

**NAME**

docker-build - Build an image from a Dockerfile

**SYNOPSIS**

```
docker build [--add-host[=]] [--build-arg[=]] [--cache-from[=]] [--cpu-shares[=0]] [--cgroup-parent[=CGROUP-PARENT]] [--help] [--iidfile[=CIDFILE]] [-f|--file[=PATH/Dockerfile]] [--squash] Experimental [--force-rm] [--isolation[=default]] [--label[=]] [--no-cache] [--pull] [--compress] [-q|--quiet] [--rm[=true]] [-t|--tag[=]] [-m|--memory[=MEMORY]] [--memory-swap[=LIMIT]] [--network[="default"]] [--shm-size[=SHM-SIZE]] [--cpu-period[=0]] [--cpu-quota[=0]] [--cpuset-cpus[=CPUSET-CPUS]] [--cpuset-mems[=CPUSET-MEMS]] [--target[=]] [--ulimit[=]] PATH | URL | -
```

**DESCRIPTION**

This will read the Dockerfile from the directory specified in **PATH**. It also sends any other files and directories found in the current directory to the Docker daemon. The contents of this directory would be used by **ADD** commands found within the Dockerfile.

Warning, this will send a lot of data to the Docker daemon depending on the contents of the current directory. The build is run by the Docker daemon, not by the CLI, so the whole context must be transferred to the daemon. The Docker CLI reports "Sending build context to Docker daemon" when the context is sent to the daemon.

When the URL to a tarball archive or to a single Dockerfile is given, no context is sent from the client to the Docker daemon. In this case, the Dockerfile at the root of the archive and the rest of the archive will get used as the context of the build. When a Git repository is set as the **URL**, the repository is cloned locally and then sent as the context.

**OPTIONS**

**-f, --file** *PATH/Dockerfile*

Path to the Dockerfile to use. If the path is a relative path and you are building from a local directory, then the path must be relative to that directory. If you are building from a remote URL pointing to either a tarball or a Git repository, then the path must be relative to the root of the remote context. In all cases, the file must be within the build context. The default is *Dockerfile*.

**--squash** *true|false*

**Experimental Only**

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.

**Note:** using this option means the new image will not be able to take advantage of layer sharing with other images and may use significantly more space.

**Note:** 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.

**--add-host []**

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

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

**--build-arg *variable***

name and value of a **buildarg**.

For example, if you want to pass a value for `http_proxy`, use

```
--build-arg=http_proxy="http://some.proxy.url"
```

Users pass these values at build-time. Docker uses the `buildargs` as the environment context for command(s) run via the Dockerfile's `RUN` instruction or for variable expansion in other Dockerfile instructions. This is not meant for passing secret values. Read more about the `buildargs` instruction (<https://docs.docker.com/engine/reference/builder/#arg>)

**--cache-from ""**

Set image that will be used as a build cache source.

**--force-rm *true|false***

Always remove intermediate containers, even after unsuccessful builds. The default is *false*.

**--isolation "*default*"**

Isolation specifies the type of isolation technology used by containers.

**--label *label***

Set metadata for an image

**--no-cache *true|false***

Do not use cache when building the image. The default is *false*.

**--iidfile ""**

Write the image ID to the file

**--help**

Print usage statement

**--pull *true|false***

Always attempt to pull a newer version of the image. The default is *false*.

**--compress *true|false***

Compress the build context using `gzip`. The default is *false*.

**-q, --quiet *true|false***

Suppress the build output and print image ID on success. The default is *false*.

**--rm** *true|false*

Remove intermediate containers after a successful build. The default is *true*.

**-t, --tag** ""

Repository names (and optionally with tags) to be applied to the resulting image in case of success. Refer to **docker-tag(1)** for more information about valid tag names.

**-m, --memory** *MEMORY*

Memory limit

**--memory-swap** *number[S]*

Combined memory plus swap limit; *S* is an optional suffix which can be one of **b** (bytes), **k** (kilobytes), **m** (megabytes), or **g** (gigabytes).

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

**--network** *type*

Set the networking mode for the RUN instructions during build. Supported standard values are: **none**, **bridge**, **host** and **container:**<*name|id*>. Any other value is taken as a custom network's name or ID which this container should connect to.

In Linux, default is **bridge**.

**--shm-size** *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.

**--cpu-shares** *0*

CPU shares (relative weight).

By default, all containers get the same proportion of CPU cycles.

CPU shares is a 'relative weight', relative to the default setting of 1024.

This default value is defined here:

```
cat /sys/fs/cgroup/cpu/cpu.shares
1024
```

You can change this proportion by adjusting the container's CPU share weighting relative to the weighting of all other running containers.

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

Container	CPU share	Flag
{C0}	60% of CPU	--cpu-shares 614 (614 is 60% of 1024)
{C1}	40% of CPU	--cpu-shares 410 (410 is 40% of 1024)

The proportion is only applied when CPU-intensive processes are running.

When tasks in one container are idle, the other containers can use the left-over CPU time. The actual amount of CPU time used varies depending on the number of containers running on the system.

For example, consider three containers, where one has **--cpu-shares 1024** and two others have **--cpu-shares 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 **--cpu-shares 1024**, the first container only gets 33% of the CPU. The remaining containers receive 16.5%, 16.5% and 33% of the CPU.

Container	CPU share	Flag	CPU time
{C0}	100%	--cpu-shares 1024	33%
{C1}	50%	--cpu-shares 512	16.5%
{C2}	50%	--cpu-shares 512	16.5%
{C4}	100%	--cpu-shares 1024	33%

On a multi-core system, the shares of CPU time are distributed across the 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 **--cpu-shares 512** running one process, and another container **{C1}** with **--cpu-shares 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 0**

Limit the CPU CFS (Completely Fair Scheduler) period.

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

#### **--cpu-quota 0**

Limit the CPU CFS (Completely Fair Scheduler) quota.

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

**--cpuset-cpus** *CPUSET-CPUS*

CPUs in which to allow execution (0-3, 0,1).

**--cpuset-mems** *CPUSET-MEMS*

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

For example, if you have four memory nodes on your system (0-3), use `--cpuset-mems 0,1` to ensure the processes in your Docker container only use memory from the first two memory nodes.

**--cgroup-parent** *CGROUP-PARENT*

Path to cgroups under which the container's cgroup are created.

If the path is not absolute, the path is considered relative to the cgroups path of the init process. Cgroups are created if they do not already exist.

**--target** ""

Set the target build stage name.

**--ulimit** []

Ulimit options

For more information about `ulimit` see [Setting ulimits in a container](https://docs.docker.com/engine/reference/commandline/run/#set-ulimits-in-container---ulimit) (<https://docs.docker.com/engine/reference/commandline/run/#set-ulimits-in-container---ulimit>)

## EXAMPLES

### Building an image using a Dockerfile located inside the current directory

Docker images can be built using the build command and a Dockerfile:

```
docker build .
```

During the build process Docker creates intermediate images. In order to keep them, you must explicitly set `--rm false`.

```
docker build --rm false .
```

A good practice is to make a sub-directory with a related name and create the Dockerfile in that directory. For example, a directory called `mongo` may contain a Dockerfile to create a Docker MongoDB image. Likewise, another directory called `httpd` may be used to store Dockerfiles for Apache web server images.

It is also a good practice to add the files required for the image to the sub-directory. These files will then be specified with the `COPY` or `ADD` instructions in the `Dockerfile`.

Note: If you include a tar file (a good practice), then Docker will automatically extract the contents of the tar file specified within the `ADD` instruction into the specified target.

### Building an image and naming that image

A good practice is to give a name to the image you are building. Note that only a-z0-9-\_. should be used for consistency. There are no hard rules here but it is best to give the names consideration.

The `-t/--tag` flag is used to rename an image. Here are some examples:

Though it is not a good practice, image names can be arbitrary:

```
docker build -t myimage .
```

A better approach is to provide a fully qualified and meaningful repository, name, and tag (where the tag in this context means the qualifier after the ":"). In this example we build a JBoss image for the Fedora repository and give it the version 1.0:

```
docker build -t fedora/jboss:1.0 .
```

The next example is for the "whenry" user repository and uses Fedora and JBoss and gives it the version 2.1 :

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

If you do not provide a version tag then Docker will assign `latest`:

```
docker build -t whenry/fedora-jboss .
```

When you list the images, the image above will have the tag `latest`.

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 .
```

So renaming an image is arbitrary but consideration should be given to a useful convention that makes sense for consumers and should also take into account Docker community conventions.

### Building an image using a URL

This will clone the specified GitHub repository from the URL and use it as context. The Dockerfile at the root of the repository is used as Dockerfile. This only works if the GitHub repository is a dedicated repository.

```
docker build github.com/scollier/purpletest
```

Note: You can set an arbitrary Git repository via the `git://` scheme.

### Building an image using a URL to a tarball'ed context

This will send the URL itself to the Docker daemon. The daemon will fetch the tarball archive, decompress it and use its contents as the build context. The Dockerfile at the root of the archive and the rest of the archive will get used as the context of the build. If you pass an `-f PATH/Dockerfile` option as well, the system will look for that file inside the contents of the tarball.

```
docker build -f dev/Dockerfile https://10.10.10.1/docker/context.tar.gz
```

Note: supported compression formats are 'xz', 'bzip2', 'gzip' and 'identity' (no compression).

### 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:

- `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"`.

## HISTORY

March 2014, Originally compiled by William Henry (whentry at redhat dot com) based on docker.com source material and internal work. June 2014, updated by Sven Dowideit [SvenDowideit@home.org.au](mailto:SvenDowideit@home.org.au) June 2015, updated by Sally O'Malley [somalley@redhat.com](mailto:somalley@redhat.com) August 2020, Updated by Des Preston [despreston@gmail.com](mailto:despreston@gmail.com)