

dockerd

Estimated reading time: 54 minutes

daemon

Usage: dockerd COMMAND

A self-sufficient runtime for containers.

Options:

<code>--add-runtime runtime</code>	Register an additional runtime
<code>--allow-nondistributable-artifacts list</code>	Push nondistributable artifacts to a registry
<code>--api-cors-header string</code>	Set CORS headers in the daemon's API
<code>--authorization-plugin list</code>	Authorization plugin to use
<code>--bip string</code>	Specify network bridge
<code>-b, --bridge string</code>	Attach containers to network bridge
<code>--cgroup-parent string</code>	Set parent cgroup for containers
<code>--cluster-advertise string</code>	Address or interface to advertise
<code>--cluster-store string</code>	URL of the distributed state store
<code>--cluster-store-opt map</code>	Set cluster store options
<code>--config-file string</code>	Daemon configuration file
<code>--containerd string</code>	Path to containerd socket
<code>--cpu-rt-period int</code>	Limit the CPU real-time period
<code>--cpu-rt-runtime int</code>	Limit the CPU real-time runtime
<code>--data-root string</code>	Root directory of persistent data
<code>-D, --debug</code>	Enable debug mode
<code>--default-gateway ip</code>	Container default gateway
<code>--default-gateway-v6 ip</code>	Container default gateway v6
<code>--default-address-pool</code>	Set the default address pool
<code>--default-runtime string</code>	Default OCI runtime
<code>--default-ulimit ulimit</code>	Default ulimits for containers
<code>--dns list</code>	DNS server to use (comma-separated)
<code>--dns-opt list</code>	DNS options to use (comma-separated)
<code>--dns-search list</code>	DNS search domains to use (comma-separated)
<code>--exec-opt list</code>	Runtime execution options
<code>--exec-root string</code>	Root directory for exec
<code>--experimental</code>	Enable experimental features
<code>--fixed-cidr string</code>	IPv4 subnet for fixed IP
<code>--fixed-cidr-v6 string</code>	IPv6 subnet for fixed IP
<code>-G, --group string</code>	Group for the unix socket
<code>--help</code>	Print usage
<code>-H, --host list</code>	Daemon socket(s) to connect to
<code>--icc</code>	Enable inter-container communication
<code>--init</code>	Run an init in the container
<code>--init-path string</code>	Path to the docker-init binary
<code>--insecure-registry list</code>	Enable insecure registry
<code>--ip ip</code>	Default IP when binding
<code>--ip-forward</code>	Enable net.ipv4.ip_forward

<code>--ip-masq</code>	Enable IP masquerading
<code>--iptables</code>	Enable addition of iptables
<code>--ipv6</code>	Enable IPv6 networking
<code>--label list</code>	Set key=value labels
<code>--live-restore</code>	Enable live restore
<code>--log-driver string</code>	Default driver for logging
<code>-l, --log-level string</code>	Set the logging level
<code>--log-opt map</code>	Default log driver options
<code>--max-concurrent-downloads int</code>	Set the max concurrent downloads
<code>--max-concurrent-uploads int</code>	Set the max concurrent uploads
<code>--metrics-addr string</code>	Set default address for metrics
<code>--mtu int</code>	Set the containers network mtu
<code>--node-generic-resources list</code>	Advertise user-defined resources
<code>--no-new-privileges</code>	Set no-new-privileges
<code>--oom-score-adjust int</code>	Set the oom_score_adj
<code>-p, --pidfile string</code>	Path to use for daemon pidfile
<code>--raw-logs</code>	Full timestamps with raw logs
<code>--registry-mirror list</code>	Preferred Docker registry mirrors
<code>--seccomp-profile string</code>	Path to seccomp profile
<code>--selinux-enabled</code>	Enable selinux support
<code>--shutdown-timeout int</code>	Set the default shutdown timeout
<code>-s, --storage-driver string</code>	Storage driver to use
<code>--storage-opt list</code>	Storage driver options
<code>--swarm-default-advertise-addr string</code>	Set default address for swarm
<code>--tls</code>	Use TLS; implied by --tls* options
<code>--tlscacert string</code>	Trust certs signed by the CA
<code>--tlscert string</code>	Path to TLS certificate file
<code>--tlskey string</code>	Path to TLS key file
<code>--tlsverify</code>	Use TLS and verify the remote
<code>--userland-proxy</code>	Use userland proxy for networking
<code>--userland-proxy-path string</code>	Path to the userland proxy binary
<code>--userns-remap string</code>	User/Group setting for remapping
<code>-v, --version</code>	Print version information

Options with [] may be specified multiple times.

Description

`dockerd` is the persistent process that manages containers. Docker uses different binaries for the daemon and client. To run the daemon you type `dockerd` .

To run the daemon with debug output, use `dockerd -D` or add `"debug": true` to the `daemon.json` file.

Note: In Docker 1.13 and higher, enable experimental features by starting `dockerd` with the `--experimental` flag or adding `"experimental": true` to the `daemon.json` file. In earlier Docker versions, a different build was required to enable experimental features.

Examples

Daemon socket option

The Docker daemon can listen for Docker Engine API (<https://docs.docker.com/engine/reference/api/>) requests via three different types of Socket: `unix` , `tcp` , and `fd` .

By default, a `unix` domain socket (or IPC socket) is created at `/var/run/docker.sock` , requiring either `root` permission, or `docker` group membership.

If you need to access the Docker daemon remotely, you need to enable the `tcp` Socket. Beware that the default setup provides un-encrypted and un-authenticated direct access to the Docker daemon - and should be secured either using the built in HTTPS encrypted socket (<https://docs.docker.com/engine/security/https/>), or by putting a secure web proxy in front of it. You can listen on port `2375` on all network interfaces with `-H tcp://0.0.0.0:2375` , or on a particular network

interface using its IP address: `-H tcp://192.168.59.103:2375` . It is conventional to use port `2375` for un-encrypted, and port `2376` for encrypted communication with the daemon.

Note: If you're using an HTTPS encrypted socket, keep in mind that only TLS1.0 and greater are supported. Protocols SSLv3 and under are not supported anymore for security reasons.

On Systemd based systems, you can communicate with the daemon via Systemd socket activation (<http://0pointer.de/blog/projects/socket-activation.html>), use `dockerd -H fd://` . Using `fd://` will work perfectly for most setups but you can also specify individual sockets:

`dockerd -H fd://3` . If the specified socket activated files aren't found, then Docker will exit. You can find examples of using Systemd socket activation with Docker and Systemd in the Docker source tree (<https://github.com/docker/docker/tree/master/contrib/init/systemd/>).

You can configure the Docker daemon to listen to multiple sockets at the same time using multiple `-H` options:

```
# listen using the default unix socket, and on 2 specific IP addresses
$ sudo dockerd -H unix:///var/run/docker.sock -H tcp://192.168.59.103:2375
```



The Docker client will honor the `DOCKER_HOST` environment variable to set the `-H` flag for the client. Use **one** of the following commands:

```
$ docker -H tcp://0.0.0.0:2375 ps
```

```
$ export DOCKER_HOST="tcp://0.0.0.0:2375"
```

```
$ docker ps
```

Setting the `DOCKER_TLS_VERIFY` environment variable to any value other than the empty string is equivalent to setting the `--tlsverify` flag. The following are equivalent:

```
$ docker --tlsverify ps
# or
$ export DOCKER_TLS_VERIFY=1
$ docker ps
```

The Docker client will honor the `HTTP_PROXY` , `HTTPS_PROXY` , and `NO_PROXY` environment variables (or the lowercase versions thereof). `HTTPS_PROXY` takes precedence over `HTTP_PROXY` .

Starting with Docker 18.09, the Docker client supports connecting to a remote daemon via SSH:

```
$ docker -H ssh://me@example.com:22 ps
$ docker -H ssh://me@example.com ps
$ docker -H ssh://example.com ps
```

To use SSH connection, you need to set up `ssh` so that it can reach the remote host with public key authentication. Password authentication is not supported. If your key is protected with passphrase, you need to set up `ssh-agent` .

Also, you need to have `docker` binary 18.09 or later on the daemon host.

BIND DOCKER TO ANOTHER HOST/PORT OR A UNIX SOCKET

Warning: Changing the default `docker` daemon binding to a TCP port or Unix `docker` user group will increase your security risks by allowing non-root users to gain *root* access on the host. Make sure you control access to `docker` . If you are binding to a TCP port, anyone with access to that port has full Docker access; so it is not advisable on an open network.

With `-H` it is possible to make the Docker daemon to listen on a specific IP and port. By default, it will listen on `unix:///var/run/docker.sock` to allow only local connections by the *root* user. You *could* set it to `0.0.0.0:2375` or a specific host IP to give access to everybody, but that is **not recommended** because then it is trivial for someone to gain root access to the host where the daemon is running.

Similarly, the Docker client can use `-H` to connect to a custom port. The Docker client will default to connecting to `unix:///var/run/docker.sock` on Linux, and `tcp://127.0.0.1:2376` on Windows.

`-H` accepts host and port assignment in the following format:

`tcp://[host]:[port][path]` or `unix://path`

For example:

- `tcp://` -> TCP connection to `127.0.0.1` on either port `2376` when TLS encryption is on, or port `2375` when communication is in plain text.
- `tcp://host:2375` -> TCP connection on host:2375
- `tcp://host:2375/path` -> TCP connection on host:2375 and prepend path to all requests
- `unix://path/to/socket` -> Unix socket located at `path/to/socket`

`-H` , when empty, will default to the same value as when no `-H` was passed in.

`-H` also accepts short form for TCP bindings: `host:` or `host:port` or `:port`

Run Docker in daemon mode:

```
$ sudo <path to>/dockerd -H 0.0.0.0:5555 &
```

Download an `ubuntu` image:

```
$ docker -H :5555 pull ubuntu
```

You can use multiple `-H` , for example, if you want to listen on both TCP and a Unix socket

```
# Run docker in daemon mode
$ sudo <path to>/dockerd -H tcp://127.0.0.1:2375 -H unix:///var/run/docker.sock
# Download an ubuntu image, use default Unix socket
$ docker pull ubuntu
# OR use the TCP port
$ docker -H tcp://127.0.0.1:2375 pull ubuntu
```



Daemon storage-driver

On Linux, the Docker daemon has support for several different image layer storage drivers: `aufs` , `devicemapper` , `btrfs` , `zfs` , `overlay` and `overlay2` .

The `aufs` driver is the oldest, but is based on a Linux kernel patch-set that is unlikely to be merged into the main kernel. These are also known to cause some serious kernel crashes. However `aufs` allows containers to share executable and shared library memory, so is a useful choice when running thousands of containers with the same program or libraries.

The `devicemapper` driver uses thin provisioning and Copy on Write (CoW) snapshots. For each `devicemapper` graph location – typically `/var/lib/docker/devicemapper` – a thin pool is created based on two block devices, one for data and one for metadata. By default, these block devices are created automatically by using loopback mounts of automatically created sparse files. Refer to [Devicemapper options \(/engine/reference/commandline/dockerd/#devicemapper-options\)](#) below for a way how to customize this setup. [~jpetazzo/Resizing Docker containers with the Device Mapper plugin \(http://jpetazzo.github.io/2014/01/29/docker-device-mapper-resize/\)](#) article explains how to tune your existing setup without the use of options.

The `btrfs` driver is very fast for `docker build` - but like `devicemapper` does not share executable memory between devices. Use `dockerd -s btrfs -g /mnt/btrfs_partition .`

The `zfs` driver is probably not as fast as `btrfs` but has a longer track record on stability. Thanks to [Single Copy ARC](#) shared blocks between clones will be cached only once. Use `dockerd -s zfs` . To select a different `zfs` filesystem set `zfs.fsname` option as described in [ZFS options \(/engine/reference/commandline/dockerd/#zfs-options\)](#).

The `overlay` is a very fast union filesystem. It is now merged in the main Linux kernel as of 3.18.0 (<https://lkml.org/lkml/2014/10/26/137>). `overlay` also supports page cache sharing, this means multiple containers accessing the same file can share a single page cache entry (or entries), it makes `overlay` as efficient with memory as `aufs` driver. Call `dockerd -s overlay` to use it.

Note: As promising as `overlay` is, the feature is still quite young and should not be used in production. Most notably, using `overlay` can cause excessive inode consumption (especially as the number of images grows), as well as being incompatible with the use of RPMs.

The `overlay2` uses the same fast union filesystem but takes advantage of additional features (<https://lkml.org/lkml/2015/2/11/106>) added in Linux kernel 4.0 to avoid excessive inode consumption. Call `dockerd -s overlay2` to use it.

Note: Both `overlay` and `overlay2` are currently unsupported on `btrfs` or any Copy on Write filesystem and should only be used over `ext4` partitions.

On Windows, the Docker daemon supports a single image layer storage driver depending on the image platform: `windowsfilter` for Windows images, and `lcow` for Linux containers on Windows.

Options per storage driver

Particular storage-driver can be configured with options specified with `--storage-opt` flags. Options for `devicemapper` are prefixed with `dm` , options for `zfs` start with `zfs` , options for `btrfs` start with `btrfs` and options for `lcow` start with `lcow` .

DEVICEMAPPER OPTIONS

This is an example of the configuration file for devicemapper on Linux:

```
{
  "storage-driver": "devicemapper",
  "storage-opts": [
    "dm.thinpooldev=/dev/mapper/thin-pool",
    "dm.use_deferred_deletion=true",
    "dm.use_deferred_removal=true"
  ]
}
```

`dm.thinpooldev`

Specifies a custom block storage device to use for the thin pool.

If using a block device for device mapper storage, it is best to use `lvm` to create and manage the thin-pool volume. This volume is then handed to Docker to exclusively create snapshot volumes needed for images and containers.

Managing the thin-pool outside of Engine makes for the most feature-rich method of having Docker utilize device mapper thin provisioning as the backing storage for Docker containers. The highlights of the lvm-based thin-pool management feature include: automatic or interactive thin-pool resize support, dynamically changing thin-pool features, automatic thinp metadata checking when lvm activates the thin-pool, etc.

As a fallback if no thin pool is provided, loopback files are created. Loopback is very slow, but can be used without any pre-configuration of storage. It is strongly recommended that you do not use loopback in production. Ensure your Engine daemon has a `--storage-opt dm.thinpooldev` argument provided.

Example:

```
$ sudo dockerd --storage-opt dm.thinpooldev=/dev/mapper/thin-pool
```


`dm.directlvm_device`

As an alternative to providing a thin pool as above, Docker can setup a block device for you.

Example:

```
$ sudo dockerd --storage-opt dm.directlvm_device=/dev/xvdf
```

`dm.thinp_percent`

Sets the percentage of passed in block device to use for storage.

Example:

```
$ sudo dockerd --storage-opt dm.thinp_percent=95
```

dm.thinp_metapercent

Sets the percentage of the passed in block device to use for metadata storage.

Example:

```
$ sudo dockerd --storage-opt dm.thinp_metapercent=1
```

dm.thinp_autoextend_threshold

Sets the value of the percentage of space used before `lvm` attempts to autoextend the available space [100 = disabled]

Example:

```
$ sudo dockerd --storage-opt dm.thinp_autoextend_threshold=80
```

dm.thinp_autoextend_percent

Sets the value percentage value to increase the thin pool by when `lvm` attempts to autoextend the available space [100 = disabled]

Example:

```
$ sudo dockerd --storage-opt dm.thinp_autoextend_percent=20
```

dm.basesize

Specifies the size to use when creating the base device, which limits the size of images and containers. The default value is 10G. Note, thin devices are inherently “sparse”, so a 10G device which is mostly empty doesn’t use 10 GB of space on the pool. However, the filesystem will use more space for the empty case the larger the device is.

The base device size can be increased at daemon restart which will allow all future images and containers (based on those new images) to be of the new base device size.

Examples

```
$ sudo dockerd --storage-opt dm.basesize=50G
```

This will increase the base device size to 50G. The Docker daemon will throw an error if existing base device size is larger than 50G. A user can use this option to expand the base device size however shrinking is not permitted.

This value affects the system-wide “base” empty filesystem that may already be initialized and inherited by pulled images. Typically, a change to this value requires additional steps to take effect:

```
$ sudo service docker stop
```

```
$ sudo rm -rf /var/lib/docker
```

```
$ sudo service docker start
```

dm.loopdatasize

Note: This option configures devicemapper loopback, which should not be used in production.

Specifies the size to use when creating the loopback file for the “data” device which is used for the thin pool. The default size is 100G. The file is sparse, so it will not initially take up this much space.

Example

```
$ sudo dockerd --storage-opt dm.loopdatasize=200G
```

dm.loopmetadatasize

Note: This option configures devicemapper loopback, which should not be used in production.

Specifies the size to use when creating the loopback file for the “metadata” device which is used for the thin pool. The default size is 2G. The file is sparse, so it will not initially take up this much space.

Example

```
$ sudo dockerd --storage-opt dm.loopmetadatasize=4G
```

dm.fs

Specifies the filesystem type to use for the base device. The supported options are “ext4” and “xfs”. The default is “xfs”

Example

```
$ sudo dockerd --storage-opt dm.fs=ext4
```

dm.mkfsarg

Specifies extra mkfs arguments to be used when creating the base device.

Example

```
$ sudo dockerd --storage-opt "dm.mkfsarg=-O ^has_journal"
```

dm.mountopt

Specifies extra mount options used when mounting the thin devices.

Example

```
$ sudo dockerd --storage-opt dm.mountopt=nodiscard
```

dm.datadev

(Deprecated, use `dm.thinpooldev`)

Specifies a custom blockdevice to use for data for the thin pool.

If using a block device for device mapper storage, ideally both `datadev` and `metadatadev` should be specified to completely avoid using the loopback device.

Example

```
$ sudo dockerd \  
  --storage-opt dm.datadev=/dev/sdb1 \  
  --storage-opt dm.metadatadev=/dev/sdc1
```

dm.metadatadev

(Deprecated, use `dm.thinpooldev`)

Specifies a custom blockdevice to use for metadata for the thin pool.

For best performance the metadata should be on a different spindle than the data, or even better on an SSD.

If setting up a new metadata pool it is required to be valid. This can be achieved by zeroing the first 4k to indicate empty metadata, like this:

```
$ dd if=/dev/zero of=$metadata_dev bs=4096 count=1
```

Example

```
$ sudo dockerd \
  --storage-opt dm.datadev=/dev/sdb1 \
  --storage-opt dm.metadatadev=/dev/sdc1
```

dm.blocksize

Specifies a custom blocksize to use for the thin pool. The default blocksize is 64K.

Example

```
$ sudo dockerd --storage-opt dm.blocksize=512K
```

dm.blkdiscard

Enables or disables the use of `blkdiscard` when removing devicemapper devices. This is enabled by default (only) if using loopback devices and is required to resparsify the loopback file on image/container removal.

Disabling this on loopback can lead to *much* faster container removal times, but will make the space used in `/var/lib/docker` directory not be returned to the system for other use when containers are removed.

Examples

```
$ sudo dockerd --storage-opt dm.blkdiscard=false
```


`dm.override_udev_sync_check`

Overrides the `udev` synchronization checks between `devicemapper` and `udev` . `udev` is the device manager for the Linux kernel.

To view the `udev` sync support of a Docker daemon that is using the `devicemapper` driver, run:

```
$ docker info
[...]  
Udev Sync Supported: true  
[...]
```

When `udev` sync support is `true` , then `devicemapper` and `udev` can coordinate the activation and deactivation of devices for containers.

When `udev` sync support is `false` , a race condition occurs between the `devicemapper` and `udev` during create and cleanup. The race condition results in errors and failures. (For information on these failures, see [docker#4036](https://github.com/docker/docker/issues/4036) (<https://github.com/docker/docker/issues/4036>))

To allow the `docker` daemon to start, regardless of `udev` sync not being supported, set `dm.override_udev_sync_check` to `true`:

```
$ sudo dockerd --storage-opt dm.override_udev_sync_check=true
```

When this value is `true` , the `devicemapper` continues and simply warns you the errors are happening.

Note: The ideal is to pursue a `docker` daemon and environment that does support synchronizing with `udev` . For further discussion on this topic, see `docker#4036` (<https://github.com/docker/docker/issues/4036>). Otherwise, set this flag for migrating existing Docker daemons to a daemon with a supported environment.

`dm.use_deferred_removal`

Enables use of deferred device removal if `libdm` and the kernel driver support the mechanism.

Deferred device removal means that if device is busy when devices are being removed/deactivated, then a deferred removal is scheduled on device. And devices automatically go away when last user of the device exits.

For example, when a container exits, its associated thin device is removed. If that device has leaked into some other mount namespace and can't be removed, the container exit still succeeds and this option causes the system to schedule the device for deferred removal. It does not wait in a loop trying to remove a busy device.

Example

```
$ sudo dockerd --storage-opt dm.use_deferred_removal=true
```

`dm.use_deferred_deletion`

Enables use of deferred device deletion for thin pool devices. By default, thin pool device deletion is synchronous. Before a container is deleted, the Docker daemon removes any associated devices. If the storage driver can not remove a device, the container deletion fails and daemon returns.

Error deleting container: Error response from daemon: Cannot destr

To avoid this failure, enable both deferred device deletion and deferred device removal on the daemon.

```
$ sudo dockerd \
  --storage-opt dm.use_deferred_deletion=true \
  --storage-opt dm.use_deferred_removal=true
```

With these two options enabled, if a device is busy when the driver is deleting a container, the driver marks the device as deleted. Later, when the device isn't in use, the driver deletes it.

In general it should be safe to enable this option by default. It will help when unintentional leaking of mount point happens across multiple mount namespaces.

`dm.min_free_space`

Specifies the min free space percent in a thin pool require for new device creation to succeed. This check applies to both free data space as well as free metadata space. Valid values are from 0% - 99%. Value 0% disables free space checking logic. If user does not specify a value for this option, the Engine uses a default value of 10%.

Whenever a new a thin pool device is created (during `docker pull` or during container creation), the Engine checks if the minimum free space is available. If sufficient space is unavailable, then device creation fails and any relevant `docker` operation fails.

To recover from this error, you must create more free space in the thin pool to recover from the error. You can create free space by deleting some images and containers from the thin pool. You can also add more storage to the thin pool.

To add more space to a LVM (logical volume management) thin pool, just add more storage to the volume group container thin pool; this should automatically resolve any errors. If your configuration uses loop devices, then stop the Engine daemon, grow the size of loop files and restart the daemon to resolve the issue.

Example

```
$ sudo dockerd --storage-opt dm.min_free_space=10%
```

`dm.xfs_nospace_max_retries`

Specifies the maximum number of retries XFS should attempt to complete IO when ENOSPC (no space) error is returned by underlying storage device.

By default XFS retries infinitely for IO to finish and this can result in unkillable process. To change this behavior one can set `xfs_nospace_max_retries` to say 0 and XFS will not retry IO after getting ENOSPC and will shutdown filesystem.

Example

```
$ sudo dockerd --storage-opt dm.xfs_nospace_max_retries=0
```

`dm.libdm_log_level`

Specifies the maximum `libdm` log level that will be forwarded to the `dockerd` log (as specified by `--log-level`). This option is primarily intended for debugging problems involving `libdm` . Using values other than the defaults may cause false-positive warnings to be logged.

Values specified must fall within the range of valid `libdm` log levels. At the time of writing, the following is the list of `libdm` log levels as well as their corresponding levels when output by `dockerd` .

<code>libdm</code>	Level	Value	<code>--log-level</code>
<code>_LOG_FATAL</code>		2	error
<code>_LOG_ERR</code>		3	error
<code>_LOG_WARN</code>		4	warn
<code>_LOG_NOTICE</code>		5	info
<code>_LOG_INFO</code>		6	info
<code>_LOG_DEBUG</code>		7	debug

Example

```
$ sudo dockerd \
    --log-level debug \
    --storage-opt dm.libdm_log_level=7
```

ZFS OPTIONS

`zfs.fsname`

Set zfs filesystem under which docker will create its own datasets. By default docker will pick up the zfs filesystem where docker graph (`/var/lib/docker`) is located.

Example

```
$ sudo dockerd -s zfs --storage-opt zfs.fsname=zroot/docker
```

BTRFS OPTIONS

`btrfs.min_space`

Specifies the minimum size to use when creating the subvolume which is used for containers. If user uses disk quota for btrfs when creating or running a container with **--storage-opt size** option, docker should ensure the **size** cannot be smaller than **btrfs.min_space**.

Example

```
$ sudo dockerd -s btrfs --storage-opt btrfs.min_space=10G
```

OVERLAY2 OPTIONS

`overlay2.override_kernel_check`

Overrides the Linux kernel version check allowing overlay2. Support for specifying multiple lower directories needed by overlay2 was added to the Linux kernel in 4.0.0. However, some older kernel versions may be patched to add multiple lower directory support for OverlayFS. This option should only be used after verifying this support exists in the kernel. Applying this option on a kernel without this support will cause failures on mount.

`overlay2.size`

Sets the default max size of the container. It is supported only when the backing fs is `xf`s and mounted with `pquota` mount option. Under these conditions the user can pass any size less than the backing fs size.

Example

```
$ sudo dockerd -s overlay2 --storage-opt overlay2.size=1G
```

WINDOWSFILTER OPTIONS

`size`

Specifies the size to use when creating the sandbox which is used for containers. Defaults to 20G.

Example

```
C:\> dockerd --storage-opt size=40G
```

LCOW (LINUX CONTAINERS ON WINDOWS) OPTIONS

`lcow.globalmode`

Specifies whether the daemon instantiates utility VM instances as required (recommended and default if omitted), or uses single global utility VM (better performance, but has security implications and not recommended for production deployments).

Example

```
C:\> dockerd --storage-opt lcow.globalmode=false
```

`lcow.kirdpath`

Specifies the folder path to the location of a pair of kernel and initrd files used for booting a utility VM. Defaults to

```
%ProgramFiles%\Linux Containers .
```

Example

```
C:\> dockerd --storage-opt lcow.kirdpath=c:\path\to\files
```

`lcow.kernel`

Specifies the filename of a kernel file located in the `lcow.kirdpath` path. Defaults to `bootx64.efi` .

Example

```
C:\> dockerd --storage-opt lcow.kernel=kernel.efi
```

lcow.initrd

Specifies the filename of an initrd file located in the `lcow.kirdpath` path. Defaults to `initrd.img` .

Example

```
C:\> dockerd --storage-opt lcow.initrd=myinitrd.img
```

lcow.bootparameters

Specifies additional boot parameters for booting utility VMs when in kernel/ initrd mode. Ignored if the utility VM is booting from VHD. These settings are kernel specific.

Example

```
C:\> dockerd --storage-opt "lcow.bootparameters='option=value'"
```



lcow.vhdx

Specifies a custom VHDX to boot a utility VM, as an alternate to kernel and initrd booting. Defaults to `uvm.vhdx` under `lcow.kirdpath` .

Example

```
C:\> dockerd --storage-opt lcow.vhdx=custom.vhdx
```

lcow.timeout

Specifies the timeout for utility VM operations in seconds. Defaults to 300.

Example


```
C:\> dockerd --storage-opt lcow.timeout=240
```

`lcow.sandboxsize`

Specifies the size in GB to use when creating the sandbox which is used for containers. Defaults to 20. Cannot be less than 20.

Example

```
C:\> dockerd --storage-opt lcow.sandboxsize=40
```

Docker runtime execution options

The Docker daemon relies on a OCI (<https://github.com/opencontainers/runtime-spec>) compliant runtime (invoked via the `containerd` daemon) as its interface to the Linux kernel namespaces , cgroups , and SELinux .

By default, the Docker daemon automatically starts `containerd` . If you want to control `containerd` startup, manually start `containerd` and pass the path to the `containerd` socket using the `--containerd` flag. For example:

```
$ sudo dockerd --containerd /var/run/dev/docker-containerd.sock
```



Runtimes can be registered with the daemon either via the configuration file or using the `--add-runtime` command line argument.

The following is an example adding 2 runtimes via the configuration:

```
{
  "default-runtime": "runc",
  "runtimes": {
    "runc": {
      "path": "runc"
    },
    "custom": {
      "path": "/usr/local/bin/my-runc-replacemer
      "runtimeArgs": [
        "--debug"
      ]
    }
  }
}
```

This is the same example via the command line:

```
$ sudo dockerd --add-runtime runc=runc --add-runtime custom=/usr/.
```

Note: Defining runtime arguments via the command line is not supported.

OPTIONS FOR THE RUNTIME

You can configure the runtime using options specified with the `--exec-opt` flag. All the flag's options have the `native` prefix. A single `native.cgroupdriver` option is available.

The `native.cgroupdriver` option specifies the management of the container's cgroups. You can only specify `cgroupfs` or `systemd`. If you specify `systemd` and it is not available, the system errors out. If you omit the `native.cgroupdriver` option, `cgroupfs` is used.

This example sets the `cgroupdriver` to `systemd` :

```
$ sudo dockerd --exec-opt native.cgroupdriver=systemd
```

Setting this option applies to all containers the daemon launches.

Also Windows Container makes use of `--exec-opt` for special purpose. Docker user can specify default container isolation technology with this, for example:

```
> dockerd --exec-opt isolation=hyperv
```

Will make `hyperv` the default isolation technology on Windows. If no isolation value is specified on daemon start, on Windows client, the default is `hyperv`, and on Windows server, the default is `process`.

Daemon DNS options

To set the DNS server for all Docker containers, use:

```
$ sudo dockerd --dns 8.8.8.8
```

To set the DNS search domain for all Docker containers, use:

```
$ sudo dockerd --dns-search example.com
```

Allow push of nondistributable artifacts

Some images (e.g., Windows base images) contain artifacts whose distribution is restricted by license. When these images are pushed to a registry, restricted artifacts are not included.

To override this behavior for specific registries, use the

`--allow-nondistributable-artifacts` option in one of the following forms:

- `--allow-nondistributable-artifacts myregistry:5000` tells the Docker daemon to push nondistributable artifacts to myregistry:5000.
- `--allow-nondistributable-artifacts 10.1.0.0/16` tells the Docker daemon to push nondistributable artifacts to all registries whose resolved IP address is within the subnet described by the CIDR syntax.

This option can be used multiple times.

This option is useful when pushing images containing nondistributable artifacts to a registry on an air-gapped network so hosts on that network can pull the images without connecting to another server.

Warning: Nondistributable artifacts typically have restrictions on how and where they can be distributed and shared. Only use this feature to push artifacts to private registries and ensure that you are in compliance with any terms that cover redistributing nondistributable artifacts.

Insecure registries

Docker considers a private registry either secure or insecure. In the rest of this section, *registry* is used for *private registry*, and `myregistry:5000` is a placeholder example for a private registry.

A secure registry uses TLS and a copy of its CA certificate is placed on the Docker host at `/etc/docker/certs.d/myregistry:5000/ca.crt`. An insecure registry is either not using TLS (i.e., listening on plain text HTTP), or is using TLS with a CA certificate not known by the Docker daemon. The latter can happen when the certificate was not found under

`/etc/docker/certs.d/myregistry:5000/`, or if the certificate verification failed (i.e., wrong CA).

By default, Docker assumes all, but local (see local registries below), registries are secure. Communicating with an insecure registry is not possible if Docker assumes that registry is secure. In order to communicate with an insecure registry, the Docker daemon requires

`--insecure-registry` in one of the following two forms:

- `--insecure-registry myregistry:5000` tells the Docker daemon that myregistry:5000 should be considered insecure.
- `--insecure-registry 10.1.0.0/16` tells the Docker daemon that all registries whose domain resolve to an IP address is part of the subnet described by the CIDR syntax, should be considered insecure.

The flag can be used multiple times to allow multiple registries to be marked as insecure.

If an insecure registry is not marked as insecure, `docker pull` , `docker push` , and `docker search` will result in an error message prompting the user to either secure or pass the `--insecure-registry` flag to the Docker daemon as described above.

Local registries, whose IP address falls in the 127.0.0.0/8 range, are automatically marked as insecure as of Docker 1.3.2. It is not recommended to rely on this, as it may change in the future.

Enabling `--insecure-registry` , i.e., allowing un-encrypted and/or untrusted communication, can be useful when running a local registry. However, because its use creates security vulnerabilities it should ONLY be enabled for testing purposes. For increased security, users should add their CA to their system's list of trusted CAs instead of enabling `--insecure-registry` .

LEGACY REGISTRIES

Starting with Docker 17.12, operations against registries supporting only the legacy v1 protocol are no longer supported. Specifically, the daemon will not attempt `push` , `pull` and `login` to v1 registries. The exception to this is `search` which can still be performed on v1 registries.

The `disable-legacy-registry` configuration option has been removed and, when used, will produce an error on daemon startup.

Running a Docker daemon behind an HTTPS_PROXY

When running inside a LAN that uses an `HTTPS` proxy, the Docker Hub certificates will be replaced by the proxy's certificates. These certificates need to be added to your Docker host's configuration:

1. Install the `ca-certificates` package for your distribution
2. Ask your network admin for the proxy's CA certificate and append them to `/etc/pki/tls/certs/ca-bundle.crt`
3. Then start your Docker daemon with `HTTPS_PROXY=http://username:password@proxy:port/ dockerd`. The `username:` and `password@` are optional - and are only needed if your proxy is set up to require authentication.

This will only add the proxy and authentication to the Docker daemon's requests - your `docker build` s and running containers will need extra configuration to use the proxy

Default `ulimit` settings

`--default-ulimit` allows you to set the default `ulimit` options to use for all containers. It takes the same options as `--ulimit` for `docker run`. If these defaults are not set, `ulimit` settings will be inherited, if not set on `docker run`, from the Docker daemon. Any `--ulimit` options passed to `docker run` will overwrite these defaults.

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 details please check the run (<https://docs.docker.com/engine/reference/commandline/run/>) reference.

Node discovery

The `--cluster-advertise` option specifies the `host:port` or `interface:port` combination that this particular daemon instance should use when advertising itself to the cluster. The daemon is reached by remote hosts through this value. If you specify an interface, make sure it includes the IP address of the actual Docker host. For Engine installation created through `docker-machine`, the interface is typically `eth1`.

The daemon uses libkv (<https://github.com/docker/libkv/>) to advertise the node within the cluster. Some key-value backends support mutual TLS. To configure the client TLS settings used by the daemon can be configured using the `--cluster-store-opt` flag, specifying the paths to PEM encoded files. For example:

```
$ sudo dockerd \
  --cluster-advertise 192.168.1.2:2376 \
  --cluster-store etcd://192.168.1.2:2379 \
  --cluster-store-opt kv.cacertfile=/path/to/ca.pem \
  --cluster-store-opt kv.certfile=/path/to/cert.pem \
  --cluster-store-opt kv.keyfile=/path/to/key.pem
```

The currently supported cluster store options are:

Option	Description
<code>discovery.heartbeat</code>	Specifies the heartbeat timer in seconds which is used by the daemon as a <code>keepalive</code> mechanism to make sure discovery module treats the node as alive in the cluster. If not configured, the default value is 20 seconds.
<code>discovery.ttl</code>	Specifies the TTL (time-to-live) in seconds which is used by the discovery module to timeout a node if a valid heartbeat is not received within the configured ttl value. If not configured, the default value is 60 seconds.

Option	Description
<code>kv.cacertfile</code>	Specifies the path to a local file with PEM encoded CA certificates to trust.
<code>kv.certfile</code>	Specifies the path to a local file with a PEM encoded certificate. This certificate is used as the client cert for communication with the Key/Value store.
<code>kv.keyfile</code>	Specifies the path to a local file with a PEM encoded private key. This private key is used as the client key for communication with the Key/Value store.
<code>kv.path</code>	Specifies the path in the Key/Value store. If not configured, the default value is 'docker/nodes'.

Access authorization

Docker's access authorization can be extended by authorization plugins that your organization can purchase or build themselves. You can install one or more authorization plugins when you start the Docker `daemon` using the `--authorization-plugin=PLUGIN_ID` option.

```
$ sudo dockerd --authorization-plugin=plugin1 --authorization-plug
```



The `PLUGIN_ID` value is either the plugin's name or a path to its specification file. The plugin's implementation determines whether you can specify a name or path. Consult with your Docker administrator to get information about the plugins available to you.

Once a plugin is installed, requests made to the `daemon` through the command line or Docker's Engine API are allowed or denied by the plugin. If you have multiple plugins installed, each plugin, in order, must allow the request for it to complete.

For information about how to create an authorization plugin, see [authorization plugin](https://docs.docker.com/engine/extend/plugins_authorization/) (https://docs.docker.com/engine/extend/plugins_authorization/) section in the Docker extend section of this documentation.

Daemon user namespace options

The Linux kernel user namespace support (http://man7.org/linux/man-pages/man7/user_namespaces.7.html) provides additional security by enabling a process, and therefore a container, to have a unique range of user and group IDs which are outside the traditional user and group range utilized by the host system. Potentially the most important security improvement is that, by default, container processes running as the `root` user will have expected administrative privilege (with some restrictions) inside the container but will effectively be mapped to an unprivileged `uid` on the host.

For details about how to use this feature, as well as limitations, see [Isolate containers with a user namespace](https://docs.docker.com/engine/security/userns-remap/) (<https://docs.docker.com/engine/security/userns-remap/>).

Miscellaneous options

IP masquerading uses address translation to allow containers without a public IP to talk to other machines on the Internet. This may interfere with some network topologies and can be disabled with `--ip-masq=false`.

Docker supports softlinks for the Docker data directory (`/var/lib/docker`) and for `/var/lib/docker/tmp`. The `DOCKER_TMPDIR` and the data directory can be set like this:

```
DOCKER_TMPDIR=/mnt/disk2/tmp /usr/local/bin/dockerd -D -g /var/lib
# or
export DOCKER_TMPDIR=/mnt/disk2/tmp
/usr/local/bin/dockerd -D -g /var/lib/docker -H unix:/// > /var/lib
```

DEFAULT CGROUP PARENT

The `--cgroup-parent` option allows you to set the default cgroup parent to use for containers. If this option is not set, it defaults to `/docker` for fs cgroup driver and `system.slice` for systemd cgroup driver.

If the cgroup has a leading forward slash (`/`), the cgroup is created under the root cgroup, otherwise the cgroup is created under the daemon cgroup.

Assuming the daemon is running in cgroup `daemoncgroup` ,

`--cgroup-parent=/foobar` creates a cgroup in `/sys/fs/cgroup/memory/foobar` , whereas using `--cgroup-parent=foobar` creates the cgroup in `/sys/fs/cgroup/memory/daemoncgroup/foobar`

The systemd cgroup driver has different rules for `--cgroup-parent` .

Systemd represents hierarchy by slice and the name of the slice encodes the location in the tree. So `--cgroup-parent` for systemd cgroups should be a slice name. A name can consist of a dash-separated series of names, which describes the path to the slice from the root slice. For example,

`--cgroup-parent=user-a-b.slice` means the memory cgroup for the container is created in

`/sys/fs/cgroup/memory/user.slice/user-a.slice/user-a-b.slice/docker-<id>.scope` .

This setting can also be set per container, using the `--cgroup-parent` option on `docker create` and `docker run` , and takes precedence over the `--cgroup-parent` option on the daemon.

DAEMON METRICS

The `--metrics-addr` option takes a tcp address to serve the metrics API. This feature is still experimental, therefore, the daemon must be running in experimental mode for this feature to work.

To serve the metrics API on `localhost:9323` you would specify `--metrics-addr 127.0.0.1:9323`, allowing you to make requests on the API at `127.0.0.1:9323/metrics` to receive metrics in the prometheus (https://prometheus.io/docs/instrumenting/exposition_formats/) format.

Port `9323` is the default port associated with Docker metrics (<https://github.com/prometheus/prometheus/wiki/Default-port-allocations>) to avoid collisions with other prometheus exporters and services.

If you are running a prometheus server you can add this address to your scrape configs to have prometheus collect metrics on Docker. For more information on prometheus you can view the website here (<https://prometheus.io/>).

```
scrape_configs:
  - job_name: 'docker'
    static_configs:
      - targets: ['127.0.0.1:9323']
```

Please note that this feature is still marked as experimental as metrics and metric names could change while this feature is still in experimental. Please provide feedback on what you would like to see collected in the API.


NODE GENERIC RESOURCES

The `--node-generic-resources` option takes a list of key-value pair (`key=value`) that allows you to advertise user defined resources in a swarm cluster.

The current expected use case is to advertise NVIDIA GPUs so that services requesting `NVIDIA-GPU=[0-16]` can land on a node that has enough GPUs for the task to run.

Example of usage:

```
{  
    "node-generic-resources": ["NVIDIA-GPU=UUID1", "NVIDIA-GPU=UUID2", "NVIDIA-GPU=UUID3"],  
}
```



Daemon configuration file

The `--config-file` option allows you to set any configuration option for the daemon in a JSON format. This file uses the same flag names as keys, except for flags that allow several entries, where it uses the plural of the flag name, e.g., `labels` for the `label` flag.

The options set in the configuration file must not conflict with options set via flags. The docker daemon fails to start if an option is duplicated between the file and the flags, regardless their value. We do this to avoid silently ignore changes introduced in configuration reloads. For example, the daemon fails to start if you set daemon labels in the configuration file and also set daemon labels via the `--label` flag. Options that are not present in the file are ignored when the daemon starts.

On Linux

The default location of the configuration file on Linux is `/etc/docker/daemon.json`. The `--config-file` flag can be used to specify a non-default location.

This is a full example of the allowed configuration options on Linux:

```
{
  "authorization-plugins": [],
  "data-root": "",
  "dns": [],
  "dns-opts": [],
  "dns-search": [],
  "exec-opts": [],
  "exec-root": "",
  "experimental": false,
  "features": {},
  "storage-driver": "",
  "storage-opts": [],
  "labels": [],
  "live-restore": true,
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "5",
    "labels": "somelabel",
    "env": "os,customer"
  },
  "mtu": 0,
  "pidfile": "",
  "cluster-store": "",
  "cluster-store-opts": {},
  "cluster-advertise": "",
  "max-concurrent-downloads": 3,
  "max-concurrent-uploads": 5,
  "default-shm-size": "64M",
  "shutdown-timeout": 15,
  "debug": true,
  "hosts": [],
  "log-level": "",
  "tls": true,
  "tlsverify": true,
  "tlscacert": "",
  "tlscert": "",
  "tlskey": "",
  "swarm-default-advertise-addr": "",
  "api-cors-header": "",
  "selinux-enabled": false,
  "userns-remap": "",
  "group": "",
```

```

"cgroup-parent": "",
"default-ulimits": {
    "nofile": {
        "Name": "nofile",
        "Hard": 64000,
        "Soft": 64000
    }
},
"init": false,
"init-path": "/usr/libexec/docker-init",
"ipv6": false,
"iptables": false,
"ip-forward": false,
"ip-masq": false,
"userland-proxy": false,
"userland-proxy-path": "/usr/libexec/docker-proxy",
"ip": "0.0.0.0",
"bridge": "",
"bip": "",
"fixed-cidr": "",
"fixed-cidr-v6": "",
"default-gateway": "",
"default-gateway-v6": "",
"icc": false,
"raw-logs": false,
"allow-nondistributable-artifacts": [],
"registry-mirrors": [],
"seccomp-profile": "",
"insecure-registries": [],
"no-new-privileges": false,
"default-runtime": "runc",
"oom-score-adjust": -500,
"node-generic-resources": ["NVIDIA-GPU=UUID1", "NVIDIA-GPU=UUID2"],
"runtimes": {
    "cc-runtime": {
        "path": "/usr/bin/cc-runtime"
    },
    "custom": {
        "path": "/usr/local/bin/my-runc-replacemer",
        "runtimeArgs": [
            "--debug"
        ]
    }
},
"default-address-pools": [{ "base": "172.80.0.0/16", "size": 24 } ]

```

```
    {"base": "172.90.0.0/16", "size": 24}]  
}
```

Note: You cannot set options in `daemon.json` that have already been set on daemon startup as a flag. On systems that use `systemd` to start the Docker daemon, `-H` is already set, so you cannot use the `hosts` key in `daemon.json` to add listening addresses. See <https://docs.docker.com/engine/admin/systemd/#custom-docker-daemon-options> for how to accomplish this task with a `systemd` drop-in file.

On Windows

The default location of the configuration file on Windows is

`%programdata%\docker\config\daemon.json`. The `--config-file` flag can be used to specify a non-default location.

This is a full example of the allowed configuration options on Windows:

```

{
  "authorization-plugins": [],
  "data-root": "",
  "dns": [],
  "dns-opts": [],
  "dns-search": [],
  "exec-opts": [],
  "experimental": false,
  "features": {},
  "storage-driver": "",
  "storage-opts": [],
  "labels": [],
  "log-driver": "",
  "mtu": 0,
  "pidfile": "",
  "cluster-store": "",
  "cluster-advertise": "",
  "max-concurrent-downloads": 3,
  "max-concurrent-uploads": 5,
  "shutdown-timeout": 15,
  "debug": true,
  "hosts": [],
  "log-level": "",
  "tlsverify": true,
  "tlscacert": "",
  "tlscert": "",
  "tlskey": "",
  "swarm-default-advertise-addr": "",
  "group": "",
  "default-ulimits": {},
  "bridge": "",
  "fixed-cidr": "",
  "raw-logs": false,
  "allow-nondistributable-artifacts": [],
  "registry-mirrors": [],
  "insecure-registries": []
}

```

FEATURE OPTIONS

The optional field `features` in `daemon.json` allows users to enable or disable specific daemon features. For example,

`{"features":{"buildkit": true}}` enables `buildkit` as the default docker image builder.

The list of currently supported feature options:

- `buildkit` : It enables `buildkit` as default builder when set to `true` or disables it by `false` . Note that if this option is not explicitly set in the daemon config file, then it is up to the cli to determine which builder to invoke.

CONFIGURATION RELOAD BEHAVIOR

Some options can be reconfigured when the daemon is running without requiring to restart the process. We use the `SIGHUP` signal in Linux to reload, and a global event in Windows with the key

`Global\docker-daemon-config-$PID` . The options can be modified in the configuration file but still will check for conflicts with the provided flags. The daemon fails to reconfigure itself if there are conflicts, but it won't stop execution.

The list of currently supported options that can be reconfigured is this:

- `debug` : it changes the daemon to debug mode when set to true.
- `cluster-store` : it reloads the discovery store with the new address.
- `cluster-store-opts` : it uses the new options to reload the discovery store.
- `cluster-advertise` : it modifies the address advertised after reloading.
- `labels` : it replaces the daemon labels with a new set of labels.
- `live-restore` : Enables keeping containers alive during daemon downtime (<https://docs.docker.com/config/containers/live-restore/>).
- `max-concurrent-downloads` : it updates the max concurrent downloads for each pull.
- `max-concurrent-uploads` : it updates the max concurrent uploads for each push.
- `default-runtime` : it updates the runtime to be used if not is specified at container creation. It defaults to "default" which is the runtime shipped with the official docker packages.

- `runtimes` : it updates the list of available OCI runtimes that can be used to run containers.
- `authorization-plugin` : it specifies the authorization plugins to use.
- `allow-nondistributable-artifacts` : Replaces the set of registries to which the daemon will push nondistributable artifacts with a new set of registries.
- `insecure-registries` : it replaces the daemon insecure registries with a new set of insecure registries. If some existing insecure registries in daemon's configuration are not in newly reloaded insecure registries, these existing ones will be removed from daemon's config.
- `registry-mirrors` : it replaces the daemon registry mirrors with a new set of registry mirrors. If some existing registry mirrors in daemon's configuration are not in newly reloaded registry mirrors, these existing ones will be removed from daemon's config.
- `shutdown-timeout` : it replaces the daemon's existing configuration timeout with a new timeout for shutting down all containers.
- `features` : it explicitly enables or disables specific features.

Updating and reloading the cluster configurations such as

`--cluster-store` , `--cluster-advertise` and `--cluster-store-opts` will take effect only if these configurations were not previously configured. If `--cluster-store` has been provided in flags and `cluster-advertise` not, `cluster-advertise` can be added in the configuration file without accompanied by `--cluster-store` . Configuration reload will log a warning message if it detects a change in previously configured cluster configurations.

Run multiple daemons

Note: Running multiple daemons on a single host is considered as "experimental". The user should be aware of unsolved problems. This solution may not work properly in some cases. Solutions are currently under development and will be delivered in the near future.

This section describes how to run multiple Docker daemons on a single host. To run multiple daemons, you must configure each daemon so that it does not conflict with other daemons on the same host. You can set these options either by providing them as flags, or by using a daemon configuration file (/engine/reference/commandline/dockerd/#daemon-configuration-file).

The following daemon options must be configured for each daemon:

<code>-b, --bridge=</code>	Attach containers to a network
<code>--exec-root=/var/run/docker</code>	Root of the Docker execution
<code>--data-root=/var/lib/docker</code>	Root of persisted Docker data
<code>-p, --pidfile=/var/run/docker.pid</code>	Path to use for daemon PID file
<code>-H, --host=[]</code>	Daemon socket(s) to connect to
<code>--iptables=true</code>	Enable addition of iptables rules
<code>--config-file=/etc/docker/daemon.json</code>	Daemon configuration file
<code>--tlscacert=~/.docker/ca.pem</code>	Trust certs signed only by this CA
<code>--tlscert=~/.docker/cert.pem</code>	Path to TLS certificate file
<code>--tlskey=~/.docker/key.pem</code>	Path to TLS key file

When your daemons use different values for these flags, you can run them on the same host without any problems. It is very important to properly understand the meaning of those options and to use them correctly.

- The `-b, --bridge=` flag is set to `docker0` as default bridge network. It is created automatically when you install Docker. If you are not using the default, you must create and configure the bridge manually or just set it to 'none': `--bridge=none`
- `--exec-root` is the path where the container state is stored. The default value is `/var/run/docker`. Specify the path for your running daemon here.
- `--data-root` is the path where persisted data such as images, volumes, and cluster state are stored. The default value is `/var/lib/docker`. To avoid any conflict with other daemons, set this parameter separately for each daemon.
- `-p, --pidfile=/var/run/docker.pid` is the path where the process ID of the daemon is stored. Specify the path for your pid file here.

- `--host=[]` specifies where the Docker daemon will listen for client connections. If unspecified, it defaults to `/var/run/docker.sock`.
- `--iptables=false` prevents the Docker daemon from adding iptables rules. If multiple daemons manage iptables rules, they may overwrite rules set by another daemon. Be aware that disabling this option requires you to manually add iptables rules to expose container ports. If you prevent Docker from adding iptables rules, Docker will also not add IP masquerading rules, even if you set `--ip-masq` to `true`. Without IP masquerading rules, Docker containers will not be able to connect to external hosts or the internet when using network other than default bridge.
- `--config-file=/etc/docker/daemon.json` is the path where configuration file is stored. You can use it instead of daemon flags. Specify the path for each daemon.
- `--tls*` Docker daemon supports `--tlsverify` mode that enforces encrypted and authenticated remote connections. The `--tls*` options enable use of specific certificates for individual daemons.

Example script for a separate “bootstrap” instance of the Docker daemon without network:

```
$ sudo dockerd \
  -H unix:///var/run/docker-bootstrap.sock \
  -p /var/run/docker-bootstrap.pid \
  --iptables=false \
  --ip-masq=false \
  --bridge=none \
  --data-root=/var/lib/docker-bootstrap \
  --exec-root=/var/run/docker-bootstrap
```

container (<https://docs.docker.com/glossary/?term=container>), daemon (<https://docs.docker.com/glossary/?term=daemon>), runtime (<https://docs.docker.com/glossary/?term=runtime>)