NAME

dockerd - Enable daemon mode

SYNOPSIS

dockerd [**--add-runtime**[=*[]*]] [**--allow-nondistributable-artifacts**[=*[]*]] [**--api-cors-header**=[=*API-CORS-HEADER*]] [**--authorization-plugin**[=*[]*]] [**-b**|**--bridge**[=*BRIDGE*]] [**--bip**[=*BIP*]] [**--cgroup-parent**[=*[]*]] [**--cluster-store**[=*[]*]] [**--cluster-advertise**[=*[]*]] [**--cluster-store-opt**[=*map[]*]] [**--config-file**[=*/etc/docker/daemon.json*]] [**--containerd**[=*SOCKET-PATH*]] [**--data-root**[=*/var/lib/docker*]] [**-D**|**--debug**] [**--default-cgroupns-mode**[=*host*]] [**--default-gateway**[=*DEFAULT-GATEWAY*]] [**--default-gateway-v6**[=*DEFAULT-GATEWAY-V6*]] [**--default-address-pool**[=*DEFAULT-ADDRESS-POOL*]] [**--default-runtime**[=*runc*]] [**--default-ipc-mode**=*MODE*] [**--default-shm-size**[=*64MiB*]] [**--default-ulimit**[=*[]*]] [**--dns**[=*[]*]] [**--dns-opt**[=*[]*]] [**--dns-search**[=*[]*]] [**--exec-opt**[=*[]*]] [**--exec-root**[=*/var/run/docker*]] [**--experimental**[=*false*]] [**--fixed-cidr**[=*FIXED-CIDR*]] [**--fixed-cidr-v6**[=*FIXED-CIDR-V6*]] [**-G**|**--group**[=*docker*]] [**-H**|**--host**[=*[]*]] [**--help**] [**--icc**[=*true*]] [**--init**[=*false*]] [**--init-path**[=*""*]] [**--insecure-registry**[=*[]*]] [**--ip**[=*0.0.0.0*]] [**--ip-forward**[=*true*]] [**--ip-masq**[=*true*]] [**--iptables**[=*true*]] [**--ipv6**] [**--isolation**[=*default*]] [**-l**|**--log-level**[=*info*]] [**--label**[=*[]*]] [**--live-restore**[=*false*]] [**--log-driver**[=*json-file*]] [**--log-opt**[=*map[]*]] [**--mtu**[=*0*]] [**--max-concurrent-downloads**[=*3*]] [**--max-concurrent-uploads**[=*5*]] [**--max-download-attempts**[=*5*]] [**--node-generic-resources**[=*[]*]] [**-p**|**--pidfile**[=*/var/run/docker.pid*]] [**--raw-logs**] [**--registry-mirror**[=*[]*]] [**-s**|**--storage-driver**[=*STORAGE-DRIVER*]] [**--seccomp-profile**[=*SECCOMP-PROFILE-PATH*]] [**--selinux-enabled**] [**--shutdown-timeout**[=*15*]] [**--storage-opt**[=*[]*]] [**--swarm-default-advertise-addr**[=*IP|INTERFACE*]] [**--tls**] [**--tlscacert**[=*~/.docker/ca.pem*]] [**--tlscert**[=*~/.docker/cert.pem*]] [**--tlskey**[=*~/.docker/key.pem*]] [**--tlsverify**] [**--userland-proxy**[=*true*]] [**--userland-proxy-path**[=*""*]] [**--userns-remap**[=*default*]]

DESCRIPTION

dockerd is used for starting the Docker daemon (i.e., to command the daemon to manage images, containers etc). So **dockerd** is a server, as a daemon.

To run the Docker daemon you can specify **dockerd**. You can check the daemon options using **dockerd --help**. Daemon options should be specified after the **dockerd** keyword in the following format.

dockerd [OPTIONS]

OPTIONS

--add-runtime=[]

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-replacement",
                        "runtimeArgs": [
```

```
                                        "--debug"
                                    ]
                                }
                        }
                }
```

This is the same example via the command line:

```
$ sudo dockerd --add-runtime runc=runc --add-runtime custom=/usr/local/bin/my-runc-replacement
```

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

--allow-nondistributable-artifacts=[]
Push nondistributable artifacts to the specified registries.

List can contain elements with CIDR notation to specify a whole subnet.

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.

--api-cors-header=""
Set CORS headers in the Engine API. Default is cors disabled. Give urls like "http://foo, http://bar, ...". Give "*" to allow all.

--authorization-plugin=""
Set authorization plugins to load

-b, **--bridge**=""
Attach containers to a pre-existing network bridge; use 'none' to disable container networking

--bip=""
Use the provided CIDR notation address for the dynamically created bridge (docker0); Mutually exclusive of -b

--cgroup-parent=""
Set parent cgroup for all containers. Default is "/docker" for fs cgroup driver and "system.slice" for systemd cgroup driver.

--cluster-store=""
URL of the distributed storage backend

--cluster-advertise=""
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 through this value.

--cluster-store-opt=""
Specifies options for the Key/Value store.

--config-file="/etc/docker/daemon.json"
Specifies the JSON file path to load the configuration from.

--containerd=""
Path to containerd socket.

--data-root=""
Path to the directory used to store persisted Docker data such as configuration for resources, swarm cluster state, and filesystem data for images, containers, and local volumes. Default is `/var/lib/docker`.

-D, **--debug**=*true*|*false*
Enable debug mode. Default is false.

--default-cgroupns-mode="**host**|**private**"
Set the default cgroup namespace mode for newly created containers. The argument can either be **host** or **private**. If unset, this defaults to `host` on cgroup v1, `private` on cgroup v2.

--default-gateway=""
IPv4 address of the container default gateway; this address must be part of the bridge subnet (which is defined by -b or --bip)

--default-gateway-v6=""
IPv6 address of the container default gateway

--default-address-pool=""
Default address pool from which IPAM driver selects a subnet for the networks. Example: base=172.30.0.0/16,size=24 will set the default address pools for the selected scope networks to {172.30.[0-255].0/24}

--default-runtime="runc"
Set default runtime if there're more than one specified by `--add-runtime`.

--default-ipc-mode="**private**|**shareable**"
Set the default IPC mode for newly created containers. The argument can either be **private** or **shareable**.

--default-shm-size=*64MiB*
Set the daemon-wide default shm size for containers. Default is `64MiB`.

--default-ulimit=[]

Default ulimits for containers.

--dns=""
Force Docker to use specific DNS servers

--dns-opt=""
DNS options to use.

--dns-search=[]
DNS search domains to use.

--exec-opt=[]
Set runtime execution options. See RUNTIME EXECUTION OPTIONS.

--exec-root=""
Path to use as the root of the Docker execution state files. Default is `/var/run/docker`.

--experimental=""
Enable the daemon experimental features.

--fixed-cidr=""
IPv4 subnet for fixed IPs (e.g., 10.20.0.0/16); this subnet must be nested in the bridge subnet (which is defined by -b or --bip).

--fixed-cidr-v6=""
IPv6 subnet for global IPv6 addresses (e.g., 2a00:1450::/64)

-G, **--group=**""
Group to assign the unix socket specified by -H when running in daemon mode. use '' (the empty string) to disable setting of a group. Default is `docker`.

-H, **--host=**[*unix:///var/run/docker.sock*]: tcp://[host:port] to bind or unix:///[/path/to/socket] to use.
The socket(s) to bind to in daemon mode specified using one or more tcp://host:port, unix:///path/to/socket, fd://* or fd://socketfd.

--help
Print usage statement

--icc=*true|false*
Allow unrestricted inter-container and Docker daemon host communication. If disabled, containers can still be linked together using the **--link** option (see **docker-run(1)**). Default is true.

--init
Run an init process inside containers for signal forwarding and process reaping.

--init-path

Path to the docker-init binary.

--insecure-registry=[]
Enable insecure registry communication, i.e., enable un-encrypted and/or untrusted communication.

List of insecure registries can contain an element with CIDR notation to specify a whole subnet. Insecure registries accept HTTP and/or accept HTTPS with certificates from unknown CAs.

Enabling `--insecure-registry` is 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 using `--insecure-registry`.

--ip=""
Default IP address to use when binding container ports. Default is `0.0.0.0`.

--ip-forward=*true|false*
Enables IP forwarding on the Docker host. The default is `true`. This flag interacts with the IP forwarding setting on your host system's kernel. If your system has IP forwarding disabled, this setting enables it. If your system has IP forwarding enabled, setting this flag to `--ip-forward=false` has no effect.

This setting will also enable IPv6 forwarding if you have both `--ip-forward=true` and `--fixed-cidr-v6` set. Note that this may reject Router Advertisements and interfere with the host's existing IPv6 configuration. For more information, please consult the documentation about "Advanced Networking - IPv6".

--ip-masq=*true|false*
Enable IP masquerading for bridge's IP range. Default is true.

--iptables=*true|false*
Enable Docker's addition of iptables rules. Default is true.

--ipv6=*true|false*
Enable IPv6 support. Default is false. Docker will create an IPv6-enabled bridge with address fe80::1 which will allow you to create IPv6-enabled containers. Use together with `--fixed-cidr-v6` to provide globally routable IPv6 addresses. IPv6 forwarding will be enabled if not used with `--ip-forward=false`. This may collide with your host's current IPv6 settings. For more information please consult the documentation about "Advanced Networking - IPv6".

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

Windows client is `hyperv`. Linux only supports `default`.

-l, **--log-level**="*debug|info|warn|error|fatal*"
Set the logging level. Default is `info`.

--label="[]"
Set key=value labels to the daemon (displayed in `docker info`)

--live-restore=*false*
Enable live restore of running containers when the daemon starts so that they are not restarted. This option is applicable only for docker daemon running on Linux host.

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

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

--mtu=*0*
Set the containers network mtu. Default is `0`.

--max-concurrent-downloads=*3*
Set the max concurrent downloads for each pull. Default is `3`.

--max-concurrent-uploads=*5*
Set the max concurrent uploads for each push. Default is `5`.

--max-download-attempts=*5*
Set the max download attempts for each pull. Default is `5`.

--node-generic-resources=*[]*
Advertise user-defined resource. Default is `[]`.
Use this if your swarm cluster has some nodes with custom resources (e.g: NVIDIA GPU, SSD, ...) and you need your services to land on nodes advertising these resources.
Usage example: `--node-generic-resources "NVIDIA-GPU=UUID1" --node-generic-resources "NVIDIA-GPU=UUID2"`

-p, **--pidfile**=""
Path to use for daemon PID file. Default is `/var/run/docker.pid`

--raw-logs
Output daemon logs in full timestamp format without ANSI coloring. If this flag is not set, the daemon outputs condensed, colorized logs if a terminal is detected, or full ("raw") output otherwise.

--registry-mirror=*://*

Prepend a registry mirror to be used for image pulls. May be specified multiple times.

-s, **--storage-driver**=""
Force the Docker runtime to use a specific storage driver.

--seccomp-profile=""
Path to seccomp profile.

--selinux-enabled=*true|false*
Enable selinux support. Default is false.

--shutdown-timeout=*15*
Set the shutdown timeout value in seconds. Default is `15`.

--storage-opt=[]
Set storage driver options. See STORAGE DRIVER OPTIONS.

--swarm-default-advertise-addr=*IP|INTERFACE*
Set default address or interface for swarm to advertise as its externally-reachable address to other cluster members. This can be a hostname, an IP address, or an interface such as `eth0`. A port cannot be specified with this option.

--tls=*true|false*
Use TLS; implied by --tlsverify. Default is false.

--tlscacert=*~/.docker/ca.pem*
Trust certs signed only by this CA.

--tlscert=*~/.docker/cert.pem*
Path to TLS certificate file.

--tlskey=*~/.docker/key.pem*
Path to TLS key file.

--tlsverify=*true|false*
Use TLS and verify the remote (daemon: verify client, client: verify daemon). Default is false.

--userland-proxy=*true|false*
Rely on a userland proxy implementation for inter-container and outside-to-container loopback communications. Default is true.

--userland-proxy-path=""
Path to the userland proxy binary.

--userns-remap=*default|uid:gid|user:group|user|uid*
Enable user namespaces for containers on the daemon. Specifying "default"

will cause a new user and group to be created to handle UID and GID range remapping for the user namespace mappings used for contained processes. Specifying a user (or uid) and optionally a group (or gid) will cause the daemon to lookup the user and group's subordinate ID ranges for use as the user namespace mappings for contained processes.

STORAGE DRIVER OPTIONS

Docker uses storage backends (known as "graphdrivers" in the Docker internals) to create writable containers from images. Many of these backends use operating system level technologies and can be configured.

Specify options to the storage backend with **--storage-opt** flags. The backends that currently take options are *devicemapper*, *zfs* and *btrfs*. Options for *devicemapper* are prefixed with *dm*, options for *zfs* start with *zfs* and options for *btrfs* start with *btrfs*.

Specifically for devicemapper, the default is a "loopback" model which requires no pre-configuration, but is extremely inefficient. Do not use it in production.

To make the best use of Docker with the devicemapper backend, you must have a recent version of LVM. Use `lvm` to create a thin pool; for more information see `man lvmthin`. Then, use `--storage-opt dm.thinpooldev` to tell the Docker engine to use that pool for allocating images and container snapshots.

Devicemapper options

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 use:

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

dm.directlvm_device

As an alternative to manually creating a thin pool as above, Docker can automatically configure a block device for you.

Example use:

$ 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 base images 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.

Example use: `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
```

Example use: `dockerd --storage-opt dm.basesize=20G`

dm.fs
Specifies the filesystem type to use for the base device. The supported options are `ext4` and `xfs`. The default is `ext4`.

Example use: `dockerd --storage-opt dm.fs=xfs`

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

Example use: `dockerd --storage-opt "dm.mkfsarg=-O ^has_journal"`

dm.mountopt
Specifies extra mount options used when mounting the thin devices.

Example use: `dockerd --storage-opt dm.mountopt=nodiscard`

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 use: `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 destroy
container
```

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

`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.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 use: `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 use: `dockerd --storage-opt dm.loopmetadatasize=4G`

dm.datadev
(Deprecated, use `dm.thinpooldev`)

Specifies a custom blockdevice to use for data for a Docker-managed thin pool. It is better to use `dm.thinpooldev` - see the documentation for it above for discussion of the advantages.

dm.metadatadev
(Deprecated, use `dm.thinpooldev`)

Specifies a custom blockdevice to use for metadata for a Docker-managed thin pool. See `dm.datadev` for why this is deprecated.

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

Example use: `dockerd --storage-opt dm.blocksize=512K`

dm.blkdiscard
Enables or disables the use of `blkdiscard` when removing devicemapper devices. This is disabled by default due to the additional latency, but as a special case with loopback devices it will be enabled, in order to re-sparsify the loopback file on image/container removal.

Disabling this on loopback can lead to *much* faster container removal times, but it also prevents the space

used in `/var/lib/docker` directory from being returned to the system for other use when containers are removed.

Example use: `dockerd --storage-opt dm.blkdiscard=false`

dm.override_udev_sync_check

By default, the devicemapper backend attempts to synchronize with the `udev` device manager for the Linux kernel. This option allows disabling that synchronization, to continue even though the configuration may be buggy.

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⟩)

To allow the `docker` daemon to start, regardless of whether `udev` sync is `false`, set `dm.override_udev_sync_check` to true:

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

When this value is `true`, the driver 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.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 the 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 tge thin pool. You can also add more storage to the thin pool.

To add more space to an LVM (logical volume management) thin pool, just add more storage to the 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 use:: `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 use:

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

dm.libdm_log_level

Specifies the maxmimum 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.

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

Example use:

```
$ 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 use: `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 use: `docker daemon -s btrfs --storage-opt btrfs.min_space=10G`

CLUSTER STORE OPTIONS

The daemon uses libkv to advertise the node within the cluster. Some Key/Value backends support mutual TLS, and the client TLS settings used by the daemon can be configured using the **--cluster-store-opt** flag, specifying the paths to PEM encoded files.

kv.cacertfile

Specifies the path to a local file with PEM encoded CA certificates to trust

kv.certfile

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.

kv.keyfile

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.

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.

```
dockerd --authorization-plugin=plugin1 --authorization-plugin=plugin2,...
```

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 access authorization plugin ⟨https://docs.docker.com/engine/extend/plugins_authorization/⟩ section in the Docker extend section of this documentation.

RUNTIME EXECUTION OPTIONS

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 on cgroup v1 hosts, `systemd` is used on cgroup v2 hosts with systemd available.

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

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

Setting this option applies to all containers the daemon launches.

HISTORY

Sept 2015, Originally compiled by Shishir Mahajan shishir.mahajan@redhat.com ⟨mailto:shishir.mahajan@redhat.com⟩ based on docker.com source material and internal work.