control certain aspects of how the docker command behaves.

You can modify the docker command behavior using environment variables or command-line options. You can also use options within config.json to modify some of the same behavior. If an environment variable and the --config flag are set, the flag takes precedent over the environment variable. Command line options override environment variables and environment variables override properties you specify in a config.json file.

**Change the .docker directory**

To specify a different directory, use the DOCKER\_CONFIG environment variable or the --config command line option. If both are specified, then the --config option overrides the DOCKER\_CONFIG environment variable. The example below overrides the docker ps command using a config.json file located in the ~/testconfigs/ directory.

$ docker --config ~/testconfigs/ ps

This flag only applies to whatever command is being ran. For persistent configuration, you can set the DOCKER\_CONFIG environment variable in your shell (e.g. ~/.profile or ~/.bashrc). The example below sets the new directory to be HOME/newdir/.docker.

echo export DOCKER\_CONFIG=$HOME/newdir/.docker > ~/.profile

**config.json properties**

The config.json file stores a JSON encoding of several properties:

The property HttpHeaders specifies a set of headers to include in all messages sent from the Docker client to the daemon. Docker does not try to interpret or understand these header; it simply puts them into the messages. Docker does not allow these headers to change any headers it sets for itself.

The property psFormat specifies the default format for docker ps output. When the --format flag is not provided with the docker ps command, Docker’s client uses this property. If this property is not set, the client falls back to the default table format. For a list of supported formatting directives, see the [**Formatting** section in the docker ps documentation](https://docs.docker.com/engine/reference/commandline/ps/)

The property imagesFormat specifies the default format for docker images output. When the --format flag is not provided with the docker images command, Docker’s client uses this property. If this property is not set, the client falls back to the default table format. For a list of supported formatting directives, see the [**Formatting** section in the docker images documentation](https://docs.docker.com/engine/reference/commandline/images/)

The property pluginsFormat specifies the default format for docker plugin ls output. When the --format flag is not provided with the docker plugin ls command, Docker’s client uses this property. If this property is not set, the client falls back to the default table format. For a list of supported formatting directives, see the [**Formatting** section in the docker plugin ls documentation](https://docs.docker.com/engine/reference/commandline/plugin_ls/)

The property servicesFormat specifies the default format for docker service ls output. When the --format flag is not provided with the docker service ls command, Docker’s client uses this property. If this property is not set, the client falls back to the default json format. For a list of supported formatting directives, see the [**Formatting** section in the docker service ls documentation](https://docs.docker.com/engine/reference/commandline/service_ls/)

The property serviceInspectFormat specifies the default format for docker service inspect output. When the --format flag is not provided with the docker service inspect command, Docker’s client uses this property. If this property is not set, the client falls back to the default json format. For a list of supported formatting directives, see the [**Formatting** section in the docker service inspect documentation](https://docs.docker.com/engine/reference/commandline/service_inspect/)

The property statsFormat specifies the default format for docker stats output. When the --format flag is not provided with the docker stats command, Docker’s client uses this property. If this property is not set, the client falls back to the default table format. For a list of supported formatting directives, see [**Formatting** section in the docker stats documentation](https://docs.docker.com/engine/reference/commandline/stats/)

The property secretFormat specifies the default format for docker secret ls output. When the --format flag is not provided with the docker secret ls command, Docker’s client uses this property. If this property is not set, the client falls back to the default table format. For a list of supported formatting directives, see [**Formatting** section in the docker secret ls documentation](https://docs.docker.com/engine/reference/commandline/secret_ls/)

The property nodesFormat specifies the default format for docker node ls output. When the --format flag is not provided with the docker node ls command, Docker’s client uses the value of nodesFormat. If the value of nodesFormat is not set, the client uses the default table format. For a list of supported formatting directives, see the [**Formatting** section in the docker node ls documentation](https://docs.docker.com/engine/reference/commandline/node_ls/)

The property configFormat specifies the default format for docker config ls output. When the --format flag is not provided with the docker config ls command, Docker’s client uses this property. If this property is not set, the client falls back to the default table format. For a list of supported formatting directives, see [**Formatting** section in the docker config ls documentation](https://docs.docker.com/engine/reference/commandline/config_ls/)

The property credsStore specifies an external binary to serve as the default credential store. When this property is set, docker login will attempt to store credentials in the binary specified by docker-credential-<value> which is visible on $PATH. If this property is not set, credentials will be stored in the auths property of the config. For more information, see the [**Credentials store** section in the docker login documentation](https://docs.docker.com/engine/reference/commandline/login/#credentials-store)

The property credHelpers specifies a set of credential helpers to use preferentially over credsStore or auths when storing and retrieving credentials for specific registries. If this property is set, the binary docker-credential-<value> will be used when storing or retrieving credentials for a specific registry. For more information, see the [**Credential helpers** section in the docker login documentation](https://docs.docker.com/engine/reference/commandline/login/#credential-helpers)

The property stackOrchestrator specifies the default orchestrator to use when running docker stack management commands. Valid values are "swarm", "kubernetes", and "all". This property can be overridden with the DOCKER\_STACK\_ORCHESTRATOR environment variable, or the --orchestrator flag.

The property proxies specifies proxy environment variables to be automatically set on containers, and set as --build-arg on containers used during docker build. A "default" set of proxies can be configured, and will be used for any docker daemon that the client connects to, or a configuration per host (docker daemon), for example, “https://docker-daemon1.example.com”. The following properties can be set for each environment:

* httpProxy (sets the value of HTTP\_PROXY and http\_proxy)
* httpsProxy (sets the value of HTTPS\_PROXY and https\_proxy)
* ftpProxy (sets the value of FTP\_PROXY and ftp\_proxy)
* noProxy (sets the value of NO\_PROXY and no\_proxy)

**Warning**: Proxy settings may contain sensitive information (for example, if the proxy requires authentication). Environment variables are stored as plain text in the container’s configuration, and as such can be inspected through the remote API or committed to an image when using docker commit.

Once attached to a container, users detach from it and leave it running using the using CTRL-p CTRL-q key sequence. This detach key sequence is customizable using the detachKeys property. Specify a <sequence> value for the property. The format of the <sequence> is a comma-separated list of either a letter [a-Z], or the ctrl- combined with any of the following:

* a-z (a single lowercase alpha character )
* @ (at sign)
* [ (left bracket)
* \\ (two backward slashes)
* \_ (underscore)
* ^ (caret)

Your customization applies to all containers started in with your Docker client. Users can override your custom or the default key sequence on a per-container basis. To do this, the user specifies the --detach-keys flag with the docker attach, docker exec, docker run or docker start command.

The property plugins contains settings specific to CLI plugins. The key is the plugin name, while the value is a further map of options, which are specific to that plugin.

Following is a sample config.json file:

{

"HttpHeaders": {

"MyHeader": "MyValue"

},

"psFormat": "table {{.ID}}\\t{{.Image}}\\t{{.Command}}\\t{{.Labels}}",

"imagesFormat": "table {{.ID}}\\t{{.Repository}}\\t{{.Tag}}\\t{{.CreatedAt}}",

"pluginsFormat": "table {{.ID}}\t{{.Name}}\t{{.Enabled}}",

"statsFormat": "table {{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}",

"servicesFormat": "table {{.ID}}\t{{.Name}}\t{{.Mode}}",

"secretFormat": "table {{.ID}}\t{{.Name}}\t{{.CreatedAt}}\t{{.UpdatedAt}}",

"configFormat": "table {{.ID}}\t{{.Name}}\t{{.CreatedAt}}\t{{.UpdatedAt}}",

"serviceInspectFormat": "pretty",

"nodesFormat": "table {{.ID}}\t{{.Hostname}}\t{{.Availability}}",

"detachKeys": "ctrl-e,e",

"credsStore": "secretservice",

"credHelpers": {

"awesomereg.example.org": "hip-star",

"unicorn.example.com": "vcbait"

},

"stackOrchestrator": "kubernetes",

"plugins": {

"plugin1": {

"option": "value"

},

"plugin2": {

"anotheroption": "anothervalue",

"athirdoption": "athirdvalue"

}

},

"proxies": {

"default": {

"httpProxy": "http://user:pass@example.com:3128",

"httpsProxy": "http://user:pass@example.com:3128",

"noProxy": "http://user:pass@example.com:3128",

"ftpProxy": "http://user:pass@example.com:3128"

},

"https://manager1.mycorp.example.com:2377": {

"httpProxy": "http://user:pass@example.com:3128",

"httpsProxy": "http://user:pass@example.com:3128"

},

}

}

**Experimental features**

Experimental features provide early access to future product functionality. These features are intended only for testing and feedback as they may change between releases without warning or can be removed entirely from a future release.

Experimental features must not be used in production environments.

To enable experimental features, edit the config.json file and set experimental to enabled. The example below enables experimental features in a config.json file that already enables a debug feature.

{

"experimental": "enabled",

"debug": true

}

You can also enable experimental features from the Docker Desktop menu. See the [Docker Desktop Getting Started page](https://docs.docker.com/docker-for-mac#experimental-features) for more information.

**Notary**

If using your own notary server and a self-signed certificate or an internal Certificate Authority, you need to place the certificate at tls/<registry\_url>/ca.crt in your docker config directory.

Alternatively you can trust the certificate globally by adding it to your system’s list of root Certificate Authorities.

## Examples

**Display help text**

To list the help on any command just execute the command, followed by the --help option.

$ docker run --help

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

Run a command in a new container

Options:

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

-a, --attach value Attach to STDIN, STDOUT or STDERR (default [])

...

**Option types**

Single character command line options can be combined, so rather than typing docker run -i -t --name test busybox sh, you can write docker run -it --name test busybox sh.

**Boolean**

Boolean options take the form -d=false. The value you see in the help text is the default value which is set if you do **not** specify that flag. If you specify a Boolean flag without a value, this will set the flag to true, irrespective of the default value.

For example, running docker run -d will set the value to true, so your container **will** run in “detached” mode, in the background.

Options which default to true (e.g., docker build --rm=true) can only be set to the non-default value by explicitly setting them to false:

$ docker build --rm=false .

**Multi**

You can specify options like -a=[] multiple times in a single command line, for example in these commands:

$ docker run -a stdin -a stdout -i -t ubuntu /bin/bash

$ docker run -a stdin -a stdout -a stderr ubuntu /bin/ls

Sometimes, multiple options can call for a more complex value string as for -v:

$ docker run -v /host:/container example/mysql

**Note**: Do not use the -t and -a stderr options together due to limitations in the pty implementation. All stderr in pty mode simply goes to stdout.

**Strings and Integers**

Options like --name="" expect a string, and they can only be specified once. Options like -c=0 expect an integer, and they can only be specified once.

# Docker run reference

Docker runs processes in isolated containers. A container is a process which runs on a host. The host may be local or remote. When an operator executes docker run, the container process that runs is isolated in that it has its own file system, its own networking, and its own isolated process tree separate from the host.

This page details how to use the docker run command to define the container’s resources at runtime.

## General form

The basic docker run command takes this form:

$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]

The docker run command must specify an [*IMAGE*](https://docs.docker.com/engine/reference/glossary/#image) to derive the container from. An image developer can define image defaults related to:

* detached or foreground running
* container identification
* network settings
* runtime constraints on CPU and memory

With the docker run [OPTIONS] an operator can add to or override the image defaults set by a developer. And, additionally, operators can override nearly all the defaults set by the Docker runtime itself. The operator’s ability to override image and Docker runtime defaults is why [*run*](https://docs.docker.com/engine/reference/commandline/run/) has more options than any other docker command.

To learn how to interpret the types of [OPTIONS], see [*Option types*](https://docs.docker.com/engine/reference/commandline/cli/#option-types).

**Note**: Depending on your Docker system configuration, you may be required to preface the docker run command with sudo. To avoid having to use sudo with the docker command, your system administrator can create a Unix group called docker and add users to it. For more information about this configuration, refer to the Docker installation documentation for your operating system.

## Operator exclusive options

Only the operator (the person executing docker run) can set the following options.

* [Detached vs foreground](https://docs.docker.com/engine/reference/run/#detached-vs-foreground)
  + [Detached (-d)](https://docs.docker.com/engine/reference/run/#detached--d)
  + [Foreground](https://docs.docker.com/engine/reference/run/#foreground)
* [Container identification](https://docs.docker.com/engine/reference/run/#container-identification)
  + [Name (--name)](https://docs.docker.com/engine/reference/run/#name---name)
  + [PID equivalent](https://docs.docker.com/engine/reference/run/#pid-equivalent)
* [IPC settings (--ipc)](https://docs.docker.com/engine/reference/run/#ipc-settings---ipc)
* [Network settings](https://docs.docker.com/engine/reference/run/#network-settings)
* [Restart policies (--restart)](https://docs.docker.com/engine/reference/run/#restart-policies---restart)
* [Clean up (--rm)](https://docs.docker.com/engine/reference/run/#clean-up---rm)
* [Runtime constraints on resources](https://docs.docker.com/engine/reference/run/#runtime-constraints-on-resources)
* [Runtime privilege and Linux capabilities](https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities)

### Detached vs foreground

When starting a Docker container, you must first decide if you want to run the container in the background in a “detached” mode or in the default foreground mode:

-d=false: Detached mode: Run container in the background, print new container id

### Detached (-d)

To start a container in detached mode, you use -d=true or just -d option. By design, containers started in detached mode exit when the root process used to run the container exits, unless you also specify the --rm option. If you use -d with --rm, the container is removed when it exits **or** when the daemon exits, whichever happens first.

Do not pass a service x start command to a detached container. For example, this command attempts to start the nginx service.

$ docker run -d -p 80:80 my\_image service nginx start

This succeeds in starting the nginx service inside the container. However, it fails the detached container paradigm in that, the root process (service nginx start) returns and the detached container stops as designed. As a result, the nginx service is started but could not be used. Instead, to start a process such as the nginx web server do the following:

$ docker run -d -p 80:80 my\_image nginx -g 'daemon off;'

To do input/output with a detached container use network connections or shared volumes. These are required because the container is no longer listening to the command line where docker run was run.

To reattach to a detached container, use docker [*attach*](https://docs.docker.com/engine/reference/commandline/attach/) command.

### Foreground

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 command line executables expect) and pass along signals. All of that is configurable:

-a=[] : Attach to `STDIN`, `STDOUT` and/or `STDERR`

-t : Allocate a pseudo-tty

--sig-proxy=true: Proxy all received signals to the process (non-TTY mode only)

-i : Keep STDIN open even if not attached

If you do not specify -a then Docker will [attach to both stdout and stderr](https://github.com/docker/docker/blob/4118e0c9eebda2412a09ae66e90c34b85fae3275/runconfig/opts/parse.go#L267) . You can specify to which of the three standard streams (STDIN, STDOUT, STDERR) you’d like to connect instead, as in:

$ docker run -a stdin -a stdout -i -t ubuntu /bin/bash

For interactive processes (like a shell), you must use -i -t together in order to allocate a tty for the container process. -i -t is often written -it as you’ll see in later examples. Specifying -t is forbidden when the client is receiving its standard input from a pipe, as in:

$ echo test | docker run -i busybox cat

**Note**: A process running as PID 1 inside a container is treated specially by Linux: it ignores any signal with the default action. So, the process will not terminate on SIGINT or SIGTERM unless it is coded to do so.

## Container identification

**Name (--name)**

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. If you do not assign a container name with the --name option, then the daemon generates a random string name for you. Defining a name can be a handy way to add meaning to a container. If you specify a name, you can use it when referencing the container within a Docker network. This works for both background and foreground Docker containers.

**Note**: Containers on the default bridge network must be linked to communicate by name.

**PID equivalent**

Finally, to help with automation, you can have Docker write the container ID out to a file of your choosing. This is similar to how some programs might write out their process ID to a file (you’ve seen them as PID files):

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

**Image[:tag]**

While not strictly a means of identifying a container, you can specify a version of an image you’d like to run the container with by adding image[:tag] to the command. For example, docker run ubuntu:14.04.

**Image[@digest]**

Images using the v2 or later image format have a content-addressable identifier called a digest. As long as the input used to generate the image is unchanged, the digest value is predictable and referenceable.

The following example runs a container from the alpine image with the sha256:9cacb71397b640eca97488cf08582ae4e4068513101088e9f96c9814bfda95e0 digest:

$ docker run alpine@sha256:9cacb71397b640eca97488cf08582ae4e4068513101088e9f96c9814bfda95e0 date

## PID settings (--pid)

--pid="" : Set the PID (Process) Namespace mode for the container,

'container:<name|id>': joins another container's PID namespace

'host': use the host's PID namespace inside the container

By default, all containers have the PID namespace enabled.

PID namespace provides separation of processes. The PID Namespace removes the view of the system processes, and allows process ids to be reused including pid 1.

In certain cases you want your container to share the host’s process namespace, basically allowing processes within the container to see all of the processes on the system. For example, you could build a container with debugging tools like strace or gdb, but want to use these tools when debugging processes within the container.

**Example: run htop inside a container**

Create this Dockerfile:

FROM alpine:latest

RUN apk add --update htop && rm -rf /var/cache/apk/\*

CMD ["htop"]

Build the Dockerfile and tag the image as myhtop:

$ docker build -t myhtop .

Use the following command to run htop inside a container:

$ docker run -it --rm --pid=host myhtop

Joining another container’s pid namespace can be used for debugging that container.

**Example**

Start a container running a redis server:

$ docker run --name my-redis -d redis

Debug the redis container by running another container that has strace in it:

$ docker run -it --pid=container:my-redis my\_strace\_docker\_image bash

$ strace -p 1

## UTS settings (--uts)

--uts="" : Set the UTS namespace mode for the container,

'host': use the host's UTS namespace inside the container

The UTS namespace is for setting the hostname and the domain that is visible to running processes in that namespace. By default, all containers, including those with --network=host, have their own UTS namespace. The host setting will result in the container using the same UTS namespace as the host. Note that --hostname and --domainname are invalid in host UTS mode.

You may wish to share the UTS namespace with the host if you would like the hostname of the container to change as the hostname of the host changes. A more advanced use case would be changing the host’s hostname from a container.

**IPC settings (--ipc)**

--ipc="MODE" : Set the IPC mode for the container

The following values are accepted:

| **Value** | **Description** |
| --- | --- |
| ”” | 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.

IPC (POSIX/SysV IPC) namespace provides separation of named shared memory segments, semaphores and message queues.

Shared memory segments are used to accelerate inter-process communication at memory speed, rather than through pipes or through the network stack. Shared memory is commonly used by databases and custom-built (typically C/OpenMPI, C++/using boost libraries) high performance applications for scientific computing and financial services industries. If these types of applications are broken into multiple containers, you might need to share the IPC mechanisms of the containers, using "shareable" mode for the main (i.e. “donor”) container, and "container:<donor-name-or-ID>" for other containers.

## Network settings

--dns=[] : Set custom dns servers for the container

--network="bridge" : Connect a container to a network

'bridge': create a network stack on the default Docker bridge

'none': no networking

'container:<name|id>': reuse another container's network stack

'host': use the Docker host network stack

'<network-name>|<network-id>': connect to a user-defined network

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

--add-host="" : Add a line to /etc/hosts (host:IP)

--mac-address="" : Sets the container's Ethernet device's MAC address

--ip="" : Sets the container's Ethernet device's IPv4 address

--ip6="" : Sets the container's Ethernet device's IPv6 address

--link-local-ip=[] : Sets one or more container's Ethernet device's link local IPv4/IPv6 addresses

By default, all containers have networking enabled and they can make any outgoing connections. The operator can completely disable networking with docker run --network none which disables all incoming and outgoing networking. In cases like this, you would perform I/O through files or STDIN and STDOUT only.

Publishing ports and linking to other containers only works with the default (bridge). The linking feature is a legacy feature. You should always prefer using Docker network drivers over linking.

Your container will use the same DNS servers as the host by default, but you can override this with --dns.

By default, the MAC address is generated using the IP address allocated to the container. You can set the container’s MAC address explicitly by providing a MAC address via the --mac-address parameter (format:12:34:56:78:9a:bc).Be aware that Docker does not check if manually specified MAC addresses are unique.

Supported networks :

| **Network** | **Description** |
| --- | --- |
| **none** | No networking in the container. |
| **bridge** (default) | Connect the container to the 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** | Connects the container to a user created network (using docker network create command) |

### Network: none

With the network is none a container will not have access to any external routes. The container will still have a loopback interface enabled in the container but it does not have any routes to external traffic.

### Network: bridge

With the network set to bridge a container will use docker’s default networking setup. A bridge is setup on the host, commonly named docker0, and a pair of veth interfaces will be created for the container. One side of the veth pair will remain on the host attached to the bridge while the other side of the pair will be placed inside the container’s namespaces in addition to the loopback interface. An IP address will be allocated for containers on the bridge’s network and traffic will be routed though this bridge to the container.

Containers can communicate via their IP addresses by default. To communicate by name, they must be linked.

### Network: host

With the network set to host a container will share the host’s network stack and all interfaces from the host will be available to the container. The container’s hostname will match the hostname on the host system. Note that --mac-address is invalid in host netmode. Even in host network mode a container has its own UTS namespace by default. As such --hostname and --domainname are allowed in host network mode and will only change the hostname and domain name inside the container. Similar to --hostname, the --add-host, --dns, --dns-search, and --dns-option options can be used in host network mode. These options update /etc/hosts or /etc/resolv.conf inside the container. No change are made to /etc/hosts and /etc/resolv.conf on the host.

Compared to the default bridge mode, the host mode gives *significantly* better networking performance since it uses the host’s native networking stack whereas the bridge has to go through one level of virtualization through the docker daemon. It is recommended to run containers in this mode when their networking performance is critical, for example, a production Load Balancer or a High Performance Web Server.

**Note**: --network="host" gives the container full access to local system services such as D-bus and is therefore considered insecure.

### Network: container

With the network set to container a container will share the network stack of another container. The other container’s name must be provided in the format of --network container:<name|id>. Note that --add-host --hostname --dns --dns-search --dns-option and --mac-address are invalid in container netmode, and --publish --publish-all --expose are also invalid in container netmode.

Example running a Redis container with Redis binding to localhost then running the redis-cli command and connecting to the Redis server over the localhost interface.

$ docker run -d --name redis example/redis --bind 127.0.0.1

$ # use the redis container's network stack to access localhost

$ docker run --rm -it --network container:redis example/redis-cli -h 127.0.0.1

### User-defined network

You can create a network using a Docker network driver or an external network driver plugin. You can connect multiple containers to the same network. Once connected to a user-defined network, the containers can communicate easily using 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.

The following example creates a network using the built-in bridge network driver and running a container in the created network

$ docker network create -d bridge my-net

$ docker run --network=my-net -itd --name=container3 busybox

### Managing /etc/hosts

Your container will have lines in /etc/hosts which define the hostname of the container itself as well as localhost and a few other common things. The --add-host flag can be used to add additional lines to /etc/hosts.

$ docker run -it --add-host db-static:86.75.30.9 ubuntu cat /etc/hosts

172.17.0.22 09d03f76bf2c

fe00::0 ip6-localnet

ff00::0 ip6-mcastprefix

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

127.0.0.1 localhost

::1 localhost ip6-localhost ip6-loopback

86.75.30.9 db-static

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.

## Restart policies (--restart)

Using the --restart flag on Docker run you can specify a restart policy for how a container should or should not be restarted on exit.

When a restart policy is active on a container, it will be shown as either Up or Restarting in [docker ps](https://docs.docker.com/engine/reference/commandline/ps/). It can also be useful to use [docker events](https://docs.docker.com/engine/reference/commandline/events/) to see the restart policy in effect.

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. |
| **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, including on daemon startup, except if the container was put into a stopped state before the Docker daemon was stopped. |

An ever increasing delay (double the previous delay, starting at 100 milliseconds) is added before each restart to prevent flooding the server. This means the daemon will wait for 100 ms, then 200 ms, 400, 800, 1600, and so on until either the on-failure limit is hit, or when you docker stop or docker rm -f the container.

If a container is successfully restarted (the container is started and runs for at least 10 seconds), the delay is reset to its default value of 100 ms.

You can specify the maximum amount of times Docker will try to restart the container when using the **on-failure** policy. The default is that Docker will try forever to restart the container. The number of (attempted) restarts for a container can be obtained via [docker inspect](https://docs.docker.com/engine/reference/commandline/inspect/). For example, to get the number of restarts for container “my-container”;

$ docker inspect -f "{{ .RestartCount }}" my-container

# 2

Or, to get the last time the container was (re)started;

$ docker inspect -f "{{ .State.StartedAt }}" my-container

# 2015-03-04T23:47:07.691840179Z

Combining --restart (restart policy) with the --rm (clean up) flag results in an error. On container restart, attached clients are disconnected. See the examples on using the [--rm (clean up)](https://docs.docker.com/engine/reference/run/#clean-up-rm) flag later in this page.

### Examples

$ 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.

$ docker run --restart=on-failure:10 redis

This will run the redis container with a restart policy of **on-failure** and a maximum restart count of 10. If the redis container exits with a non-zero exit status more than 10 times in a row Docker will abort trying to restart the container. Providing a maximum restart limit is only valid for the **on-failure** policy.

## 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 $?

# docker: Error response from daemon: Container command '/etc' could not be invoked.

126

***127*** if the ***contained command*** cannot be found

$ docker run busybox foo; echo $?

# docker: Error response from daemon: Container command 'foo' not found or does not exist.

127

***Exit code*** of ***contained command*** otherwise

$ docker run busybox /bin/sh -c 'exit 3'; echo $?

# 3

## Clean up (--rm)

By default a container’s file system persists even after the container exits. This makes debugging a lot easier (since you can inspect the final state) and you retain all your data by default. But if you are running short-term **foreground** processes, these container file systems can really pile up. If instead you’d like Docker to **automatically clean up the container and remove the file system when the container exits**, you can add the --rm flag:

--rm=false: Automatically remove the container when it exits

**Note**: When you set the --rm flag, Docker also removes the anonymous volumes associated with the container when the container is removed. This is similar to running docker rm -v my-container. Only volumes that are specified without a name are removed. For example, with docker run --rm -v /foo -v awesome:/bar busybox top, the volume for /foo will be removed, but the volume for /bar will not. Volumes inherited via --volumes-from will be removed with the same logic -- if the original volume was specified with a name it will **not** be removed.

## Security configuration

--security-opt="label=user:USER" : Set the label user for the container

--security-opt="label=role:ROLE" : Set the label role for the container

--security-opt="label=type:TYPE" : Set the label type for the container

--security-opt="label=level:LEVEL" : Set the label level for the container

--security-opt="label=disable" : Turn off label confinement for the container

--security-opt="apparmor=PROFILE" : Set the apparmor profile to be applied to the container

--security-opt="no-new-privileges:true|false" : Disable/enable container processes from gaining new privileges

--security-opt="seccomp=unconfined" : Turn off seccomp confinement for the container

--security-opt="seccomp=profile.json": White listed syscalls seccomp Json file to be used as a seccomp filter

You can override the default labeling scheme for each container by specifying the --security-opt flag. 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 -it fedora bash

**Note**: Automatic translation of MLS labels is not currently supported.

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

$ docker run --security-opt label=disable -it 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 -it centos bash

**Note**: You would have to write policy defining a svirt\_apache\_t type.

If you want to prevent your container processes from gaining additional privileges, you can execute the following command:

$ docker run --security-opt no-new-privileges -it centos bash

This means that commands that raise privileges such as su or sudo will no longer work. It also causes any seccomp filters to be applied later, after privileges have been dropped which may mean you can have a more restrictive set of filters. For more details, see the [kernel documentation](https://www.kernel.org/doc/Documentation/prctl/no_new_privs.txt).

## Specify an init process

You can use the --init flag to indicate that an init process should be used as the PID 1 in the container. Specifying an init process ensures the usual responsibilities of an init system, such as reaping zombie processes, are performed inside the created container.

The default init process used is the first docker-init executable found in the system path of the Docker daemon process. This docker-init binary, included in the default installation, is backed by [tini](https://github.com/krallin/tini).

## Specify custom cgroups

Using the --cgroup-parent flag, you can pass a specific cgroup to run a container in. This allows you to create and manage cgroups on their own. You can define custom resources for those cgroups and put containers under a common parent group.

## Runtime constraints on resources

The operator can also adjust the performance parameters of the container:

| **Option** | **Description** |
| --- | --- |
| -m, --memory="" | Memory limit (format: <number>[<unit>]). Number is a positive integer. Unit can be one of b, k, m, or g. Minimum is 4M. |
| --memory-swap="" | Total memory limit (memory + swap, format: <number>[<unit>]). Number is a positive integer. Unit can be one of b, k, m, or g. |
| --memory-reservation="" | Memory soft limit (format: <number>[<unit>]). Number is a positive integer. Unit can be one of b, k, m, or g. |
| --kernel-memory="" | Kernel memory limit (format: <number>[<unit>]). Number is a positive integer. Unit can be one of b, k, m, or g. Minimum is 4M. |
| -c, --cpu-shares=0 | CPU shares (relative weight) |
| --cpus=0.000 | Number of CPUs. Number is a fractional number. 0.000 means no limit. |
| --cpu-period=0 | Limit the CPU CFS (Completely Fair Scheduler) period |
| --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. |
| --cpu-quota=0 | Limit the CPU CFS (Completely Fair Scheduler) quota |
| --cpu-rt-period=0 | Limit the CPU real-time period. In microseconds. Requires parent cgroups be set and cannot be higher than parent. Also check rtprio ulimits. |
| --cpu-rt-runtime=0 | Limit the CPU real-time runtime. In microseconds. Requires parent cgroups be set and cannot be higher than parent. Also check rtprio ulimits. |
| --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) |
| --device-read-bps="" | Limit read rate from a device (format: <device-path>:<number>[<unit>]). Number is a positive integer. Unit can be one of kb, mb, or gb. |
| --device-write-bps="" | Limit write rate to a device (format: <device-path>:<number>[<unit>]). Number is a positive integer. Unit can be one of kb, mb, or gb. |
| --device-read-iops="" | Limit read rate (IO per second) from a device (format: <device-path>:<number>). Number is a positive integer. |
| --device-write-iops="" | Limit write rate (IO per second) to a device (format: <device-path>:<number>). Number is a positive integer. |
| --oom-kill-disable=false | Whether to disable OOM Killer for the container or not. |
| --oom-score-adj=0 | Tune container’s OOM preferences (-1000 to 1000) |
| --memory-swappiness="" | Tune a container’s memory swappiness behavior. Accepts an integer between 0 and 100. |
| --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. |

**User memory constraints**

We have four ways to set user memory usage:

| **Option** | **Result** |
| --- | --- |
| **memory=inf, memory-swap=inf** (default) | There is no memory limit for the container. The container can use as much memory as needed. |
| **memory=L<inf, memory-swap=inf** | (specify memory and set memory-swap as -1) The container is not allowed to use more than L bytes of memory, but can use as much swap as is needed (if the host supports swap memory). |
| **memory=L<inf, memory-swap=2\*L** | (specify memory without memory-swap) The container is not allowed to use more than L bytes of memory, swap *plus* memory usage is double of that. |
| **memory=L<inf, memory-swap=S<inf, L<=S** | (specify both memory and memory-swap) The container is not allowed to use more than L bytes of memory, swap *plus* memory usage is limited by S. |

Examples:

$ docker run -it ubuntu:14.04 /bin/bash

We set nothing about memory, this means the processes in the container can use as much memory and swap memory as they need.

$ docker run -it -m 300M --memory-swap -1 ubuntu:14.04 /bin/bash

We set memory limit and disabled swap memory limit, this means the processes in the container can use 300M memory and as much swap memory as they need (if the host supports swap memory).

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

We set memory limit only, this means the processes in the container can use 300M memory and 300M swap memory, by default, the total virtual memory size (--memory-swap) will be set as double of memory, in this case, memory + swap would be 2\*300M, so processes can use 300M swap memory as well.

$ docker run -it -m 300M --memory-swap 1G ubuntu:14.04 /bin/bash

We set both memory and swap memory, so the processes in the container can use 300M memory and 700M swap memory.

Memory reservation is a kind of memory soft limit that allows for greater sharing of memory. Under normal circumstances, containers can use as much of the memory as needed and are constrained only by the hard limits set with the -m/--memory option. When memory reservation is set, Docker detects memory contention or low memory and forces containers to restrict their consumption to a reservation limit.

Always set the memory reservation value below the hard limit, otherwise the hard limit takes precedence. A reservation of 0 is the same as setting no reservation. By default (without reservation set), memory reservation is the same as the hard memory limit.

Memory reservation is a soft-limit feature and does not guarantee the limit won’t be exceeded. Instead, the feature attempts to ensure that, when memory is heavily contended for, memory is allocated based on the reservation hints/setup.

The following example limits the memory (-m) to 500M and sets the memory reservation to 200M.

$ docker run -it -m 500M --memory-reservation 200M ubuntu:14.04 /bin/bash

Under this configuration, when the container consumes memory more than 200M and less than 500M, the next system memory reclaim attempts to shrink container memory below 200M.

The following example set memory reservation to 1G without a hard memory limit.

$ docker run -it --memory-reservation 1G ubuntu:14.04 /bin/bash

The container can use as much memory as it needs. The memory reservation setting ensures the container doesn’t consume too much memory for long time, because every memory reclaim shrinks the container’s consumption to the reservation.

By default, kernel kills processes in a container if an out-of-memory (OOM) error occurs. To change this behaviour, use the --oom-kill-disable option. Only disable the OOM killer on containers where you have also set the -m/--memory option. If the -m flag is not set, this can result in the host running out of memory and require killing the host’s system processes to free memory.

The following example limits the memory to 100M and disables the OOM killer for this container:

$ docker run -it -m 100M --oom-kill-disable ubuntu:14.04 /bin/bash

The following example, illustrates a dangerous way to use the flag:

$ docker run -it --oom-kill-disable ubuntu:14.04 /bin/bash

The container has unlimited memory which can cause the host to run out memory and require killing system processes to free memory. The --oom-score-adj parameter can be changed to select the priority of which containers will be killed when the system is out of memory, with negative scores making them less likely to be killed, and positive scores more likely.

**Kernel memory constraints**

Kernel memory is fundamentally different than user memory as kernel memory can’t be swapped out. The inability to swap makes it possible for the container to block system services by consuming too much kernel memory. Kernel memory includes：

* stack pages
* slab pages
* sockets memory pressure
* tcp memory pressure

You can setup kernel memory limit to constrain these kinds of memory. For example, every process consumes some stack pages. By limiting kernel memory, you can prevent new processes from being created when the kernel memory usage is too high.

Kernel memory is never completely independent of user memory. Instead, you limit kernel memory in the context of the user memory limit. Assume “U” is the user memory limit and “K” the kernel limit. There are three possible ways to set limits:

| **Option** | **Result** |
| --- | --- |
| **U != 0, K = inf** (default) | This is the standard memory limitation mechanism already present before using kernel memory. Kernel memory is completely ignored. |
| **U != 0, K < U** | Kernel memory is a subset of the user memory. This setup is useful in deployments where the total amount of memory per-cgroup is overcommitted. Overcommitting kernel memory limits is definitely not recommended, since the box can still run out of non-reclaimable memory. In this case, you can configure K so that the sum of all groups is never greater than the total memory. Then, freely set U at the expense of the system's service quality. |
| **U != 0, K > U** | Since kernel memory charges are also fed to the user counter and reclamation is triggered for the container for both kinds of memory. This configuration gives the admin a unified view of memory. It is also useful for people who just want to track kernel memory usage. |

Examples:

$ docker run -it -m 500M --kernel-memory 50M ubuntu:14.04 /bin/bash

We set memory and kernel memory, so the processes in the container can use 500M memory in total, in this 500M memory, it can be 50M kernel memory tops.

$ docker run -it --kernel-memory 50M ubuntu:14.04 /bin/bash

We set kernel memory without **-m**, so the processes in the container can use as much memory as they want, but they can only use 50M kernel memory.

**Swappiness constraint**

By default, a container’s kernel can swap out a percentage of anonymous pages. To set this percentage for a container, specify a --memory-swappiness value between 0 and 100. A value of 0 turns off anonymous page swapping. A value of 100 sets all anonymous pages as swappable. By default, if you are not using --memory-swappiness, memory swappiness value will be inherited from the parent.

For example, you can set:

$ docker run -it --memory-swappiness=0 ubuntu:14.04 /bin/bash

Setting the --memory-swappiness option is helpful when you want to retain the container’s working set and to avoid swapping performance penalties.

**CPU share constraint**

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 -c or --cpu-shares flag to set the weighting to 2 or higher. If 0 is set, the system will ignore the value and use the default of 1024.

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

**CPU period constraint**

The default CPU CFS (Completely Fair Scheduler) period is 100ms. We can use --cpu-period to set the period of CPUs to limit the container’s CPU usage. And usually --cpu-period should work with --cpu-quota.

Examples:

$ docker run -it --cpu-period=50000 --cpu-quota=25000 ubuntu:14.04 /bin/bash

If there is 1 CPU, this means the container can get 50% CPU worth of run-time every 50ms.

In addition to use --cpu-period and --cpu-quota for setting CPU period constraints, it is possible to specify --cpus with a float number to achieve the same purpose. For example, if there is 1 CPU, then --cpus=0.5 will achieve the same result as setting --cpu-period=50000 and --cpu-quota=25000 (50% CPU).

The default value for --cpus is 0.000, which means there is no limit.

For more information, see the [CFS documentation on bandwidth limiting](https://www.kernel.org/doc/Documentation/scheduler/sched-bwc.txt).

**Cpuset constraint**

We can set cpus in which to allow execution for containers.

Examples:

$ docker run -it --cpuset-cpus="1,3" ubuntu:14.04 /bin/bash

This means processes in container can be executed on cpu 1 and cpu 3.

$ docker run -it --cpuset-cpus="0-2" ubuntu:14.04 /bin/bash

This means processes in container can be executed on cpu 0, cpu 1 and cpu 2.

We can set mems in which to allow execution for containers. Only effective on NUMA systems.

Examples:

$ docker run -it --cpuset-mems="1,3" ubuntu:14.04 /bin/bash

This example restricts the processes in the container to only use memory from memory nodes 1 and 3.

$ docker run -it --cpuset-mems="0-2" ubuntu:14.04 /bin/bash

This example restricts the processes in the container to only use memory from memory nodes 0, 1 and 2.

**CPU quota constraint**

The --cpu-quota flag limits the container’s CPU usage. The default 0 value allows the container to take 100% of a CPU resource (1 CPU). The CFS (Completely Fair Scheduler) handles resource allocation for executing processes and is default Linux Scheduler used by the kernel. Set this value to 50000 to limit the container to 50% of a CPU resource. For multiple CPUs, adjust the --cpu-quota as necessary. For more information, see the [CFS documentation on bandwidth limiting](https://www.kernel.org/doc/Documentation/scheduler/sched-bwc.txt).

**Block IO bandwidth (Blkio) constraint**

By default, all containers get the same proportion of block IO bandwidth (blkio). This proportion is 500. To modify this proportion, change the container’s blkio weight relative to the weighting of all other running containers using the --blkio-weight flag.

**Note:** The blkio weight setting is only available for direct IO. Buffered IO is not currently supported.

The --blkio-weight flag can set the weighting to a value between 10 to 1000. For example, the commands below create two containers with different blkio weight:

$ docker run -it --name c1 --blkio-weight 300 ubuntu:14.04 /bin/bash

$ docker run -it --name c2 --blkio-weight 600 ubuntu:14.04 /bin/bash

If you do block IO in the two containers at the same time, by, for example:

$ time dd if=/mnt/zerofile of=test.out bs=1M count=1024 oflag=direct

You’ll find that the proportion of time is the same as the proportion of blkio weights of the two containers.

The --blkio-weight-device="DEVICE\_NAME:WEIGHT" flag sets a specific device weight. The DEVICE\_NAME:WEIGHT is a string containing a colon-separated device name and weight. For example, to set /dev/sda device weight to 200:

$ docker run -it \

--blkio-weight-device "/dev/sda:200" \

ubuntu

If you specify both the --blkio-weight and --blkio-weight-device, Docker uses the --blkio-weight as the default weight and uses --blkio-weight-device to override this default with a new value on a specific device. The following example uses a default weight of 300 and overrides this default on /dev/sda setting that weight to 200:

$ docker run -it \

--blkio-weight 300 \

--blkio-weight-device "/dev/sda:200" \

ubuntu

The --device-read-bps flag limits the read rate (bytes per second) from a device. For example, this command creates a container and limits the read rate to 1mb per second from /dev/sda:

$ docker run -it --device-read-bps /dev/sda:1mb ubuntu

The --device-write-bps flag limits the write rate (bytes per second) to a device. For example, this command creates a container and limits the write rate to 1mb per second for /dev/sda:

$ docker run -it --device-write-bps /dev/sda:1mb ubuntu

Both flags take limits in the <device-path>:<limit>[unit] format. Both read and write rates must be a positive integer. You can specify the rate in kb (kilobytes), mb (megabytes), or gb (gigabytes).

The --device-read-iops flag limits read rate (IO per second) from a device. For example, this command creates a container and limits the read rate to 1000 IO per second from /dev/sda:

$ docker run -ti --device-read-iops /dev/sda:1000 ubuntu

The --device-write-iops flag limits write rate (IO per second) to a device. For example, this command creates a container and limits the write rate to 1000 IO per second to /dev/sda:

$ docker run -ti --device-write-iops /dev/sda:1000 ubuntu

Both flags take limits in the <device-path>:<limit> format. Both read and write rates must be a positive integer.

**Additional groups**

--group-add: Add additional groups to run as

By default, the docker container process runs with the supplementary groups looked up for the specified user. If one wants to add more to that list of groups, then one can use this flag:

$ docker run --rm --group-add audio --group-add nogroup --group-add 777 busybox id

uid=0(root) gid=0(root) groups=10(wheel),29(audio),99(nogroup),777

**Runtime privilege and Linux capabilities**

--cap-add: Add Linux capabilities

--cap-drop: Drop Linux capabilities

--privileged=false: Give extended privileges to this container

--device=[]: Allows you to run devices inside the container without the --privileged flag.

By default, Docker containers are “unprivileged” and cannot, for example, run a Docker daemon inside a Docker container. This is because by default a container is not allowed to access any devices, but a “privileged” container is given access to all devices (see the documentation on [cgroups devices](https://www.kernel.org/doc/Documentation/cgroup-v1/devices.txt)).

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 or SELinux to allow the container nearly all the same access to the host as processes running outside containers on the host. Additional information about running with --privileged is available on the [Docker Blog](http://blog.docker.com/2013/09/docker-can-now-run-within-docker/).

If you want to limit access to a specific device or devices you can use the --device flag. It allows you to specify one or more devices that will be accessible within the container.

$ docker run --device=/dev/snd:/dev/snd ...

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:w --rm -it ubuntu fdisk /dev/xvdc

crash....

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

fdisk: unable to open /dev/xvdc: Operation not permitted

In addition to --privileged, the operator can have fine grain control over the capabilities using --cap-add and --cap-drop. By default, Docker has a default list of capabilities that are kept. The following table lists the Linux capability options which are allowed by default and can be dropped.

| **Capability Key** | **Capability Description** |
| --- | --- |
| SETPCAP | Modify process capabilities. |
| MKNOD | Create special files using mknod(2). |
| AUDIT\_WRITE | Write records to kernel auditing log. |
| CHOWN | Make arbitrary changes to file UIDs and GIDs (see chown(2)). |
| NET\_RAW | Use RAW and PACKET sockets. |
| DAC\_OVERRIDE | Bypass file read, write, and execute permission checks. |
| FOWNER | Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file. |
| FSETID | Don’t clear set-user-ID and set-group-ID permission bits when a file is modified. |
| KILL | Bypass permission checks for sending signals. |
| SETGID | Make arbitrary manipulations of process GIDs and supplementary GID list. |
| SETUID | Make arbitrary manipulations of process UIDs. |
| NET\_BIND\_SERVICE | Bind a socket to internet domain privileged ports (port numbers less than 1024). |
| SYS\_CHROOT | Use chroot(2), change root directory. |
| SETFCAP | Set file capabilities. |

The next table shows the capabilities which are not granted by default and may be added.

| **Capability Key** | **Capability Description** |
| --- | --- |
| SYS\_MODULE | Load and unload kernel modules. |
| SYS\_RAWIO | Perform I/O port operations (iopl(2) and ioperm(2)). |
| SYS\_PACCT | Use acct(2), switch process accounting on or off. |
| SYS\_ADMIN | Perform a range of system administration operations. |
| SYS\_NICE | Raise process nice value (nice(2), setpriority(2)) and change the nice value for arbitrary processes. |
| SYS\_RESOURCE | Override resource Limits. |
| SYS\_TIME | Set system clock (settimeofday(2), stime(2), adjtimex(2)); set real-time (hardware) clock. |
| SYS\_TTY\_CONFIG | Use vhangup(2); employ various privileged ioctl(2) operations on virtual terminals. |
| AUDIT\_CONTROL | Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules. |
| MAC\_ADMIN | Allow MAC configuration or state changes. Implemented for the Smack LSM. |
| MAC\_OVERRIDE | Override Mandatory Access Control (MAC). Implemented for the Smack Linux Security Module (LSM). |
| NET\_ADMIN | Perform various network-related operations. |
| SYSLOG | Perform privileged syslog(2) operations. |
| DAC\_READ\_SEARCH | Bypass file read permission checks and directory read and execute permission checks. |
| LINUX\_IMMUTABLE | Set the FS\_APPEND\_FL and FS\_IMMUTABLE\_FL i-node flags. |
| NET\_BROADCAST | Make socket broadcasts, and listen to multicasts. |
| IPC\_LOCK | Lock memory (mlock(2), mlockall(2), mmap(2), shmctl(2)). |
| IPC\_OWNER | Bypass permission checks for operations on System V IPC objects. |
| SYS\_PTRACE | Trace arbitrary processes using ptrace(2). |
| SYS\_BOOT | Use reboot(2) and kexec\_load(2), reboot and load a new kernel for later execution. |
| LEASE | Establish leases on arbitrary files (see fcntl(2)). |
| WAKE\_ALARM | Trigger something that will wake up the system. |
| BLOCK\_SUSPEND | Employ features that can block system suspend. |

Further reference information is available on the [capabilities(7) - Linux man page](http://man7.org/linux/man-pages/man7/capabilities.7.html)

Both flags support the value ALL, so if the operator wants to have all capabilities but MKNOD they could use:

$ docker run --cap-add=ALL --cap-drop=MKNOD ...

For interacting with the network stack, instead of using --privileged they should use --cap-add=NET\_ADMIN to modify the network interfaces.

$ docker run -it --rm ubuntu:14.04 ip link add dummy0 type dummy

RTNETLINK answers: Operation not permitted

$ docker run -it --rm --cap-add=NET\_ADMIN ubuntu:14.04 ip link add dummy0 type dummy

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

$ docker run --rm -it --cap-add SYS\_ADMIN sshfs sshfs sven@10.10.10.20:/home/sven /mnt

fuse: failed to open /dev/fuse: Operation not permitted

$ docker run --rm -it --device /dev/fuse sshfs sshfs sven@10.10.10.20:/home/sven /mnt

fusermount: mount failed: Operation not permitted

$ docker run --rm -it --cap-add SYS\_ADMIN --device /dev/fuse sshfs

# sshfs sven@10.10.10.20:/home/sven /mnt

The authenticity of host '10.10.10.20 (10.10.10.20)' can't be established.

ECDSA key fingerprint is 25:34:85:75:25:b0:17:46:05:19:04:93:b5:dd:5f:c6.

Are you sure you want to continue connecting (yes/no)? yes

sven@10.10.10.20's password:

root@30aa0cfaf1b5:/# ls -la /mnt/src/docker

total 1516

drwxrwxr-x 1 1000 1000 4096 Dec 4 06:08 .

drwxrwxr-x 1 1000 1000 4096 Dec 4 11:46 ..

-rw-rw-r-- 1 1000 1000 16 Oct 8 00:09 .dockerignore

-rwxrwxr-x 1 1000 1000 464 Oct 8 00:09 .drone.yml

drwxrwxr-x 1 1000 1000 4096 Dec 4 06:11 .git

-rw-rw-r-- 1 1000 1000 461 Dec 4 06:08 .gitignore

....

The default seccomp profile will adjust to the selected capabilities, in order to allow use of facilities allowed by the capabilities, so you should not have to adjust this, since Docker 1.12. In Docker 1.10 and 1.11 this did not happen and it may be necessary to use a custom seccomp profile or use --security-opt seccomp=unconfined when adding capabilities.

**Logging drivers (--log-driver)**

The container can have a different logging driver than the Docker daemon. Use the --log-driver=VALUE with the docker run command to configure the container’s logging driver. The following options are supported:

| **Driver** | **Description** |
| --- | --- |
| none | Disables any logging for the container. docker logs won’t be available with this driver. |
| json-file | Default logging driver for Docker. Writes JSON messages to file. No logging options are supported for this driver. |
| syslog | Syslog logging driver for Docker. Writes log messages to syslog. |
| journald | Journald logging driver for Docker. Writes log messages to journald. |
| gelf | Graylog Extended Log Format (GELF) logging driver for Docker. Writes log messages to a GELF endpoint likeGraylog or Logstash. |
| fluentd | Fluentd logging driver for Docker. Writes log messages to fluentd (forward input). |
| awslogs | Amazon CloudWatch Logs logging driver for Docker. Writes log messages to Amazon CloudWatch Logs |
| splunk | Splunk logging driver for Docker. Writes log messages to splunk using Event Http Collector. |

The docker logs command is available only for the json-file and journald logging drivers. For detailed information on working with logging drivers, see [Configure logging drivers](https://docs.docker.com/config/containers/logging/configure/).

**Overriding Dockerfile image defaults**

When a developer builds an image from a [*Dockerfile*](https://docs.docker.com/engine/reference/builder/) or when she commits it, the developer can set a number of default parameters that take effect when the image starts up as a container.

Four of the Dockerfile commands cannot be overridden at runtime: FROM, MAINTAINER, RUN, and ADD. Everything else has a corresponding override in docker run. We’ll go through what the developer might have set in each Dockerfile instruction and how the operator can override that setting.

* [CMD (Default Command or Options)](https://docs.docker.com/engine/reference/run/#cmd-default-command-or-options)
* [ENTRYPOINT (Default Command to Execute at Runtime)](https://docs.docker.com/engine/reference/run/#entrypoint-default-command-to-execute-at-runtime)
* [EXPOSE (Incoming Ports)](https://docs.docker.com/engine/reference/run/#expose-incoming-ports)
* [ENV (Environment Variables)](https://docs.docker.com/engine/reference/run/#env-environment-variables)
* [HEALTHCHECK](https://docs.docker.com/engine/reference/run/#healthcheck)
* [VOLUME (Shared Filesystems)](https://docs.docker.com/engine/reference/run/#volume-shared-filesystems)
* [USER](https://docs.docker.com/engine/reference/run/#user)
* [WORKDIR](https://docs.docker.com/engine/reference/run/#workdir)

**CMD (default command or options)**

Recall the optional COMMAND in the Docker commandline:

$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]

This command is optional because the person who created the IMAGE may have already provided a default COMMAND using the Dockerfile CMD instruction. As the operator (the person running a container from the image), you can override that CMD instruction just by specifying a new COMMAND.

If the image also specifies an ENTRYPOINT then the CMD or COMMAND get appended as arguments to the ENTRYPOINT.

**ENTRYPOINT (default command to execute at runtime)**

--entrypoint="": Overwrite the default entrypoint set by the image

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 string to specify the new ENTRYPOINT. Here is an example of how to run a shell in a container that has been set up to automatically run something else (like /usr/bin/redis-server):

$ docker run -it --entrypoint /bin/bash example/redis

or two examples of how to pass more parameters to that ENTRYPOINT:

$ docker run -it --entrypoint /bin/bash example/redis -c ls -l

$ docker run -it --entrypoint /usr/bin/redis-cli example/redis --help

You can reset a containers entrypoint by passing an empty string, for example:

$ docker run -it --entrypoint="" mysql bash

**Note**: Passing --entrypoint will clear out any default command set on the image (i.e. any CMD instruction in the Dockerfile used to build it).

**EXPOSE (incoming ports)**

The following run command options work with container networking:

--expose=[]: Expose a port or a range of ports inside the container.

These are additional to those exposed by the `EXPOSE` instruction

-P : Publish all exposed ports to the host interfaces

-p=[] : Publish a container's port or a range of ports to the host

format: ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort | containerPort

Both hostPort and containerPort can be specified as a

range of ports. When specifying ranges for both, the

number of container ports in the range must match the

number of host ports in the range, for example:

-p 1234-1236:1234-1236/tcp

When specifying a range for hostPort only, the

containerPort must not be a range. In this case the

container port is published somewhere within the

specified hostPort range. (e.g., `-p 1234-1236:1234/tcp`)

(use 'docker port' to see the actual mapping)

--link="" : Add link to another container (<name or id>:alias or <name or id>)

With the exception of the EXPOSE directive, an image developer hasn’t got much control over networking. The EXPOSE instruction defines the initial incoming ports that provide services. These ports are available to processes inside the container. An operator can use the --expose option to add to the exposed ports.

To expose a container’s internal port, an operator can start the container with the -P or -p flag. The exposed port is accessible on the host and the ports are available to any client that can reach the host.

The -P option publishes all the ports to the host interfaces. Docker binds each exposed port to a random port on the host. The range of ports are within an *ephemeral port range* defined by /proc/sys/net/ipv4/ip\_local\_port\_range. Use the -p flag to explicitly map a single port or range of ports.

The port number inside the container (where the service listens) does not need to match the port number exposed on the outside of the container (where clients connect). For example, inside the container an HTTP service is listening on port 80 (and so the image developer specifies EXPOSE 80 in the Dockerfile). At runtime, the port might be bound to 42800 on the host. To find the mapping between the host ports and the exposed ports, use docker port.

If the operator uses --link when starting a new client container in the default bridge network, then the client container can access the exposed port via a private networking interface. If --link is used when starting a container in a user-defined network as described in [*Networking overview*](https://docs.docker.com/network/), it will provide a named alias for the container being linked to.

**ENV (environment variables)**

Docker automatically sets some environment variables when creating a Linux container. Docker does not set any environment variables when creating a Windows container.

The following environment variables are set for Linux containers:

| **Variable** | **Value** |
| --- | --- |
| HOME | Set based on the value of USER |
| HOSTNAME | The hostname associated with the container |
| PATH | Includes popular directories, such as /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin |
| TERM | xterm if the container is allocated a pseudo-TTY |

Additionally, the operator can **set any environment variable** in the container by using one or more -e flags, even overriding those mentioned above, or already defined by the developer with a Dockerfile ENV. If the operator names an environment variable without specifying a value, then the current value of the named variable is propagated into the container’s environment:

$ export today=Wednesday

$ docker run -e "deep=purple" -e today --rm alpine env

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

HOSTNAME=d2219b854598

deep=purple

today=Wednesday

HOME=/root

PS C:\> docker run --rm -e "foo=bar" microsoft/nanoserver cmd /s /c set

ALLUSERSPROFILE=C:\ProgramData

APPDATA=C:\Users\ContainerAdministrator\AppData\Roaming

CommonProgramFiles=C:\Program Files\Common Files

CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files

CommonProgramW6432=C:\Program Files\Common Files

COMPUTERNAME=C2FAEFCC8253

ComSpec=C:\Windows\system32\cmd.exe

foo=bar

LOCALAPPDATA=C:\Users\ContainerAdministrator\AppData\Local

NUMBER\_OF\_PROCESSORS=8

OS=Windows\_NT

Path=C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Users\ContainerAdministrator\AppData\Local\Microsoft\WindowsApps

PATHEXT=.COM;.EXE;.BAT;.CMD

PROCESSOR\_ARCHITECTURE=AMD64

PROCESSOR\_IDENTIFIER=Intel64 Family 6 Model 62 Stepping 4, GenuineIntel

PROCESSOR\_LEVEL=6

PROCESSOR\_REVISION=3e04

ProgramData=C:\ProgramData

ProgramFiles=C:\Program Files

ProgramFiles(x86)=C:\Program Files (x86)

ProgramW6432=C:\Program Files

PROMPT=$P$G

PUBLIC=C:\Users\Public

SystemDrive=C:

SystemRoot=C:\Windows

TEMP=C:\Users\ContainerAdministrator\AppData\Local\Temp

TMP=C:\Users\ContainerAdministrator\AppData\Local\Temp

USERDOMAIN=User Manager

USERNAME=ContainerAdministrator

USERPROFILE=C:\Users\ContainerAdministrator

windir=C:\Windows

Similarly the operator can set the **HOSTNAME** (Linux) or **COMPUTERNAME** (Windows) with -h.

**HEALTHCHECK**

--health-cmd Command to run to check health

--health-interval Time between running the check

--health-retries Consecutive failures needed to report unhealthy

--health-timeout Maximum time to allow one check to run

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

--no-healthcheck Disable any container-specified HEALTHCHECK

Example:

$ docker run --name=test -d \

--health-cmd='stat /etc/passwd || exit 1' \

--health-interval=2s \

busybox sleep 1d

$ sleep 2; docker inspect --format='{{.State.Health.Status}}' test

healthy

$ docker exec test rm /etc/passwd

$ sleep 2; docker inspect --format='{{json .State.Health}}' test

{

"Status": "unhealthy",

"FailingStreak": 3,

"Log": [

{

"Start": "2016-05-25T17:22:04.635478668Z",

"End": "2016-05-25T17:22:04.7272552Z",

"ExitCode": 0,

"Output": " File: /etc/passwd\n Size: 334 \tBlocks: 8 IO Block: 4096 regular file\nDevice: 32h/50d\tInode: 12 Links: 1\nAccess: (0664/-rw-rw-r--) Uid: ( 0/ root) Gid: ( 0/ root)\nAccess: 2015-12-05 22:05:32.000000000\nModify: 2015..."

},

{

"Start": "2016-05-25T17:22:06.732900633Z",

"End": "2016-05-25T17:22:06.822168935Z",

"ExitCode": 0,

"Output": " File: /etc/passwd\n Size: 334 \tBlocks: 8 IO Block: 4096 regular file\nDevice: 32h/50d\tInode: 12 Links: 1\nAccess: (0664/-rw-rw-r--) Uid: ( 0/ root) Gid: ( 0/ root)\nAccess: 2015-12-05 22:05:32.000000000\nModify: 2015..."

},

{

"Start": "2016-05-25T17:22:08.823956535Z",

"End": "2016-05-25T17:22:08.897359124Z",

"ExitCode": 1,

"Output": "stat: can't stat '/etc/passwd': No such file or directory\n"

},

{

"Start": "2016-05-25T17:22:10.898802931Z",

"End": "2016-05-25T17:22:10.969631866Z",

"ExitCode": 1,

"Output": "stat: can't stat '/etc/passwd': No such file or directory\n"

},

{

"Start": "2016-05-25T17:22:12.971033523Z",

"End": "2016-05-25T17:22:13.082015516Z",

"ExitCode": 1,

"Output": "stat: can't stat '/etc/passwd': No such file or directory\n"

}

]

}

The health status is also displayed in the docker ps output.

**TMPFS (mount tmpfs filesystems)**

--tmpfs=[]: Create a tmpfs mount with: container-dir[:<options>],

where the options are identical to the Linux

'mount -t tmpfs -o' command.

The example below mounts an empty tmpfs into the container with the rw, noexec, nosuid, and size=65536k options.

$ docker run -d --tmpfs /run:rw,noexec,nosuid,size=65536k my\_image

**VOLUME (shared filesystems)**

-v, --volume=[host-src:]container-dest[:<options>]: Bind mount a volume.

The comma-delimited `options` are [rw|ro], [z|Z],

[[r]shared|[r]slave|[r]private], and [nocopy].

The 'host-src' is an absolute path or a name value.

If neither 'rw' or 'ro' is specified then the volume is mounted in

read-write mode.

The `nocopy` mode is used to disable automatically copying the requested volume

path in the container to the volume storage location.

For named volumes, `copy` is the default mode. Copy modes are not supported

for bind-mounted volumes.

--volumes-from="": Mount all volumes from the given container(s)

**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.

The volumes commands are complex enough to have their own documentation in section [*Use volumes*](https://docs.docker.com/storage/volumes/). A developer can define one or more VOLUME’s associated with an image, but only the operator can give access from one container to another (or from a container to a volume mounted on the host).

The container-dest must always be an absolute path such as /src/docs. The host-src can either be an absolute path or a name value. If you supply an absolute path for the host-dir, Docker bind-mounts to the path you specify. If you supply a name, Docker creates a named volume by that name.

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).

For example, you can specify either /foo or foo for a host-src value. If you supply the /foo value, Docker creates a bind mount. If you supply the foo specification, Docker creates a named volume.

**USER**

root (id = 0) is the default user within a container. The image developer can create additional users. Those users are accessible by name. When passing a numeric ID, the user does not have to exist in the container.

The developer can set a default user to run the first process with the Dockerfile USER instruction. When starting a container, the operator can override the USER instruction by passing the -u option.

-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 ]

**Note:** if you pass a numeric uid, it must be in the range of 0-2147483647.

**WORKDIR**

The default working directory for running binaries within a container is the root directory (/), but the developer can set a different default with the Dockerfile WORKDIR command. The operator can override this with:

-w="": Working directory inside the container

# Docker (base command)

## Description

The base command for the Docker CLI.

## Child commands

| Command | Description |
| --- | --- |
| [docker attach](https://docs.docker.com/engine/reference/commandline/attach/) | Attach local standard input, output, and error streams to a running container |
| [docker build](https://docs.docker.com/engine/reference/commandline/build/) | Build an image from a Dockerfile |
| [docker builder](https://docs.docker.com/engine/reference/commandline/builder/) | Manage builds |
| [docker checkpoint](https://docs.docker.com/engine/reference/commandline/checkpoint/) | Manage checkpoints |
| [docker commit](https://docs.docker.com/engine/reference/commandline/commit/) | Create a new image from a container’s changes |
| [docker config](https://docs.docker.com/engine/reference/commandline/config/) | Manage Docker configs |
| [docker container](https://docs.docker.com/engine/reference/commandline/container/) | Manage containers |
| [docker context](https://docs.docker.com/engine/reference/commandline/context/) | Manage contexts |
| [docker cp](https://docs.docker.com/engine/reference/commandline/cp/) | Copy files/folders between a container and the local filesystem |
| [docker create](https://docs.docker.com/engine/reference/commandline/create/) | Create a new container |
| [docker diff](https://docs.docker.com/engine/reference/commandline/diff/) | Inspect changes to files or directories on a container’s filesystem |
| [docker events](https://docs.docker.com/engine/reference/commandline/events/) | Get real time events from the server |
| [docker exec](https://docs.docker.com/engine/reference/commandline/exec/) | Run a command in a running container |
| [docker export](https://docs.docker.com/engine/reference/commandline/export/) | Export a container’s filesystem as a tar archive |
| [docker history](https://docs.docker.com/engine/reference/commandline/history/) | Show the history of an image |
| [docker image](https://docs.docker.com/engine/reference/commandline/image/) | Manage images |
| [docker images](https://docs.docker.com/engine/reference/commandline/images/) | List images |
| [docker import](https://docs.docker.com/engine/reference/commandline/import/) | Import the contents from a tarball to create a filesystem image |
| [docker info](https://docs.docker.com/engine/reference/commandline/info/) | Display system-wide information |
| [docker inspect](https://docs.docker.com/engine/reference/commandline/inspect/) | Return low-level information on Docker objects |
| [docker kill](https://docs.docker.com/engine/reference/commandline/kill/) | Kill one or more running containers |
| [docker load](https://docs.docker.com/engine/reference/commandline/load/) | Load an image from a tar archive or STDIN |
| [docker login](https://docs.docker.com/engine/reference/commandline/login/) | Log in to a Docker registry |
| [docker logout](https://docs.docker.com/engine/reference/commandline/logout/) | Log out from a Docker registry |
| [docker logs](https://docs.docker.com/engine/reference/commandline/logs/) | Fetch the logs of a container |
| [docker manifest](https://docs.docker.com/engine/reference/commandline/manifest/) | Manage Docker image manifests and manifest lists |
| [docker network](https://docs.docker.com/engine/reference/commandline/network/) | Manage networks |
| [docker node](https://docs.docker.com/engine/reference/commandline/node/) | Manage Swarm nodes |
| [docker pause](https://docs.docker.com/engine/reference/commandline/pause/) | Pause all processes within one or more containers |
| [docker plugin](https://docs.docker.com/engine/reference/commandline/plugin/) | Manage plugins |
| [docker port](https://docs.docker.com/engine/reference/commandline/port/) | List port mappings or a specific mapping for the container |
| [docker ps](https://docs.docker.com/engine/reference/commandline/ps/) | List containers |
| [docker pull](https://docs.docker.com/engine/reference/commandline/pull/) | Pull an image or a repository from a registry |
| [docker push](https://docs.docker.com/engine/reference/commandline/push/) | Push an image or a repository to a registry |
| [docker rename](https://docs.docker.com/engine/reference/commandline/rename/) | Rename a container |
| [docker restart](https://docs.docker.com/engine/reference/commandline/restart/) | Restart one or more containers |
| [docker rm](https://docs.docker.com/engine/reference/commandline/rm/) | Remove one or more containers |
| [docker rmi](https://docs.docker.com/engine/reference/commandline/rmi/) | Remove one or more images |
| [docker run](https://docs.docker.com/engine/reference/commandline/run/) | Run a command in a new container |
| [docker save](https://docs.docker.com/engine/reference/commandline/save/) | Save one or more images to a tar archive (streamed to STDOUT by default) |
| [docker search](https://docs.docker.com/engine/reference/commandline/search/) | Search the Docker Hub for images |
| [docker secret](https://docs.docker.com/engine/reference/commandline/secret/) | Manage Docker secrets |
| [docker service](https://docs.docker.com/engine/reference/commandline/service/) | Manage services |
| [docker stack](https://docs.docker.com/engine/reference/commandline/stack/) | Manage Docker stacks |
| [docker start](https://docs.docker.com/engine/reference/commandline/start/) | Start one or more stopped containers |
| [docker stats](https://docs.docker.com/engine/reference/commandline/stats/) | Display a live stream of container(s) resource usage statistics |
| [docker stop](https://docs.docker.com/engine/reference/commandline/stop/) | Stop one or more running containers |
| [docker swarm](https://docs.docker.com/engine/reference/commandline/swarm/) | Manage Swarm |
| [docker system](https://docs.docker.com/engine/reference/commandline/system/) | Manage Docker |
| [docker tag](https://docs.docker.com/engine/reference/commandline/tag/) | Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE |
| [docker top](https://docs.docker.com/engine/reference/commandline/top/) | Display the running processes of a container |
| [docker trust](https://docs.docker.com/engine/reference/commandline/trust/) | Manage trust on Docker images |
| [docker unpause](https://docs.docker.com/engine/reference/commandline/unpause/) | Unpause all processes within one or more containers |
| [docker update](https://docs.docker.com/engine/reference/commandline/update/) | Update configuration of one or more containers |
| [docker version](https://docs.docker.com/engine/reference/commandline/version/) | Show the Docker version information |
| [docker volume](https://docs.docker.com/engine/reference/commandline/volume/) | Manage volumes |
| [docker wait](https://docs.docker.com/engine/reference/commandline/wait/) | Block until one or more containers stop, then print their exit codes |

# docker container

**Description**

Manage containers

**Usage**

docker container COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker container attach](#_docker_container_attach) | Attach local standard input, output, and error streams to a running container |
| [docker container commit](https://docs.docker.com/engine/reference/commandline/container_commit/) | Create a new image from a container’s changes |
| [docker container cp](https://docs.docker.com/engine/reference/commandline/container_cp/) | Copy files/folders between a container and the local filesystem |
| [docker container create](https://docs.docker.com/engine/reference/commandline/container_create/) | Create a new container |
| [docker container diff](https://docs.docker.com/engine/reference/commandline/container_diff/) | Inspect changes to files or directories on a container’s filesystem |
| [docker container exec](https://docs.docker.com/engine/reference/commandline/container_exec/) | Run a command in a running container |
| [docker container export](https://docs.docker.com/engine/reference/commandline/container_export/) | Export a container’s filesystem as a tar archive |
| [docker container inspect](https://docs.docker.com/engine/reference/commandline/container_inspect/) | Display detailed information on one or more containers |
| [docker container kill](https://docs.docker.com/engine/reference/commandline/container_kill/) | Kill one or more running containers |
| [docker container logs](https://docs.docker.com/engine/reference/commandline/container_logs/) | Fetch the logs of a container |
| [docker container ls](https://docs.docker.com/engine/reference/commandline/container_ls/) | List containers |
| [docker container pause](https://docs.docker.com/engine/reference/commandline/container_pause/) | Pause all processes within one or more containers |
| [docker container port](https://docs.docker.com/engine/reference/commandline/container_port/) | List port mappings or a specific mapping for the container |
| [docker container prune](https://docs.docker.com/engine/reference/commandline/container_prune/) | Remove all stopped containers |
| [docker container rename](https://docs.docker.com/engine/reference/commandline/container_rename/) | Rename a container |
| [docker container restart](https://docs.docker.com/engine/reference/commandline/container_restart/) | Restart one or more containers |
| [docker container rm](https://docs.docker.com/engine/reference/commandline/container_rm/) | Remove one or more containers |
| [docker container run](https://docs.docker.com/engine/reference/commandline/container_run/) | Run a command in a new container |
| [docker container start](https://docs.docker.com/engine/reference/commandline/container_start/) | Start one or more stopped containers |
| [docker container stats](https://docs.docker.com/engine/reference/commandline/container_stats/) | Display a live stream of container(s) resource usage statistics |
| [docker container stop](https://docs.docker.com/engine/reference/commandline/container_stop/) | Stop one or more running containers |
| [docker container top](https://docs.docker.com/engine/reference/commandline/container_top/) | Display the running processes of a container |
| [docker container unpause](https://docs.docker.com/engine/reference/commandline/container_unpause/) | Unpause all processes within one or more containers |
| [docker container update](https://docs.docker.com/engine/reference/commandline/container_update/) | Update configuration of one or more containers |
| [docker container wait](https://docs.docker.com/engine/reference/commandline/container_wait/) | Block until one or more containers stop, then print their exit codes |

## docker container attach

### Description

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

### Usage

docker container attach [OPTIONS] CONTAINER

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --detach-keys |  | Override the key sequence for detaching a container |
| --no-stdin |  | Do not attach STDIN |
| --sig-proxy | true | Proxy all received signals to the process |

## docker container commit

### Description

Create a new image from a container’s changes

### Usage

docker container commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --author , -a |  | Author (e.g., “John Hannibal Smith [hannibal@a-team.com](mailto:hannibal@a-team.com)”) |
| --change , -c |  | Apply Dockerfile instruction to the created image |
| --message , -m |  | Commit message |
| --pause , -p | true | Pause container during commit |

## docker container cp

### Description

Copy files/folders between a container and the local filesystem

### Usage

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

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

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --archive , -a |  | Archive mode (copy all uid/gid information) |
| --follow-link , -L |  | Always follow symbol link in SRC\_PATH |

### Extended description

Copy files/folders between a container and the local filesystem

Use ‘-‘ as the source to read a tar archive from stdin and extract it to a directory destination in a container. Use ‘-‘ as the destination to stream a tar archive of a container source to stdout.

## docker container create

### Description

Create a new container

### Usage

docker container create [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) |
| --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 |
| --domainname |  | Container NIS domain name |
| --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 |
| --gpus |  | [API 1.40+](https://docs.docker.com/engine/api/v1.40/) GPU devices to add to the container (‘all’ to pass all GPUs) |
| --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) |
| --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 |
| --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 |
| --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 |

## docker container diff

### Description

Inspect changes to files or directories on a container’s filesystem

### Usage

docker container diff CONTAINER

## docker container exec

### Description

Run a command in a running container

### Usage

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

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --detach , -d |  | Detached mode: run command in the background |
| --detach-keys |  | Override the key sequence for detaching a container |
| --env , -e |  | [API 1.25+](https://docs.docker.com/engine/api/v1.25/) Set environment variables |
| --interactive , -i |  | Keep STDIN open even if not attached |
| --privileged |  | Give extended privileges to the command |
| --tty , -t |  | Allocate a pseudo-TTY |
| --user , -u |  | Username or UID (format: <name|uid>[:<group|gid>]) |
| --workdir , -w |  | [API 1.35+](https://docs.docker.com/engine/api/v1.35/) Working directory inside the container |

## docker container export

### Description

Export a container’s filesystem as a tar archive

### Usage

docker container export [OPTIONS] CONTAINER

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --output , -o |  | Write to a file, instead of STDOUT |

## docker container inspect

### Description

Display detailed information on one or more containers

### Usage

docker container inspect [OPTIONS] CONTAINER [CONTAINER...]

## docker container kill

### Description

Kill one or more running containers

### Usage

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

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --signal , -s | KILL | Signal to send to the container |

## docker container ls

### Description

List containers

### Usage

docker container ls [OPTIONS]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all , -a |  | Show all containers (default shows just running) |
| --filter , -f |  | Filter output based on conditions provided |
| --format |  | Pretty-print containers using a Go template |
| --last , -n | -1 | Show n last created containers (includes all states) |
| --latest , -l |  | Show the latest created container (includes all states) |
| --no-trunc |  | Don’t truncate output |
| --quiet , -q |  | Only display numeric IDs |
| --size , -s |  | Display total file sizes |

## docker container pause

### Description

Pause all processes within one or more containers

### Usage

docker container pause CONTAINER [CONTAINER...]

## docker container port

### Description

List port mappings or a specific mapping for the container

### Usage

docker container port CONTAINER [PRIVATE\_PORT[/PROTO]]

## docker container prune

### Description

Remove all stopped containers

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

### Usage

docker container prune [OPTIONS]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --filter |  | Provide filter values (e.g. ‘until=') |
| --force , -f |  | Do not prompt for confirmation |

## docker container rename

### Description

Rename a container

### Usage

docker container rename CONTAINER NEW\_NAME

## docker container restart

### Description

Restart one or more containers

### Usage

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

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --time , -t | 10 | Seconds to wait for stop before killing the container |

## docker container rm

### Description

Remove one or more containers

### Usage

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

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --force , -f |  | Force the removal of a running container (uses SIGKILL) |
| --link , -l |  | Remove the specified link |
| --volumes , -v |  | Remove anonymous volumes associated with the container |

## docker container run

### Description

Run a command in a new container

### Usage

docker container 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) |
| --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 |
| --domainname |  | Container NIS domain name |
| --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 |
| --gpus |  | [API 1.40+](https://docs.docker.com/engine/api/v1.40/) GPU devices to add to the container (‘all’ to pass all GPUs) |
| --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) |
| --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 |
| --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 |

## docker container start

### Description

Start one or more stopped containers

### Usage

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

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --attach , -a |  | Attach STDOUT/STDERR and forward signals |
| --checkpoint |  | [experimental (daemon)](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file) Restore from this checkpoint |
| --checkpoint-dir |  | [experimental (daemon)](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file) Use a custom checkpoint storage directory |
| --detach-keys |  | Override the key sequence for detaching a container |
| --interactive , -i |  | Attach container’s STDIN |

## docker container stats

### Description

Display a live stream of container(s) resource usage statistics

### Usage

docker container stats [OPTIONS] [CONTAINER...]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all , -a |  | Show all containers (default shows just running) |
| --format |  | Pretty-print images using a Go template |
| --no-stream |  | Disable streaming stats and only pull the first result |
| --no-trunc |  | Do not truncate output |

## docker container stop

### Description

Stop one or more running containers

### Usage

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

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --time , -t | 10 | Seconds to wait for stop before killing it |

## docker container top

### Description

Display the running processes of a container

### Usage

docker container top CONTAINER [ps OPTIONS]

## docker container unpause

### Description

Unpause all processes within one or more containers

### Usage

docker container unpause CONTAINER [CONTAINER...]

## docker container update

### Description

Update configuration of one or more containers

### Usage

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

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --blkio-weight |  | Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0) |
| --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 the CPU real-time period in microseconds |
| --cpu-rt-runtime |  | [API 1.25+](https://docs.docker.com/engine/api/v1.25/) Limit the CPU real-time runtime in microseconds |
| --cpu-shares , -c |  | CPU shares (relative weight) |
| --cpus |  | [API 1.29+](https://docs.docker.com/engine/api/v1.29/) 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) |
| --kernel-memory |  | Kernel memory limit |
| --memory , -m |  | Memory limit |
| --memory-reservation |  | Memory soft limit |
| --memory-swap |  | Swap limit equal to memory plus swap: ‘-1’ to enable unlimited swap |
| --pids-limit |  | [API 1.40+](https://docs.docker.com/engine/api/v1.40/) Tune container pids limit (set -1 for unlimited) |
| --restart |  | Restart policy to apply when a container exits |

## docker container wait

### Description

Block until one or more containers stop, then print their exit codes

### Usage

docker container wait CONTAINER [CONTAINER...]

**docker attach**

**Description**

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

**Usage**

docker attach [OPTIONS] CONTAINER

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --detach-keys |  | Override the key sequence for detaching a container |
| --no-stdin |  | Do not attach STDIN |
| --sig-proxy | true | Proxy all received signals to the process |

**Extended description**

Use docker attach to attach your terminal’s standard input, output, and error (or any combination of the three) to a running container using the container’s ID or name. This allows you to view its ongoing output or to control it interactively, as though the commands were running directly in your terminal.

**Note:** The attach command will display the output of the ENTRYPOINT/CMD process. This can appear as if the attach command is hung when in fact the process may simply not be interacting with the terminal at that time.

You can attach to the same contained process multiple times simultaneously, from different sessions on the Docker host.

To stop a container, use CTRL-c. This key sequence sends SIGKILL to the container. If --sig-proxy is true (the default),CTRL-c sends a SIGINT to the container. If the container was run with -i and -t, you can detach from a container and leave it running using the CTRL-p CTRL-q key sequence.

**Note:** A process running as PID 1 inside a container is treated specially by Linux: it ignores any signal with the default action. So, the process will not terminate on SIGINT or SIGTERM unless it is coded to do so.

It is forbidden to redirect the standard input of a docker attach command while attaching to a tty-enabled container (i.e.: launched with -t).

While a client is connected to container’s stdio using docker attach, Docker uses a ~1MB memory buffer to maximize the throughput of the application. If this buffer is filled, the speed of the API connection will start to have an effect on the process output writing speed. This is similar to other applications like SSH. Because of this, it is not recommended to run performance critical applications that generate a lot of output in the foreground over a slow client connection. Instead, users should use the docker logs command to get access to the logs.

**Override the detach sequence**

If you want, you can configure an override the Docker key sequence for detach. This is useful if the Docker default sequence conflicts with key sequence you use for other applications. There are two ways to define your own detach key sequence, as a per-container override or as a configuration property on your entire configuration.

To override the sequence for an individual container, use the --detach-keys="<sequence>" flag with the docker attach command. The format of the <sequence> is either a letter [a-Z], or the ctrl- combined with any of the following:

* a-z (a single lowercase alpha character )
* @ (at sign)
* [ (left bracket)
* \\ (two backward slashes)
* \_ (underscore)
* ^ (caret)

These a, ctrl-a, X, or ctrl-\\ values are all examples of valid key sequences. To configure a different configuration default key sequence for all containers, see [**Configuration file** section](https://docs.docker.com/engine/reference/commandline/cli/#configuration-files).

**Examples**

**Attach to and detach from a running container**

$ docker run -d --name topdemo ubuntu /usr/bin/top -b

$ docker attach topdemo

top - 02:05:52 up 3:05, 0 users, load average: 0.01, 0.02, 0.05

Tasks: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie

Cpu(s): 0.1%us, 0.2%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

Mem: 373572k total, 355560k used, 18012k free, 27872k buffers

Swap: 786428k total, 0k used, 786428k free, 221740k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

1 root 20 0 17200 1116 912 R 0 0.3 0:00.03 top

top - 02:05:55 up 3:05, 0 users, load average: 0.01, 0.02, 0.05

Tasks: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie

Cpu(s): 0.0%us, 0.2%sy, 0.0%ni, 99.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

Mem: 373572k total, 355244k used, 18328k free, 27872k buffers

Swap: 786428k total, 0k used, 786428k free, 221776k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

1 root 20 0 17208 1144 932 R 0 0.3 0:00.03 top

top - 02:05:58 up 3:06, 0 users, load average: 0.01, 0.02, 0.05

Tasks: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie

Cpu(s): 0.2%us, 0.3%sy, 0.0%ni, 99.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

Mem: 373572k total, 355780k used, 17792k free, 27880k buffers

Swap: 786428k total, 0k used, 786428k free, 221776k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

1 root 20 0 17208 1144 932 R 0 0.3 0:00.03 top

^C$

$ echo $?

0

$ docker ps -a | grep topdemo

7998ac8581f9 ubuntu:14.04 "/usr/bin/top -b" 38 seconds ago Exited (0) 21 seconds ago topdemo

**Get the exit code of the container’s command**

And in this second example, you can see the exit code returned by the bash process is returned by the docker attach command to its caller too:

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

275c44472aebd77c926d4527885bb09f2f6db21d878c75f0a1c212c03d3bcfab

$ docker attach test

root@f38c87f2a42d:/# exit 13

exit

$ echo $?

13

$ docker ps -a | grep test

275c44472aeb debian:7 "/bin/bash" 26 seconds ago Exited (13) 17 seconds ago test

**docker build**

**Description**

Build an image from a Dockerfile

**Usage**

docker build [OPTIONS] PATH | URL | -

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --add-host |  | Add a custom host-to-IP mapping (host:ip) |
| --build-arg |  | Set build-time variables |
| --cache-from |  | Images to consider as cache sources |
| --cgroup-parent |  | Optional parent cgroup for the container |
| --compress |  | Compress the build context using gzip |
| --cpu-period |  | Limit the CPU CFS (Completely Fair Scheduler) period |
| --cpu-quota |  | Limit the CPU CFS (Completely Fair Scheduler) quota |
| --cpu-shares , -c |  | CPU shares (relative weight) |
| --cpuset-cpus |  | CPUs in which to allow execution (0-3, 0,1) |
| --cpuset-mems |  | MEMs in which to allow execution (0-3, 0,1) |
| --disable-content-trust | true | Skip image verification |
| --file , -f |  | Name of the Dockerfile (Default is ‘PATH/Dockerfile’) |
| --force-rm |  | Always remove intermediate containers |
| --iidfile |  | Write the image ID to the file |
| --isolation |  | Container isolation technology |
| --label |  | Set metadata for an image |
| --memory , -m |  | Memory limit |
| --memory-swap |  | Swap limit equal to memory plus swap: ‘-1’ to enable unlimited swap |
| --network |  | [API 1.25+](https://docs.docker.com/engine/api/v1.25/) Set the networking mode for the RUN instructions during build |
| --no-cache |  | Do not use cache when building the image |
| --output , -o |  | [API 1.40+](https://docs.docker.com/engine/api/v1.40/) Output destination (format: type=local,dest=path) |
| --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 |
| --progress | auto | Set type of progress output (auto, plain, tty). Use plain to show container output |
| --pull |  | Always attempt to pull a newer version of the image |
| --quiet , -q |  | Suppress the build output and print image ID on success |
| --rm | true | Remove intermediate containers after a successful build |
| --secret |  | [API 1.39+](https://docs.docker.com/engine/api/v1.39/) Secret file to expose to the build (only if BuildKit enabled): id=mysecret,src=/local/secret |
| --security-opt |  | Security options |
| --shm-size |  | Size of /dev/shm |
| --squash |  | [experimental (daemon)](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file)[API 1.25+](https://docs.docker.com/engine/api/v1.25/) Squash newly built layers into a single new layer |
| --ssh |  | [API 1.39+](https://docs.docker.com/engine/api/v1.39/) SSH agent socket or keys to expose to the build (only if BuildKit enabled) (format: default|[=|[,]]) |
| --stream |  | [experimental (daemon)](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file)[API 1.31+](https://docs.docker.com/engine/api/v1.31/) Stream attaches to server to negotiate build context |
| --tag , -t |  | Name and optionally a tag in the ‘name:tag’ format |
| --target |  | Set the target build stage to build. |
| --ulimit |  | Ulimit options |

**Extended description**

The docker build command builds Docker images from a Dockerfile and a “context”. A build’s context is the set of files located in the specified PATH or URL. The build process can refer to any of the files in the context. For example, your build can use a [*COPY*](https://docs.docker.com/engine/reference/builder/#copy) instruction to reference a file in the context.

The URL parameter can refer to three kinds of resources: Git repositories, pre-packaged tarball contexts and plain text files.

**Git repositories**

When the URL parameter points to the location of a Git repository, the repository acts as the build context. The system recursively fetches the repository and its submodules. The commit history is not preserved. A repository is first pulled into a temporary directory on your local host. After that succeeds, the directory is sent to the Docker daemon as the context. Local copy gives you the ability to access private repositories using local user credentials, VPN’s, and so forth.

**Note:** If the URL parameter contains a fragment the system will recursively clone the repository and its submodules using a git clone --recursive command.

Git URLs accept context configuration in their fragment section, separated by a colon :. The first part represents the reference that Git will check out, and can be either a branch, a tag, or a remote reference. The second part represents a subdirectory inside the repository that will be used as a build context.

For example, run this command to use a directory called docker in the branch container:

$ docker build https://github.com/docker/rootfs.git#container:docker

The following table represents all the valid suffixes with their build contexts:

| **Build Syntax Suffix** | **Commit Used** | **Build Context Used** |
| --- | --- | --- |
| myrepo.git | refs/heads/master | / |
| myrepo.git#mytag | refs/tags/mytag | / |
| myrepo.git#mybranch | refs/heads/mybranch | / |
| myrepo.git#pull/42/head | refs/pull/42/head | / |
| myrepo.git#:myfolder | refs/heads/master | /myfolder |
| myrepo.git#master:myfolder | refs/heads/master | /myfolder |
| myrepo.git#mytag:myfolder | refs/tags/mytag | /myfolder |
| myrepo.git#mybranch:myfolder | refs/heads/mybranch | /myfolder |

**Tarball contexts**

If you pass an URL to a remote tarball, the URL itself is sent to the daemon:

$ docker build http://server/context.tar.gz

The download operation will be performed on the host the Docker daemon is running on, which is not necessarily the same host from which the build command is being issued. The Docker daemon will fetch context.tar.gz and use it as the build context. Tarball contexts must be tar archives conforming to the standard tar UNIX format and can be compressed with any one of the ‘xz’, ‘bzip2’, ‘gzip’ or ‘identity’ (no compression) formats.

**Text files**

Instead of specifying a context, you can pass a single Dockerfile in the URL or pipe the file in via STDIN. To pipe a Dockerfile from STDIN:

$ docker build - < Dockerfile

With Powershell on Windows, you can run:

Get-Content Dockerfile | docker build -

If you use STDIN or specify a URL pointing to a plain text file, the system places the contents into a file called Dockerfile, and any -f, --file option is ignored. In this scenario, there is no context.

By default the docker build command will look for a Dockerfile at the root of the build context. The -f, --file, option lets you specify the path to an alternative file to use instead. This is useful in cases where the same set of files are used for multiple builds. The path must be to a file within the build context. If a relative path is specified then it is interpreted as relative to the root of the context.

In most cases, it’s best to put each Dockerfile in an empty directory. Then, add to that directory only the files needed for building the Dockerfile. To increase the build’s performance, you can exclude files and directories by adding a .dockerignore file to that directory as well. For information on creating one, see the [.dockerignore file](https://docs.docker.com/engine/reference/builder/#dockerignore-file).

If the Docker client loses connection to the daemon, the build is canceled. This happens if you interrupt the Docker client with CTRL-c or if the Docker client is killed for any reason. If the build initiated a pull which is still running at the time the build is cancelled, the pull is cancelled as well.

**Examples**

**Build with PATH**

$ docker build .

Uploading context 10240 bytes

Step 1/3 : FROM busybox

Pulling repository busybox

---> e9aa60c60128MB/2.284 MB (100%) endpoint: https://cdn-registry-1.docker.io/v1/

Step 2/3 : RUN ls -lh /

---> Running in 9c9e81692ae9

total 24

drwxr-xr-x 2 root root 4.0K Mar 12 2013 bin

drwxr-xr-x 5 root root 4.0K Oct 19 00:19 dev

drwxr-xr-x 2 root root 4.0K Oct 19 00:19 etc

drwxr-xr-x 2 root root 4.0K Nov 15 23:34 lib

lrwxrwxrwx 1 root root 3 Mar 12 2013 lib64 -> lib

dr-xr-xr-x 116 root root 0 Nov 15 23:34 proc

lrwxrwxrwx 1 root root 3 Mar 12 2013 sbin -> bin

dr-xr-xr-x 13 root root 0 Nov 15 23:34 sys

drwxr-xr-x 2 root root 4.0K Mar 12 2013 tmp

drwxr-xr-x 2 root root 4.0K Nov 15 23:34 usr

---> b35f4035db3f

Step 3/3 : CMD echo Hello world

---> Running in 02071fceb21b

---> f52f38b7823e

Successfully built f52f38b7823e

Removing intermediate container 9c9e81692ae9

Removing intermediate container 02071fceb21b

This example specifies that the PATH is ., and so all the files in the local directory get tard and sent to the Docker daemon. The PATH specifies where to find the files for the “context” of the build on the Docker daemon. Remember that the daemon could be running on a remote machine and that no parsing of the Dockerfile happens at the client side (where you’re running docker build). That means that *all* the files at PATH get sent, not just the ones listed to [*ADD*](https://docs.docker.com/engine/reference/builder/#add) in the Dockerfile.

The transfer of context from the local machine to the Docker daemon is what the docker client means when you see the “Sending build context” message.

If you wish to keep the intermediate containers after the build is complete, you must use --rm=false. This does not affect the build cache.

**Build with URL**

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

This will clone the GitHub repository and use the cloned repository as context. The Dockerfile at the root of the repository is used as Dockerfile. You can specify an arbitrary Git repository by using the git:// or git@ scheme.

$ docker build -f ctx/Dockerfile http://server/ctx.tar.gz

Downloading context: http://server/ctx.tar.gz [===================>] 240 B/240 B

Step 1/3 : FROM busybox

---> 8c2e06607696

Step 2/3 : ADD ctx/container.cfg /

---> e7829950cee3

Removing intermediate container b35224abf821

Step 3/3 : CMD /bin/ls

---> Running in fbc63d321d73

---> 3286931702ad

Removing intermediate container fbc63d321d73

Successfully built 377c409b35e4

This sends the URL http://server/ctx.tar.gz to the Docker daemon, which downloads and extracts the referenced tarball. The -f ctx/Dockerfile parameter specifies a path inside ctx.tar.gz to the Dockerfile that is used to build the image. Any ADD commands in that Dockerfile that refers to local paths must be relative to the root of the contents inside ctx.tar.gz. In the example above, the tarball contains a directory ctx/, so the ADD ctx/container.cfg / operation works as expected.

**Build with -**

$ docker build - < Dockerfile

This will read a Dockerfile from STDIN without context. Due to the lack of a context, no contents of any local directory will be sent to the Docker daemon. Since there is no context, a Dockerfile ADD only works if it refers to a remote URL.

$ docker build - < context.tar.gz

This will build an image for a compressed context read from STDIN. Supported formats are: bzip2, gzip and xz.

**Use a .dockerignore file**

$ docker build .

Uploading context 18.829 MB

Uploading context

Step 1/2 : FROM busybox

---> 769b9341d937

Step 2/2 : CMD echo Hello world

---> Using cache

---> 99cc1ad10469

Successfully built 99cc1ad10469

$ echo ".git" > .dockerignore

$ docker build .

Uploading context 6.76 MB

Uploading context

Step 1/2 : FROM busybox

---> 769b9341d937

Step 2/2 : CMD echo Hello world

---> Using cache

---> 99cc1ad10469

Successfully built 99cc1ad10469

This example shows the use of the .dockerignore file to exclude the .git directory from the context. Its effect can be seen in the changed size of the uploaded context. The builder reference contains detailed information on [creating a .dockerignore file](https://docs.docker.com/engine/reference/builder/#dockerignore-file).

When using the [BuildKit backend](https://docs.docker.com/engine/reference/builder/#buildkit), docker build searches for a .dockerignore file relative to the Dockerfile name. For example, running docker build -f myapp.Dockerfile . will first look for an ignore file named myapp.Dockerfile.dockerignore. If such a file is not found, the .dockerignore file is used if present. Using a Dockerfile based .dockerignore is useful if a project contains multiple Dockerfiles that expect to ignore different sets of files.

**Tag an image (-t)**

$ docker build -t vieux/apache:2.0 .

This will build like the previous example, but it will then tag the resulting image. The repository name will be vieux/apache and the tag will be 2.0. [Read more about valid tags](https://docs.docker.com/engine/reference/commandline/tag/).

You can apply multiple tags to an image. For example, you can apply the latest tag to a newly built image and add another tag that references a specific version. For example, to tag an image both as whenry/fedora-jboss:latest and whenry/fedora-jboss:v2.1, use the following:

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

**Specify a Dockerfile (-f)**

$ docker build -f Dockerfile.debug .

This will use a file called Dockerfile.debug for the build instructions instead of Dockerfile.

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

The above command will use the current directory as the build context and read a Dockerfile from stdin.

$ docker build -f dockerfiles/Dockerfile.debug -t myapp\_debug .

$ docker build -f dockerfiles/Dockerfile.prod -t myapp\_prod .

The above commands will build the current build context (as specified by the .) twice, once using a debug version of a Dockerfile and once using a production version.

$ cd /home/me/myapp/some/dir/really/deep

$ docker build -f /home/me/myapp/dockerfiles/debug /home/me/myapp

$ docker build -f ../../../../dockerfiles/debug /home/me/myapp

These two docker build commands do the exact same thing. They both use the contents of the debug file instead of looking for a Dockerfile and will use /home/me/myapp as the root of the build context. Note that debug is in the directory structure of the build context, regardless of how you refer to it on the command line.

**Note:** docker build will return a no such file or directory error if the file or directory does not exist in the uploaded context. This may happen if there is no context, or if you specify a file that is elsewhere on the Host system. The context is limited to the current directory (and its children) for security reasons, and to ensure repeatable builds on remote Docker hosts. This is also the reason why ADD ../file will not work.

**Use a custom parent cgroup (--cgroup-parent)**

When docker build is run with the --cgroup-parent option the containers used in the build will be run with the [corresponding docker run flag](https://docs.docker.com/engine/reference/run/#specifying-custom-cgroups).

**Set ulimits in container (--ulimit)**

Using the --ulimit option with docker build will cause each build step’s container to be started using those [--ulimit flag values](https://docs.docker.com/engine/reference/commandline/run/#set-ulimits-in-container-ulimit).

**Set build-time variables (--build-arg)**

You can use ENV instructions in a Dockerfile to define variable values. These values persist in the built image. However, often persistence is not what you want. Users want to specify variables differently depending on which host they build an image on.

A good example is http\_proxy or source versions for pulling intermediate files. The ARG instruction lets Dockerfile authors define values that users can set at build-time using the --build-arg flag:

$ docker build --build-arg HTTP\_PROXY=http://10.20.30.2:1234 --build-arg FTP\_PROXY=http://40.50.60.5:4567 .

This flag allows you to pass the build-time variables that are accessed like regular environment variables in the RUN instruction of the Dockerfile. Also, these values don’t persist in the intermediate or final images like ENV values do. You must add --build-arg for each build argument.

Using this flag will not alter the output you see when the ARG lines from the Dockerfile are echoed during the build process.

For detailed information on using ARG and ENV instructions, see the [Dockerfile reference](https://docs.docker.com/engine/reference/builder/).

You may also use the --build-arg flag without a value, in which case the value from the local environment will be propagated into the Docker container being built:

$ export HTTP\_PROXY=http://10.20.30.2:1234

$ docker build --build-arg HTTP\_PROXY .

This is similar to how docker run -e works. Refer to the [docker run documentation](https://docs.docker.com/engine/reference/commandline/run/#set-environment-variables--e---env---env-file) for more information.

**Optional security options (--security-opt)**

This flag is only supported on a daemon running on Windows, and only supports the credentialspec option. The credentialspec must be in the format file://spec.txt or registry://keyname.

**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. On Microsoft Windows, you can specify these values:

| **Value** | **Description** |
| --- | --- |
| 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. |

Specifying the --isolation flag without a value is the same as setting --isolation="default".

**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 build --add-host=docker:10.180.0.1 .

**Specifying target build stage (--target)**

When building a Dockerfile with multiple build stages, --target can be used to specify an intermediate build stage by name as a final stage for the resulting image. Commands after the target stage will be skipped.

FROM debian AS build-env

...

FROM alpine AS production-env

...

$ docker build -t mybuildimage --target build-env .

**Custom build outputs**

By default, a local container image is created from the build result. The --output (or -o) flag allows you to override this behavior, and a specify a custom exporter. For example, custom exporters allow you to export the build artifacts as files on the local filesystem instead of a Docker image, which can be useful for generating local binaries, code generation etc.

The value for --output is a CSV-formatted string defining the exporter type and options. Currently, local and tar exporters are supported. The local exporter writes the resulting build files to a directory on the client side. The tar exporter is similar but writes the files as a single tarball (.tar).

If no type is specified, the value defaults to the output directory of the local exporter. Use a hyphen (-) to write the output tarball to standard output (STDOUT).

The following example builds an image using the current directory (.) as build context, and exports the files to a directory named out in the current directory. If the directory does not exist, Docker creates the directory automatically:

$ docker build -o out .

The example above uses the short-hand syntax, omitting the type options, and thus uses the default (local) exporter. The example below shows the equivalent using the long-hand CSV syntax, specifying both type and dest (destination path):

$ docker build --output type=local,dest=out .

Use the tar type to export the files as a .tar archive:

$ docker build --output type=tar,dest=out.tar .

The example below shows the equivalent when using the short-hand syntax. In this case, - is specified as destination, which automatically selects the tar type, and writes the output tarball to standard output, which is then redirected to the out.tar file:

docker build -o - . > out.tar

The --output option exports all files from the target stage. A common pattern for exporting only specific files is to do multi-stage builds and to copy the desired files to a new scratch stage with [COPY --from](https://docs.docker.com/engine/reference/builder/#copy).

The example Dockerfile below uses a separate stage to collect the build-artifacts for exporting:

FROM golang AS build-stage

RUN go get -u github.com/LK4D4/vndr

FROM scratch AS export-stage

COPY --from=build-stage /go/bin/vndr /

When building the Dockerfile with the -o option, only the files from the final stage are exported to the out directory, in this case, the vndr binary:

$ docker build -o out .

[+] Building 2.3s (7/7) FINISHED

=> [internal] load build definition from Dockerfile 0.1s

=> => transferring dockerfile: 176B 0.0s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> [internal] load metadata for docker.io/library/golang:latest 1.6s

=> [build-stage 1/2] FROM docker.io/library/golang@sha256:2df96417dca0561bf1027742dcc5b446a18957cd28eba6aa79269f23f1846d3f 0.0s

=> => resolve docker.io/library/golang@sha256:2df96417dca0561bf1027742dcc5b446a18957cd28eba6aa79269f23f1846d3f 0.0s

=> CACHED [build-stage 2/2] RUN go get -u github.com/LK4D4/vndr 0.0s

=> [export-stage 1/1] COPY --from=build-stage /go/bin/vndr / 0.2s

=> exporting to client 0.4s

=> => copying files 10.30MB 0.3s

$ ls ./out

vndr

**Note**: This feature requires the BuildKit backend. You can either [enable BuildKit](https://docs.docker.com/engine/reference/builder/#buildkit) or use the [buildx](https://github.com/docker/buildx) plugin which provides more output type options.

**Specifying external cache sources**

In addition to local build cache, the builder can reuse the cache generated from previous builds with the --cache-from flag pointing to an image in the registry.

To use an image as a cache source, cache metadata needs to be written into the image on creation. This can be done by setting --build-arg BUILDKIT\_INLINE\_CACHE=1 when building the image. After that, the built image can be used as a cache source for subsequent builds.

Upon importing the cache, the builder will only pull the JSON metadata from the registry and determine possible cache hits based on that information. If there is a cache hit, the matched layers are pulled into the local environment.

In addition to images, the cache can also be pulled from special cache manifests generated by [buildx](https://github.com/docker/buildx) or the BuildKit CLI (buildctl). These manifests (when built with the type=registry and mode=max options) allow pulling layer data for intermediate stages in multi-stage builds.

The following example builds an image with inline-cache metadata and pushes it to a registry, then uses the image as a cache source on another machine:

$ docker build -t myname/myapp --build-arg BUILDKIT\_INLINE\_CACHE=1 .

$ docker push myname/myapp

After pushing the image, the image is used as cache source on another machine. BuildKit automatically pulls the image from the registry if needed.

# on another machine

$ docker build --cache-from myname/myapp .

**Note**: This feature requires the BuildKit backend. You can either [enable BuildKit](https://docs.docker.com/engine/reference/builder/#buildkit) or use the [buildx](https://github.com/docker/buildx) plugin. The previous builder has limited support for reusing cache from pre-pulled images.

**Squash an image’s layers (--squash) (experimental)**

**Overview**

Once the image is built, squash the new layers into a new image with a single new layer. Squashing does not destroy any existing image, rather it creates a new image with the content of the squashed layers. This effectively makes it look like all Dockerfile commands were created with a single layer. The build cache is preserved with this method.

The --squash option is an experimental feature, and should not be considered stable.

Squashing layers can be beneficial if your Dockerfile produces multiple layers modifying the same files, for example, files that are created in one step, and removed in another step. For other use-cases, squashing images may actually have a negative impact on performance; when pulling an image consisting of multiple layers, layers can be pulled in parallel, and allows sharing layers between images (saving space).

For most use cases, multi-stage builds are a better alternative, as they give more fine-grained control over your build, and can take advantage of future optimizations in the builder. Refer to the [use multi-stage builds](https://docs.docker.com/develop/develop-images/multistage-build/) section in the userguide for more information.

**Known limitations**

The --squash option has a number of known limitations:

* When squashing layers, the resulting image cannot take advantage of layer sharing with other images, and may use significantly more space. Sharing the base image is still supported.
* When using this option you may see significantly more space used due to storing two copies of the image, one for the build cache with all the cache layers in tact, and one for the squashed version.
* While squashing layers may produce smaller images, it may have a negative impact on performance, as a single layer takes longer to extract, and downloading a single layer cannot be parallelized.
* When attempting to squash an image that does not make changes to the filesystem (for example, the Dockerfile only contains ENV instructions), the squash step will fail (see [issue #33823](https://github.com/moby/moby/issues/33823)).

**Prerequisites**

The example on this page is using experimental mode in Docker 1.13.

Experimental mode can be enabled by using the --experimental flag when starting the Docker daemon or setting experimental: true in the daemon.json configuration file.

By default, experimental mode is disabled. To see the current configuration, use the docker version command.

Server:

Version: 1.13.1

API version: 1.26 (minimum version 1.12)

Go version: go1.7.5

Git commit: 092cba3

Built: Wed Feb 8 06:35:24 2017

OS/Arch: linux/amd64

Experimental: false

[...]

To enable experimental mode, users need to restart the docker daemon with the experimental flag enabled.

**Enable Docker experimental**

Experimental features are now included in the standard Docker binaries as of version 1.13.0. For enabling experimental features, you need to start the Docker daemon with --experimental flag. You can also enable the daemon flag via /etc/docker/daemon.json. e.g.

{

"experimental": true

}

Then make sure the experimental flag is enabled:

$ docker version -f '{{.Server.Experimental}}'

true

**Build an image with --squash argument**

The following is an example of docker build with --squash argument

FROM busybox

RUN echo hello > /hello

RUN echo world >> /hello

RUN touch remove\_me /remove\_me

ENV HELLO world

RUN rm /remove\_me

An image named test is built with --squash argument.

$ docker build --squash -t test .

[...]

If everything is right, the history will look like this:

$ docker history test

IMAGE CREATED CREATED BY SIZE COMMENT

4e10cb5b4cac 3 seconds ago 12 B merge sha256:88a7b0112a41826885df0e7072698006ee8f621c6ab99fca7fe9151d7b599702 to sha256:47bcc53f74dc94b1920f0b34f6036096526296767650f223433fe65c35f149eb

<missing> 5 minutes ago /bin/sh -c rm /remove\_me 0 B

<missing> 5 minutes ago /bin/sh -c #(nop) ENV HELLO=world 0 B

<missing> 5 minutes ago /bin/sh -c touch remove\_me /remove\_me 0 B

<missing> 5 minutes ago /bin/sh -c echo world >> /hello 0 B

<missing> 6 minutes ago /bin/sh -c echo hello > /hello 0 B

<missing> 7 weeks ago /bin/sh -c #(nop) CMD ["sh"] 0 B

<missing> 7 weeks ago /bin/sh -c #(nop) ADD file:47ca6e777c36a4cfff 1.113 MB

We could find that all layer’s name is <missing>, and there is a new layer with COMMENT merge.

Test the image, check for /remove\_me being gone, make sure hello\nworld is in /hello, make sure the HELLO envvar’s value is world.

# docker builder

### Description

Manage builds

[API 1.31+](https://docs.docker.com/engine/api/v1.31/)  The client and daemon API must both be at least [1.31](https://docs.docker.com/engine/api/v1.31/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

### Usage

docker builder COMMAND

### Child commands

| Command | Description |
| --- | --- |
| [docker builder build](https://docs.docker.com/engine/reference/commandline/builder_build/) | Build an image from a Dockerfile |
| [docker builder prune](https://docs.docker.com/engine/reference/commandline/builder_prune/) | Remove build cache |

## docker builder build

### Description

Build an image from a Dockerfile

[API 1.31+](https://docs.docker.com/engine/api/v1.31/)  The client and daemon API must both be at least [1.31](https://docs.docker.com/engine/api/v1.31/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

### Usage

docker builder build [OPTIONS] PATH | URL | -

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --add-host |  | Add a custom host-to-IP mapping (host:ip) |
| --build-arg |  | Set build-time variables |
| --cache-from |  | Images to consider as cache sources |
| --cgroup-parent |  | Optional parent cgroup for the container |
| --compress |  | Compress the build context using gzip |
| --cpu-period |  | Limit the CPU CFS (Completely Fair Scheduler) period |
| --cpu-quota |  | Limit the CPU CFS (Completely Fair Scheduler) quota |
| --cpu-shares , -c |  | CPU shares (relative weight) |
| --cpuset-cpus |  | CPUs in which to allow execution (0-3, 0,1) |
| --cpuset-mems |  | MEMs in which to allow execution (0-3, 0,1) |
| --disable-content-trust | true | Skip image verification |
| --file , -f |  | Name of the Dockerfile (Default is ‘PATH/Dockerfile’) |
| --force-rm |  | Always remove intermediate containers |
| --iidfile |  | Write the image ID to the file |
| --isolation |  | Container isolation technology |
| --label |  | Set metadata for an image |
| --memory , -m |  | Memory limit |
| --memory-swap |  | Swap limit equal to memory plus swap: ‘-1’ to enable unlimited swap |
| --network |  | [API 1.25+](https://docs.docker.com/engine/api/v1.25/) Set the networking mode for the RUN instructions during build |
| --no-cache |  | Do not use cache when building the image |
| --output , -o |  | [API 1.40+](https://docs.docker.com/engine/api/v1.40/) Output destination (format: type=local,dest=path) |
| --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 |
| --progress | auto | Set type of progress output (auto, plain, tty). Use plain to show container output |
| --pull |  | Always attempt to pull a newer version of the image |
| --quiet , -q |  | Suppress the build output and print image ID on success |
| --rm | true | Remove intermediate containers after a successful build |
| --secret |  | [API 1.39+](https://docs.docker.com/engine/api/v1.39/) Secret file to expose to the build (only if BuildKit enabled): id=mysecret,src=/local/secret |
| --security-opt |  | Security options |
| --shm-size |  | Size of /dev/shm |
| --squash |  | [experimental (daemon)](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file)[API 1.25+](https://docs.docker.com/engine/api/v1.25/) Squash newly built layers into a single new layer |
| --ssh |  | [API 1.39+](https://docs.docker.com/engine/api/v1.39/) SSH agent socket or keys to expose to the build (only if BuildKit enabled) (format: default|[=|[,]]) |
| --stream |  | [experimental (daemon)](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file)[API 1.31+](https://docs.docker.com/engine/api/v1.31/) Stream attaches to server to negotiate build context |
| --tag , -t |  | Name and optionally a tag in the ‘name:tag’ format |
| --target |  | Set the target build stage to build. |
| --ulimit |  | Ulimit options |

## docker builder prune

### Description

Remove build cache

[API 1.39+](https://docs.docker.com/engine/api/v1.39/)  The client and daemon API must both be at least [1.39](https://docs.docker.com/engine/api/v1.39/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

### Usage

docker builder prune

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all , -a |  | Remove all unused images, not just dangling ones |
| --filter |  | Provide filter values (e.g. ‘unused-for=24h’) |
| --force , -f |  | Do not prompt for confirmation |
| --keep-storage |  | Amount of disk space to keep for cache |
|  |  |  |
|  |  |  |

# docker cp

**Description**

Copy files/folders between a container and the local filesystem

**Usage**

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

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

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --archive , -a |  | Archive mode (copy all uid/gid information) |
| --follow-link , -L |  | Always follow symbol link in SRC\_PATH |

**Parent command**

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

**Extended description**

The docker cp utility copies the contents of SRC\_PATH to the DEST\_PATH. You can copy from the container’s file system to the local machine or the reverse, from the local filesystem to the container. If - is specified for either the SRC\_PATH or DEST\_PATH, you can also stream a tar archive from STDIN or to STDOUT. The CONTAINER can be a running or stopped container. The SRC\_PATH or DEST\_PATH can be a file or directory.

The docker cp command assumes container paths are relative to the container’s / (root) directory. This means supplying the initial forward slash is optional; The command sees compassionate\_darwin:/tmp/foo/myfile.txt and compassionate\_darwin:tmp/foo/myfile.txt as identical. Local machine paths can be an absolute or relative value. The command interprets a local machine’s relative paths as relative to the current working directory where docker cp is run.

The cp command behaves like the Unix cp -a command in that directories are copied recursively with permissions preserved if possible. Ownership is set to the user and primary group at the destination. For example, files copied to a container are created with UID:GID of the root user. Files copied to the local machine are created with the UID:GID of the user which invoked the docker cp command. However, if you specify the -a option, docker cp sets the ownership to the user and primary group at the source. If you specify the -L option, docker cp follows any symbolic link in the SRC\_PATH. docker cp does *not* create parent directories for DEST\_PATH if they do not exist.

Assuming a path separator of /, a first argument of SRC\_PATH and second argument of DEST\_PATH, the behavior is as follows:

* SRC\_PATH specifies a file
  + DEST\_PATH does not exist
    - the file is saved to a file created at DEST\_PATH
  + DEST\_PATH does not exist and ends with /
    - Error condition: the destination directory must exist.
  + DEST\_PATH exists and is a file
    - the destination is overwritten with the source file’s contents
  + DEST\_PATH exists and is a directory
    - the file is copied into this directory using the basename from SRC\_PATH
* SRC\_PATH specifies a directory
  + DEST\_PATH does not exist
    - DEST\_PATH is created as a directory and the *contents* of the source directory are copied into this directory
  + DEST\_PATH exists and is a file
    - Error condition: cannot copy a directory to a file
  + DEST\_PATH exists and is a directory
    - SRC\_PATH does not end with /. (that is: *slash* followed by *dot*)
      * the source directory is copied into this directory
    - SRC\_PATH does end with /. (that is: *slash* followed by *dot*)
      * the *content* of the source directory is copied into this directory

The command requires SRC\_PATH and DEST\_PATH to exist according to the above rules. If SRC\_PATH is local and is a symbolic link, the symbolic link, not the target, is copied by default. To copy the link target and not the link, specify the -L option.

A colon (:) is used as a delimiter between CONTAINER and its path. You can also use : when specifying paths to a SRC\_PATH or DEST\_PATH on a local machine, for example file:name.txt. If you use a : in a local machine path, you must be explicit with a relative or absolute path, for example:

`/path/to/file:name.txt` or `./file:name.txt`

It is not possible to copy certain system files such as resources under /proc, /sys, /dev, [tmpfs](https://docs.docker.com/engine/reference/commandline/run/#mount-tmpfs-tmpfs), and mounts created by the user in the container. However, you can still copy such files by manually running tar in docker exec. Both of the following examples do the same thing in different ways (consider SRC\_PATH and DEST\_PATH are directories):

$ docker exec CONTAINER tar Ccf $(dirname SRC\_PATH) - $(basename SRC\_PATH) | tar Cxf DEST\_PATH -

$ tar Ccf $(dirname SRC\_PATH) - $(basename SRC\_PATH) | docker exec -i CONTAINER tar Cxf DEST\_PATH -

Using - as the SRC\_PATH streams the contents of STDIN as a tar archive. The command extracts the content of the tar to the DEST\_PATH in container’s filesystem. In this case, DEST\_PATH must specify a directory. Using - as the DEST\_PATH streams the contents of the resource as a tar archive to STDOUT.

# docker checkpoint

## Description

Manage checkpoints

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

This command is experimental.

This command is experimental on the Docker daemon. It should not be used in production environments. To enable experimental features on the Docker daemon, edit the [daemon.json](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file) and set experimental to true.

Experimental features provide early access to future product functionality. These features are intended for testing and feedback only as they may change between releases without warning or can be removed entirely from a future release. Experimental features must not be used in production environments. Docker does not offer support for experimental features.

To enable experimental features in the Docker CLI, edit the config.json file and set experimental to enabled.

To enable experimental features from the Docker Desktop menu, click **Settings** (**Preferences** on macOS) > **Command Line** and then turn on the **Enable experimental features** toggle. Click **Apply & Restart**.

## Usage

docker checkpoint COMMAND

## Child commands

| Command | Description |
| --- | --- |
| [docker checkpoint create](https://docs.docker.com/engine/reference/commandline/checkpoint_create/) | Create a checkpoint from a running container |
| [docker checkpoint ls](https://docs.docker.com/engine/reference/commandline/checkpoint_ls/) | List checkpoints for a container |
| [docker checkpoint rm](https://docs.docker.com/engine/reference/commandline/checkpoint_rm/) | Remove a checkpoint |

## docker checkpoint create

## Description

Create a checkpoint from a running container

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

This command is experimental.

This command is experimental on the Docker daemon. It should not be used in production environments. To enable experimental features on the Docker daemon, edit the [daemon.json](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file) and set experimental to true.

Experimental features provide early access to future product functionality. These features are intended for testing and feedback only as they may change between releases without warning or can be removed entirely from a future release. Experimental features must not be used in production environments. Docker does not offer support for experimental features.

To enable experimental features in the Docker CLI, edit the config.json file and set experimental to enabled.

To enable experimental features from the Docker Desktop menu, click **Settings** (**Preferences** on macOS) > **Command Line** and then turn on the **Enable experimental features** toggle. Click **Apply & Restart**.

## Usage

docker checkpoint create [OPTIONS] CONTAINER CHECKPOINT

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --checkpoint-dir |  | Use a custom checkpoint storage directory |
| --leave-running |  | Leave the container running after checkpoint |

## docker checkpoint ls

## Description

List checkpoints for a container

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

This command is experimental.

This command is experimental on the Docker daemon. It should not be used in production environments. To enable experimental features on the Docker daemon, edit the [daemon.json](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file) and set experimental to true.

Experimental features provide early access to future product functionality. These features are intended for testing and feedback only as they may change between releases without warning or can be removed entirely from a future release. Experimental features must not be used in production environments. Docker does not offer support for experimental features.

To enable experimental features in the Docker CLI, edit the config.json file and set experimental to enabled.

To enable experimental features from the Docker Desktop menu, click **Settings** (**Preferences** on macOS) > **Command Line** and then turn on the **Enable experimental features** toggle. Click **Apply & Restart**.

## Usage

docker checkpoint ls [OPTIONS] CONTAINER

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --checkpoint-dir |  | Use a custom checkpoint storage directory |

## docker checkpoint rm

### Description

Remove a checkpoint

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

This command is experimental.

This command is experimental on the Docker daemon. It should not be used in production environments. To enable experimental features on the Docker daemon, edit the [daemon.json](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file) and set experimental to true.

Experimental features provide early access to future product functionality. These features are intended for testing and feedback only as they may change between releases without warning or can be removed entirely from a future release. Experimental features must not be used in production environments. Docker does not offer support for experimental features.

To enable experimental features in the Docker CLI, edit the config.json file and set experimental to enabled.

To enable experimental features from the Docker Desktop menu, click **Settings** (**Preferences** on macOS) > **Command Line** and then turn on the **Enable experimental features** toggle. Click **Apply & Restart**.

### Usage

docker checkpoint rm [OPTIONS] CONTAINER CHECKPOINT

### Options

--checkpoint-dir Use a custom checkpoint storage directory

**docker cluster**

## Description

Docker Cluster

This command is only available on Docker Enterprise Edition.

Learn more about [Docker Enterprise products](https://docs.docker.com/ee/supported-platforms/).

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --dry-run |  | Skip provisioning resources |
| --log-level | warn | Set the logging level (“trace”|”debug”|”info”|”warn”|”error”|”fatal”) |

# docker config

## Description

Manage Docker configs

[API 1.30+](https://docs.docker.com/engine/api/v1.30/)  The client and daemon API must both be at least [1.30](https://docs.docker.com/engine/api/v1.30/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

Swarm This command works with the Swarm orchestrator.

## Usage

docker config COMMAND

## Child commands

| Command | Description |
| --- | --- |
| [docker config create](https://docs.docker.com/engine/reference/commandline/config_create/) | Create a config from a file or STDIN |
| [docker config inspect](https://docs.docker.com/engine/reference/commandline/config_inspect/) | Display detailed information on one or more configs |
| [docker config ls](https://docs.docker.com/engine/reference/commandline/config_ls/) | List configs |
| [docker config rm](https://docs.docker.com/engine/reference/commandline/config_rm/) | Remove one or more configs |

## docker config create

## Description

Create a config from a file or STDIN

[API 1.30+](https://docs.docker.com/engine/api/v1.30/)  The client and daemon API must both be at least [1.30](https://docs.docker.com/engine/api/v1.30/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

Swarm This command works with the Swarm orchestrator.

## Usage

docker config create [OPTIONS] CONFIG file|-

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --label , -l |  | Config labels |
| --template-driver |  | [API 1.37+](https://docs.docker.com/engine/api/v1.37/) Template driver |

## docker config inspect

## Description

Display detailed information on one or more configs

[API 1.30+](https://docs.docker.com/engine/api/v1.30/)  The client and daemon API must both be at least [1.30](https://docs.docker.com/engine/api/v1.30/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

Swarm This command works with the Swarm orchestrator.

## Usage

docker config inspect [OPTIONS] CONFIG [CONFIG...]

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format , -f |  | Format the output using the given Go template |
| --pretty |  | Print the information in a human friendly format |

## docker config ls

## Description

List configs

[API 1.30+](https://docs.docker.com/engine/api/v1.30/)  The client and daemon API must both be at least [1.30](https://docs.docker.com/engine/api/v1.30/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

Swarm This command works with the Swarm orchestrator.

## Usage

docker config ls [OPTIONS]

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --filter , -f |  | Filter output based on conditions provided |
| --format |  | Pretty-print configs using a Go template |
| --quiet , -q |  | Only display IDs |

## docker config rm

## Description

Remove one or more configs

[API 1.30+](https://docs.docker.com/engine/api/v1.30/)  The client and daemon API must both be at least [1.30](https://docs.docker.com/engine/api/v1.30/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

Swarm This command works with the Swarm orchestrator.

## Usage

docker config rm CONFIG [CONFIG...]

**docker commit**

**Description**

Create a new image from a container’s changes

**Usage**

docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --author , -a |  | Author (e.g., “John Hannibal Smith [hannibal@a-team.com](mailto:hannibal@a-team.com)”) |
| --change , -c |  | Apply Dockerfile instruction to the created image |
| --message , -m |  | Commit message |
| --pause , -p | true | Pause container during commit |

**docker create**

**Description**

Create a new container

**Usage**

docker create [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) |
| --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 |
| --domainname |  | Container NIS domain name |
| --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 |
| --gpus |  | [API 1.40+](https://docs.docker.com/engine/api/v1.40/) GPU devices to add to the container (‘all’ to pass all GPUs) |
| --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) |
| --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 |
| --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 |
| --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 create command creates a writeable container layer over the specified image and prepares it for running the specified command. The container ID is then printed to STDOUT. This is similar to docker run -d except the container is never started. You can then use the docker start <container\_id> command to start the container at any point.

This is useful when you want to set up a container configuration ahead of time so that it is ready to start when you need it. The initial status of the new container is created.

Please see the [run command](https://docs.docker.com/engine/reference/commandline/run/) section and the [Docker run reference](https://docs.docker.com/engine/reference/run/) for more details.

**Examples**

**Create and start a container**

$ docker create -t -i fedora bash

6d8af538ec541dd581ebc2a24153a28329acb5268abe5ef868c1f1a261221752

$ docker start -a -i 6d8af538ec5

bash-4.2#

**Initialize volumes**

As of v1.4.0 container volumes are initialized during the docker create phase (i.e., docker run too). For example, this allows you to create the data volume container, and then use it from another container:

$ docker create -v /data --name data ubuntu

240633dfbb98128fa77473d3d9018f6123b99c454b3251427ae190a7d951ad57

$ docker run --rm --volumes-from data ubuntu ls -la /data

total 8

drwxr-xr-x 2 root root 4096 Dec 5 04:10 .

drwxr-xr-x 48 root root 4096 Dec 5 04:11 ..

Similarly, create a host directory bind mounted volume container, which can then be used from the subsequent container:

$ docker create -v /home/docker:/docker --name docker ubuntu

9aa88c08f319cd1e4515c3c46b0de7cc9aa75e878357b1e96f91e2c773029f03

$ docker run --rm --volumes-from docker ubuntu ls -la /docker

total 20

drwxr-sr-x 5 1000 staff 180 Dec 5 04:00 .

drwxr-xr-x 48 root root 4096 Dec 5 04:13 ..

-rw-rw-r-- 1 1000 staff 3833 Dec 5 04:01 .ash\_history

-rw-r--r-- 1 1000 staff 446 Nov 28 11:51 .ashrc

-rw-r--r-- 1 1000 staff 25 Dec 5 04:00 .gitconfig

drwxr-sr-x 3 1000 staff 60 Dec 1 03:28 .local

-rw-r--r-- 1 1000 staff 920 Nov 28 11:51 .profile

drwx--S--- 2 1000 staff 460 Dec 5 00:51 .ssh

drwxr-xr-x 32 1000 staff 1140 Dec 5 04:01 docker

Set storage driver options per container.

$ docker create -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.

**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. On Microsoft Windows, you can specify these values:

| **Value** | **Description** |
| --- | --- |
| 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 if the |
| daemon is running on Windows server, or hyperv if running on Windows client. |  |
| process | Namespace isolation only. |
| hyperv | Hyper-V hypervisor partition-based isolation. |

Specifying the --isolation flag without a value is the same as setting --isolation="default".

**Dealing with dynamically created devices (--device-cgroup-rule)**

Devices available to a container are assigned at creation time. The assigned devices will both be added to the cgroup.allow file and created into the container once it is run. This poses a problem when a new device needs to be added to running container.

One of the solution is to add a more permissive rule to a container allowing it access to a wider range of devices. For example, supposing our container needs access to a character device with major 42 and any number of minor number (added as new devices appear), the following rule would be added:

docker create --device-cgroup-rule='c 42:\* rmw' -name my-container my-image

Then, a user could ask udev to execute a script that would docker exec my-container mknod newDevX c 42 <minor> the required device when it is added.

NOTE: initially present devices still need to be explicitly added to the create/run command

# docker context

## Description

Manage contexts

## Usage

docker context COMMAND

## Child commands

| Command | Description |
| --- | --- |
| [docker context create](https://docs.docker.com/engine/reference/commandline/context_create/) | Create a context |
| [docker context export](https://docs.docker.com/engine/reference/commandline/context_export/) | Export a context to a tar or kubeconfig file |
| [docker context import](https://docs.docker.com/engine/reference/commandline/context_import/) | Import a context from a tar or zip file |
| [docker context inspect](https://docs.docker.com/engine/reference/commandline/context_inspect/) | Display detailed information on one or more contexts |
| [docker context ls](https://docs.docker.com/engine/reference/commandline/context_ls/) | List contexts |
| [docker context rm](https://docs.docker.com/engine/reference/commandline/context_rm/) | Remove one or more contexts |
| [docker context update](https://docs.docker.com/engine/reference/commandline/context_update/) | Update a context |
| [docker context use](https://docs.docker.com/engine/reference/commandline/context_use/) | Set the current docker context |

## docker context create

### Description

Create a context

### Usage

docker context create [OPTIONS] CONTEXT

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --default-stack-orchestrator |  | Default orchestrator for stack operations to use with this context (swarm|kubernetes|all) |
| --description |  | Description of the context |
| --docker |  | set the docker endpoint |
| --from |  | create context from a named context |
| --kubernetes |  | set the kubernetes endpoint |

## docker context export

### Description

Export a context to a tar or kubeconfig file

### Usage

docker context export [OPTIONS] CONTEXT [FILE|-]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --kubeconfig |  | Export as a kubeconfig file |

## docker context import

### Description

Import a context from a tar or zip file

### Usage

docker context import CONTEXT FILE|-

### Parent command

| **Command** | **Description** |
| --- | --- |
| [docker context](https://docs.docker.com/engine/reference/commandline/context) | Manage contexts |

## docker context inspect

### Description

Display detailed information on one or more contexts

### Usage

docker context inspect [OPTIONS] [CONTEXT] [CONTEXT...]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format , -f |  | Format the output using the given Go template |

## docker context ls

### Description

List contexts

### Usage

docker context ls [OPTIONS]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format |  | Pretty-print contexts using a Go template |
| --quiet , -q |  | Only show context names |

## docker context rm

### Description

Remove one or more contexts

### Usage

docker context rm CONTEXT [CONTEXT...]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --force , -f |  | Force the removal of a context in use |

## docker context update

### Description

Update a context

### Usage

docker context update [OPTIONS] CONTEXT

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --default-stack-orchestrator |  | Default orchestrator for stack operations to use with this context (swarm|kubernetes|all) |
| --description |  | Description of the context |
| --docker |  | set the docker endpoint |
| --kubernetes |  | set the kubernetes endpoint |

## docker context use

### Description

Set the current docker context

### Usage

docker context use CONTEXT

### Parent command

| **Command** | **Description** |
| --- | --- |
| [docker context](https://docs.docker.com/engine/reference/commandline/context) | Manage contexts |

# docker diff

## Description

Inspect changes to files or directories on a container’s filesystem

## Usage

docker diff CONTAINER

## Parent command

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

## Extended description

List the changed files and directories in a container᾿s filesystem since the container was created. Three different types of change are tracked:

| **Symbol** | **Description** |
| --- | --- |
| A | A file or directory was added |
| D | A file or directory was deleted |
| C | A file or directory was changed |

You can use the full or shortened container ID or the container name set using docker run --name option.

## Examples

Inspect the changes to an nginx container:

$ docker diff 1fdfd1f54c1b

C /dev

C /dev/console

C /dev/core

C /dev/stdout

C /dev/fd

C /dev/ptmx

C /dev/stderr

C /dev/stdin

C /run

A /run/nginx.pid

C /var/lib/nginx/tmp

A /var/lib/nginx/tmp/client\_body

A /var/lib/nginx/tmp/fastcgi

A /var/lib/nginx/tmp/proxy

A /var/lib/nginx/tmp/scgi

A /var/lib/nginx/tmp/uwsgi

C /var/log/nginx

A /var/log/nginx/access.log

A /var/log/nginx/error.log

# docker events

## Description

Get real time events from the server

## Usage

docker events [OPTIONS]

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --filter , -f |  | Filter output based on conditions provided |
| --format |  | Format the output using the given Go template |
| --since |  | Show all events created since timestamp |
| --until |  | Stream events until this timestamp |

# docker export

## Description

Export a container’s filesystem as a tar archive

## Usage

docker export [OPTIONS] CONTAINER

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --output , -o |  | Write to a file, instead of STDOUT |

# docker exec

## Description

Run a command in a running container

## Usage

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

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --detach , -d |  | Detached mode: run command in the background |
| --detach-keys |  | Override the key sequence for detaching a container |
| --env , -e |  | [API 1.25+](https://docs.docker.com/engine/api/v1.25/) Set environment variables |
| --interactive , -i |  | Keep STDIN open even if not attached |
| --privileged |  | Give extended privileges to the command |
| --tty , -t |  | Allocate a pseudo-TTY |
| --user , -u |  | Username or UID (format: <name|uid>[:<group|gid>]) |
| --workdir , -w |  | [API 1.35+](https://docs.docker.com/engine/api/v1.35/) 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 exec command runs a new command in a running container.

The command started using docker exec only runs while the container’s primary process (PID 1) is running, and it is not restarted if the container is restarted.

COMMAND will run in the default directory of the container. If the underlying image has a custom directory specified with the WORKDIR directive in its Dockerfile, this will be used instead.

COMMAND should be an executable, a chained or a quoted command will not work. Example: docker exec -ti my\_container "echo a && echo b" will not work, but docker exec -ti my\_container sh -c "echo a && echo b" will.

## Examples

### Run docker exec on a running container

First, start a container.

$ docker run --name ubuntu\_bash --rm -i -t ubuntu bash

This will create a container named ubuntu\_bash and start a Bash session.

Next, execute a command on the container.

$ docker exec -d ubuntu\_bash touch /tmp/execWorks

This will create a new file /tmp/execWorks inside the running container ubuntu\_bash, in the background.

Next, execute an interactive bash shell on the container.

$ docker exec -it ubuntu\_bash bash

This will create a new Bash session in the container ubuntu\_bash.

Next, set an environment variable in the current bash session.

$ docker exec -it -e VAR=1 ubuntu\_bash bash

This will create a new Bash session in the container ubuntu\_bash with environment variable $VAR set to “1”. Note that this environment variable will only be valid on the current Bash session.

By default docker exec command runs in the same working directory set when container was created.

$ docker exec -it ubuntu\_bash pwd

/

You can select working directory for the command to execute into

$ docker exec -it -w /root ubuntu\_bash pwd

/root

### Try to run docker exec on a paused container

If the container is paused, then the docker exec command will fail with an error:

$ docker pause test

test

$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1ae3b36715d2 ubuntu:latest "bash" 17 seconds ago Up 16 seconds (Paused) test

$ docker exec test ls

FATA[0000] Error response from daemon: Container test is paused, unpause the container before exec

$ echo $?

1

# docker history

## Description

Show the history of an image

## Usage

docker history [OPTIONS] IMAGE

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format |  | Pretty-print images using a Go template |
| --human , -H | true | Print sizes and dates in human readable format |
| --no-trunc |  | Don’t truncate output |
| --quiet , -q |  | Only show numeric IDs |

# docker import

## Description

Import the contents from a tarball to create a filesystem image

## Usage

docker import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --change , -c |  | Apply Dockerfile instruction to the created image |
| --message , -m |  | Set commit message for imported image |
| --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 |

## Parent command

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

## Extended description

You can specify a URL or - (dash) to take data directly from STDIN. The URL can point to an archive (.tar, .tar.gz, .tgz, .bzip, .tar.xz, or .txz) containing a filesystem or to an individual file on the Docker host. If you specify an archive, Docker untars it in the container relative to the / (root). If you specify an individual file, you must specify the full path within the host. To import from a remote location, specify a URI that begins with the http:// or https:// protocol.

The --change option will apply Dockerfile instructions to the image that is created. Supported Dockerfile instructions: CMD|ENTRYPOINT|ENV|EXPOSE|ONBUILD|USER|VOLUME|WORKDIR

## Examples

### Import from a remote location

This will create a new untagged image.

$ docker import http://example.com/exampleimage.tgz

### Import from a local file

* Import to docker via pipe and STDIN.
* $ cat exampleimage.tgz | docker import - exampleimagelocal:new
* Import with a commit message.
* $ cat exampleimage.tgz | docker import --message "New image imported from tarball" - exampleimagelocal:new
* Import to docker from a local archive.
* $ docker import /path/to/exampleimage.tgz

### Import from a local directory

$ sudo tar -c . | docker import - exampleimagedir

### Import from a local directory with new configurations

$ sudo tar -c . | docker import --change "ENV DEBUG true" - exampleimagedir

Note the sudo in this example – you must preserve the ownership of the files (especially root ownership) during the archiving with tar. If you are not root (or the sudo command) when you tar, then the ownerships might not get preserved.

# docker image

## Description

Manage images

## Usage

docker image COMMAND

## Child commands

| Command | Description |
| --- | --- |
| [docker image build](https://docs.docker.com/engine/reference/commandline/image_build/) | Build an image from a Dockerfile |
| [docker image history](https://docs.docker.com/engine/reference/commandline/image_history/) | Show the history of an image |
| [docker image import](https://docs.docker.com/engine/reference/commandline/image_import/) | Import the contents from a tarball to create a filesystem image |
| [docker image inspect](https://docs.docker.com/engine/reference/commandline/image_inspect/) | Display detailed information on one or more images |
| [docker image load](https://docs.docker.com/engine/reference/commandline/image_load/) | Load an image from a tar archive or STDIN |
| [docker image ls](https://docs.docker.com/engine/reference/commandline/image_ls/) | List images |
| [docker image prune](https://docs.docker.com/engine/reference/commandline/image_prune/) | Remove unused images |
| [docker image pull](https://docs.docker.com/engine/reference/commandline/image_pull/) | Pull an image or a repository from a registry |
| [docker image push](https://docs.docker.com/engine/reference/commandline/image_push/) | Push an image or a repository to a registry |
| [docker image rm](https://docs.docker.com/engine/reference/commandline/image_rm/) | Remove one or more images |
| [docker image save](https://docs.docker.com/engine/reference/commandline/image_save/) | Save one or more images to a tar archive (streamed to STDOUT by default) |
| [docker image tag](https://docs.docker.com/engine/reference/commandline/image_tag/) | Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE |

## docker image build

### Description

Build an image from a Dockerfile

### Usage

docker image build [OPTIONS] PATH | URL | -

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --add-host |  | Add a custom host-to-IP mapping (host:ip) |
| --build-arg |  | Set build-time variables |
| --cache-from |  | Images to consider as cache sources |
| --cgroup-parent |  | Optional parent cgroup for the container |
| --compress |  | Compress the build context using gzip |
| --cpu-period |  | Limit the CPU CFS (Completely Fair Scheduler) period |
| --cpu-quota |  | Limit the CPU CFS (Completely Fair Scheduler) quota |
| --cpu-shares , -c |  | CPU shares (relative weight) |
| --cpuset-cpus |  | CPUs in which to allow execution (0-3, 0,1) |
| --cpuset-mems |  | MEMs in which to allow execution (0-3, 0,1) |
| --disable-content-trust | true | Skip image verification |
| --file , -f |  | Name of the Dockerfile (Default is ‘PATH/Dockerfile’) |
| --force-rm |  | Always remove intermediate containers |
| --iidfile |  | Write the image ID to the file |
| --isolation |  | Container isolation technology |
| --label |  | Set metadata for an image |
| --memory , -m |  | Memory limit |
| --memory-swap |  | Swap limit equal to memory plus swap: ‘-1’ to enable unlimited swap |
| --network |  | [API 1.25+](https://docs.docker.com/engine/api/v1.25/) Set the networking mode for the RUN instructions during build |
| --no-cache |  | Do not use cache when building the image |
| --output , -o |  | [API 1.40+](https://docs.docker.com/engine/api/v1.40/) Output destination (format: type=local,dest=path) |
| --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 |
| --progress | auto | Set type of progress output (auto, plain, tty). Use plain to show container output |
| --pull |  | Always attempt to pull a newer version of the image |
| --quiet , -q |  | Suppress the build output and print image ID on success |
| --rm | true | Remove intermediate containers after a successful build |
| --secret |  | [API 1.39+](https://docs.docker.com/engine/api/v1.39/) Secret file to expose to the build (only if BuildKit enabled): id=mysecret,src=/local/secret |
| --security-opt |  | Security options |
| --shm-size |  | Size of /dev/shm |
| --squash |  | [experimental (daemon)](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file)[API 1.25+](https://docs.docker.com/engine/api/v1.25/) Squash newly built layers into a single new layer |
| --ssh |  | [API 1.39+](https://docs.docker.com/engine/api/v1.39/) SSH agent socket or keys to expose to the build (only if BuildKit enabled) (format: default|[=|[,]]) |
| --stream |  | [experimental (daemon)](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file)[API 1.31+](https://docs.docker.com/engine/api/v1.31/) Stream attaches to server to negotiate build context |
| --tag , -t |  | Name and optionally a tag in the ‘name:tag’ format |
| --target |  | Set the target build stage to build. |
| --ulimit |  | Ulimit options |

## docker image history

### Description

Show the history of an image

### Usage

docker image history [OPTIONS] IMAGE

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format |  | Pretty-print images using a Go template |
| --human , -H | true | Print sizes and dates in human readable format |
| --no-trunc |  | Don’t truncate output |
| --quiet , -q |  | Only show numeric IDs |

## docker image import

### Description

Import the contents from a tarball to create a filesystem image

### Usage

docker image import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --change , -c |  | Apply Dockerfile instruction to the created image |
| --message , -m |  | Set commit message for imported image |
| --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 |

## docker image inspect

### Description

Display detailed information on one or more images

### Usage

docker image inspect [OPTIONS] IMAGE [IMAGE...]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format , -f |  | Format the output using the given Go template |

## docker image load

### Description

Load an image from a tar archive or STDIN

### Usage

docker image load [OPTIONS]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --input , -i |  | Read from tar archive file, instead of STDIN |
| --quiet , -q |  | Suppress the load output |

## docker image ls

### Description

List images

### Usage

docker image ls [OPTIONS] [REPOSITORY[:TAG]]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all , -a |  | Show all images (default hides intermediate images) |
| --digests |  | Show digests |
| --filter , -f |  | Filter output based on conditions provided |
| --format |  | Pretty-print images using a Go template |
| --no-trunc |  | Don’t truncate output |
| --quiet , -q |  | Only show numeric IDs |

## docker image prune

### Description

Remove unused images

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

### Usage

docker image prune [OPTIONS]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all , -a |  | Remove all unused images, not just dangling ones |
| --filter |  | Provide filter values (e.g. ‘until=') |
| --force , -f |  | Do not prompt for confirmation |

## docker image pull

### Description

Pull an image or a repository from a registry

### Usage

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

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all-tags , -a |  | Download all tagged images in the repository |
| --disable-content-trust | true | Skip image verification |
| --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 |
| --quiet , -q |  | Suppress verbose output |

## docker image push

### Description

Push an image or a repository to a registry

### Usage

docker image push [OPTIONS] NAME[:TAG]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --disable-content-trust | true | Skip image signing |

## docker image rm

### Description

Remove one or more images

### Usage

docker image rm [OPTIONS] IMAGE [IMAGE...]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --force , -f |  | Force removal of the image |
| --no-prune |  | Do not delete untagged parents |

## docker image save

**Description**

Save one or more images to a tar archive (streamed to STDOUT by default)

**Usage**

docker image save [OPTIONS] IMAGE [IMAGE...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --output , -o |  | Write to a file, instead of STDOUT |

## docker image tag

### Description

Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

### Usage

docker image tag SOURCE\_IMAGE[:TAG] TARGET\_IMAGE[:TAG]

**docker info**

**Description**

Display system-wide information

**Usage**

docker info [OPTIONS]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format , -f |  | Format the output using the given Go template |

**docker inspect**

**Description**

Return low-level information on Docker objects

**Usage**

docker inspect [OPTIONS] NAME|ID [NAME|ID...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format , -f |  | Format the output using the given Go template |
| --size , -s |  | Display total file sizes if the type is container |
| --type |  | Return JSON for specified type |

# docker images

## Description

List images

## Usage

docker images [OPTIONS] [REPOSITORY[:TAG]]

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all , -a |  | Show all images (default hides intermediate images) |
| --digests |  | Show digests |
| --filter , -f |  | Filter output based on conditions provided |
| --format |  | Pretty-print images using a Go template |
| --no-trunc |  | Don’t truncate output |
| --quiet , -q |  | Only show numeric IDs |

## Parent command

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

## Extended description

The default docker images will show all top level images, their repository and tags, and their size.

Docker images have intermediate layers that increase reusability, decrease disk usage, and speed up docker build by allowing each step to be cached. These intermediate layers are not shown by default.

The SIZE is the cumulative space taken up by the image and all its parent images. This is also the disk space used by the contents of the Tar file created when you docker save an image.

An image will be listed more than once if it has multiple repository names or tags. This single image (identifiable by its matching IMAGE ID) uses up the SIZE listed only once.

## Examples

### List the most recently created images

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

<none> <none> 77af4d6b9913 19 hours ago 1.089 GB

committ latest b6fa739cedf5 19 hours ago 1.089 GB

<none> <none> 78a85c484f71 19 hours ago 1.089 GB

docker latest 30557a29d5ab 20 hours ago 1.089 GB

<none> <none> 5ed6274db6ce 24 hours ago 1.089 GB

postgres 9 746b819f315e 4 days ago 213.4 MB

postgres 9.3 746b819f315e 4 days ago 213.4 MB

postgres 9.3.5 746b819f315e 4 days ago 213.4 MB

postgres latest 746b819f315e 4 days ago 213.4 MB

### List images by name and tag

The docker images command takes an optional [REPOSITORY[:TAG]] argument that restricts the list to images that match the argument. If you specify REPOSITORYbut no TAG, the docker images command lists all images in the given repository.

For example, to list all images in the “java” repository, run this command :

$ docker images java

REPOSITORY TAG IMAGE ID CREATED SIZE

java 8 308e519aac60 6 days ago 824.5 MB

java 7 493d82594c15 3 months ago 656.3 MB

java latest 2711b1d6f3aa 5 months ago 603.9 MB

The [REPOSITORY[:TAG]] value must be an “exact match”. This means that, for example, docker images jav does not match the image java.

If both REPOSITORY and TAG are provided, only images matching that repository and tag are listed. To find all local images in the “java” repository with tag “8” you can use:

$ docker images java:8

REPOSITORY TAG IMAGE ID CREATED SIZE

java 8 308e519aac60 6 days ago 824.5 MB

If nothing matches REPOSITORY[:TAG], the list is empty.

$ docker images java:0

REPOSITORY TAG IMAGE ID CREATED SIZE

### List the full length image IDs

$ docker images --no-trunc

REPOSITORY TAG IMAGE ID CREATED SIZE

<none> <none> sha256:77af4d6b9913e693e8d0b4b294fa62ade6054e6b2f1ffb617ac955dd63fb0182 19 hours ago 1.089 GB

committest latest sha256:b6fa739cedf5ea12a620a439402b6004d057da800f91c7524b5086a5e4749c9f 19 hours ago 1.089 GB

<none> <none> sha256:78a85c484f71509adeaace20e72e941f6bdd2b25b4c75da8693efd9f61a37921 19 hours ago 1.089 GB

docker latest sha256:30557a29d5abc51e5f1d5b472e79b7e296f595abcf19fe6b9199dbbc809c6ff4 20 hours ago 1.089 GB

<none> <none> sha256:0124422dd9f9cf7ef15c0617cda3931ee68346455441d66ab8bdc5b05e9fdce5 20 hours ago 1.089 GB

<none> <none> sha256:18ad6fad340262ac2a636efd98a6d1f0ea775ae3d45240d3418466495a19a81b 22 hours ago 1.082 GB

<none> <none> sha256:f9f1e26352f0a3ba6a0ff68167559f64f3e21ff7ada60366e2d44a04befd1d3a 23 hours ago 1.089 GB

tryout latest sha256:2629d1fa0b81b222fca63371ca16cbf6a0772d07759ff80e8d1369b926940074 23 hours ago 131.5 MB

<none> <none> sha256:5ed6274db6ceb2397844896966ea239290555e74ef307030ebb01ff91b1914df 24 hours ago 1.089 GB

### List image digests

Images that use the v2 or later format have a content-addressable identifier called a digest. As long as the input used to generate the image is unchanged, the digest value is predictable. To list image digest values, use the --digests flag:

$ docker images --digests

REPOSITORY TAG DIGEST IMAGE ID CREATED SIZE

localhost:5000/test/busybox <none> sha256:cbbf2f9a99b47fc460d422812b6a5adff7dfee951d8fa2e4a98caa0382cfbdbf 4986bf8c1536 9 weeks ago 2.43 MB

When pushing or pulling to a 2.0 registry, the push or pull command output includes the image digest. You can pull using a digest value. You can also reference by digest in create, run, and rmi commands, as well as the FROM image reference in a Dockerfile.

### Filtering

The filtering flag (-f or --filter) format is of “key=value”. If there is more than one filter, then pass multiple flags (e.g., --filter "foo=bar" --filter "bif=baz")

The currently supported filters are:

* dangling (boolean - true or false)
* label (label=<key> or label=<key>=<value>)
* before (<image-name>[:<tag>], <image id> or <image@digest>) - filter images created before given id or references
* since (<image-name>[:<tag>], <image id> or <image@digest>) - filter images created since given id or references
* reference (pattern of an image reference) - filter images whose reference matches the specified pattern

#### Show untagged images (dangling)

$ docker images --filter "dangling=true"

REPOSITORY TAG IMAGE ID CREATED SIZE

<none> <none> 8abc22fbb042 4 weeks ago 0 B

<none> <none> 48e5f45168b9 4 weeks ago 2.489 MB

<none> <none> bf747efa0e2f 4 weeks ago 0 B

<none> <none> 980fe10e5736 12 weeks ago 101.4 MB

<none> <none> dea752e4e117 12 weeks ago 101.4 MB

<none> <none> 511136ea3c5a 8 months ago 0 B

This will display untagged images that are the leaves of the images tree (not intermediary layers). These images occur when a new build of an image takes the repo:tag away from the image ID, leaving it as <none>:<none> or untagged. A warning will be issued if trying to remove an image when a container is presently using it. By having this flag it allows for batch cleanup.

You can use this in conjunction with docker rmi ...:

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

8abc22fbb042

48e5f45168b9

bf747efa0e2f

980fe10e5736

dea752e4e117

511136ea3c5a

**Note**: Docker warns you if any containers exist that are using these untagged images.

#### Show images with a given label

The label filter matches images based on the presence of a label alone or a label and a value.

The following filter matches images with the com.example.version label regardless of its value.

$ docker images --filter "label=com.example.version"

REPOSITORY TAG IMAGE ID CREATED SIZE

match-me-1 latest eeae25ada2aa About a minute ago 188.3 MB

match-me-2 latest dea752e4e117 About a minute ago 188.3 MB

The following filter matches images with the com.example.version label with the 1.0 value.

$ docker images --filter "label=com.example.version=1.0"

REPOSITORY TAG IMAGE ID CREATED SIZE

match-me latest 511136ea3c5a About a minute ago 188.3 MB

In this example, with the 0.1 value, it returns an empty set because no matches were found.

$ docker images --filter "label=com.example.version=0.1"

REPOSITORY TAG IMAGE ID CREATED SIZE

#### Filter images by time

The before filter shows only images created before the image with given id or reference. For example, having these images:

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

image1 latest eeae25ada2aa 4 minutes ago 188.3 MB

image2 latest dea752e4e117 9 minutes ago 188.3 MB

image3 latest 511136ea3c5a 25 minutes ago 188.3 MB

Filtering with before would give:

$ docker images --filter "before=image1"

REPOSITORY TAG IMAGE ID CREATED SIZE

image2 latest dea752e4e117 9 minutes ago 188.3 MB

image3 latest 511136ea3c5a 25 minutes ago 188.3 MB

Filtering with since would give:

$ docker images --filter "since=image3"

REPOSITORY TAG IMAGE ID CREATED SIZE

image1 latest eeae25ada2aa 4 minutes ago 188.3 MB

image2 latest dea752e4e117 9 minutes ago 188.3 MB

#### Filter images by reference

The reference filter shows only images whose reference matches the specified pattern.

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox latest e02e811dd08f 5 weeks ago 1.09 MB

busybox uclibc e02e811dd08f 5 weeks ago 1.09 MB

busybox musl 733eb3059dce 5 weeks ago 1.21 MB

busybox glibc 21c16b6787c6 5 weeks ago 4.19 MB

Filtering with reference would give:

$ docker images --filter=reference='busy\*:\*libc'

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox uclibc e02e811dd08f 5 weeks ago 1.09 MB

busybox glibc 21c16b6787c6 5 weeks ago 4.19 MB

Filtering with multiple reference would give, either match A or B:

$ docker images --filter=reference='busy\*:uclibc' --filter=reference='busy\*:glibc'

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox uclibc e02e811dd08f 5 weeks ago 1.09 MB

busybox glibc 21c16b6787c6 5 weeks ago 4.19 MB

### Format the output

The formatting option (--format) will pretty print container output using a Go template.

Valid placeholders for the Go template are listed below:

| **Placeholder** | **Description** |
| --- | --- |
| .ID | Image ID |
| .Repository | Image repository |
| .Tag | Image tag |
| .Digest | Image digest |
| .CreatedSince | Elapsed time since the image was created |
| .CreatedAt | Time when the image was created |
| .Size | Image disk size |

When using the --format option, the image command will either output the data exactly as the template declares or, when using the table directive, will include column headers as well.

The following example uses a template without headers and outputs the ID and Repository entries separated by a colon for all images:

$ docker images --format "{{.ID}}: {{.Repository}}"

77af4d6b9913: <none>

b6fa739cedf5: committ

78a85c484f71: <none>

30557a29d5ab: docker

5ed6274db6ce: <none>

746b819f315e: postgres

746b819f315e: postgres

746b819f315e: postgres

746b819f315e: postgres

To list all images with their repository and tag in a table format you can use:

$ docker images --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}"

IMAGE ID REPOSITORY TAG

77af4d6b9913 <none> <none>

b6fa739cedf5 committ latest

78a85c484f71 <none> <none>

30557a29d5ab docker latest

5ed6274db6ce <none> <none>

746b819f315e postgres 9

746b819f315e postgres 9.3

746b819f315e postgres 9.3.5

746b819f315e postgres latest

**docker kill**

**Description**

Kill one or more running containers

**Usage**

docker kill [OPTIONS] CONTAINER [CONTAINER...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --signal , -s | KILL | Signal to send to 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 kill subcommand kills one or more containers. The main process inside the container is sent SIGKILL signal (default), or the signal that is specified with the --signal option. You can kill a container using the container’s ID, ID-prefix, or name.

**Note**: ENTRYPOINT and CMD in the *shell* form run as a subcommand of /bin/sh -c, which does not pass signals. This means that the executable is not the container’s PID 1 and does not receive Unix signals.

**Examples**

**Send a KILL signal to a container**

The following example sends the default KILL signal to the container named my\_container:

$ docker kill my\_container

**Send a custom signal to a container**

The following example sends a SIGHUP signal to the container named my\_container:

$ docker kill --signal=SIGHUP my\_container

You can specify a custom signal either by *name*, or *number*. The SIG prefix is optional, so the following examples are equivalent:

$ docker kill --signal=SIGHUP my\_container

$ docker kill --signal=HUP my\_container

$ docker kill --signal=1 my\_container

Refer to the [signal(7)](http://man7.org/linux/man-pages/man7/signal.7.html) man-page for a list of standard Linux signals.

**docker load**

**Description**

Load an image from a tar archive or STDIN

**Usage**

docker load [OPTIONS]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --input , -i |  | Read from tar archive file, instead of STDIN |
| --quiet , -q |  | Suppress the load output |

**Parent command**

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

**Extended description**

Load an image or repository from a tar archive (even if compressed with gzip, bzip2, or xz) from a file or STDIN. It restores both images and tags.

**Examples**

$ docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

$ docker load < busybox.tar.gz

Loaded image: busybox:latest

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox latest 769b9341d937 7 weeks ago 2.489 MB

$ docker load --input fedora.tar

Loaded image: fedora:rawhide

Loaded image: fedora:20

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox latest 769b9341d937 7 weeks ago 2.489 MB

fedora rawhide 0d20aec6529d 7 weeks ago 387 MB

fedora 20 58394af37342 7 weeks ago 385.5 MB

fedora heisenbug 58394af37342 7 weeks ago 385.5 MB

fedora

# docker login

## Description

Log in to a Docker registry

## Usage

docker login [OPTIONS] [SERVER]

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --password , -p |  | Password |
| --password-stdin |  | Take the password from stdin |
| --username , -u |  | Username |

## Parent command

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

## Extended description

Login to a registry.

### Login to a self-hosted registry

If you want to login to a self-hosted registry you can specify this by adding the server name.

$ docker login localhost:8080

### Provide a password using STDIN

To run the docker login command non-interactively, you can set the --password-stdin flag to provide a password through STDIN. Using STDIN prevents the password from ending up in the shell’s history, or log-files.

The following example reads a password from a file, and passes it to the docker login command using STDIN:

$ cat ~/my\_password.txt | docker login --username foo --password-stdin

### Privileged user requirement

docker login requires user to use sudo or be root, except when:

1. connecting to a remote daemon, such as a docker-machine provisioned docker engine.
2. user is added to the docker group. This will impact the security of your system; the docker group is root equivalent. See [Docker Daemon Attack Surface](https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface) for details.

You can log into any public or private repository for which you have credentials. When you log in, the command stores credentials in $HOME/.docker/config.json on Linux or %USERPROFILE%/.docker/config.json on Windows, via the procedure described below.

### Credentials store

The Docker Engine can keep user credentials in an external credentials store, such as the native keychain of the operating system. Using an external store is more secure than storing credentials in the Docker configuration file.

To use a credentials store, you need an external helper program to interact with a specific keychain or external store. Docker requires the helper program to be in the client’s host $PATH.

This is the list of currently available credentials helpers and where you can download them from:

* D-Bus Secret Service: https://github.com/docker/docker-credential-helpers/releases
* Apple macOS keychain: https://github.com/docker/docker-credential-helpers/releases
* Microsoft Windows Credential Manager: https://github.com/docker/docker-credential-helpers/releases
* [pass](https://www.passwordstore.org/): https://github.com/docker/docker-credential-helpers/releases

#### Configure the credentials store

You need to specify the credentials store in $HOME/.docker/config.json to tell the docker engine to use it. The value of the config property should be the suffix of the program to use (i.e. everything after docker-credential-). For example, to use docker-credential-osxkeychain:

{

"credsStore": "osxkeychain"

}

If you are currently logged in, run docker logout to remove the credentials from the file and run docker login again.

#### Default behavior

By default, Docker looks for the native binary on each of the platforms, i.e. “osxkeychain” on macOS, “wincred” on windows, and “pass” on Linux. A special case is that on Linux, Docker will fall back to the “secretservice” binary if it cannot find the “pass” binary. If none of these binaries are present, it stores the credentials (i.e. password) in base64 encoding in the config files described above.

#### Credential helper protocol

Credential helpers can be any program or script that follows a very simple protocol. This protocol is heavily inspired by Git, but it differs in the information shared.

The helpers always use the first argument in the command to identify the action. There are only three possible values for that argument: store, get, and erase.

The store command takes a JSON payload from the standard input. That payload carries the server address, to identify the credential, the user name, and either a password or an identity token.

{

"ServerURL": "https://index.docker.io/v1",

"Username": "david",

"Secret": "passw0rd1"

}

If the secret being stored is an identity token, the Username should be set to <token>.

The store command can write error messages to STDOUT that the docker engine will show if there was an issue.

The get command takes a string payload from the standard input. That payload carries the server address that the docker engine needs credentials for. This is an example of that payload: https://index.docker.io/v1.

The get command writes a JSON payload to STDOUT. Docker reads the user name and password from this payload:

{

"Username": "david",

"Secret": "passw0rd1"

}

The erase command takes a string payload from STDIN. That payload carries the server address that the docker engine wants to remove credentials for. This is an example of that payload: https://index.docker.io/v1.

The erase command can write error messages to STDOUT that the docker engine will show if there was an issue.

### Credential helpers

Credential helpers are similar to the credential store above, but act as the designated programs to handle credentials for specific registries. The default credential store (credsStore or the config file itself) will not be used for operations concerning credentials of the specified registries.

#### Configure credential helpers

If you are currently logged in, run docker logout to remove the credentials from the default store.

Credential helpers are specified in a similar way to credsStore, but allow for multiple helpers to be configured at a time. Keys specify the registry domain, and values specify the suffix of the program to use (i.e. everything after docker-credential-). For example:

{

"credHelpers": {

"registry.example.com": "registryhelper",

"awesomereg.example.org": "hip-star",

"unicorn.example.io": "vcbait"

}

}

**docker logout**

**Description**

Log out from a Docker registry

**Usage**

docker logout [SERVER]

**Parent command**

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

**Examples**

$ docker logout localhost:8080

**docker logs**

**Description**

Fetch the logs of a container

**Usage**

docker logs [OPTIONS] CONTAINER

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --details |  | Show extra details provided to logs |
| --follow , -f |  | Follow log output |
| --since |  | Show logs since timestamp (e.g. 2013-01-02T13:23:37) or relative (e.g. 42m for 42 minutes) |
| --tail | all | Number of lines to show from the end of the logs |
| --timestamps , -t |  | Show timestamps |
| --until |  | [API 1.35+](https://docs.docker.com/engine/api/v1.35/) Show logs before a timestamp (e.g. 2013-01-02T13:23:37) or relative (e.g. 42m for 42 minutes) |

**Parent command**

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

**Extended description**

The docker logs command batch-retrieves logs present at the time of execution.

**Note**: this command is only functional for containers that are started with the json-file or journald logging driver.

For more information about selecting and configuring logging drivers, refer to [Configure logging drivers](https://docs.docker.com/config/containers/logging/configure/).

The docker logs --follow command will continue streaming the new output from the container’s STDOUT and STDERR.

Passing a negative number or a non-integer to --tail is invalid and the value is set to all in that case.

The docker logs --timestamps command will add an [RFC3339Nano timestamp](https://golang.org/pkg/time/#pkg-constants) , for example 2014-09-16T06:17:46.000000000Z, to each log entry. To ensure that the timestamps are aligned the nano-second part of the timestamp will be padded with zero when necessary.

The docker logs --details command will add on extra attributes, such as environment variables and labels, provided to --log-opt when creating the container.

The --since option shows only the container logs generated after a given date. You can specify the date as an RFC 3339 date, a UNIX timestamp, or a Go duration string (e.g. 1m30s, 3h). Besides RFC3339 date format you may also use RFC3339Nano, 2006-01-02T15:04:05, 2006-01-02T15:04:05.999999999, 2006-01-02Z07:00, and 2006-01-02. The local timezone on the client will be used if you do not provide either a Z or a +-00:00 timezone offset at the end of the timestamp. When providing Unix timestamps enter seconds[.nanoseconds], where seconds is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap seconds (aka Unix epoch or Unix time), and the optional .nanoseconds field is a fraction of a second no more than nine digits long. You can combine the --since option with either or both of the --follow or --tail options.

**Examples**

**Retrieve logs until a specific point in time**

In order to retrieve logs before a specific point in time, run:

$ docker run --name test -d busybox sh -c "while true; do $(echo date); sleep 1; done"

$ date

Tue 14 Nov 2017 16:40:00 CET

$ docker logs -f --until=2s

Tue 14 Nov 2017 16:40:00 CET

Tue 14 Nov 2017 16:40:01 CET

Tue 14 Nov 2017 16:40:02 CET

**docker manifest**

**Description**

Manage Docker image manifests and manifest lists

This command is experimental on the Docker client.

**It should not be used in production environments.**

To enable experimental features in the Docker CLI, edit the [config.json](https://docs.docker.com/engine/reference/commandline/cli/#configuration-files) and set experimental to enabled. You can go [here](https://docs.docker.com/engine/reference/commandline/cli/#experimental-features) for more information.

**Usage**

docker manifest COMMAND COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker manifest annotate](https://docs.docker.com/engine/reference/commandline/manifest_annotate/) | Add additional information to a local image manifest |
| [docker manifest create](https://docs.docker.com/engine/reference/commandline/manifest_create/) | Create a local manifest list for annotating and pushing to a registry |
| [docker manifest inspect](https://docs.docker.com/engine/reference/commandline/manifest_inspect/) | Display an image manifest, or manifest list |
| [docker manifest push](https://docs.docker.com/engine/reference/commandline/manifest_push/) | Push a manifest list to a repository |

**Parent command**

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

**Extended description**

The docker manifest command by itself performs no action. In order to operate on a manifest or manifest list, one of the subcommands must be used.

A single manifest is information about an image, such as layers, size, and digest. The docker manifest command also gives users additional information such as the os and architecture an image was built for.

A manifest list is a list of image layers that is created by specifying one or more (ideally more than one) image names. It can then be used in the same way as an image name in docker pull and docker run commands, for example.

Ideally a manifest list is created from images that are identical in function for different os/arch combinations. For this reason, manifest lists are often referred to as “multi-arch images”. However, a user could create a manifest list that points to two images -- one for windows on amd64, and one for darwin on amd64.

**manifest inspect**

manifest inspect --help

Usage: docker manifest inspect [OPTIONS] [MANIFEST\_LIST] MANIFEST

Display an image manifest, or manifest list

Options:

--help Print usage

--insecure Allow communication with an insecure registry

-v, --verbose Output additional info including layers and platform

**manifest create**

Usage: docker manifest create MANIFEST\_LIST MANIFEST [MANIFEST...]

Create a local manifest list for annotating and pushing to a registry

Options:

-a, --amend Amend an existing manifest list

--insecure Allow communication with an insecure registry

--help Print usage

**manifest annotate**

Usage: docker manifest annotate [OPTIONS] MANIFEST\_LIST MANIFEST

Add additional information to a local image manifest

Options:

--arch string Set architecture

--help Print usage

--os string Set operating system

--os-features stringSlice Set operating system feature

--variant string Set architecture variant

**manifest push**

Usage: docker manifest push [OPTIONS] MANIFEST\_LIST

Push a manifest list to a repository

Options:

--help Print usage

--insecure Allow push to an insecure registry

-p, --purge Remove the local manifest list after push

**Working with insecure registries**

The manifest command interacts solely with a Docker registry. Because of this, it has no way to query the engine for the list of allowed insecure registries. To allow the CLI to interact with an insecure registry, some docker manifest commands have an --insecure flag. For each transaction, such as a create, which queries a registry, the --insecure flag must be specified. This flag tells the CLI that this registry call may ignore security concerns like missing or self-signed certificates. Likewise, on a manifest push to an insecure registry, the --insecure flag must be specified. If this is not used with an insecure registry, the manifest command fails to find a registry that meets the default requirements.

**Examples**

**Inspect an image’s manifest object**

$ docker manifest inspect hello-world

{

"schemaVersion": 2,

"mediaType": "application/vnd.docker.distribution.manifest.v2+json",

"config": {

"mediaType": "application/vnd.docker.container.image.v1+json",

"size": 1520,

"digest": "sha256:1815c82652c03bfd8644afda26fb184f2ed891d921b20a0703b46768f9755c57"

},

"layers": [

{

"mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",

"size": 972,

"digest": "sha256:b04784fba78d739b526e27edc02a5a8cd07b1052e9283f5fc155828f4b614c28"

}

]

}

**Inspect an image’s manifest and get the os/arch info**

The docker manifest inspect command takes an optional --verbose flag that gives you the image’s name (Ref), and architecture and os (Platform).

Just as with other docker commands that take image names, you can refer to an image with or without a tag, or by digest (e.g. hello-world@sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26ff74f).

Here is an example of inspecting an image’s manifest with the --verbose flag:

$ docker manifest inspect --verbose hello-world

{

"Ref": "docker.io/library/hello-world:latest",

"Digest": "sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26ff74f",

"SchemaV2Manifest": {

"schemaVersion": 2,

"mediaType": "application/vnd.docker.distribution.manifest.v2+json",

"config": {

"mediaType": "application/vnd.docker.container.image.v1+json",

"size": 1520,

"digest": "sha256:1815c82652c03bfd8644afda26fb184f2ed891d921b20a0703b46768f9755c57"

},

"layers": [

{

"mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",

"size": 972,

"digest": "sha256:b04784fba78d739b526e27edc02a5a8cd07b1052e9283f5fc155828f4b614c28"

}

]

},

"Platform": {

"architecture": "amd64",

"os": "linux"

}

}

**Create and push a manifest list**

To create a manifest list, you first create the manifest list locally by specifying the constituent images you would like to have included in your manifest list. Keep in mind that this is pushed to a registry, so if you want to push to a registry other than the docker registry, you need to create your manifest list with the registry name or IP and port. This is similar to tagging an image and pushing it to a foreign registry.

After you have created your local copy of the manifest list, you may optionally annotate it. Annotations allowed are the architecture and operating system (overriding the image’s current values), os features, and an architecture variant.

Finally, you need to push your manifest list to the desired registry. Below are descriptions of these three commands, and an example putting them all together.

$ docker manifest create 45.55.81.106:5000/coolapp:v1 \

45.55.81.106:5000/coolapp-ppc64le-linux:v1 \

45.55.81.106:5000/coolapp-arm-linux:v1 \

45.55.81.106:5000/coolapp-amd64-linux:v1 \

45.55.81.106:5000/coolapp-amd64-windows:v1

Created manifest list 45.55.81.106:5000/coolapp:v1

$ docker manifest annotate 45.55.81.106:5000/coolapp:v1 45.55.81.106:5000/coolapp-arm-linux --arch arm

$ docker manifest push 45.55.81.106:5000/coolapp:v1

Pushed manifest 45.55.81.106:5000/coolapp@sha256:9701edc932223a66e49dd6c894a11db8c2cf4eccd1414f1ec105a623bf16b426 with digest: sha256:f67dcc5fc786f04f0743abfe0ee5dae9bd8caf8efa6c8144f7f2a43889dc513b

Pushed manifest 45.55.81.106:5000/coolapp@sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26ff74f with digest: sha256:b64ca0b60356a30971f098c92200b1271257f100a55b351e6bbe985638352f3a

Pushed manifest 45.55.81.106:5000/coolapp@sha256:39dc41c658cf25f33681a41310372f02728925a54aac3598310bfb1770615fc9 with digest: sha256:df436846483aff62bad830b730a0d3b77731bcf98ba5e470a8bbb8e9e346e4e8

Pushed manifest 45.55.81.106:5000/coolapp@sha256:f91b1145cd4ac800b28122313ae9e88ac340bb3f1e3a4cd3e59a3648650f3275 with digest: sha256:5bb8e50aa2edd408bdf3ddf61efb7338ff34a07b762992c9432f1c02fc0e5e62

sha256:050b213d49d7673ba35014f21454c573dcbec75254a08f4a7c34f66a47c06aba

**Inspect a manifest list**

$ docker manifest inspect coolapp:v1

{

"schemaVersion": 2,

"mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",

"manifests": [

{

"mediaType": "application/vnd.docker.distribution.manifest.v2+json",

"size": 424,

"digest": "sha256:f67dcc5fc786f04f0743abfe0ee5dae9bd8caf8efa6c8144f7f2a43889dc513b",

"platform": {

"architecture": "arm",

"os": "linux"

}

},

{

"mediaType": "application/vnd.docker.distribution.manifest.v2+json",

"size": 424,

"digest": "sha256:b64ca0b60356a30971f098c92200b1271257f100a55b351e6bbe985638352f3a",

"platform": {

"architecture": "amd64",

"os": "linux"

}

},

{

"mediaType": "application/vnd.docker.distribution.manifest.v2+json",

"size": 425,

"digest": "sha256:df436846483aff62bad830b730a0d3b77731bcf98ba5e470a8bbb8e9e346e4e8",

"platform": {

"architecture": "ppc64le",

"os": "linux"

}

},

{

"mediaType": "application/vnd.docker.distribution.manifest.v2+json",

"size": 425,

"digest": "sha256:5bb8e50aa2edd408bdf3ddf61efb7338ff34a07b762992c9432f1c02fc0e5e62",

"platform": {

"architecture": "s390x",

"os": "linux"

}

}

]

}

**Push to an insecure registry**

Here is an example of creating and pushing a manifest list using a known insecure registry.

$ docker manifest create --insecure myprivateregistry.mycompany.com/repo/image:1.0 \

myprivateregistry.mycompany.com/repo/image-linux-ppc64le:1.0 \

myprivateregistry.mycompany.com/repo/image-linux-s390x:1.0 \

myprivateregistry.mycompany.com/repo/image-linux-arm:1.0 \

myprivateregistry.mycompany.com/repo/image-linux-armhf:1.0 \

myprivateregistry.mycompany.com/repo/image-windows-amd64:1.0 \

myprivateregistry.mycompany.com/repo/image-linux-amd64:1.0

$ docker manifest push --insecure myprivateregistry.mycompany.com/repo/image:tag

Note that the --insecure flag is not required to annotate a manifest list, since annotations are to a locally-stored copy of a manifest list. You may also skip the --insecure flag if you are performing a docker manifest inspect on a locally-stored manifest list. Be sure to keep in mind that locally-stored manifest lists are never used by the engine on a docker pull.

**docker network**

**Description**

Manage networks

[API 1.21+](https://docs.docker.com/engine/api/v1.21/)  The client and daemon API must both be at least [1.21](https://docs.docker.com/engine/api/v1.21/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

**Usage**

docker network COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker network connect](https://docs.docker.com/engine/reference/commandline/network_connect/) | Connect a container to a network |
| [docker network create](https://docs.docker.com/engine/reference/commandline/network_create/) | Create a network |
| [docker network disconnect](https://docs.docker.com/engine/reference/commandline/network_disconnect/) | Disconnect a container from a network |
| [docker network inspect](https://docs.docker.com/engine/reference/commandline/network_inspect/) | Display detailed information on one or more networks |
| [docker network ls](https://docs.docker.com/engine/reference/commandline/network_ls/) | List networks |
| [docker network prune](https://docs.docker.com/engine/reference/commandline/network_prune/) | Remove all unused networks |
| [docker network rm](https://docs.docker.com/engine/reference/commandline/network_rm/) | Remove one or more networks |

**Parent command**

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

**Extended description**

Manage networks. You can use subcommands to create, inspect, list, remove, prune, connect, and disconnect networks.

**docker node**

**Description**

Manage Swarm nodes

[API 1.24+](https://docs.docker.com/engine/api/v1.24/)  The client and daemon API must both be at least [1.24](https://docs.docker.com/engine/api/v1.24/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

Swarm This command works with the Swarm orchestrator.

**Usage**

docker node COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker node demote](https://docs.docker.com/engine/reference/commandline/node_demote/) | Demote one or more nodes from manager in the swarm |
| [docker node inspect](https://docs.docker.com/engine/reference/commandline/node_inspect/) | Display detailed information on one or more nodes |
| [docker node ls](https://docs.docker.com/engine/reference/commandline/node_ls/) | List nodes in the swarm |
| [docker node promote](https://docs.docker.com/engine/reference/commandline/node_promote/) | Promote one or more nodes to manager in the swarm |
| [docker node ps](https://docs.docker.com/engine/reference/commandline/node_ps/) | List tasks running on one or more nodes, defaults to current node |
| [docker node rm](https://docs.docker.com/engine/reference/commandline/node_rm/) | Remove one or more nodes from the swarm |
| [docker node update](https://docs.docker.com/engine/reference/commandline/node_update/) | Update a node |

**Parent command**

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

**Extended description**

Manage nodes.

# docker ps

## Description

List containers

## Usage

docker ps [OPTIONS]

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all , -a |  | Show all containers (default shows just running) |
| --filter , -f |  | Filter output based on conditions provided |
| --format |  | Pretty-print containers using a Go template |
| --last , -n | -1 | Show n last created containers (includes all states) |
| --latest , -l |  | Show the latest created container (includes all states) |
| --no-trunc |  | Don’t truncate output |
| --quiet , -q |  | Only display numeric IDs |
| --size , -s |  | Display total file sizes |

**docker pause**

**Description**

Pause all processes within one or more containers

**Usage**

docker pause CONTAINER [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 pause command suspends all processes in the specified containers. On Linux, this uses the freezer cgroup. Traditionally, when suspending a process the SIGSTOP signal is used, which is observable by the process being suspended. With the freezer cgroup the process is unaware, and unable to capture, that it is being suspended, and subsequently resumed. On Windows, only Hyper-V containers can be paused.

See the [freezer cgroup documentation](https://www.kernel.org/doc/Documentation/cgroup-v1/freezer-subsystem.txt) for further details.

**Examples**

$ docker pause my\_container

**docker port**

**Description**

List port mappings or a specific mapping for the container

**Usage**

docker port CONTAINER [PRIVATE\_PORT[/PROTO]]

**Parent command**

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

**Examples**

**Show all mapped ports**

You can find out all the ports mapped by not specifying a PRIVATE\_PORT, or just a specific mapping:

$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

b650456536c7 busybox:latest top 54 minutes ago Up 54 minutes 0.0.0.0:1234->9876/tcp, 0.0.0.0:4321->7890/tcp test

$ docker port test

7890/tcp -> 0.0.0.0:4321

9876/tcp -> 0.0.0.0:1234

$ docker port test 7890/tcp

0.0.0.0:4321

$ docker port test 7890/udp

2014/06/24 11:53:36 Error: No public port '7890/udp' published for test

$ docker port test 7890

0.0.0.0:4321

**docker plugin**

**Description**

Manage plugins

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

**Usage**

docker plugin COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker plugin create](https://docs.docker.com/engine/reference/commandline/plugin_create/) | Create a plugin from a rootfs and configuration. Plugin data directory must contain config.json and rootfs directory. |
| [docker plugin disable](https://docs.docker.com/engine/reference/commandline/plugin_disable/) | Disable a plugin |
| [docker plugin enable](https://docs.docker.com/engine/reference/commandline/plugin_enable/) | Enable a plugin |
| [docker plugin inspect](https://docs.docker.com/engine/reference/commandline/plugin_inspect/) | Display detailed information on one or more plugins |
| [docker plugin install](https://docs.docker.com/engine/reference/commandline/plugin_install/) | Install a plugin |
| [docker plugin ls](https://docs.docker.com/engine/reference/commandline/plugin_ls/) | List plugins |
| [docker plugin push](https://docs.docker.com/engine/reference/commandline/plugin_push/) | Push a plugin to a registry |
| [docker plugin rm](https://docs.docker.com/engine/reference/commandline/plugin_rm/) | Remove one or more plugins |
| [docker plugin set](https://docs.docker.com/engine/reference/commandline/plugin_set/) | Change settings for a plugin |
| [docker plugin upgrade](https://docs.docker.com/engine/reference/commandline/plugin_upgrade/) | Upgrade an existing plugin |

**Parent command**

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

**Extended description**

Manage plugins.

# docker pull

## Description

Pull an image or a repository from a registry

## Usage

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

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all-tags , -a |  | Download all tagged images in the repository |
| --disable-content-trust | true | Skip image verification |
| --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 |
| --quiet , -q |  | Suppress verbose output |

## Parent command

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

## Extended description

Most of your images will be created on top of a base image from the [Docker Hub](https://hub.docker.com) registry.

[Docker Hub](https://hub.docker.com) contains many pre-built images that you can pull and try without needing to define and configure your own.

To download a particular image, or set of images (i.e., a repository), use docker pull.

### Proxy configuration

If you are behind an HTTP proxy server, for example in corporate settings, before open a connect to registry, you may need to configure the Docker daemon’s proxy settings, using the HTTP\_PROXY, HTTPS\_PROXY, and NO\_PROXY environment variables. To set these environment variables on a host using systemd, refer to the [control and configure Docker with systemd](https://docs.docker.com/engine/admin/systemd/#http-proxy) for variables configuration.

### Concurrent downloads

By default the Docker daemon will pull three layers of an image at a time. If you are on a low bandwidth connection this may cause timeout issues and you may want to lower this via the --max-concurrent-downloads daemon option. See the [daemon documentation](https://docs.docker.com/engine/reference/commandline/dockerd/) for more details.

## Examples

### Pull an image from Docker Hub

To download a particular image, or set of images (i.e., a repository), use docker pull. If no tag is provided, Docker Engine uses the :latest tag as a default. This command pulls the debian:latest image:

$ docker pull debian

Using default tag: latest

latest: Pulling from library/debian

fdd5d7827f33: Pull complete

a3ed95caeb02: Pull complete

Digest: sha256:e7d38b3517548a1c71e41bffe9c8ae6d6d29546ce46bf62159837aad072c90aa

Status: Downloaded newer image for debian:latest

Docker images can consist of multiple layers. In the example above, the image consists of two layers; fdd5d7827f33 and a3ed95caeb02.

Layers can be reused by images. For example, the debian:jessie image shares both layers with debian:latest. Pulling the debian:jessie image therefore only pulls its metadata, but not its layers, because all layers are already present locally:

$ docker pull debian:jessie

jessie: Pulling from library/debian

fdd5d7827f33: Already exists

a3ed95caeb02: Already exists

Digest: sha256:a9c958be96d7d40df920e7041608f2f017af81800ca5ad23e327bc402626b58e

Status: Downloaded newer image for debian:jessie

To see which images are present locally, use the [docker images](https://docs.docker.com/engine/reference/commandline/images/) command:

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

debian jessie f50f9524513f 5 days ago 125.1 MB

debian latest f50f9524513f 5 days ago 125.1 MB

Docker uses a content-addressable image store, and the image ID is a SHA256 digest covering the image’s configuration and layers. In the example above, debian:jessie and debian:latest have the same image ID because they are actually the same image tagged with different names. Because they are the same image, their layers are stored only once and do not consume extra disk space.

For more information about images, layers, and the content-addressable store, refer to [understand images, containers, and storage drivers](https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/).

### Pull an image by digest (immutable identifier)

So far, you’ve pulled images by their name (and “tag”). Using names and tags is a convenient way to work with images. When using tags, you can docker pull an image again to make sure you have the most up-to-date version of that image. For example, docker pull ubuntu:14.04 pulls the latest version of the Ubuntu 14.04 image.

In some cases you don’t want images to be updated to newer versions, but prefer to use a fixed version of an image. Docker enables you to pull an image by its digest. When pulling an image by digest, you specify exactly which version of an image to pull. Doing so, allows you to “pin” an image to that version, and guarantee that the image you’re using is always the same.

To know the digest of an image, pull the image first. Let’s pull the latest ubuntu:14.04 image from Docker Hub:

$ docker pull ubuntu:14.04

14.04: Pulling from library/ubuntu

5a132a7e7af1: Pull complete

fd2731e4c50c: Pull complete

28a2f68d1120: Pull complete

a3ed95caeb02: Pull complete

Digest: sha256:45b23dee08af5e43a7fea6c4cf9c25ccf269ee113168c19722f87876677c5cb2

Status: Downloaded newer image for ubuntu:14.04

Docker prints the digest of the image after the pull has finished. In the example above, the digest of the image is:

sha256:45b23dee08af5e43a7fea6c4cf9c25ccf269ee113168c19722f87876677c5cb2

Docker also prints the digest of an image when pushing to a registry. This may be useful if you want to pin to a version of the image you just pushed.

A digest takes the place of the tag when pulling an image, for example, to pull the above image by digest, run the following command:

$ docker pull ubuntu@sha256:45b23dee08af5e43a7fea6c4cf9c25ccf269ee113168c19722f87876677c5cb2

sha256:45b23dee08af5e43a7fea6c4cf9c25ccf269ee113168c19722f87876677c5cb2: Pulling from library/ubuntu

5a132a7e7af1: Already exists

fd2731e4c50c: Already exists

28a2f68d1120: Already exists

a3ed95caeb02: Already exists

Digest: sha256:45b23dee08af5e43a7fea6c4cf9c25ccf269ee113168c19722f87876677c5cb2

Status: Downloaded newer image for ubuntu@sha256:45b23dee08af5e43a7fea6c4cf9c25ccf269ee113168c19722f87876677c5cb2

Digest can also be used in the FROM of a Dockerfile, for example:

FROM ubuntu@sha256:45b23dee08af5e43a7fea6c4cf9c25ccf269ee113168c19722f87876677c5cb2

MAINTAINER some maintainer <maintainer@example.com>

**Note**: Using this feature “pins” an image to a specific version in time. Docker will therefore not pull updated versions of an image, which may include security updates. If you want to pull an updated image, you need to change the digest accordingly.

### Pull from a different registry

By default, docker pull pulls images from [Docker Hub](https://hub.docker.com). It is also possible to manually specify the path of a registry to pull from. For example, if you have set up a local registry, you can specify its path to pull from it. A registry path is similar to a URL, but does not contain a protocol specifier (https://).

The following command pulls the testing/test-image image from a local registry listening on port 5000 (myregistry.local:5000):

$ docker pull myregistry.local:5000/testing/test-image

Registry credentials are managed by [docker login](https://docs.docker.com/engine/reference/commandline/login/).

Docker uses the https:// protocol to communicate with a registry, unless the registry is allowed to be accessed over an insecure connection. Refer to the [insecure registries](https://docs.docker.com/engine/reference/commandline/dockerd/#insecure-registries) section for more information.

### Pull a repository with multiple images

By default, docker pull pulls a single image from the registry. A repository can contain multiple images. To pull all images from a repository, provide the -a (or --all-tags) option when using docker pull.

This command pulls all images from the fedora repository:

$ docker pull --all-tags fedora

Pulling repository fedora

ad57ef8d78d7: Download complete

105182bb5e8b: Download complete

511136ea3c5a: Download complete

73bd853d2ea5: Download complete

....

Status: Downloaded newer image for fedora

After the pull has completed use the docker images command to see the images that were pulled. The example below shows all the fedora images that are present locally:

$ docker images fedora

REPOSITORY TAG IMAGE ID CREATED SIZE

fedora rawhide ad57ef8d78d7 5 days ago 359.3 MB

fedora 20 105182bb5e8b 5 days ago 372.7 MB

fedora heisenbug 105182bb5e8b 5 days ago 372.7 MB

fedora latest 105182bb5e8b 5 days ago 372.7 MB

### Cancel a pull

Killing the docker pull process, for example by pressing CTRL-c while it is running in a terminal, will terminate the pull operation.

$ docker pull fedora

Using default tag: latest

latest: Pulling from library/fedora

a3ed95caeb02: Pulling fs layer

236608c7b546: Pulling fs layer

^C

**Note**: Technically, the Engine terminates a pull operation when the connection between the Docker Engine daemon and the Docker Engine client initiating the pull is lost. If the connection with the Engine daemon is lost for other reasons than a manual interaction, the pull is also aborted.

**docker push**

**Description**

Push an image or a repository to a registry

**Usage**

docker push [OPTIONS] NAME[:TAG]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --disable-content-trust | true | Skip image signing |

**Parent command**

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

**Extended description**

Use docker push to share your images to the [Docker Hub](https://hub.docker.com) registry or to a self-hosted one.

Refer to the [docker tag](https://docs.docker.com/engine/reference/commandline/tag/) reference for more information about valid image and tag names.

Killing the docker push process, for example by pressing CTRL-c while it is running in a terminal, terminates the push operation.

Progress bars are shown during docker push, which show the uncompressed size. The actual amount of data that’s pushed will be compressed before sending, so the uploaded size will not be reflected by the progress bar.

Registry credentials are managed by [docker login](https://docs.docker.com/engine/reference/commandline/login/).

**Concurrent uploads**

By default the Docker daemon will push five layers of an image at a time. If you are on a low bandwidth connection this may cause timeout issues and you may want to lower this via the --max-concurrent-uploads daemon option. See the [daemon documentation](https://docs.docker.com/engine/reference/commandline/dockerd/) for more details.

**Examples**

**Push a new image to a registry**

First save the new image by finding the container ID (using [docker ps](https://docs.docker.com/engine/reference/commandline/ps/)) and then committing it to a new image name. Note that only a-z0-9-\_. are allowed when naming images:

$ docker commit c16378f943fe rhel-httpd

Now, push the image to the registry using the image ID. In this example the registry is on host named registry-host and listening on port 5000. To do this, tag the image with the host name or IP address, and the port of the registry:

$ docker tag rhel-httpd registry-host:5000/myadmin/rhel-httpd

$ docker push registry-host:5000/myadmin/rhel-httpd

Check that this worked by running:

$ docker images

You should see both rhel-httpd and registry-host:5000/myadmin/rhel-httpd listed.

**docker registry**

**Description**

Manage Docker registries

This command is only available on Docker Enterprise Edition.

Learn more about [Docker Enterprise products](https://docs.docker.com/ee/supported-platforms/).

This command is experimental on the Docker client.

**It should not be used in production environments.**

To enable experimental features in the Docker CLI, edit the [config.json](https://docs.docker.com/engine/reference/commandline/cli/#configuration-files) and set experimental to enabled. You can go [here](https://docs.docker.com/engine/reference/commandline/cli/#experimental-features) for more information.

**Usage**

docker registry COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker registry events](https://docs.docker.com/engine/reference/commandline/registry_events/) | List registry events (DTR Only) |
| [docker registry history](https://docs.docker.com/engine/reference/commandline/registry_history/) | Inspect registry image history (DTR Only) |
| [docker registry info](https://docs.docker.com/engine/reference/commandline/registry_info/) | Display information about a registry (DTR Only) |
| [docker registry inspect](https://docs.docker.com/engine/reference/commandline/registry_inspect/) | Inspect registry image |
| [docker registry joblogs](https://docs.docker.com/engine/reference/commandline/registry_joblogs/) | List registry job logs (DTR Only) |
| [docker registry jobs](https://docs.docker.com/engine/reference/commandline/registry_jobs/) | List registry jobs (DTR Only) |
| [docker registry ls](https://docs.docker.com/engine/reference/commandline/registry_ls/) | List registry images |
| [docker registry rmi](https://docs.docker.com/engine/reference/commandline/registry_rmi/) | Remove a registry image (DTR Only) |

**Parent command**

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

**docker rename**

**Description**

Rename a container

**Usage**

docker rename CONTAINER NEW\_NAME

**Parent command**

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

**Extended description**

The docker rename command renames a container.

**Examples**

$ docker rename my\_container my\_new\_container

**docker restart**

**Description**

Restart one or more containers

**Usage**

docker restart [OPTIONS] CONTAINER [CONTAINER...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --time , -t | 10 | Seconds to wait for stop before killing the container |

**Parent command**

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

**Examples**

$ docker restart my\_container

**docker rm**

**Description**

Remove one or more containers

**Usage**

docker rm [OPTIONS] CONTAINER [CONTAINER...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --force , -f |  | Force the removal of a running container (uses SIGKILL) |
| --link , -l |  | Remove the specified link |
| --volumes , -v |  | Remove anonymous volumes associated with the container |

**Parent command**

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

**Examples**

**Remove a container**

This will remove the container referenced under the link /redis.

$ docker rm /redis

/redis

**Remove a link specified with --link on the default bridge network**

This will remove the underlying link between /webapp and the /redis containers on the default bridge network, removing all network communication between the two containers. This does not apply when --link is used with user-specified networks.

$ docker rm --link /webapp/redis

/webapp/redis

**Force-remove a running container**

This command will force-remove a running container.

$ docker rm --force redis

redis

The main process inside the container referenced under the link redis will receive SIGKILL, then the container will be removed.

**Remove all stopped containers**

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

This command will delete all stopped containers. The command docker ps -a -q will return all existing container IDs and pass them to the rm command which will delete them. Any running containers will not be deleted.

**Remove a container and its volumes**

$ docker rm -v redis

redis

This command will remove the container and any volumes associated with it. Note that if a volume was specified with a name, it will not be removed.

**Remove a container and selectively remove volumes**

$ docker create -v awesome:/foo -v /bar --name hello redis

hello

$ docker rm -v hello

In this example, the volume for /foo will remain intact, but the volume for /bar will be removed. The same behavior holds for volumes inherited with --volumes-from.

**docker rmi**

**Description**

Remove one or more images

**Usage**

docker rmi [OPTIONS] IMAGE [IMAGE...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --force , -f |  | Force removal of the image |
| --no-prune |  | Do not delete untagged parents |

**Parent command**

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

**Extended description**

Removes (and un-tags) one or more images from the host node. If an image has multiple tags, using this command with the tag as a parameter only removes the tag. If the tag is the only one for the image, both the image and the tag are removed.

This does not remove images from a registry. You cannot remove an image of a running container unless you use the -f option. To see all images on a host use the [docker image ls](https://docs.docker.com/engine/reference/commandline/images/) command.

**Examples**

You can remove an image using its short or long ID, its tag, or its digest. If an image has one or more tags referencing it, you must remove all of them before the image is removed. Digest references are removed automatically when an image is removed by tag.

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

test1 latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)

test latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)

test2 latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)

$ docker rmi fd484f19954f

Error: Conflict, cannot delete image fd484f19954f because it is tagged in multiple repositories, use -f to force

2013/12/11 05:47:16 Error: failed to remove one or more images

$ docker rmi test1:latest

Untagged: test1:latest

$ docker rmi test2:latest

Untagged: test2:latest

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

test latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)

$ docker rmi test:latest

Untagged: test:latest

Deleted: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8

If you use the -f flag and specify the image’s short or long ID, then this command untags and removes all images that match the specified ID.

$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

test1 latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)

test latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)

test2 latest fd484f19954f 23 seconds ago 7 B (virtual 4.964 MB)

$ docker rmi -f fd484f19954f

Untagged: test1:latest

Untagged: test:latest

Untagged: test2:latest

Deleted: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8

An image pulled by digest has no tag associated with it:

$ docker images --digests

REPOSITORY TAG DIGEST IMAGE ID CREATED SIZE

localhost:5000/test/busybox <none> sha256:cbbf2f9a99b47fc460d422812b6a5adff7dfee951d8fa2e4a98caa0382cfbdbf 4986bf8c1536 9 weeks ago 2.43 MB

To remove an image using its digest:

$ docker rmi localhost:5000/test/busybox@sha256:cbbf2f9a99b47fc460d422812b6a5adff7dfee951d8fa2e4a98caa0382cfbdbf

Untagged: localhost:5000/test/busybox@sha256:cbbf2f9a99b47fc460d422812b6a5adff7dfee951d8fa2e4a98caa0382cfbdbf

Deleted: 4986bf8c15363d1c5d15512d5266f8777bfba4974ac56e3270e7760f6f0a8125

Deleted: ea13149945cb6b1e746bf28032f02e9b5a793523481a0a18645fc77ad53c4ea2

Deleted: df7546f9f060a2268024

# docker run

## 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) |
| --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 |
| --domainname |  | Container NIS domain name |
| --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 |
| --gpus |  | [API 1.40+](https://docs.docker.com/engine/api/v1.40/) GPU devices to add to the container (‘all’ to pass all GPUs) |
| --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) |
| --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 |
| --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 seconds 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.

Note that ports which are not bound to the host (i.e., -p 80:80 instead of -p 127.0.0.1:80:80) will be accessible from the outside. This also applies if you configured UFW to block this specific port, as Docker manages his own iptables rules. [Read more](https://docs.docker.com/network/iptables/)

$ 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.

For Windows, the format of the string passed to the --device option is in the form of --device=<IdType>/<Id>. Beginning with Windows Server 2019 and Windows 10 October 2018 Update, Windows only supports an IdType of class and the Id as a [device interface class GUID](https://docs.microsoft.com/en-us/windows-hardware/drivers/install/overview-of-device-interface-classes). Refer to the table defined in the [Windows container docs](https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/hardware-devices-in-containers) for a list of container-supported device interface class GUIDs.

If this option is specified for a process-isolated Windows container, all devices that implement the requested device interface class GUID are made available in the container. For example, the command below makes all COM ports on the host visible in the container.

PS C:\> docker run --device=class/86E0D1E0-8089-11D0-9CE4-08003E301F73 mcr.microsoft.com/windows/servercore:ltsc2019

**Note**: the --device option is only supported on process-isolated Windows containers. This option fails if the container isolation is hyperv or when running Linux Containers on Windows (LCOW).

### Access an NVIDIA GPU

The --gpus­ flag allows you to access NVIDIA GPU resources. First you need to install [nvidia-container-runtime](https://nvidia.github.io/nvidia-container-runtime/). Visit [Specify a container’s resources](https://docs.docker.com/config/containers/resource_constraints/) for more information.

To use --gpus, specify which GPUs (or all) to use. If no value is provied, all available GPUs are used. The example below exposes all available GPUs.

$ docker run -it --rm --gpus all ubuntu nvidia-smi

Use the device option to specify GPUs. The example below exposes a specific GPU.

$ docker run -it --rm --gpus device=GPU-3a23c669-1f69-c64e-cf85-44e9b07e7a2a ubuntu nvidia-smi

The example below exposes the first and third GPUs.

$ docker run -it --rm --gpus device=0,2 nvidia-smi

### 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 | sed -n 1p`

$ 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 older than Windows 10 1809). |
| hyperv | Hyper-V hypervisor partition-based isolation. |

The default isolation on Windows server operating systems is process. The default isolation on Windows client operating systems is hyperv. An attempt to start a container on a client operating system older than Windows 10 1809 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-ComputerInfo \*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-ComputerInfo \*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, 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:

Sysctls beginning with net.\*

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

**docker save**

**Description**

Save one or more images to a tar archive (streamed to STDOUT by default)

**Usage**

docker save [OPTIONS] IMAGE [IMAGE...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --output , -o |  | Write to a file, instead of STDOUT |

**Parent command**

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

**Extended description**

Produces a tarred repository to the standard output stream. Contains all parent layers, and all tags + versions, or specified repo:tag, for each argument provided.

**Examples**

**Create a backup that can then be used with docker load.**

$ docker save busybox > busybox.tar

$ ls -sh busybox.tar

2.7M busybox.tar

$ docker save --output busybox.tar busybox

$ ls -sh busybox.tar

2.7M busybox.tar

$ docker save -o fedora-all.tar fedora

$ docker save -o fedora-latest.tar fedora:latest

**Save an image to a tar.gz file using gzip**

You can use gzip to save the image file and make the backup smaller.

docker save myimage:latest | gzip > myimage\_latest.tar.gz

**Cherry-pick particular tags**

You can even cherry-pick particular tags of an image repository.

$ docker save -o ubuntu.tar ubuntu:lucid ubuntu:saucy

# docker search

## Description

Search the Docker Hub for images

## Usage

docker search [OPTIONS] TERM

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --automated |  | [deprecated](https://docs.docker.com/engine/deprecated/) Only show automated builds |
| --filter , -f |  | Filter output based on conditions provided |
| --format |  | Pretty-print search using a Go template |
| --limit | 25 | Max number of search results |
| --no-trunc |  | Don’t truncate output |
| --stars , -s |  | [deprecated](https://docs.docker.com/engine/deprecated/) Only displays with at least x stars |

## Parent command

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

## Extended description

Search [Docker Hub](https://hub.docker.com) for images

See [Find Public Images on Docker Hub](https://docs.docker.com/engine/tutorials/dockerrepos/#searching-for-images) for more details on finding shared images from the command line.

**Note**: Search queries return a maximum of 25 results.

## Examples

### Search images by name

This example displays images with a name containing ‘busybox’:

$ docker search busybox

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

busybox Busybox base image. 316 [OK]

progrium/busybox 50 [OK]

radial/busyboxplus Full-chain, Internet enabled, busybox made... 8 [OK]

odise/busybox-python 2 [OK]

azukiapp/busybox This image is meant to be used as the base... 2 [OK]

ofayau/busybox-jvm Prepare busybox to install a 32 bits JVM. 1 [OK]

shingonoide/archlinux-busybox Arch Linux, a lightweight and flexible Lin... 1 [OK]

odise/busybox-curl 1 [OK]

ofayau/busybox-libc32 Busybox with 32 bits (and 64 bits) libs 1 [OK]

peelsky/zulu-openjdk-busybox 1 [OK]

skomma/busybox-data Docker image suitable for data volume cont... 1 [OK]

elektritter/busybox-teamspeak Lightweight teamspeak3 container based on... 1 [OK]

socketplane/busybox 1 [OK]

oveits/docker-nginx-busybox This is a tiny NginX docker image based on... 0 [OK]

ggtools/busybox-ubuntu Busybox ubuntu version with extra goodies 0 [OK]

nikfoundas/busybox-confd Minimal busybox based distribution of confd 0 [OK]

openshift/busybox-http-app 0 [OK]

jllopis/busybox 0 [OK]

swyckoff/busybox 0 [OK]

powellquiring/busybox 0 [OK]

williamyeh/busybox-sh Docker image for BusyBox's sh 0 [OK]

simplexsys/busybox-cli-powered Docker busybox images, with a few often us... 0 [OK]

fhisamoto/busybox-java Busybox java 0 [OK]

scottabernethy/busybox 0 [OK]

marclop/busybox-solr

### Display non-truncated description (--no-trunc)

This example displays images with a name containing ‘busybox’, at least 3 stars and the description isn’t truncated in the output:

$ docker search --filter=stars=3 --no-trunc busybox

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

busybox Busybox base image. 325 [OK]

progrium/busybox 50 [OK]

radial/busyboxplus Full-chain, Internet enabled, busybox made from scratch. Comes in git and cURL flavors. 8 [OK]

### Limit search results (--limit)

The flag --limit is the maximum number of results returned by a search. This value could be in the range between 1 and 100. The default value of --limit is 25.

### Filtering

The filtering flag (-f or --filter) format is a key=value pair. If there is more than one filter, then pass multiple flags (e.g. --filter is-automated=true --filter stars=3)

The currently supported filters are:

* stars (int - number of stars the image has)
* is-automated (boolean - true or false) - is the image automated or not
* is-official (boolean - true or false) - is the image official or not

#### stars

This example displays images with a name containing ‘busybox’ and at least 3 stars:

$ docker search --filter stars=3 busybox

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

busybox Busybox base image. 325 [OK]

progrium/busybox 50 [OK]

radial/busyboxplus Full-chain, Internet enabled, busybox made... 8 [OK]

#### is-automated

This example displays images with a name containing ‘busybox’ and are automated builds:

$ docker search --filter is-automated=true busybox

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

progrium/busybox 50 [OK]

radial/busyboxplus Full-chain, Internet enabled, busybox made... 8 [OK]

#### is-official

This example displays images with a name containing ‘busybox’, at least 3 stars and are official builds:

$ docker search --filter is-official=true --filter stars=3 busybox

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

progrium/busybox 50 [OK]

radial/busyboxplus Full-chain, Internet enabled, busybox made... 8 [OK]

### Format the output

The formatting option (--format) pretty-prints search output using a Go template.

Valid placeholders for the Go template are:

| **Placeholder** | **Description** |
| --- | --- |
| .Name | Image Name |
| .Description | Image description |
| .StarCount | Number of stars for the image |
| .IsOfficial | “OK” if image is official |
| .IsAutomated | “OK” if image build was automated |

When you use the --format option, the search command will output the data exactly as the template declares. If you use the table directive, column headers are included as well.

The following example uses a template without headers and outputs the Name and StarCount entries separated by a colon for all images:

{% raw %}

$ docker search --format "{{.Name}}: {{.StarCount}}" nginx

nginx: 5441

jwilder/nginx-proxy: 953

richarvey/nginx-php-fpm: 353

million12/nginx-php: 75

webdevops/php-nginx: 70

h3nrik/nginx-ldap: 35

bitnami/nginx: 23

evild/alpine-nginx: 14

million12/nginx: 9

maxexcloo/nginx: 7

{% endraw %}

This example outputs a table format:

{% raw %}

$ docker search --format "table {{.Name}}\t{{.IsAutomated}}\t{{.IsOfficial}}" nginx

NAME AUTOMATED OFFICIAL

nginx [OK]

jwilder/nginx-proxy [OK]

richarvey/nginx-php-fpm [OK]

jrcs/letsencrypt-nginx-proxy-companion [OK]

million12/nginx-php [OK]

webdevops/php-nginx [OK]

{% endraw %}

**docker system**

**Description**

Manage Docker

**Usage**

docker system COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker system df](https://docs.docker.com/engine/reference/commandline/system_df/) | Show docker disk usage |
| [docker system events](https://docs.docker.com/engine/reference/commandline/system_events/) | Get real time events from the server |
| [docker system info](https://docs.docker.com/engine/reference/commandline/system_info/) | Display system-wide information |
| [docker system prune](https://docs.docker.com/engine/reference/commandline/system_prune/) | Remove unused data |

**Parent command**

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

**Extended description**

Manage Docker.

## docker system df

### Description

Show docker disk usage

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

### Usage

docker system df [OPTIONS]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format |  | Pretty-print images using a Go template |
| --verbose , -v |  | Show detailed information on space usage |

### Extended description

The docker system df command displays information regarding the amount of disk space used by the docker daemon.

### Examples

By default the command will just show a summary of the data used:

$ docker system df

TYPE TOTAL ACTIVE SIZE RECLAIMABLE

Images 5 2 16.43 MB 11.63 MB (70%)

Containers 2 0 212 B 212 B (100%)

Local Volumes 2 1 36 B 0 B (0%)

A more detailed view can be requested using the -v, --verbose flag:

$ docker system df -v

Images space usage:

REPOSITORY TAG IMAGE ID CREATED SIZE SHARED SIZE UNIQUE SIZE CONTAINERS

my-curl latest b2789dd875bf 6 minutes ago 11 MB 11 MB 5 B 0

my-jq latest ae67841be6d0 6 minutes ago 9.623 MB 8.991 MB 632.1 kB 0

<none> <none> a0971c4015c1 6 minutes ago 11 MB 11 MB 0 B 0

alpine latest 4e38e38c8ce0 9 weeks ago 4.799 MB 0 B 4.799 MB 1

alpine 3.3 47cf20d8c26c 9 weeks ago 4.797 MB 4.797 MB 0 B 1

Containers space usage:

CONTAINER ID IMAGE COMMAND LOCAL VOLUMES SIZE CREATED STATUS NAMES

4a7f7eebae0f alpine:latest "sh" 1 0 B 16 minutes ago Exited (0) 5 minutes ago hopeful\_yalow

f98f9c2aa1ea alpine:3.3 "sh" 1 212 B 16 minutes ago Exited (0) 48 seconds ago anon-vol

Local Volumes space usage:

NAME LINKS SIZE

07c7bdf3e34ab76d921894c2b834f073721fccfbbcba792aa7648e3a7a664c2e 2 36 B

my-named-vol 0 0 B

* SHARED SIZE is the amount of space that an image shares with another one (i.e. their common data)
* UNIQUE SIZE is the amount of space that is only used by a given image
* SIZE is the virtual size of the image, it is the sum of SHARED SIZE and UNIQUE SIZE

**Note**: Network information is not shown because it doesn’t consume the disk space.

## docker system events

### Description

Get real time events from the server

### Usage

docker system events [OPTIONS]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --filter , -f |  | Filter output based on conditions provided |
| --format |  | Format the output using the given Go template |
| --since |  | Show all events created since timestamp |
| --until |  | Stream events until this timestamp |

## docker system info

### Description

Display system-wide information

### Usage

docker system info [OPTIONS]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format , -f |  | Format the output using the given Go template |

## docker system prune

### Description

Remove unused data

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

### Usage

docker system prune [OPTIONS]

### Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all , -a |  | Remove all unused images not just dangling ones |
| --filter |  | [API 1.28+](https://docs.docker.com/engine/api/v1.28/) Provide filter values (e.g. ‘label==') |
| --force , -f |  | Do not prompt for confirmation |
| --volumes |  | Prune volumes |

**docker secret**

**Description**

Manage Docker secrets

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

Swarm This command works with the Swarm orchestrator.

**Usage**

docker secret COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker secret create](https://docs.docker.com/engine/reference/commandline/secret_create/) | Create a secret from a file or STDIN as content |
| [docker secret inspect](https://docs.docker.com/engine/reference/commandline/secret_inspect/) | Display detailed information on one or more secrets |
| [docker secret ls](https://docs.docker.com/engine/reference/commandline/secret_ls/) | List secrets |
| [docker secret rm](https://docs.docker.com/engine/reference/commandline/secret_rm/) | Remove one or more secrets |

**Parent command**

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

**Extended description**

Manage secrets.

**docker service**

**Description**

Manage services

[API 1.24+](https://docs.docker.com/engine/api/v1.24/)  The client and daemon API must both be at least [1.24](https://docs.docker.com/engine/api/v1.24/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

Swarm This command works with the Swarm orchestrator.

**Usage**

docker service COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker service create](https://docs.docker.com/engine/reference/commandline/service_create/) | Create a new service |
| [docker service inspect](https://docs.docker.com/engine/reference/commandline/service_inspect/) | Display detailed information on one or more services |
| [docker service logs](https://docs.docker.com/engine/reference/commandline/service_logs/) | Fetch the logs of a service or task |
| [docker service ls](https://docs.docker.com/engine/reference/commandline/service_ls/) | List services |
| [docker service ps](https://docs.docker.com/engine/reference/commandline/service_ps/) | List the tasks of one or more services |
| [docker service rm](https://docs.docker.com/engine/reference/commandline/service_rm/) | Remove one or more services |
| [docker service rollback](https://docs.docker.com/engine/reference/commandline/service_rollback/) | Revert changes to a service’s configuration |
| [docker service scale](https://docs.docker.com/engine/reference/commandline/service_scale/) | Scale one or multiple replicated services |
| [docker service update](https://docs.docker.com/engine/reference/commandline/service_update/) | Update a service |

**Parent command**

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

**Extended description**

Manage services.

**Note**: This is a cluster management command, and must be executed on a swarm manager node. To learn about managers and workers, refer to the [Swarm mode section](https://docs.docker.com/engine/swarm/) in the documentation.

**docker stack**

**Description**

Manage Docker stacks

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

**Usage**

docker stack [OPTIONS] COMMAND

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --kubeconfig |  | Kubernetes Kubernetes config file |
| --orchestrator |  | Orchestrator to use (swarm|kubernetes|all) |

**Child commands**

| Command | Description |
| --- | --- |
| [docker stack deploy](https://docs.docker.com/engine/reference/commandline/stack_deploy/) | Deploy a new stack or update an existing stack |
| [docker stack ls](https://docs.docker.com/engine/reference/commandline/stack_ls/) | List stacks |
| [docker stack ps](https://docs.docker.com/engine/reference/commandline/stack_ps/) | List the tasks in the stack |
| [docker stack rm](https://docs.docker.com/engine/reference/commandline/stack_rm/) | Remove one or more stacks |
| [docker stack services](https://docs.docker.com/engine/reference/commandline/stack_services/) | List the services in the stack |

**Parent command**

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

**Extended description**

Manage stacks.

**docker start**

**Description**

Start one or more stopped containers

**Usage**

docker start [OPTIONS] CONTAINER [CONTAINER...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --attach , -a |  | Attach STDOUT/STDERR and forward signals |
| --checkpoint |  | [experimental (daemon)](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file) Restore from this checkpoint |
| --checkpoint-dir |  | [experimental (daemon)](https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file) Use a custom checkpoint storage directory |
| --detach-keys |  | Override the key sequence for detaching a container |
| --interactive , -i |  | Attach container’s STDIN |

**Parent command**

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

**Examples**

$ docker start my\_container

**docker stop**

**Description**

Stop one or more running containers

**Usage**

docker stop [OPTIONS] CONTAINER [CONTAINER...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --time , -t | 10 | Seconds to wait for stop before killing it |

**Parent command**

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

**Extended description**

The main process inside the container will receive SIGTERM, and after a grace period, SIGKILL.

**Examples**

$ docker stop my\_container

**docker stats**

**Description**

Display a live stream of container(s) resource usage statistics

**Usage**

docker stats [OPTIONS] [CONTAINER...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --all , -a |  | Show all containers (default shows just running) |
| --format |  | Pretty-print images using a Go template |
| --no-stream |  | Disable streaming stats and only pull the first result |
| --no-trunc |  | Do not truncate output |

**Parent command**

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

**Extended description**

The docker stats command returns a live data stream for running containers. To limit data to one or more specific containers, specify a list of container names or ids separated by a space. You can specify a stopped container but stopped containers do not return any data.

If you want more detailed information about a container’s resource usage, use the /containers/(id)/stats API endpoint.

**Note**: On Linux, the Docker CLI reports memory usage by subtracting page cache usage from the total memory usage. The API does not perform such a calculation but rather provides the total memory usage and the amount from the page cache so that clients can use the data as needed.

**Note**: The PIDS column contains the number of processes and kernel threads created by that container. Threads is the term used by Linux kernel. Other equivalent terms are “lightweight process” or “kernel task”, etc. A large number in the PIDS column combined with a small number of processes (as reported by ps or top) may indicate that something in the container is creating many threads.

**Examples**

Running docker stats on all running containers against a Linux daemon.

$ docker stats

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

b95a83497c91 awesome\_brattain 0.28% 5.629MiB / 1.952GiB 0.28% 916B / 0B 147kB / 0B 9

67b2525d8ad1 foobar 0.00% 1.727MiB / 1.952GiB 0.09% 2.48kB / 0B 4.11MB / 0B 2

e5c383697914 test-1951.1.kay7x1lh1twk9c0oig50sd5tr 0.00% 196KiB / 1.952GiB 0.01% 71.2kB / 0B 770kB / 0B 1

4bda148efbc0 random.1.vnc8on831idyr42slu578u3cr 0.00% 1.672MiB / 1.952GiB 0.08% 110kB / 0B 578kB / 0B 2

If you don’t [specify a format string using --format](https://docs.docker.com/engine/reference/commandline/stats/#formatting), the following columns are shown.

| **Column name** | **Description** |
| --- | --- |
| CONTAINER ID and Name | the ID and name of the container |
| CPU % and MEM % | the percentage of the host’s CPU and memory the container is using |
| MEM USAGE / LIMIT | the total memory the container is using, and the total amount of memory it is allowed to use |
| NET I/O | The amount of data the container has sent and received over its network interface |
| BLOCK I/O | The amount of data the container has read to and written from block devices on the host |
| PIDs | the number of processes or threads the container has created |

Running docker stats on multiple containers by name and id against a Linux daemon.

$ docker stats awesome\_brattain 67b2525d8ad1

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

b95a83497c91 awesome\_brattain 0.28% 5.629MiB / 1.952GiB 0.28% 916B / 0B 147kB / 0B 9

67b2525d8ad1 foobar 0.00% 1.727MiB / 1.952GiB 0.09% 2.48kB / 0B 4.11MB / 0B 2

Running docker stats with customized format on all (Running and Stopped) containers.

$ docker stats --all --format "table {{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}" fervent\_panini 5acfcb1b4fd1 drunk\_visvesvaraya big\_heisenberg

CONTAINER CPU % MEM USAGE / LIMIT

fervent\_panini 0.00% 56KiB / 15.57GiB

5acfcb1b4fd1 0.07% 32.86MiB / 15.57GiB

drunk\_visvesvaraya 0.00% 0B / 0B

big\_heisenberg 0.00% 0B / 0B

drunk\_visvesvaraya and big\_heisenberg are stopped containers in the above example.

Running docker stats on all running containers against a Windows daemon.

PS E:\> docker stats

CONTAINER ID CPU % PRIV WORKING SET NET I/O BLOCK I/O

09d3bb5b1604 6.61% 38.21 MiB 17.1 kB / 7.73 kB 10.7 MB / 3.57 MB

9db7aa4d986d 9.19% 38.26 MiB 15.2 kB / 7.65 kB 10.6 MB / 3.3 MB

3f214c61ad1d 0.00% 28.64 MiB 64 kB / 6.84 kB 4.42 MB / 6.93 MB

Running docker stats on multiple containers by name and id against a Windows daemon.

PS E:\> docker ps -a

CONTAINER ID NAME IMAGE COMMAND CREATED STATUS PORTS NAMES

3f214c61ad1d awesome\_brattain nanoserver "cmd" 2 minutes ago Up 2 minutes big\_minsky

9db7aa4d986d mad\_wilson windowsservercore "cmd" 2 minutes ago Up 2 minutes mad\_wilson

09d3bb5b1604 fervent\_panini windowsservercore "cmd" 2 minutes ago Up 2 minutes affectionate\_easley

PS E:\> docker stats 3f214c61ad1d mad\_wilson

CONTAINER ID NAME CPU % PRIV WORKING SET NET I/O BLOCK I/O

3f214c61ad1d awesome\_brattain 0.00% 46.25 MiB 76.3 kB / 7.92 kB 10.3 MB / 14.7 MB

9db7aa4d986d mad\_wilson 9.59% 40.09 MiB 27.6 kB / 8.81 kB 17 MB / 20.1 MB

**Formatting**

The formatting option (--format) pretty prints container output using a Go template.

Valid placeholders for the Go template are listed below:

| **Placeholder** | **Description** |
| --- | --- |
| .Container | Container name or ID (user input) |
| .Name | Container name |
| .ID | Container ID |
| .CPUPerc | CPU percentage |
| .MemUsage | Memory usage |
| .NetIO | Network IO |
| .BlockIO | Block IO |
| .MemPerc | Memory percentage (Not available on Windows) |
| .PIDs | Number of PIDs (Not available on Windows) |

When using the --format option, the stats command either outputs the data exactly as the template declares or, when using the table directive, includes column headers as well.

The following example uses a template without headers and outputs the Container and CPUPerc entries separated by a colon for all images:

$ docker stats --format "{{.Container}}: {{.CPUPerc}}"

09d3bb5b1604: 6.61%

9db7aa4d986d: 9.19%

3f214c61ad1d: 0.00%

To list all containers statistics with their name, CPU percentage and memory usage in a table format you can use:

$ docker stats --format "table {{.Container}}\t{{.CPUPerc}}\t{{.MemUsage}}"

CONTAINER CPU % PRIV WORKING SET

1285939c1fd3 0.07% 796 KiB / 64 MiB

9c76f7834ae2 0.07% 2.746 MiB / 64 MiB

d1ea048f04e4 0.03% 4.583 MiB / 64 MiB

The default format is as follows:

On Linux:

"table {{.ID}}\t{{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.MemPerc}}\t{{.NetIO}}\t{{.BlockIO}}\t{{.PIDs}}"

On Windows:

"table {{.ID}}\t{{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.NetIO}}\t{{.BlockIO}}"

**Note**: On Docker 17.09 and older, the {{.Container}} column was used, instead of {{.ID}}\t{{.Name}}.

**docker swarm**

**Description**

Manage Swarm

[API 1.24+](https://docs.docker.com/engine/api/v1.24/)  The client and daemon API must both be at least [1.24](https://docs.docker.com/engine/api/v1.24/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

Swarm This command works with the Swarm orchestrator.

**Usage**

docker swarm COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker swarm ca](https://docs.docker.com/engine/reference/commandline/swarm_ca/) | Display and rotate the root CA |
| [docker swarm init](https://docs.docker.com/engine/reference/commandline/swarm_init/) | Initialize a swarm |
| [docker swarm join](https://docs.docker.com/engine/reference/commandline/swarm_join/) | Join a swarm as a node and/or manager |
| [docker swarm join-token](https://docs.docker.com/engine/reference/commandline/swarm_join-token/) | Manage join tokens |
| [docker swarm leave](https://docs.docker.com/engine/reference/commandline/swarm_leave/) | Leave the swarm |
| [docker swarm unlock](https://docs.docker.com/engine/reference/commandline/swarm_unlock/) | Unlock swarm |
| [docker swarm unlock-key](https://docs.docker.com/engine/reference/commandline/swarm_unlock-key/) | Manage the unlock key |
| [docker swarm update](https://docs.docker.com/engine/reference/commandline/swarm_update/) | Update the swarm |

**Parent command**

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

**Extended description**

Manage the swarm.

**docker tag**

**Description**

Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

**Usage**

docker tag SOURCE\_IMAGE[:TAG] TARGET\_IMAGE[:TAG]

**Parent command**

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

**Extended description**

An image name is made up of slash-separated name components, optionally prefixed by a registry hostname. The hostname must comply with standard DNS rules, but may not contain underscores. If a hostname is present, it may optionally be followed by a port number in the format :8080. If not present, the command uses Docker’s public registry located at registry-1.docker.io by default. Name components may contain lowercase letters, digits and separators. A separator is defined as a period, one or two underscores, or one or more dashes. A name component may not start or end with a separator.

A tag name must be valid ASCII and may contain lowercase and uppercase letters, digits, underscores, periods and dashes. A tag name may not start with a period or a dash and may contain a maximum of 128 characters.

You can group your images together using names and tags, and then upload them to [*Share Images via Repositories*](https://docs.docker.com/engine/tutorials/dockerrepos/#/contributing-to-docker-hub).

**Examples**

**Tag an image referenced by ID**

To tag a local image with ID “0e5574283393” into the “fedora” repository with “version1.0”:

$ docker tag 0e5574283393 fedora/httpd:version1.0

**Tag an image referenced by Name**

To tag a local image with name “httpd” into the “fedora” repository with “version1.0”:

$ docker tag httpd fedora/httpd:version1.0

Note that since the tag name is not specified, the alias is created for an existing local version httpd:latest.

**Tag an image referenced by Name and Tag**

To tag a local image with name “httpd” and tag “test” into the “fedora” repository with “version1.0.test”:

$ docker tag httpd:test fedora/httpd:version1.0.test

**Tag an image for a private repository**

To push an image to a private registry and not the central Docker registry you must tag it with the registry hostname and port (if needed).

$ docker tag 0e5574283393 myregistryhost:5000/fedora/httpd:version1.0

**docker template**

**Description**

Use templates to quickly create new services

This command is only available on Docker Enterprise Edition.

Learn more about [Docker Enterprise products](https://docs.docker.com/ee/supported-platforms/).

This command is experimental on the Docker client.

**It should not be used in production environments.**

To enable experimental features in the Docker CLI, edit the [config.json](https://docs.docker.com/engine/reference/commandline/cli/#configuration-files) and set experimental to enabled. You can go [here](https://docs.docker.com/engine/reference/commandline/cli/#experimental-features) for more information.

**Child commands**

| Command | Description |
| --- | --- |
| [docker template config](https://docs.docker.com/engine/reference/commandline/template_config/) | Modify docker template configuration |
| [docker template inspect](https://docs.docker.com/engine/reference/commandline/template_inspect/) | Inspect service templates or application templates |
| [docker template list](https://docs.docker.com/engine/reference/commandline/template_list/) | List available templates with their informations |
| [docker template scaffold](https://docs.docker.com/engine/reference/commandline/template_scaffold/) | Choose an application template or service template(s) and scaffold a new project |
| [docker template version](https://docs.docker.com/engine/reference/commandline/template_version/) | Print version information |

**Parent command**

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

**docker trust**

**Description**

Manage trust on Docker images

**Usage**

docker trust COMMAND

**Child commands**

| Command | Description |
| --- | --- |
| [docker trust inspect](https://docs.docker.com/engine/reference/commandline/trust_inspect/) | Return low-level information about keys and signatures |
| [docker trust key](https://docs.docker.com/engine/reference/commandline/trust_key/) | Manage keys for signing Docker images |
| [docker trust revoke](https://docs.docker.com/engine/reference/commandline/trust_revoke/) | Remove trust for an image |
| [docker trust sign](https://docs.docker.com/engine/reference/commandline/trust_sign/) | Sign an image |
| [docker trust signer](https://docs.docker.com/engine/reference/commandline/trust_signer/) | Manage entities who can sign Docker images |

**Parent command**

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

**docker update**

**Description**

Update configuration of one or more containers

**Usage**

docker update [OPTIONS] CONTAINER [CONTAINER...]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --blkio-weight |  | Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0) |
| --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 the CPU real-time period in microseconds |
| --cpu-rt-runtime |  | [API 1.25+](https://docs.docker.com/engine/api/v1.25/) Limit the CPU real-time runtime in microseconds |
| --cpu-shares , -c |  | CPU shares (relative weight) |
| --cpus |  | [API 1.29+](https://docs.docker.com/engine/api/v1.29/) 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) |
| --kernel-memory |  | Kernel memory limit |
| --memory , -m |  | Memory limit |
| --memory-reservation |  | Memory soft limit |
| --memory-swap |  | Swap limit equal to memory plus swap: ‘-1’ to enable unlimited swap |
| --pids-limit |  | [API 1.40+](https://docs.docker.com/engine/api/v1.40/) Tune container pids limit (set -1 for unlimited) |
| --restart |  | Restart policy to apply when a container exits |

**Parent command**

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

**Extended description**

The docker update command dynamically updates container configuration. You can use this command to prevent containers from consuming too many resources from their Docker host. With a single command, you can place limits on a single container or on many. To specify more than one container, provide space-separated list of container names or IDs.

With the exception of the --kernel-memory option, you can specify these options on a running or a stopped container. On kernel version older than 4.6, you can only update --kernel-memory on a stopped container or on a running container with kernel memory initialized.

**Warning**: The docker update and docker container update commands are not supported for Windows containers.

**Examples**

The following sections illustrate ways to use this command.

**Update a container’s cpu-shares**

To limit a container’s cpu-shares to 512, first identify the container name or ID. You can use docker ps to find these values. You can also use the ID returned from the docker run command. Then, do the following:

$ docker update --cpu-shares 512 abebf7571666

**Update a container with cpu-shares and memory**

To update multiple resource configurations for multiple containers:

$ docker update --cpu-shares 512 -m 300M abebf7571666 hopeful\_morse

**Update a container’s kernel memory constraints**

You can update a container’s kernel memory limit using the --kernel-memory option. On kernel version older than 4.6, this option can be updated on a running container only if the container was started with --kernel-memory. If the container was started *without* --kernel-memory you need to stop the container before updating kernel memory.

For example, if you started a container with this command:

$ docker run -dit --name test --kernel-memory 50M ubuntu bash

You can update kernel memory while the container is running:

$ docker update --kernel-memory 80M test

If you started a container *without* kernel memory initialized:

$ docker run -dit --name test2 --memory 300M ubuntu bash

Update kernel memory of running container test2 will fail. You need to stop the container before updating the --kernel-memory setting. The next time you start it, the container uses the new value.

Kernel version newer than (include) 4.6 does not have this limitation, you can use --kernel-memory the same way as other options.

**Update a container’s restart policy**

You can change a container’s restart policy on a running container. The new restart policy takes effect instantly after you run docker update on a container.

To update restart policy for one or more containers:

$ docker update --restart=on-failure:3 abebf7571666 hopeful\_morse

Note that if the container is started with “--rm” flag, you cannot update the restart policy for it. The AutoRemove and RestartPolicy are mutually exclusive for the container.

**docker version**

**Description**

Show the Docker version information

**Usage**

docker version [OPTIONS]

**Options**

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --format , -f |  | Format the output using the given Go template |
| --kubeconfig |  | Kubernetes Kubernetes config file |

**Parent command**

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

**Extended description**

By default, this will render all version information in an easy to read layout. If a format is specified, the given template will be executed instead.

Go’s [text/template](http://golang.org/pkg/text/template/) package describes all the details of the format.

**Examples**

**Default output**

$ docker version

Client:

Version: 1.8.0

API version: 1.20

Go version: go1.4.2

Git commit: f5bae0a

Built: Tue Jun 23 17:56:00 UTC 2015

OS/Arch: linux/amd64

Server:

Version: 1.8.0

API version: 1.20

Go version: go1.4.2

Git commit: f5bae0a

Built: Tue Jun 23 17:56:00 UTC 2015

OS/Arch: linux/amd64

**Get the server version**

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

1.8.0

**Dump raw JSON data**

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

{"Client":{"Version":"1.8.0","ApiVersion":"1.20","GitCommit":"f5bae0a","GoVersion":"go1.4.2","Os":"linux","Arch":"amd64","BuildTime":"Tue Jun 23 17:56:00 UTC 2015"},"ServerOK":true,"Server":{"Version":"1.8.0","ApiVersion":"1.20","GitCommit":"f5bae0a","GoVersion":"go1.4.2","Os":"linux","Arch":"amd64","KernelVersion":"3.13.2-gentoo","BuildTime":"Tue Jun 23 17:56:00 UTC 2015"}}

# docker volume

## Description

Manage volumes

[API 1.21+](https://docs.docker.com/engine/api/v1.21/)  The client and daemon API must both be at least [1.21](https://docs.docker.com/engine/api/v1.21/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

## Usage

docker volume COMMAND COMMAND

## docker volume create

## Description

Create a volume

[API 1.21+](https://docs.docker.com/engine/api/v1.21/)  The client and daemon API must both be at least [1.21](https://docs.docker.com/engine/api/v1.21/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

## Usage

docker volume create [OPTIONS] [VOLUME]

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --driver , -d | local | Specify volume driver name |
| --label |  | Set metadata for a volume |
| --name |  | Specify volume name |
| --opt , -o |  | Set driver specific options |

## docker volume inspect

## Description

Display detailed information on one or more volumes

[API 1.21+](https://docs.docker.com/engine/api/v1.21/)  The client and daemon API must both be at least [1.21](https://docs.docker.com/engine/api/v1.21/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

## Usage

docker volume inspect [OPTIONS] VOLUME [VOLUME...]

## docker volume ls

## Description

List volumes

[API 1.21+](https://docs.docker.com/engine/api/v1.21/)  The client and daemon API must both be at least [1.21](https://docs.docker.com/engine/api/v1.21/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

## Usage

docker volume ls [OPTIONS]

## docker volume prune

## Description

Remove all unused local volumes

[API 1.25+](https://docs.docker.com/engine/api/v1.25/)  The client and daemon API must both be at least [1.25](https://docs.docker.com/engine/api/v1.25/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

## Usage

docker volume prune [OPTIONS]

## docker volume rm

## Description

Remove one or more volumes

[API 1.21+](https://docs.docker.com/engine/api/v1.21/)  The client and daemon API must both be at least [1.21](https://docs.docker.com/engine/api/v1.21/) to use this command. Use the docker version command on the client to check your client and daemon API versions.

## Usage

docker volume rm [OPTIONS] VOLUME [VOLUME...]

## Options

| Name, shorthand | Default | Description |
| --- | --- | --- |
| --force , -f |  | [API 1.25+](https://docs.docker.com/engine/api/v1.25/) Force the removal of one or more volumes |

|  |
| --- |
|  |
|  |

# Swarm mode overview

Estimated reading time: 3 minutes

To use Docker in swarm mode, install Docker. See [installation instructions](https://docs.docker.com/install/) for all operating systems and platforms.

Current versions of Docker include swarm mode for natively managing a cluster of Docker Engines called a swarm. Use the Docker CLI to create a swarm, deploy application services to a swarm, and manage swarm behavior.

## Feature highlights

* **Cluster management integrated with Docker Engine:** Use the Docker Engine CLI to create a swarm of Docker Engines where you can deploy application services. You don’t need additional orchestration software to create or manage a swarm.
* **Decentralized design:** Instead of handling differentiation between node roles at deployment time, the Docker Engine handles any specialization at runtime. You can deploy both kinds of nodes, managers and workers, using the Docker Engine. This means you can build an entire swarm from a single disk image.
* **Declarative service model:** Docker Engine uses a declarative approach to let you define the desired state of the various services in your application stack. For example, you might describe an application comprised of a web front end service with message queueing services and a database backend.
* **Scaling:** For each service, you can declare the number of tasks you want to run. When you scale up or down, the swarm manager automatically adapts by adding or removing tasks to maintain the desired state.
* **Desired state reconciliation:** The swarm manager node constantly monitors the cluster state and reconciles any differences between the actual state and your expressed desired state. For example, if you set up a service to run 10 replicas of a container, and a worker machine hosting two of those replicas crashes, the manager creates two new replicas to replace the replicas that crashed. The swarm manager assigns the new replicas to workers that are running and available.
* **Multi-host networking:** You can specify an overlay network for your services. The swarm manager automatically assigns addresses to the containers on the overlay network when it initializes or updates the application.
* **Service discovery:** Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers. You can query every container running in the swarm through a DNS server embedded in the swarm.
* **Load balancing:** You can expose the ports for services to an external load balancer. Internally, the swarm lets you specify how to distribute service containers between nodes.
* **Secure by default:** Each node in the swarm enforces TLS mutual authentication and encryption to secure communications between itself and all other nodes. You have the option to use self-signed root certificates or certificates from a custom root CA.
* **Rolling updates:** At rollout time you can apply service updates to nodes incrementally. The swarm manager lets you control the delay between service deployment to different sets of nodes. If anything goes wrong, you can roll back to a previous version of the service.

## What’s next?

### Swarm mode key concepts and tutorial

* Learn swarm mode [key concepts](https://docs.docker.com/engine/swarm/key-concepts/).
* Get started with the [Swarm mode tutorial](https://docs.docker.com/engine/swarm/swarm-tutorial/).

### Swarm mode CLI commands

Explore swarm mode CLI commands

* [swarm init](https://docs.docker.com/engine/reference/commandline/swarm_init/)
* [swarm join](https://docs.docker.com/engine/reference/commandline/swarm_join/)
* [service create](https://docs.docker.com/engine/reference/commandline/service_create/)
* [service inspect](https://docs.docker.com/engine/reference/commandline/service_inspect/)
* [service ls](https://docs.docker.com/engine/reference/commandline/service_ls/)
* [service rm](https://docs.docker.com/engine/reference/commandline/service_rm/)
* [service scale](https://docs.docker.com/engine/reference/commandline/service_scale/)
* [service ps](https://docs.docker.com/engine/reference/commandline/service_ps/)
* [service update](https://docs.docker.com/engine/reference/commandline/service_update/)

# Dockerfile reference

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

This page describes the commands you can use in a Dockerfile. When you are done reading this page, refer to the [Dockerfile Best Practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/) for a tip-oriented guide.

## Usage

The [docker build](https://docs.docker.com/engine/reference/commandline/build/) command builds an image from a Dockerfile and a context. The build’s context is the set of files at a specified location PATH or URL. The PATH is a directory on your local filesystem. The URL is a Git repository location.

A context is processed recursively. So, a PATH includes any subdirectories and the URL includes the repository and its submodules. This example shows a build command that uses the current directory as context:

$ docker build .

Sending build context to Docker daemon 6.51 MB

...

The build is run by the Docker daemon, not by the CLI. The first thing a build process does is send the entire context (recursively) to the daemon. In most cases, it’s best to start with an empty directory as context and keep your Dockerfile in that directory. Add only the files needed for building the Dockerfile.

**Warning**: Do not use your root directory, /, as the PATH as it causes the build to transfer the entire contents of your hard drive to the Docker daemon.

To use a file in the build context, the Dockerfile refers to the file specified in an instruction, for example, a COPY instruction. To increase the build’s performance, exclude files and directories by adding a .dockerignore file to the context directory. For information about how to [create a .dockerignore file](https://docs.docker.com/engine/reference/builder/#dockerignore-file) see the documentation on this page.

Traditionally, the Dockerfile is called Dockerfile and located in the root of the context. You use the -f flag with docker build to point to a Dockerfile anywhere in your file system.

$ docker build -f /path/to/a/Dockerfile .

You can specify a repository and tag at which to save the new image if the build succeeds:

$ docker build -t shykes/myapp .

To tag the image into multiple repositories after the build, add multiple -t parameters when you run the build command:

$ docker build -t shykes/myapp:1.0.2 -t shykes/myapp:latest .

Before the Docker daemon runs the instructions in the Dockerfile, it performs a preliminary validation of the Dockerfile and returns an error if the syntax is incorrect:

$ docker build -t test/myapp .

Sending build context to Docker daemon 2.048 kB

Error response from daemon: Unknown instruction: RUNCMD

The Docker daemon runs the instructions in the Dockerfile one-by-one, committing the result of each instruction to a new image if necessary, before finally outputting the ID of your new image. The Docker daemon will automatically clean up the context you sent.

Note that each instruction is run independently, and causes a new image to be created - so RUN cd /tmp will not have any effect on the next instructions.

Whenever possible, Docker will re-use the intermediate images (cache), to accelerate the docker build process significantly. This is indicated by the Using cache message in the console output. (For more information, see the [Dockerfile best practices guide](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/):

$ docker build -t svendowideit/ambassador .

Sending build context to Docker daemon 15.36 kB

Step 1/4 : FROM alpine:3.2

---> 31f630c65071

Step 2/4 : MAINTAINER SvenDowideit@home.org.au

---> Using cache

---> 2a1c91448f5f

Step 3/4 : RUN apk update && apk add socat && rm -r /var/cache/

---> Using cache

---> 21ed6e7fbb73

Step 4/4 : CMD env | grep \_TCP= | (sed 's/.\*\_PORT\_\([0-9]\*\)\_TCP=tcp:\/\/\(.\*\):\(.\*\)/socat -t 100000000 TCP4-LISTEN:\1,fork,reuseaddr TCP4:\2:\3 \&/' && echo wait) | sh

---> Using cache

---> 7ea8aef582cc

Successfully built 7ea8aef582cc

Build cache is only used from images that have a local parent chain. This means that these images were created by previous builds or the whole chain of images was loaded with docker load. If you wish to use build cache of a specific image you can specify it with --cache-from option. Images specified with --cache-from do not need to have a parent chain and may be pulled from other registries.

When you’re done with your build, you’re ready to look into [Pushing a repository to its registry](https://docs.docker.com/engine/tutorials/dockerrepos/#/contributing-to-docker-hub).

## BuildKit

Starting with version 18.09, Docker supports a new backend for executing your builds that is provided by the [moby/buildkit](https://github.com/moby/buildkit) project. The BuildKit backend provides many benefits compared to the old implementation. For example, BuildKit can:

* Detect and skip executing unused build stages
* Parallelize building independent build stages
* Incrementally transfer only the changed files in your build context between builds
* Detect and skip transferring unused files in your build context
* Use external Dockerfile implementations with many new features
* Avoid side-effects with rest of the API (intermediate images and containers)
* Prioritize your build cache for automatic pruning

To use the BuildKit backend, you need to set an environment variable DOCKER\_BUILDKIT=1 on the CLI before invoking docker build.

To learn about the experimental Dockerfile syntax available to BuildKit-based builds [refer to the documentation in the BuildKit repository](https://github.com/moby/buildkit/blob/master/frontend/dockerfile/docs/experimental.md).

## Format

Here is the format of the Dockerfile:

# Comment

INSTRUCTION arguments

The instruction is not case-sensitive. However, convention is for them to be UPPERCASE to distinguish them from arguments more easily.

Docker runs instructions in a Dockerfile in order. A Dockerfile **must begin with a `FROM` instruction**. This may be after [parser directives](https://docs.docker.com/engine/reference/builder/#parser-directives), [comments](https://docs.docker.com/engine/reference/builder/#format), and globally scoped [ARGs](https://docs.docker.com/engine/reference/builder/#arg). The FROM instruction specifies the [Parent Image](https://docs.docker.com/engine/reference/glossary/#parent-image) from which you are building. FROM may only be preceded by one or more ARG instructions, which declare arguments that are used in FROM lines in the Dockerfile.

Docker treats lines that begin with # as a comment, unless the line is a valid [parser directive](https://docs.docker.com/engine/reference/builder/#parser-directives). A # marker anywhere else in a line is treated as an argument. This allows statements like:

# Comment

RUN echo 'we are running some # of cool things'

Line continuation characters are not supported in comments.

## Parser directives

Parser directives are optional, and affect the way in which subsequent lines in a Dockerfile are handled. Parser directives do not add layers to the build, and will not be shown as a build step. Parser directives are written as a special type of comment in the form # directive=value. A single directive may only be used once.

Once a comment, empty line or builder instruction has been processed, Docker no longer looks for parser directives. Instead it treats anything formatted as a parser directive as a comment and does not attempt to validate if it might be a parser directive. Therefore, all parser directives must be at the very top of a Dockerfile.

Parser directives are not case-sensitive. However, convention is for them to be lowercase. Convention is also to include a blank line following any parser directives. Line continuation characters are not supported in parser directives.

Due to these rules, the following examples are all invalid:

Invalid due to line continuation:

# direc \

tive=value

Invalid due to appearing twice:

# directive=value1

# directive=value2

FROM ImageName

Treated as a comment due to appearing after a builder instruction:

FROM ImageName

# directive=value

Treated as a comment due to appearing after a comment which is not a parser directive:

# About my dockerfile

# directive=value

FROM ImageName

The unknown directive is treated as a comment due to not being recognized. In addition, the known directive is treated as a comment due to appearing after a comment which is not a parser directive.

# unknowndirective=value

# knowndirective=value

Non line-breaking whitespace is permitted in a parser directive. Hence, the following lines are all treated identically:

#directive=value

# directive =value

# directive= value

# directive = value

# dIrEcTiVe=value

The following parser directives are supported:

* syntax
* escape

## syntax

# syntax=[remote image reference]

For example:

# syntax=docker/dockerfile

# syntax=docker/dockerfile:1.0

# syntax=docker.io/docker/dockerfile:1

# syntax=docker/dockerfile:1.0.0-experimental

# syntax=example.com/user/repo:tag@sha256:abcdef...

This feature is only enabled if the [BuildKit](https://docs.docker.com/engine/reference/builder/#buildkit) backend is used.

The syntax directive defines the location of the Dockerfile builder that is used for building the current Dockerfile. The BuildKit backend allows to seamlessly use external implementations of builders that are distributed as Docker images and execute inside a container sandbox environment.

Custom Dockerfile implementation allows you to:

* Automatically get bugfixes without updating the daemon
* Make sure all users are using the same implementation to build your Dockerfile
* Use the latest features without updating the daemon
* Try out new experimental or third-party features

### Official releases

Docker distributes official versions of the images that can be used for building Dockerfiles under docker/dockerfile repository on Docker Hub. There are two channels where new images are released: stable and experimental.

Stable channel follows semantic versioning. For example:

* docker/dockerfile:1.0.0 - only allow immutable version 1.0.0
* docker/dockerfile:1.0 - allow versions 1.0.\*
* docker/dockerfile:1 - allow versions 1..
* docker/dockerfile:latest - latest release on stable channel

The experimental channel uses incremental versioning with the major and minor component from the stable channel on the time of the release. For example:

* docker/dockerfile:1.0.1-experimental - only allow immutable version 1.0.1-experimental
* docker/dockerfile:1.0-experimental - latest experimental releases after 1.0
* docker/dockerfile:experimental - latest release on experimental channel

You should choose a channel that best fits your needs. If you only want bugfixes, you should use docker/dockerfile:1.0. If you want to benefit from experimental features, you should use the experimental channel. If you are using the experimental channel, newer releases may not be backwards compatible, so it is recommended to use an immutable full version variant.

For master builds and nightly feature releases refer to the description in [the source repository](https://github.com/moby/buildkit/blob/master/README.md).

## escape

# escape=\ (backslash)

Or

# escape=` (backtick)

The escape directive sets the character used to escape characters in a Dockerfile. If not specified, the default escape character is \.

The escape character is used both to escape characters in a line, and to escape a newline. This allows a Dockerfile instruction to span multiple lines. Note that regardless of whether the escape parser directive is included in a Dockerfile, escaping is not performed in a *RUN* command, except at the end of a line.

Setting the escape character to ` is especially useful on Windows, where \ is the directory path separator. ` is consistent with [Windows PowerShell](https://technet.microsoft.com/en-us/library/hh847755.aspx).

Consider the following example which would fail in a non-obvious way on Windows. The second \ at the end of the second line would be interpreted as an escape for the newline, instead of a target of the escape from the first \. Similarly, the \ at the end of the third line would, assuming it was actually handled as an instruction, cause it be treated as a line continuation. The result of this dockerfile is that second and third lines are considered a single instruction:

FROM microsoft/nanoserver

COPY testfile.txt c:\\

RUN dir c:\

Results in:

PS C:\John> docker build -t cmd .

Sending build context to Docker daemon 3.072 kB

Step 1/2 : FROM microsoft/nanoserver

---> 22738ff49c6d

Step 2/2 : COPY testfile.txt c:\RUN dir c:

GetFileAttributesEx c:RUN: The system cannot find the file specified.

PS C:\John>

One solution to the above would be to use / as the target of both the COPY instruction, and dir. However, this syntax is, at best, confusing as it is not natural for paths on Windows, and at worst, error prone as not all commands on Windows support / as the path separator.

By adding the escape parser directive, the following Dockerfile succeeds as expected with the use of natural platform semantics for file paths on Windows:

# escape=`

FROM microsoft/nanoserver

COPY testfile.txt c:\

RUN dir c:\

Results in:

PS C:\John> docker build -t succeeds --no-cache=true .

Sending build context to Docker daemon 3.072 kB

Step 1/3 : FROM microsoft/nanoserver

---> 22738ff49c6d

Step 2/3 : COPY testfile.txt c:\

---> 96655de338de

Removing intermediate container 4db9acbb1682

Step 3/3 : RUN dir c:\

---> Running in a2c157f842f5

Volume in drive C has no label.

Volume Serial Number is 7E6D-E0F7

Directory of c:\

10/05/2016 05:04 PM 1,894 License.txt

10/05/2016 02:22 PM <DIR> Program Files

10/05/2016 02:14 PM <DIR> Program Files (x86)

10/28/2016 11:18 AM 62 testfile.txt

10/28/2016 11:20 AM <DIR> Users

10/28/2016 11:20 AM <DIR> Windows

2 File(s) 1,956 bytes

4 Dir(s) 21,259,096,064 bytes free

---> 01c7f3bef04f

Removing intermediate container a2c157f842f5

Successfully built 01c7f3bef04f

PS C:\John>

## Environment replacement

Environment variables (declared with [the ENV statement](https://docs.docker.com/engine/reference/builder/#env)) can also be used in certain instructions as variables to be interpreted by the Dockerfile. Escapes are also handled for including variable-like syntax into a statement literally.

Environment variables are notated in the Dockerfile either with $variable\_name or ${variable\_name}. They are treated equivalently and the brace syntax is typically used to address issues with variable names with no whitespace, like ${foo}\_bar.

The ${variable\_name} syntax also supports a few of the standard bash modifiers as specified below:

* ${variable:-word} indicates that if variable is set then the result will be that value. If variable is not set then word will be the result.
* ${variable:+word} indicates that if variable is set then word will be the result, otherwise the result is the empty string.

In all cases, word can be any string, including additional environment variables.

Escaping is possible by adding a \ before the variable: \$foo or \${foo}, for example, will translate to $foo and ${foo} literals respectively.

Example (parsed representation is displayed after the #):

FROM busybox

ENV foo /bar

WORKDIR ${foo} # WORKDIR /bar

ADD . $foo # ADD . /bar

COPY \$foo /quux # COPY $foo /quux

Environment variables are supported by the following list of instructions in the Dockerfile:

* ADD
* COPY
* ENV
* EXPOSE
* FROM
* LABEL
* STOPSIGNAL
* USER
* VOLUME
* WORKDIR

as well as:

* ONBUILD (when combined with one of the supported instructions above)

**Note**: prior to 1.4, ONBUILD instructions did **NOT** support environment variable, even when combined with any of the instructions listed above.

Environment variable substitution will use the same value for each variable throughout the entire instruction. In other words, in this example:

ENV abc=hello

ENV abc=bye def=$abc

ENV ghi=$abc

will result in def having a value of hello, not bye. However, ghi will have a value of bye because it is not part of the same instruction that set abc to bye.

## .dockerignore file

Before the docker CLI sends the context to the docker daemon, it looks for a file named .dockerignore in the root directory of the context. If this file exists, the CLI modifies the context to exclude files and directories that match patterns in it. This helps to avoid unnecessarily sending large or sensitive files and directories to the daemon and potentially adding them to images using ADD or COPY.

The CLI interprets the .dockerignore file as a newline-separated list of patterns similar to the file globs of Unix shells. For the purposes of matching, the root of the context is considered to be both the working and the root directory. For example, the patterns /foo/bar and foo/bar both exclude a file or directory named bar in the foo subdirectory of PATH or in the root of the git repository located at URL. Neither excludes anything else.

If a line in .dockerignore file starts with # in column 1, then this line is considered as a comment and is ignored before interpreted by the CLI.

Here is an example .dockerignore file:

# comment

\*/temp\*

\*/\*/temp\*

temp?

This file causes the following build behavior:

| **Rule** | **Behavior** |
| --- | --- |
| # comment | Ignored. |
| \*/temp\* | Exclude files and directories whose names start with temp in any immediate subdirectory of the root. For example, the plain file /somedir/temporary.txt is excluded, as is the directory /somedir/temp. |
| \*/\*/temp\* | Exclude files and directories starting with temp from any subdirectory that is two levels below the root. For example, /somedir/subdir/temporary.txt is excluded. |
| temp? | Exclude files and directories in the root directory whose names are a one-character extension of temp. For example, /tempa and /tempb are excluded. |

Matching is done using Go’s [filepath.Match](http://golang.org/pkg/path/filepath#Match) rules. A preprocessing step removes leading and trailing whitespace and eliminates . and .. elements using Go’s [filepath.Clean](http://golang.org/pkg/path/filepath/#Clean). Lines that are blank after preprocessing are ignored.

Beyond Go’s filepath.Match rules, Docker also supports a special wildcard string \*\* that matches any number of directories (including zero). For example, \*\*/\*.go will exclude all files that end with .go that are found in all directories, including the root of the build context.

Lines starting with ! (exclamation mark) can be used to make exceptions to exclusions. The following is an example .dockerignore file that uses this mechanism:

\*.md

!README.md

All markdown files except README.md are excluded from the context.

The placement of ! exception rules influences the behavior: the last line of the .dockerignore that matches a particular file determines whether it is included or excluded. Consider the following example:

\*.md

!README\*.md

README-secret.md

No markdown files are included in the context except README files other than README-secret.md.

Now consider this example:

\*.md

README-secret.md

!README\*.md

All of the README files are included. The middle line has no effect because !README\*.md matches README-secret.md and comes last.

You can even use the .dockerignore file to exclude the Dockerfile and .dockerignore files. These files are still sent to the daemon because it needs them to do its job. But the ADD and COPY instructions do not copy them to the image.

Finally, you may want to specify which files to include in the context, rather than which to exclude. To achieve this, specify \* as the first pattern, followed by one or more ! exception patterns.

**Note**: For historical reasons, the pattern . is ignored.

## FROM

FROM [--platform=<platform>] <image> [AS <name>]

Or

FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]

Or

FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]

The FROM instruction initializes a new build stage and sets the [Base Image](https://docs.docker.com/glossary/#base-image) for subsequent instructions. As such, a valid Dockerfile must start with a FROM instruction. The image can be any valid image – it is especially easy to start by **pulling an image** from the [Public Repositories](https://docs.docker.com/engine/tutorials/dockerrepos/).

* ARG is the only instruction that may precede FROM in the Dockerfile. See [Understand how ARG and FROM interact](https://docs.docker.com/engine/reference/builder/#understand-how-arg-and-from-interact).
* FROM can appear multiple times within a single Dockerfile to create multiple images or use one build stage as a dependency for another. Simply make a note of the last image ID output by the commit before each new FROM instruction. Each FROM instruction clears any state created by previous instructions.
* Optionally a name can be given to a new build stage by adding AS name to the FROM instruction. The name can be used in subsequent FROM and COPY --from=<name|index> instructions to refer to the image built in this stage.
* The tag or digest values are optional. If you omit either of them, the builder assumes a latest tag by default. The builder returns an error if it cannot find the tag value.

The optional --platform flag can be used to specify the platform of the image in case FROM references a multi-platform image. For example, linux/amd64, linux/arm64, or windows/amd64. By default, the target platform of the build request is used. Global build arguments can be used in the value of this flag, for example [automatic platform ARGs](https://docs.docker.com/engine/reference/builder/#automatic-platform-args-in-the-global-scope) allow you to force a stage to native build platform (--platform=$BUILDPLATFORM), and use it to cross-compile to the target platform inside the stage.

### Understand how ARG and FROM interact

FROM instructions support variables that are declared by any ARG instructions that occur before the first FROM.

ARG CODE\_VERSION=latest

FROM base:${CODE\_VERSION}

CMD /code/run-app

FROM extras:${CODE\_VERSION}

CMD /code/run-extras

An ARG declared before a FROM is outside of a build stage, so it can’t be used in any instruction after a FROM. To use the default value of an ARG declared before the first FROM use an ARG instruction without a value inside of a build stage:

ARG VERSION=latest

FROM busybox:$VERSION

ARG VERSION

RUN echo $VERSION > image\_version

## RUN

RUN has 2 forms:

* RUN <command> (shell form, the command is run in a shell, which by default is /bin/sh -c on Linux or cmd /S /C on Windows)
* RUN ["executable", "param1", "param2"] (exec form)

The RUN instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.

Layering RUN instructions and generating commits conforms to the core concepts of Docker where commits are cheap and containers can be created from any point in an image’s history, much like source control.

The exec form makes it possible to avoid shell string munging, and to RUN commands using a base image that does not contain the specified shell executable.

The default shell for the shell form can be changed using the SHELL command.

In the shell form you can use a \ (backslash) to continue a single RUN instruction onto the next line. For example, consider these two lines:

RUN /bin/bash -c 'source $HOME/.bashrc; \

echo $HOME'

Together they are equivalent to this single line:

RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'

**Note**: To use a different shell, other than ‘/bin/sh’, use the exec form passing in the desired shell. For example, RUN ["/bin/bash", "-c", "echo hello"]

**Note**: The exec form is parsed as a JSON array, which means that you must use double-quotes (“) around words not single-quotes (‘).

**Note**: Unlike the shell form, the exec form does not invoke a command shell. This means that normal shell processing does not happen. For example, RUN [ "echo", "$HOME" ] will not do variable substitution on $HOME. If you want shell processing then either use the shell form or execute a shell directly, for example: RUN [ "sh", "-c", "echo $HOME" ]. When using the exec form and executing a shell directly, as in the case for the shell form, it is the shell that is doing the environment variable expansion, not docker.

**Note**: In the JSON form, it is necessary to escape backslashes. This is particularly relevant on Windows where the backslash is the path separator. The following line would otherwise be treated as shell form due to not being valid JSON, and fail in an unexpected way: RUN ["c:\windows\system32\tasklist.exe"] The correct syntax for this example is: RUN ["c:\\windows\\system32\\tasklist.exe"]

The cache for RUN instructions isn’t invalidated automatically during the next build. The cache for an instruction like RUN apt-get dist-upgrade -y will be reused during the next build. The cache for RUN instructions can be invalidated by using the --no-cache flag, for example docker build --no-cache.

See the [Dockerfile Best Practices guide](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/) for more information.

The cache for RUN instructions can be invalidated by ADD instructions. See [below](https://docs.docker.com/engine/reference/builder/#add) for details.

### Known issues (RUN)

* [Issue 783](https://github.com/docker/docker/issues/783) is about file permissions problems that can occur when using the AUFS file system. You might notice it during an attempt to rm a file, for example.

For systems that have recent aufs version (i.e., dirperm1 mount option can be set), docker will attempt to fix the issue automatically by mounting the layers with dirperm1 option. More details on dirperm1 option can be found at [aufs man page](https://github.com/sfjro/aufs3-linux/tree/aufs3.18/Documentation/filesystems/aufs)

If your system doesn’t have support for dirperm1, the issue describes a workaround.

## CMD

The CMD instruction has three forms:

* CMD ["executable","param1","param2"] (exec form, this is the preferred form)
* CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
* CMD command param1 param2 (shell form)

There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect.

**The main purpose of a CMD is to provide defaults for an executing container.** These defaults can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well.

**Note**: If CMD is used to provide default arguments for the ENTRYPOINT instruction, both the CMD and ENTRYPOINT instructions should be specified with the JSON array format.

**Note**: The exec form is parsed as a JSON array, which means that you must use double-quotes (“) around words not single-quotes (‘).

**Note**: Unlike the shell form, the exec form does not invoke a command shell. This means that normal shell processing does not happen. For example, CMD [ "echo", "$HOME" ] will not do variable substitution on $HOME. If you want shell processing then either use the shell form or execute a shell directly, for example: CMD [ "sh", "-c", "echo $HOME" ]. When using the exec form and executing a shell directly, as in the case for the shell form, it is the shell that is doing the environment variable expansion, not docker.

When used in the shell or exec formats, the CMD instruction sets the command to be executed when running the image.

If you use the shell form of the CMD, then the <command> will execute in /bin/sh -c:

FROM ubuntu

CMD echo "This is a test." | wc -

If you want to **run your** <command> **without a shell** then you must express the command as a JSON array and give the full path to the executable. **This array form is the preferred format of CMD.** Any additional parameters must be individually expressed as strings in the array:

FROM ubuntu

CMD ["/usr/bin/wc","--help"]

If you would like your container to run the same executable every time, then you should consider using ENTRYPOINT in combination with CMD. See [ENTRYPOINT](https://docs.docker.com/engine/reference/builder/#entrypoint).

If the user specifies arguments to docker run then they will override the default specified in CMD.

**Note**: Don’t confuse RUN with CMD. RUN actually runs a command and commits the result; CMD does not execute anything at build time, but specifies the intended command for the image.

## LABEL

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

The LABEL instruction adds metadata to an image. A LABEL is a key-value pair. To include spaces within a LABEL value, use quotes and backslashes as you would in command-line parsing. A few usage examples:

LABEL "com.example.vendor"="ACME Incorporated"

LABEL com.example.label-with-value="foo"

LABEL version="1.0"

LABEL description="This text illustrates \

that label-values can span multiple lines."

An image can have more than one label. You can specify multiple labels on a single line. Prior to Docker 1.10, this decreased the size of the final image, but this is no longer the case. You may still choose to specify multiple labels in a single instruction, in one of the following two ways:

LABEL multi.label1="value1" multi.label2="value2" other="value3"

LABEL multi.label1="value1" \

multi.label2="value2" \

other="value3"

Labels included in base or parent images (images in the FROM line) are inherited by your image. If a label already exists but with a different value, the most-recently-applied value overrides any previously-set value.

To view an image’s labels, use the docker inspect command.

"Labels": {

"com.example.vendor": "ACME Incorporated"

"com.example.label-with-value": "foo",

"version": "1.0",

"description": "This text illustrates that label-values can span multiple lines.",

"multi.label1": "value1",

"multi.label2": "value2",

"other": "value3"

},

## MAINTAINER (deprecated)

MAINTAINER <name>

The MAINTAINER instruction sets the Author field of the generated images. The LABEL instruction is a much more flexible version of this and you should use it instead, as it enables setting any metadata you require, and can be viewed easily, for example with docker inspect. To set a label corresponding to the MAINTAINER field you could use:

LABEL maintainer="SvenDowideit@home.org.au"

This will then be visible from docker inspect with the other labels.

## EXPOSE

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

The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

The EXPOSE instruction does not actually publish the port. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published. To actually publish the port when running the container, use the -p flag on docker run to publish and map one or more ports, or the -P flag to publish all exposed ports and map them to high-order ports.

By default, EXPOSE assumes TCP. You can also specify UDP:

EXPOSE 80/udp

To expose on both TCP and UDP, include two lines:

EXPOSE 80/tcp

EXPOSE 80/udp

In this case, if you use -P with docker run, the port will be exposed once for TCP and once for UDP. Remember that -P uses an ephemeral high-ordered host port on the host, so the port will not be the same for TCP and UDP.

Regardless of the EXPOSE settings, you can override them at runtime by using the -p flag. For example

docker run -p 80:80/tcp -p 80:80/udp ...

To set up port redirection on the host system, see [using the -P flag](https://docs.docker.com/engine/reference/run/#expose-incoming-ports). The docker network command supports creating networks for communication among containers without the need to expose or publish specific ports, because the containers connected to the network can communicate with each other over any port. For detailed information, see the [overview of this feature](https://docs.docker.com/engine/userguide/networking/)).

## ENV

ENV <key> <value>

ENV <key>=<value> ...

The ENV instruction sets the environment variable <key> to the value <value>. This value will be in the environment for all subsequent instructions in the build stage and can be [replaced inline](https://docs.docker.com/engine/reference/builder/#environment-replacement) in many as well.

The ENV instruction has two forms. The first form, ENV <key> <value>, will set a single variable to a value. The entire string after the first space will be treated as the <value> - including whitespace characters. The value will be interpreted for other environment variables, so quote characters will be removed if they are not escaped.

The second form, ENV <key>=<value> ..., allows for multiple variables to be set at one time. Notice that the second form uses the equals sign (=) in the syntax, while the first form does not. Like command line parsing, quotes and backslashes can be used to include spaces within values.

For example:

ENV myName="John Doe" myDog=Rex\ The\ Dog \

myCat=fluffy

and

ENV myName John Doe

ENV myDog Rex The Dog

ENV myCat fluffy

will yield the same net results in the final image.

The environment variables set using ENV will persist when a container is run from the resulting image. You can view the values using docker inspect, and change them using docker run --env <key>=<value>.

**Note**: Environment persistence can cause unexpected side effects. For example, setting ENV DEBIAN\_FRONTEND noninteractive may confuse apt-get users on a Debian-based image. To set a value for a single command, use RUN <key>=<value> <command>.

## ADD

ADD has two forms:

* ADD [--chown=<user>:<group>] <src>... <dest>
* ADD [--chown=<user>:<group>] ["<src>",... "<dest>"] (this form is required for paths containing whitespace)

**Note**: The --chown feature is only supported on Dockerfiles used to build Linux containers, and will not work on Windows containers. Since user and group ownership concepts do not translate between Linux and Windows, the use of /etc/passwd and /etc/group for translating user and group names to IDs restricts this feature to only be viable for Linux OS-based containers.

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

Multiple <src> resources may be specified but if they are files or directories, their paths are interpreted as relative to the source of the context of the build.

Each <src> may contain wildcards and matching will be done using Go’s [filepath.Match](http://golang.org/pkg/path/filepath#Match) rules. For example:

ADD hom\* /mydir/ # adds all files starting with "hom"

ADD hom?.txt /mydir/ # ? is replaced with any single character, e.g., "home.txt"

The <dest> is an absolute path, or a path relative to WORKDIR, into which the source will be copied inside the destination container.

ADD test relativeDir/ # adds "test" to `WORKDIR`/relativeDir/

ADD test /absoluteDir/ # adds "test" to /absoluteDir/

When adding files or directories that contain special characters (such as [ and ]), you need to escape those paths following the Golang rules to prevent them from being treated as a matching pattern. For example, to add a file named arr[0].txt, use the following;

ADD arr[[]0].txt /mydir/ # copy a file named "arr[0].txt" to /mydir/

All new files and directories are created with a UID and GID of 0, unless the optional --chown flag specifies a given username, groupname, or UID/GID combination to request specific ownership of the content added. The format of the --chown flag allows for either username and groupname strings or direct integer UID and GID in any combination. Providing a username without groupname or a UID without GID will use the same numeric UID as the GID. If a username or groupname is provided, the container’s root filesystem /etc/passwd and /etc/group files will be used to perform the translation from name to integer UID or GID respectively. The following examples show valid definitions for the --chown flag:

ADD --chown=55:mygroup files\* /somedir/

ADD --chown=bin files\* /somedir/

ADD --chown=1 files\* /somedir/

ADD --chown=10:11 files\* /somedir/

If the container root filesystem does not contain either /etc/passwd or /etc/group files and either user or group names are used in the --chown flag, the build will fail on the ADD operation. Using numeric IDs requires no lookup and will not depend on container root filesystem content.

In the case where <src> is a remote file URL, the destination will have permissions of 600. If the remote file being retrieved has an HTTP Last-Modified header, the timestamp from that header will be used to set the mtime on the destination file. However, like any other file processed during an ADD, mtime will not be included in the determination of whether or not the file has changed and the cache should be updated.

**Note**: If you build by passing a Dockerfile through STDIN (docker build - < somefile), there is no build context, so the Dockerfile can only contain a URL based ADD instruction. You can also pass a compressed archive through STDIN: (docker build - < archive.tar.gz), the Dockerfile at the root of the archive and the rest of the archive will be used as the context of the build.

**Note**: If your URL files are protected using authentication, you will need to use RUN wget, RUN curl or use another tool from within the container as the ADD instruction does not support authentication.

**Note**: The first encountered ADD instruction will invalidate the cache for all following instructions from the Dockerfile if the contents of <src> have changed. This includes invalidating the cache for RUN instructions. See the [Dockerfile Best Practices guide](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/) for more information.

ADD obeys the following rules:

* The <src> path must be inside the context of the build; you cannot ADD ../something /something, because the first step of a docker build is to send the context directory (and subdirectories) to the docker daemon.
* If <src> is a URL and <dest> does not end with a trailing slash, then a file is downloaded from the URL and copied to <dest>.
* If <src> is a URL and <dest> does end with a trailing slash, then the filename is inferred from the URL and the file is downloaded to <dest>/<filename>. For instance, ADD http://example.com/foobar / would create the file /foobar. The URL must have a nontrivial path so that an appropriate filename can be discovered in this case (http://example.com will not work).
* If <src> is a directory, the entire contents of the directory are copied, including filesystem metadata.

**Note**: The directory itself is not copied, just its contents.

* If <src> is a local tar archive in a recognized compression format (identity, gzip, bzip2 or xz) then it is unpacked as a directory. Resources from remote URLs are **not** decompressed. When a directory is copied or unpacked, it has the same behavior as tar -x, the result is the union of:
  1. Whatever existed at the destination path and
  2. The contents of the source tree, with conflicts resolved in favor of “2.” on a file-by-file basis.

**Note**: Whether a file is identified as a recognized compression format or not is done solely based on the contents of the file, not the name of the file. For example, if an empty file happens to end with .tar.gz this will not be recognized as a compressed file and **will not** generate any kind of decompression error message, rather the file will simply be copied to the destination.

* If <src> is any other kind of file, it is copied individually along with its metadata. In this case, if <dest> ends with a trailing slash /, it will be considered a directory and the contents of <src> will be written at <dest>/base(<src>).
* If multiple <src> resources are specified, either directly or due to the use of a wildcard, then <dest> must be a directory, and it must end with a slash /.
* If <dest> does not end with a trailing slash, it will be considered a regular file and the contents of <src> will be written at <dest>.
* If <dest> doesn’t exist, it is created along with all missing directories in its path.

## COPY

COPY has two forms:

* COPY [--chown=<user>:<group>] <src>... <dest>
* COPY [--chown=<user>:<group>] ["<src>",... "<dest>"] (this form is required for paths containing whitespace)

**Note**: The --chown feature is only supported on Dockerfiles used to build Linux containers, and will not work on Windows containers. Since user and group ownership concepts do not translate between Linux and Windows, the use of /etc/passwd and /etc/group for translating user and group names to IDs restricts this feature to only be viable for Linux OS-based containers.

The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.

Multiple <src> resources may be specified but the paths of files and directories will be interpreted as relative to the source of the context of the build.

Each <src> may contain wildcards and matching will be done using Go’s [filepath.Match](http://golang.org/pkg/path/filepath#Match) rules. For example:

COPY hom\* /mydir/ # adds all files starting with "hom"

COPY hom?.txt /mydir/ # ? is replaced with any single character, e.g., "home.txt"

The <dest> is an absolute path, or a path relative to WORKDIR, into which the source will be copied inside the destination container.

COPY test relativeDir/ # adds "test" to `WORKDIR`/relativeDir/

COPY test /absoluteDir/ # adds "test" to /absoluteDir/

When copying files or directories that contain special characters (such as [ and ]), you need to escape those paths following the Golang rules to prevent them from being treated as a matching pattern. For example, to copy a file named arr[0].txt, use the following;

COPY arr[[]0].txt /mydir/ # copy a file named "arr[0].txt" to /mydir/

All new files and directories are created with a UID and GID of 0, unless the optional --chown flag specifies a given username, groupname, or UID/GID combination to request specific ownership of the copied content. The format of the --chown flag allows for either username and groupname strings or direct integer UID and GID in any combination. Providing a username without groupname or a UID without GID will use the same numeric UID as the GID. If a username or groupname is provided, the container’s root filesystem /etc/passwd and /etc/group files will be used to perform the translation from name to integer UID or GID respectively. The following examples show valid definitions for the --chown flag:

COPY --chown=55:mygroup files\* /somedir/

COPY --chown=bin files\* /somedir/

COPY --chown=1 files\* /somedir/

COPY --chown=10:11 files\* /somedir/

If the container root filesystem does not contain either /etc/passwd or /etc/group files and either user or group names are used in the --chown flag, the build will fail on the COPY operation. Using numeric IDs requires no lookup and will not depend on container root filesystem content.

**Note**: If you build using STDIN (docker build - < somefile), there is no build context, so COPY can’t be used.

Optionally COPY accepts a flag --from=<name|index> that can be used to set the source location to a previous build stage (created with FROM .. AS <name>) that will be used instead of a build context sent by the user. The flag also accepts a numeric index assigned for all previous build stages started with FROM instruction. In case a build stage with a specified name can’t be found an image with the same name is attempted to be used instead.

COPY obeys the following rules:

* The <src> path must be inside the context of the build; you cannot COPY ../something /something, because the first step of a docker build is to send the context directory (and subdirectories) to the docker daemon.
* If <src> is a directory, the entire contents of the directory are copied, including filesystem metadata.

**Note**: The directory itself is not copied, just its contents.

* If <src> is any other kind of file, it is copied individually along with its metadata. In this case, if <dest> ends with a trailing slash /, it will be considered a directory and the contents of <src> will be written at <dest>/base(<src>).
* If multiple <src> resources are specified, either directly or due to the use of a wildcard, then <dest> must be a directory, and it must end with a slash /.
* If <dest> does not end with a trailing slash, it will be considered a regular file and the contents of <src> will be written at <dest>.
* If <dest> doesn’t exist, it is created along with all missing directories in its path.

## ENTRYPOINT

ENTRYPOINT has two forms:

* ENTRYPOINT ["executable", "param1", "param2"] (exec form, preferred)
* ENTRYPOINT command param1 param2 (shell form)

An ENTRYPOINT allows you to configure a container that will run as an executable.

For example, the following will start nginx with its default content, listening on port 80:

docker run -i -t --rm -p 80:80 nginx

Command line arguments to docker run <image> will be appended after all elements in an exec form ENTRYPOINT, and will override all elements specified using CMD. This allows arguments to be passed to the entry point, i.e., docker run <image> -d will pass the -d argument to the entry point. You can override the ENTRYPOINT instruction using the docker run --entrypoint flag.

The shell form prevents any CMD or run command line arguments from being used, but has the disadvantage that your ENTRYPOINT will be started as a subcommand of /bin/sh -c, which does not pass signals. This means that the executable will not be the container’s PID 1 - and will not receive Unix signals - so your executable will not receive a SIGTERM from docker stop <container>.

Only the last ENTRYPOINT instruction in the Dockerfile will have an effect.

### Exec form ENTRYPOINT example

You can use the exec form of ENTRYPOINT to set fairly stable default commands and arguments and then use either form of CMD to set additional defaults that are more likely to be changed.

FROM ubuntu

ENTRYPOINT ["top", "-b"]

CMD ["-c"]

When you run the container, you can see that top is the only process:

$ docker run -it --rm --name test top -H

top - 08:25:00 up 7:27, 0 users, load average: 0.00, 0.01, 0.05

Threads: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie

%Cpu(s): 0.1 us, 0.1 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem: 2056668 total, 1616832 used, 439836 free, 99352 buffers

KiB Swap: 1441840 total, 0 used, 1441840 free. 1324440 cached Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

1 root 20 0 19744 2336 2080 R 0.0 0.1 0:00.04 top

To examine the result further, you can use docker exec:

$ docker exec -it test ps aux

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 2.6 0.1 19752 2352 ? Ss+ 08:24 0:00 top -b -H

root 7 0.0 0.1 15572 2164 ? R+ 08:25 0:00 ps aux

And you can gracefully request top to shut down using docker stop test.

The following Dockerfile shows using the ENTRYPOINT to run Apache in the foreground (i.e., as PID 1):

FROM debian:stable

RUN apt-get update && apt-get install -y --force-yes apache2

EXPOSE 80 443

VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"]

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]

If you need to write a starter script for a single executable, you can ensure that the final executable receives the Unix signals by using exec and gosu commands:

#!/usr/bin/env bash

set -e

if [ "$1" = 'postgres' ]; then

chown -R postgres "$PGDATA"

if [ -z "$(ls -A "$PGDATA")" ]; then

gosu postgres initdb

fi

exec gosu postgres "$@"

fi

exec "$@"

Lastly, if you need to do some extra cleanup (or communicate with other containers) on shutdown, or are co-ordinating more than one executable, you may need to ensure that the ENTRYPOINT script receives the Unix signals, passes them on, and then does some more work:

#!/bin/sh

# Note: I've written this using sh so it works in the busybox container too

# USE the trap if you need to also do manual cleanup after the service is stopped,

# or need to start multiple services in the one container

trap "echo TRAPed signal" HUP INT QUIT TERM

# start service in background here

/usr/sbin/apachectl start

echo "[hit enter key to exit] or run 'docker stop <container>'"

read

# stop service and clean up here

echo "stopping apache"

/usr/sbin/apachectl stop

echo "exited $0"

If you run this image with docker run -it --rm -p 80:80 --name test apache, you can then examine the container’s processes with docker exec, or docker top, and then ask the script to stop Apache:

$ docker exec -it test ps aux

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 0.1 0.0 4448 692 ? Ss+ 00:42 0:00 /bin/sh /run.sh 123 cmd cmd2

root 19 0.0 0.2 71304 4440 ? Ss 00:42 0:00 /usr/sbin/apache2 -k start

www-data 20 0.2 0.2 360468 6004 ? Sl 00:42 0:00 /usr/sbin/apache2 -k start

www-data 21 0.2 0.2 360468 6000 ? Sl 00:42 0:00 /usr/sbin/apache2 -k start

root 81 0.0 0.1 15572 2140 ? R+ 00:44 0:00 ps aux

$ docker top test

PID USER COMMAND

10035 root {run.sh} /bin/sh /run.sh 123 cmd cmd2

10054 root /usr/sbin/apache2 -k start

10055 33 /usr/sbin/apache2 -k start

10056 33 /usr/sbin/apache2 -k start

$ /usr/bin/time docker stop test

test

real 0m 0.27s

user 0m 0.03s

sys 0m 0.03s

**Note:** you can override the ENTRYPOINT setting using --entrypoint, but this can only set the binary to exec (no sh -c will be used).

**Note**: The exec form is parsed as a JSON array, which means that you must use double-quotes (“) around words not single-quotes (‘).

**Note**: Unlike the shell form, the exec form does not invoke a command shell. This means that normal shell processing does not happen. For example, ENTRYPOINT [ "echo", "$HOME" ] will not do variable substitution on $HOME. If you want shell processing then either use the shell form or execute a shell directly, for example: ENTRYPOINT [ "sh", "-c", "echo $HOME" ]. When using the exec form and executing a shell directly, as in the case for the shell form, it is the shell that is doing the environment variable expansion, not docker.

### Shell form ENTRYPOINT example

You can specify a plain string for the ENTRYPOINT and it will execute in /bin/sh -c. This form will use shell processing to substitute shell environment variables, and will ignore any CMD or docker run command line arguments. To ensure that docker stop will signal any long running ENTRYPOINT executable correctly, you need to remember to start it with exec:

FROM ubuntu

ENTRYPOINT exec top -b

When you run this image, you’ll see the single PID 1 process:

$ docker run -it --rm --name test top

Mem: 1704520K used, 352148K free, 0K shrd, 0K buff, 140368121167873K cached

CPU: 5% usr 0% sys 0% nic 94% idle 0% io 0% irq 0% sirq

Load average: 0.08 0.03 0.05 2/98 6

PID PPID USER STAT VSZ %VSZ %CPU COMMAND

1 0 root R 3164 0% 0% top -b

Which will exit cleanly on docker stop:

$ /usr/bin/time docker stop test

test

real 0m 0.20s

user 0m 0.02s

sys 0m 0.04s

If you forget to add exec to the beginning of your ENTRYPOINT:

FROM ubuntu

ENTRYPOINT top -b

CMD --ignored-param1

You can then run it (giving it a name for the next step):

$ docker run -it --name test top --ignored-param2

Mem: 1704184K used, 352484K free, 0K shrd, 0K buff, 140621524238337K cached

CPU: 9% usr 2% sys 0% nic 88% idle 0% io 0% irq 0% sirq

Load average: 0.01 0.02 0.05 2/101 7

PID PPID USER STAT VSZ %VSZ %CPU COMMAND

1 0 root S 3168 0% 0% /bin/sh -c top -b cmd cmd2

7 1 root R 3164 0% 0% top -b

You can see from the output of top that the specified ENTRYPOINT is not PID 1.

If you then run docker stop test, the container will not exit cleanly - the stop command will be forced to send a SIGKILL after the timeout:

$ docker exec -it test ps aux

PID USER COMMAND

1 root /bin/sh -c top -b cmd cmd2

7 root top -b

8 root ps aux

$ /usr/bin/time docker stop test

test

real 0m 10.19s

user 0m 0.04s

sys 0m 0.03s

### Understand how CMD and ENTRYPOINT interact

Both CMD and ENTRYPOINT instructions define what command gets executed when running a container. There are few rules that describe their co-operation.

1. Dockerfile should specify at least one of CMD or ENTRYPOINT commands.
2. ENTRYPOINT should be defined when using the container as an executable.
3. CMD should be used as a way of defining default arguments for an ENTRYPOINT command or for executing an ad-hoc command in a container.
4. CMD will be overridden when running the container with alternative arguments.

The table below shows what command is executed for different ENTRYPOINT / CMD combinations:

|  | **No ENTRYPOINT** | **ENTRYPOINT exec\_entry p1\_entry** | **ENTRYPOINT [“exec\_entry”, “p1\_entry”]** |
| --- | --- | --- | --- |
| **No CMD** | error, not allowed | /bin/sh -c exec\_entry p1\_entry | exec\_entry p1\_entry |
| **CMD [“exec\_cmd”, “p1\_cmd”]** | exec\_cmd p1\_cmd | /bin/sh -c exec\_entry p1\_entry | exec\_entry p1\_entry exec\_cmd p1\_cmd |
| **CMD [“p1\_cmd”, “p2\_cmd”]** | p1\_cmd p2\_cmd | /bin/sh -c exec\_entry p1\_entry | exec\_entry p1\_entry p1\_cmd p2\_cmd |
| **CMD exec\_cmd p1\_cmd** | /bin/sh -c exec\_cmd p1\_cmd | /bin/sh -c exec\_entry p1\_entry | exec\_entry p1\_entry /bin/sh -c exec\_cmd p1\_cmd |

**Note**: If CMD is defined from the base image, setting ENTRYPOINT will reset CMD to an empty value. In this scenario, CMD must be defined in the current image to have a value.

## VOLUME

VOLUME ["/data"]

The VOLUME instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers. The value can be a JSON array, VOLUME ["/var/log/"], or a plain string with multiple arguments, such as VOLUME /var/log or VOLUME /var/log /var/db. For more information/examples and mounting instructions via the Docker client, refer to [Share Directories via Volumes](https://docs.docker.com/engine/tutorials/dockervolumes/) documentation.

The docker run command initializes the newly created volume with any data that exists at the specified location within the base image. For example, consider the following Dockerfile snippet:

FROM ubuntu

RUN mkdir /myvol

RUN echo "hello world" > /myvol/greeting

VOLUME /myvol

This Dockerfile results in an image that causes docker run to create a new mount point at /myvol and copy the greeting file into the newly created volume.

### Notes about specifying volumes

Keep the following things in mind about volumes in the Dockerfile.

* **Volumes on Windows-based containers**: When using Windows-based containers, the destination of a volume inside the container must be one of:
  + a non-existing or empty directory
  + a drive other than C:
* **Changing the volume from within the Dockerfile**: If any build steps change the data within the volume after it has been declared, those changes will be discarded.
* **JSON formatting**: The list is parsed as a JSON array. You must enclose words with double quotes (") rather than single quotes (').
* **The host directory is declared at container run-time**: The host directory (the mountpoint) is, by its nature, host-dependent. This is to preserve image portability, since a given host directory can’t be guaranteed to be available on all hosts. For this reason, you can’t mount a host directory from within the Dockerfile. The VOLUME instruction does not support specifying a host-dir parameter. You must specify the mountpoint when you create or run the container.

## USER

USER <user>[:<group>] or

USER <UID>[:<GID>]

The USER instruction 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.

**Warning**: When the user doesn’t have a primary group then the image (or the next instructions) will be run with the root group.

On Windows, the user must be created first if it’s not a built-in account. This can be done with the net user command called as part of a Dockerfile.

FROM microsoft/windowsservercore

# Create Windows user in the container

RUN net user /add patrick

# Set it for subsequent commands

USER patrick

## WORKDIR

WORKDIR /path/to/workdir

The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile. If the WORKDIR doesn’t exist, it will be created even if it’s not used in any subsequent Dockerfile instruction.

The WORKDIR instruction can be used multiple times in a Dockerfile. If a relative path is provided, it will be relative to the path of the previous WORKDIR instruction. For example:

WORKDIR /a

WORKDIR b

WORKDIR c

RUN pwd

The output of the final pwd command in this Dockerfile would be /a/b/c.

The WORKDIR instruction can resolve environment variables previously set using ENV. You can only use environment variables explicitly set in the Dockerfile. For example:

ENV DIRPATH /path

WORKDIR $DIRPATH/$DIRNAME

RUN pwd

The output of the final pwd command in this Dockerfile would be /path/$DIRNAME

## ARG

ARG <name>[=<default value>]

The ARG instruction 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. If a user specifies a build argument that was not defined in the Dockerfile, the build outputs a warning.

[Warning] One or more build-args [foo] were not consumed.

A Dockerfile may include one or more ARG instructions. For example, the following is a valid Dockerfile:

FROM busybox

ARG user1

ARG buildno

...

**Warning:** It is not recommended to use build-time variables for passing secrets like github keys, user credentials etc. Build-time variable values are visible to any user of the image with the docker history command.

### Default values

An ARG instruction can optionally include a default value:

FROM busybox

ARG user1=someuser

ARG buildno=1

...

If an ARG instruction has a default value and if there is no value passed at build-time, the builder uses the default.

### Scope

An ARG variable definition comes into effect from the line on which it is defined in the Dockerfile not from the argument’s use on the command-line or elsewhere. For example, consider this Dockerfile:

1 FROM busybox

2 USER ${user:-some\_user}

3 ARG user

4 USER $user

...

A user builds this file by calling:

$ docker build --build-arg user=what\_user .

The USER at line 2 evaluates to some\_user as the user variable is defined on the subsequent line 3. The USER at line 4 evaluates to what\_user as user is defined and the what\_user value was passed on the command line. Prior to its definition by an ARG instruction, any use of a variable results in an empty string.

An ARG instruction goes out of scope at the end of the build stage where it was defined. To use an arg in multiple stages, each stage must include the ARG instruction.

FROM busybox

ARG SETTINGS

RUN ./run/setup $SETTINGS

FROM busybox

ARG SETTINGS

RUN ./run/other $SETTINGS

### Using ARG variables

You can use an ARG or an ENV instruction to specify variables that are available to the RUN instruction. Environment variables defined using the ENV instruction always override an ARG instruction of the same name. Consider this Dockerfile with an ENV and ARG instruction.

1 FROM ubuntu

2 ARG CONT\_IMG\_VER

3 ENV CONT\_IMG\_VER v1.0.0

4 RUN echo $CONT\_IMG\_VER

Then, assume this image is built with this command:

$ docker build --build-arg CONT\_IMG\_VER=v2.0.1 .

In this case, the RUN instruction uses v1.0.0 instead of the ARG setting passed by the user:v2.0.1 This behavior is similar to a shell script where a locally scoped variable overrides the variables passed as arguments or inherited from environment, from its point of definition.

Using the example above but a different ENV specification you can create more useful interactions between ARG and ENV instructions:

1 FROM ubuntu

2 ARG CONT\_IMG\_VER

3 ENV CONT\_IMG\_VER ${CONT\_IMG\_VER:-v1.0.0}

4 RUN echo $CONT\_IMG\_VER

Unlike an ARG instruction, ENV values are always persisted in the built image. Consider a docker build without the --build-arg flag:

$ docker build .

Using this Dockerfile example, CONT\_IMG\_VER is still persisted in the image but its value would be v1.0.0 as it is the default set in line 3 by the ENV instruction.

The variable expansion technique in this example allows you to pass arguments from the command line and persist them in the final image by leveraging the ENV instruction. Variable expansion is only supported for [a limited set of Dockerfile instructions.](https://docs.docker.com/engine/reference/builder/#environment-replacement)

### Predefined ARGs

Docker has a set of predefined ARG variables that you can use without a corresponding ARG instruction in the Dockerfile.

* HTTP\_PROXY
* http\_proxy
* HTTPS\_PROXY
* https\_proxy
* FTP\_PROXY
* ftp\_proxy
* NO\_PROXY
* no\_proxy

To use these, simply pass them on the command line using the flag:

--build-arg <varname>=<value>

By default, these pre-defined variables are excluded from the output of docker history. Excluding them reduces the risk of accidentally leaking sensitive authentication information in an HTTP\_PROXY variable.

For example, consider building the following Dockerfile using --build-arg HTTP\_PROXY=http://user:pass@proxy.lon.example.com

FROM ubuntu

RUN echo "Hello World"

In this case, the value of the HTTP\_PROXY variable is not available in the docker history and is not cached. If you were to change location, and your proxy server changed to http://user:pass@proxy.sfo.example.com, a subsequent build does not result in a cache miss.

If you need to override this behaviour then you may do so by adding an ARG statement in the Dockerfile as follows:

FROM ubuntu

ARG HTTP\_PROXY

RUN echo "Hello World"

When building this Dockerfile, the HTTP\_PROXY is preserved in the docker history, and changing its value invalidates the build cache.

### Automatic platform ARGs in the global scope

This feature is only available when using the [BuildKit](https://docs.docker.com/engine/reference/builder/#buildkit) backend.

Docker predefines a set of ARG variables with information on the platform of the node performing the build (build platform) and on the platform of the resulting image (target platform). The target platform can be specified with the --platform flag on docker build.

The following ARG variables are set automatically:

* TARGETPLATFORM - platform of the build result. Eg linux/amd64, linux/arm/v7, windows/amd64.
* TARGETOS - OS component of TARGETPLATFORM
* TARGETARCH - architecture component of TARGETPLATFORM
* TARGETVARIANT - variant component of TARGETPLATFORM
* BUILDPLATFORM - platform of the node performing the build.
* BUILDOS - OS component of BUILDPLATFORM
* BUILDARCH - architecture component of BUILDPLATFORM
* BUILDVARIANT - variant component of BUILDPLATFORM

These arguments are defined in the global scope so are not automatically available inside build stages or for your RUN commands. To expose one of these arguments inside the build stage redefine it without value.

For example:

FROM alpine

ARG TARGETPLATFORM

RUN echo "I'm building for $TARGETPLATFORM"

### Impact on build caching

ARG variables are not persisted into the built image as ENV variables are. However, ARG variables do impact the build cache in similar ways. If a Dockerfile defines an ARG variable whose value is different from a previous build, then a “cache miss” occurs upon its first usage, not its definition. In particular, all RUN instructions following an ARG instruction use the ARG variable implicitly (as an environment variable), thus can cause a cache miss. All predefined ARG variables are exempt from caching unless there is a matching ARG statement in the Dockerfile.

For example, consider these two Dockerfile:

1 FROM ubuntu

2 ARG CONT\_IMG\_VER

3 RUN echo $CONT\_IMG\_VER

1 FROM ubuntu

2 ARG CONT\_IMG\_VER

3 RUN echo hello

If you specify --build-arg CONT\_IMG\_VER=<value> on the command line, in both cases, the specification on line 2 does not cause a cache miss; line 3 does cause a cache miss.ARG CONT\_IMG\_VER causes the RUN line to be identified as the same as running CONT\_IMG\_VER=<value> echo hello, so if the <value> changes, we get a cache miss.

Consider another example under the same command line:

1 FROM ubuntu

2 ARG CONT\_IMG\_VER

3 ENV CONT\_IMG\_VER $CONT\_IMG\_VER

4 RUN echo $CONT\_IMG\_VER

In this example, the cache miss occurs on line 3. The miss happens because the variable’s value in the ENV references the ARG variable and that variable is changed through the command line. In this example, the ENV command causes the image to include the value.

If an ENV instruction overrides an ARG instruction of the same name, like this Dockerfile:

1 FROM ubuntu

2 ARG CONT\_IMG\_VER

3 ENV CONT\_IMG\_VER hello

4 RUN echo $CONT\_IMG\_VER

Line 3 does not cause a cache miss because the value of CONT\_IMG\_VER is a constant (hello). As a result, the environment variables and values used on the RUN (line 4) doesn’t change between builds.

## ONBUILD

ONBUILD [INSTRUCTION]

The ONBUILD instruction adds to the image a trigger instruction to be executed at a later time, when the image is used as the base for another build. The trigger will be executed in the context of the downstream build, as if it had been inserted immediately after the FROM instruction in the downstream Dockerfile.

Any build instruction can be registered as a trigger.

This is useful if you are building an image which will be used as a base to build other images, for example an application build environment or a daemon which may be customized with user-specific configuration.

For example, if your image is a reusable Python application builder, it will require application source code to be added in a particular directory, and it might require a build script to be called after that. You can’t just call ADD and RUN now, because you don’t yet have access to the application source code, and it will be different for each application build. You could simply provide application developers with a boilerplate Dockerfile to copy-paste into their application, but that is inefficient, error-prone and difficult to update because it mixes with application-specific code.

The solution is to use ONBUILD to register advance instructions to run later, during the next build stage.

Here’s how it works:

1. When it encounters an ONBUILD instruction, the builder adds a trigger to the metadata of the image being built. The instruction does not otherwise affect the current build.
2. At the end of the build, a list of all triggers is stored in the image manifest, under the key OnBuild. They can be inspected with the docker inspect command.
3. Later the image may be used as a base for a new build, using the FROM instruction. As part of processing the FROM instruction, the downstream builder looks for ONBUILD triggers, and executes them in the same order they were registered. If any of the triggers fail, the FROM instruction is aborted which in turn causes the build to fail. If all triggers succeed, the FROM instruction completes and the build continues as usual.
4. Triggers are cleared from the final image after being executed. In other words they are not inherited by “grand-children” builds.

For example you might add something like this:

[...]

ONBUILD ADD . /app/src

ONBUILD RUN /usr/local/bin/python-build --dir /app/src

[...]

**Warning**: Chaining ONBUILD instructions using ONBUILD ONBUILD isn’t allowed.

**Warning**: The ONBUILD instruction may not trigger FROM or MAINTAINER instructions.

## STOPSIGNAL

STOPSIGNAL signal

The STOPSIGNAL instruction 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.

## HEALTHCHECK

The HEALTHCHECK instruction has two forms:

* HEALTHCHECK [OPTIONS] CMD command (check container health by running a command inside the container)
* HEALTHCHECK NONE (disable any healthcheck inherited from the base image)

The HEALTHCHECK instruction tells Docker how to test a container to check that it is still working. This can detect cases such as a web server that is stuck in an infinite loop and unable to handle new connections, even though the server process is still running.

When a container has a healthcheck specified, it has a health status in addition to its normal status. This status is initially starting. Whenever a health check passes, it becomes healthy (whatever state it was previously in). After a certain number of consecutive failures, it becomes unhealthy.

The options that can appear before CMD are:

* --interval=DURATION (default: 30s)
* --timeout=DURATION (default: 30s)
* --start-period=DURATION (default: 0s)
* --retries=N (default: 3)

The health check will first run **interval** seconds after the container is started, and then again **interval** seconds after each previous check completes.

If a single run of the check takes longer than **timeout** seconds then the check is considered to have failed.

It takes **retries** consecutive failures of the health check for the container to be considered unhealthy.

**start period** provides initialization time for containers that need time to bootstrap. Probe failure during that period will not be counted towards the maximum number of retries. However, if a health check succeeds during the start period, the container is considered started and all consecutive failures will be counted towards the maximum number of retries.

There can only be one HEALTHCHECK instruction in a Dockerfile. If you list more than one then only the last HEALTHCHECK will take effect.

The command after the CMD keyword can be either a shell command (e.g. HEALTHCHECK CMD /bin/check-running) or an exec array (as with other Dockerfile commands; see e.g. ENTRYPOINT for details).

The command’s exit status indicates the health status of the container. The possible values are:

* 0: success - the container is healthy and ready for use
* 1: unhealthy - the container is not working correctly
* 2: reserved - do not use this exit code

For example, to check every five minutes or so that a web-server is able to serve the site’s main page within three seconds:

HEALTHCHECK --interval=5m --timeout=3s \

CMD curl -f http://localhost/ || exit 1

To help debug failing probes, any output text (UTF-8 encoded) that the command writes on stdout or stderr will be stored in the health status and can be queried with docker inspect. Such output should be kept short (only the first 4096 bytes are stored currently).

When the health status of a container changes, a health\_status event is generated with the new status.

The HEALTHCHECK feature was added in Docker 1.12.

## SHELL

SHELL ["executable", "parameters"]

The SHELL instruction allows the default shell used for the shell form of commands to be overridden. The default shell on Linux is ["/bin/sh", "-c"], and on Windows is ["cmd", "/S", "/C"]. The SHELL instruction must be written in JSON form in a Dockerfile.

The SHELL instruction is particularly useful on Windows where there are two commonly used and quite different native shells: cmd and powershell, as well as alternate shells available including sh.

The SHELL instruction can appear multiple times. Each SHELL instruction overrides all previous SHELL instructions, and affects all subsequent instructions. For example:

FROM microsoft/windowsservercore

# Executed as cmd /S /C echo default

RUN echo default

# Executed as cmd /S /C powershell -command Write-Host default

RUN powershell -command Write-Host default

# Executed as powershell -command Write-Host hello

SHELL ["powershell", "-command"]

RUN Write-Host hello

# Executed as cmd /S /C echo hello

SHELL ["cmd", "/S", "/C"]

RUN echo hello

The following instructions can be affected by the SHELL instruction when the shell form of them is used in a Dockerfile: RUN, CMD and ENTRYPOINT.

The following example is a common pattern found on Windows which can be streamlined by using the SHELL instruction:

...

RUN powershell -command Execute-MyCmdlet -param1 "c:\foo.txt"

...

The command invoked by docker will be:

cmd /S /C powershell -command Execute-MyCmdlet -param1 "c:\foo.txt"

This is inefficient for two reasons. First, there is an un-necessary cmd.exe command processor (aka shell) being invoked. Second, each RUN instruction in the shell form requires an extra powershell -command prefixing the command.

To make this more efficient, one of two mechanisms can be employed. One is to use the JSON form of the RUN command such as:

...

RUN ["powershell", "-command", "Execute-MyCmdlet", "-param1 \"c:\\foo.txt\""]

...

While the JSON form is unambiguous and does not use the un-necessary cmd.exe, it does require more verbosity through double-quoting and escaping. The alternate mechanism is to use the SHELL instruction and the shell form, making a more natural syntax for Windows users, especially when combined with the escape parser directive:

# escape=`

FROM microsoft/nanoserver

SHELL ["powershell","-command"]

RUN New-Item -ItemType Directory C:\Example

ADD Execute-MyCmdlet.ps1 c:\example\

RUN c:\example\Execute-MyCmdlet -sample 'hello world'

Resulting in:

PS E:\docker\build\shell> docker build -t shell .

Sending build context to Docker daemon 4.096 kB

Step 1/5 : FROM microsoft/nanoserver

---> 22738ff49c6d

Step 2/5 : SHELL powershell -command

---> Running in 6fcdb6855ae2

---> 6331462d4300

Removing intermediate container 6fcdb6855ae2

Step 3/5 : RUN New-Item -ItemType Directory C:\Example

---> Running in d0eef8386e97

Directory: C:\

Mode LastWriteTime Length Name

---- ------------- ------ ----

d----- 10/28/2016 11:26 AM Example

---> 3f2fbf1395d9

Removing intermediate container d0eef8386e97

Step 4/5 : ADD Execute-MyCmdlet.ps1 c:\example\

---> a955b2621c31

Removing intermediate container b825593d39fc

Step 5/5 : RUN c:\example\Execute-MyCmdlet 'hello world'

---> Running in be6d8e63fe75

hello world

---> 8e559e9bf424

Removing intermediate container be6d8e63fe75

Successfully built 8e559e9bf424

PS E:\docker\build\shell>

The SHELL instruction could also be used to modify the way in which a shell operates. For example, using SHELL cmd /S /C /V:ON|OFF on Windows, delayed environment variable expansion semantics could be modified.

The SHELL instruction can also be used on Linux should an alternate shell be required such as zsh, csh, tcsh and others.

The SHELL feature was added in Docker 1.12.

## External implementation features

This feature is only available when using the [BuildKit](https://docs.docker.com/engine/reference/builder/#buildkit) backend.

Docker build supports experimental features like cache mounts, build secrets and ssh forwarding that are enabled by using an external implementation of the builder with a syntax directive. To learn about these features, [refer to the documentation in BuildKit repository](https://github.com/moby/buildkit/blob/master/frontend/dockerfile/docs/experimental.md).

## Dockerfile examples

Below you can see some examples of Dockerfile syntax.

# Nginx

#

# VERSION 0.0.1

FROM ubuntu

LABEL Description="This image is used to start the foobar executable" Vendor="ACME Products" Version="1.0"

RUN apt-get update && apt-get install -y inotify-tools nginx apache2 openssh-server

# Firefox over VNC

#

# VERSION 0.3

FROM ubuntu

# Install vnc, xvfb in order to create a 'fake' display and firefox

RUN apt-get update && apt-get install -y x11vnc xvfb firefox

RUN mkdir ~/.vnc

# Setup a password

RUN x11vnc -storepasswd 1234 ~/.vnc/passwd

# Autostart firefox (might not be the best way, but it does the trick)

RUN bash -c 'echo "firefox" >> /.bashrc'

EXPOSE 5900

CMD ["x11vnc", "-forever", "-usepw", "-create"]

# Multiple images example

#

# VERSION 0.1

FROM ubuntu

RUN echo foo > bar

# Will output something like ===> 907ad6c2736f

FROM ubuntu

RUN echo moo > oink

# Will output something like ===> 695d7793cbe4

# You'll now have two images, 907ad6c2736f with /bar, and 695d7793cbe4 with

# /oink.