dockerd

Table of contents

daemon

```
Usage:  dockerd [OPTIONS]

A self-sufficient runtime for containers.

Options:
      --add-runtime runtime                    Register an adc
      --allow-direct-routing                   Allow remote ac
      --authorization-plugin list              Authorization p
      --bip string                             IPv4 address fc
      --bip6 string                            IPv6 address fc
  -b, --bridge string                          Attach containe
      --bridge-accept-fwmark string            In bridge netwc
      --cdi-spec-dir list                      CDI specificati
      --cgroup-parent string                   Set parent cgrc
      --config-file string                     Daemon configur
```

```
      --containerd string                       containerd grpc
      --containerd-namespace string             Containerd name
      --containerd-plugins-namespace string     Containerd name
      --cpu-rt-period int                       Limit the CPU r
                                                parent cgroup f
      --cpu-rt-runtime int                      Limit the CPU r
                                                parent cgroup f
      --cri-containerd                          start containe
      --data-root string                        Root directory
  -D, --debug                                   Enable debug mo
      --default-address-pool pool-options       Default address
      --default-cgroupns-mode string            Default mode fo
      --default-gateway ip                      Default gateway
      --default-gateway-v6 ip                   Default gateway
      --default-ipc-mode string                 Default mode fo
      --default-network-opt mapmap              Default network
      --default-runtime string                  Default OCI run
      --default-shm-size bytes                  Default shm siz
      --default-ulimit ulimit                   Default ulimits
      --dns list                                DNS server to u
      --dns-opt list                            DNS options to
      --dns-search list                         DNS search doma
      --exec-opt list                           Runtime executi
      --exec-root string                        Root directory
      --experimental                            Enable experime
      --feature map                             Enable feature
      --firewall-backend string                 Firewall backer
      --fixed-cidr string                       IPv4 subnet for
      --fixed-cidr-v6 string                    IPv6 subnet for
```

```
  -G, --group string                     Group for the u
      --help                             Print usage
  -H, --host list                        Daemon socket(s
      --host-gateway-ip list             IP addresses th
                                         Defaults to the
      --http-proxy string                HTTP proxy URL
      --https-proxy string               HTTPS proxy URL
      --icc                              Enable inter-cc
      --init                             Run an init in
      --init-path string                 Path to the doc
      --insecure-registry list           Enable insecure
      --ip ip                            Host IP for por
      --ip-forward                       Enable IP forwa
      --ip-forward-no-drop               Do not set the
      --ip-masq                          Enable IP masqu
      --ip6tables                        Enable addition
      --iptables                         Enable addition
      --ipv6                             Enable IPv6 net
      --label list                       Set key=value l
      --live-restore                     Enable live res
      --log-driver string                Default driver
      --log-format string                Set the logging
  -l, --log-level string                 Set the logging
      --log-opt map                      Default log dri
      --max-concurrent-downloads int     Set the max cor
      --max-concurrent-uploads int       Set the max cor
      --max-download-attempts int        Set the max dow
      --metrics-addr string              Set default adc
      --mtu int                          Set the MTU for
```

```
      --network-control-plane-mtu int        Network Control
      --no-new-privileges                    Set no-new-priv
      --no-proxy string                      Comma-separated
      --node-generic-resource list           Advertise user-
  -p, --pidfile string                       Path to use for
      --raw-logs                             Full timestamps
      --registry-mirror list                 Preferred regis
      --rootless                             Enable rootless
      --seccomp-profile string               Path to seccomp
      --selinux-enabled                      Enable selinux
      --shutdown-timeout int                 Set the default
  -s, --storage-driver string                Storage driver
      --storage-opt list                     Storage driver
      --swarm-default-advertise-addr string  Set default adc
      --tls                                  Use TLS; implie
      --tlscacert string                     Trust certs sig
      --tlscert string                       Path to TLS cer
      --tlskey string                        Path to TLS key
      --tlsverify                            Use TLS and ver
      --userland-proxy                       Use userland pr
      --userland-proxy-path string           Path to the use
      --userns-remap string                  User/Group sett
      --validate                             Validate daemor
  -v, --version                              Print version i
```

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 --debug` or add `"debug": true` to the `daemon.json` file.

> **Note**
>
> **Enabling experimental features**
>
> Enable experimental features by starting `dockerd` with the `--experimental` flag or adding `"experimental": true` to the `daemon.json` file.

Environment variables

The following list of environment variables are supported by the `dockerd` daemon. Some of these environment variables are supported both by the Docker Daemon and the `docker` CLI. Refer to Environment variables to learn about environment variables supported by the `docker` CLI.

Variable	Description
`DOCKER_CERT_PATH`	Location of your authentication keys. This variable is used both by the `docker` CLI and the `dockerd` daemon.
`DOCKER_DRIVER`	The storage driver to use.
`DOCKER_RAMDISK`	If set this disables `pivot_root`.
`DOCKER_TLS_VERIFY`	When set Docker uses TLS and verifies the remote. This variable is used both by the `docker` CLI and the `dockerd` daemon.
`DOCKER_TMPDIR`	Location for temporary files created by the daemon.
`HTTP_PROXY`	Proxy URL for HTTP requests unless overridden by NoProxy. See the Go specification for details.
`HTTPS_PROXY`	Proxy URL for HTTPS requests unless overridden by NoProxy. See the Go specification for details.
`MOBY_DISABLE_PIGZ`	Disables the use of `unpigz` to decompress layers in parallel when pulling images, even if it is installed.
`NO_PROXY`	Comma-separated values specifying hosts that should be excluded from proxying. See the Go specification for details.

Examples

Proxy configuration

> **Note**
>
> Refer to the Docker Desktop manual if you are running Docker Desktop.

If you are behind an HTTP proxy server, for example in corporate settings, you may have to configure the Docker daemon to use the proxy server for operations such as pulling and pushing images. The daemon can be configured in three ways:

1. Using environment variables (`HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY`).
2. Using the `http-proxy`, `https-proxy`, and `no-proxy` fields in the daemon configuration file (Docker Engine version 23.0 or later).
3. Using the `--http-proxy`, `--https-proxy`, and `--no-proxy` command-line options. (Docker Engine version 23.0 or later).

The command-line and configuration file options take precedence over environment variables. Refer to control and configure Docker with systemd to set these environment variables on a host using `systemd`.

Daemon socket option

The Docker daemon can listen for Docker Engine 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. When using a TCP socket, the Docker daemon provides un-encrypted and un-authenticated direct access to the Docker daemon by default. You should secure the daemon either using the built in HTTPS encrypted socket, 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 TLS version 1.0 and higher is supported. Protocols SSLv3 and below are not supported for security reasons.

On systemd based systems, you can communicate with the daemon via systemd socket activation, with `dockerd -H fd://`. Using `fd://`

works for most setups, but you can also specify individual sockets: `dockerd -H fd://3`. If the specified socket activated files aren't found, the daemon exits. You can find examples of using systemd socket activation with Docker and systemd in the Docker source tree.

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

The example below runs the daemon listening on the default Unix socket, and on 2 specific IP addresses on this host:

```
$ sudo dockerd -H unix:///var/run/docker.sock -H tcp://192.16
```

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

The Docker client supports connecting to a remote daemon via SSH:

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

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 introduces security risks, as it may allow

> 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's not advisable on an open network.

With `-H` it's possible to make the Docker daemon to listen on a specific IP and port. By default, it listens 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 isn't recommended because someone could 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 defaults 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, defaults 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

```
$ sudo dockerd -H tcp://127.0.0.1:2375 -H unix:///var/run/doc
# 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: `overlay2`, `fuse-overlayfs`, `btrfs`, and `zfs`.

`overlay2` is the preferred storage driver for all currently supported Linux distributions, and is selected by default. Unless users have a strong reason to prefer another storage driver, `overlay2` should be used.

You can find out more about storage drivers and how to select one in Select a storage driver.

On Windows, the Docker daemon only supports the `windowsfilter` storage driver.

Options per storage driver

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

ZFS options

`zfs.fsname`

Specifies the ZFS filesystem that the daemon should use to create its datasets. By default, the ZFS filesystem in `/var/lib/docker` is used.

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** can't be smaller than **btrfs.min_space**.

Example

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

Overlay2 options

`overlay2.size`

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

Runtime options

The Docker daemon relies on a OCI compliant runtime (invoked via the `containerd` daemon) as its interface to the Linux kernel `namespaces`, `cgroups`, and `SELinux`.

Configure container runtimes

By default, the Docker daemon uses runc as a container runtime. You can configure the daemon to add additional runtimes.

containerd shims installed on `PATH` can be used directly, without the need to edit the daemon's configuration. For example, if you install the Kata Containers shim (`containerd-shim-kata-v2`) on `PATH`, then you

can select that runtime with `docker run` without having to edit the daemon's configuration:

```
$ docker run --runtime io.containerd.kata.v2
```

Container runtimes that don't implement containerd shims, or containerd shims installed outside of `PATH`, must be registered with the daemon, either via the configuration file or using the `--add-runtime` command line flag.

For examples on how to use other container runtimes, see Alternative container runtimes

Configure runtimes using `daemon.json`

To register and configure container runtimes using the daemon's configuration file, add the runtimes as entries under `runtimes`:

```
{
  "runtimes": {
    "<runtime>": {}
  }
}
```

The key of the entry (`<runtime>` in the previous example) represents the name of the runtime. This is the name that you reference when you run a container, using `docker run --runtime <runtime>`.

The runtime entry contains an object specifying the configuration for your runtime. The properties of the object depends on what kind of runtime you're looking to register:

- If the runtime implements its own containerd shim, the object shall contain a `runtimeType` field and an optional `options` field.

  ```
  {
    "runtimes": {
      "<runtime>": {
        "runtimeType": "<name-or-path>",
        "options": {}
      }
    }
  }
  ```

  See Configure shims.

- If the runtime is designed to be a drop-in replacement for runc, the object contains a `path` field, and an optional `runtimeArgs` field.

  ```
  {
    "runtimes": {
      "<runtime>": {
        "path": "/path/to/bin",
        "runtimeArgs": ["...args"]
      }
    }
  }
  ```

```
}
```

See Configure runc drop-in replacements.

After changing the runtimes configuration in the configuration file, you must reload or restart the daemon for changes to take effect:

```
$ sudo systemctl reload dockerd
```

Configure containerd shims

If the runtime that you want to register implements a containerd shim, or if you want to register a runtime which uses the runc shim, use the following format for the runtime entry:

```
{
  "runtimes": {
    "<runtime>": {
      "runtimeType": "<name-or-path>",
      "options": {}
    }
  }
}
```

`runtimeType` refers to either:

- A fully qualified name of a containerd shim.

The fully qualified name of a shim is the same as the `runtime_type` used to register the runtime in containerd's CRI configuration. For example, `io.containerd.runsc.v1`.

- The path of a containerd shim binary.

  This option is useful if you installed the containerd shim binary outside of `PATH`.

`options` is optional. It lets you specify the runtime configuration that you want to use for the shim. The configuration parameters that you can specify in `options` depends on the runtime you're registering. For most shims, the supported configuration options are `TypeUrl` and `ConfigPath`. For example:

```
{
  "runtimes": {
    "gvisor": {
      "runtimeType": "io.containerd.runsc.v1",
      "options": {
        "TypeUrl": "io.containerd.runsc.v1.options",
        "ConfigPath": "/etc/containerd/runsc.toml"
      }
    }
  }
}
```

You can configure multiple runtimes using the same runtimeType. For example:

```
{
  "runtimes": {
    "gvisor-foo": {
      "runtimeType": "io.containerd.runsc.v1",
      "options": {
        "TypeUrl": "io.containerd.runsc.v1.options",
        "ConfigPath": "/etc/containerd/runsc-foo.toml"
      }
    },
    "gvisor-bar": {
      "runtimeType": "io.containerd.runsc.v1",
      "options": {
        "TypeUrl": "io.containerd.runsc.v1.options",
        "ConfigPath": "/etc/containerd/runsc-bar.toml"
      }
    }
  }
}
```

The `options` field takes a special set of configuration parameters when used with `"runtimeType": "io.containerd.runc.v2"`. For more information about runc parameters, refer to the runc configuration section in CRI Plugin Config Guide.

Configure runc drop-in replacements

If the runtime that you want to register can act as a drop-in replacement for runc, you can register the runtime either using the daemon configuration file, or using the `--add-runtime` flag for the `dockerd` cli.

When you use the configuration file, the entry uses the following format:

```
{
  "runtimes": {
    "<runtime>": {
      "path": "/path/to/binary",
      "runtimeArgs": ["...args"]
    }
  }
}
```

Where `path` is either the absolute path to the runtime executable, or the name of an executable installed on `PATH`:

```
{
  "runtimes": {
    "runc": {
      "path": "runc"
    }
  }
}
```

And `runtimeArgs` lets you optionally pass additional arguments to the runtime. Entries with this format use the containerd runc shim to invoke a custom runtime binary.

When you use the `--add-runtime` CLI flag, use the following format:

```
$ sudo dockerd --add-runtime <runtime>=<path>
```

Defining runtime arguments via the command line is not supported.

For an example configuration for a runc drop-in replacment, see Alternative container runtimes > youki

Configure the default container runtime

You can specify either the name of a fully qualified containerd runtime shim, or the name of a registered runtime. You can specify the default runtime either using the daemon configuration file, or using the `--default-runtime` flag for the `dockerd` cli.

When you use the configuration file, the entry uses the following format:

```
{
  "default-runtime": "io.containerd.runsc.v1"
}
```

When you use the `--default-runtime` CLI flag, use the following format:

```
$ dockerd --default-runtime io.containerd.runsc.v1
```

Run containerd standalone

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 /run/containerd/containerd.sock
```

Configure cgroup driver

You can configure how the runtime should manage container cgroups, using the `--exec-opt native.cgroupdriver` CLI flag.

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.

Configure container isolation technology (Windows)

For Windows containers, you can specify the default container isolation technology to use, using the `--exec-opt isolation` flag.

The following example makes `hyperv` the default isolation technology:

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

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
```

Insecure registries

In this section, "registry" refers to a private registry, and `myregistry:5000` is a placeholder example of a private registry.

Docker considers a private registry either secure or insecure. 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 wasn't found under `/etc/docker/certs.d/myregistry:5000/`, or if the certificate verification failed (i.e., wrong CA).

By default, Docker assumes all registries to be secure, except for local registries. Communicating with an insecure registry isn't 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 isn't marked as insecure, `docker pull`, `docker push`, and `docker search` result in error messages, 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 isn't 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

Operations against registries supporting only the legacy v1 protocol are no longer supported. Specifically, the daemon doesn't attempt to push, pull or sign in to v1 registries. The exception to this is `search` which can still be performed on v1 registries.

Running a Docker daemon behind an HTTPS_PROXY

When running inside a LAN that uses an `HTTPS` proxy, the proxy's certificates replace Docker Hub's certificates. These certificates must 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 only adds the proxy and authentication to the Docker daemon's requests. To use the proxy when building images and running containers, see Configure Docker to use a proxy server

Default `ulimit` settings

The `--default-ulimit` flag lets you set the default `ulimit` options to use for all containers. It takes the same options as `--ulimit` for `docker run`. If these defaults aren't set, `ulimit` settings are inherited from the Docker daemon. Any `--ulimit` options passed to `docker run` override the daemon 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, see `docker run` reference.

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 --authorizatior
```

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, refer to the authorization plugin section.

Daemon user namespace options

The Linux kernel user namespace support 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. One of the most important

security improvements is that, by default, container processes running as the `root` user have expected administrative privileges it expects (with some restrictions) inside the container, but are effectively 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.

Configure host gateway IP

The Docker daemon supports a special `host-gateway` value for the `--add-host` flag for the `docker run` and `docker build` commands. This value resolves to addresses on the host, so that containers can connect to services running on the host.

By default, `host-gateway` resolves to the IPv4 address of the default bridge, and its IPv6 address if it has one.

You can configure this to resolve to a different IP using the `--host-gateway-ip` flag for the dockerd command line interface, or the `host-gateway-ip` key in the daemon configuration file.

To supply both IPv4 and IPv6 addresses on the command line, use two `--host-gateway-ip` options.

To supply addresses in the daemon configuration file, use `"host-gateway-ips"` with a JSON array, as shown below. For compatibility

with older versions of the daemon, a single IP address can also be specified as a JSON string in option `"host-gateway-ip"`.

```
$ cat > /etc/docker/daemon.json
{ "host-gateway-ips": ["192.0.2.1", "2001:db8::1111"]}
$ sudo systemctl restart docker
$ docker run -it --add-host host.docker.internal:host-gateway
  busybox ping host.docker.internal
PING host.docker.internal (192.0.2.1): 56 data bytes
$ docker run -it --add-host host.docker.internal:host-gateway
  busybox ping -6 host.docker.internal
PING host.docker.internal (2001:db8::1111): 56 data bytes
```

Configure CDI devices

Container Device Interface (CDI) is a standardized mechanism for container runtimes to create containers which are able to interact with third party devices.

CDI is currently only supported for Linux containers and is enabled by default since Docker Engine 28.3.0.

The Docker daemon supports running containers with CDI devices if the requested device specifications are available on the filesystem of the daemon.

The default specification directories are:

- `/etc/cdi/` for static CDI Specs
- `/var/run/cdi` for generated CDI Specs

Set custom locations

To set custom locations for CDI specifications, use the `cdi-spec-dirs` option in the `daemon.json` configuration file, or the `--cdi-spec-dir` flag for the `dockerd` CLI:

```
{
  "cdi-spec-dirs": ["/etc/cdi/", "/var/run/cdi"]
}
```

You can view the configured CDI specification directories using the `docker info` command.

Disable CDI devices

The feature in enabled by default. To disable it, use the `cdi` options in the `daemon.json` file:

```
"features": {
  "cdi": false
},
```

To check the status of the CDI devices, run `docker info`.

Daemon logging format

The `--log-format` option or "log-format" option in the daemon configuration file lets you set the format for logs produced by the daemon. The logging format should only be configured either through the `--log-format` command line option or through the "log-format" field in the configuration file; using both the command-line option and the "log-format" field in the configuration file produces an error. If this option is not set, the default is "text".

The following example configures the daemon through the `--log-format` command line option to use `json` formatted logs;

```
$ dockerd --log-format=json
# ...
{"level":"info","msg":"API listen on /var/run/docker.sock","t
```

The following example shows a `daemon.json` configuration file with the "log-format" set;

```
{
  "log-format": "json"
}
```

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 soft links 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:

```
$ export DOCKER_TMPDIR=/mnt/disk2/tmp
$ sudo -E dockerd --data-root /var/lib/docker -H unix://
```

Default cgroup parent

The `--cgroup-parent` option lets you set the default cgroup parent for containers. If this option isn't set, it defaults to `/docker` for the cgroupfs driver, and `system.slice` for the 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 format.

Port `9323` is the default port associated with Docker metrics 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, see Collect Docker metrics with Prometheus.

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"
  ]
}
```

Enable feature in the daemon (--feature)

The `--feature` option lets you enable or disable a feature in the daemon. This option corresponds with the "features" field in the daemon.json configuration file. Features should only be configured either through the `--feature` command line option or through the "features" field in the configuration file; using both the command-line option and the "features" field in the configuration file produces an error. The feature option can be specified multiple times to configure

multiple features. The `--feature` option accepts a name and optional boolean value. When omitting the value, the default is `true`.

The following example runs the daemon with the `cdi` and `containerd-snapshotter` features enabled. The `cdi` option is provided with a value;

```
$ dockerd --feature cdi=true --feature containerd-snapshotter
```

The following example is the equivalent using the `daemon.json` configuration file;

```
{
  "features": {
    "cdi": true,
    "containerd-snapshotter": true
  }
}
```

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 using flags. The Docker daemon fails to start if an option is duplicated between the file and the flags, regardless of their value. This is intentional, and avoids 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.

The `--validate` option allows to validate a configuration file without starting the Docker daemon. A non-zero exit code is returned for invalid configuration files.

```
$ dockerd --validate --config-file=/tmp/valid-config.json
configuration OK

$ echo $?
0

$ dockerd --validate --config-file /tmp/invalid-config.json
unable to configure the Docker daemon with file /tmp/invalid-

$ echo $?
1
```

On Linux

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

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

```
{
  "allow-direct-routing": false,
  "authorization-plugins": [],
  "bip": "",
  "bip6": "",
  "bridge": "",
  "bridge-accept-fwmark": "",
  "builder": {
    "gc": {
      "enabled": true,
      "defaultReservedSpace": "10GB",
      "policy": [
        { "maxUsedSpace": "512MB", "keepDuration": "48h", "fi
        { "reservedSpace": "10GB", "maxUsedSpace": "100GB", '
        { "reservedSpace": "50GB", "minFreeSpace": "20GB", "m
      ]
    }
  },
  "cgroup-parent": "",
  "containerd": "/run/containerd/containerd.sock",
  "containerd-namespace": "docker",
  "containerd-plugins-namespace": "docker-plugins",
```

```
  "data-root": "",
  "debug": true,
  "default-address-pools": [
    {
      "base": "172.30.0.0/16",
      "size": 24
    },
    {
      "base": "172.31.0.0/16",
      "size": 24
    }
  ],
  "default-cgroupns-mode": "private",
  "default-gateway": "",
  "default-gateway-v6": "",
  "default-network-opts": {},
  "default-runtime": "runc",
  "default-shm-size": "64M",
  "default-ulimits": {
    "nofile": {
      "Hard": 64000,
      "Name": "nofile",
      "Soft": 64000
    }
  },
  "dns": [],
  "dns-opts": [],
  "dns-search": [],
  "exec-opts": [],
```

```
  "exec-root": "",
  "experimental": false,
  "features": {
    "cdi": true,
    "containerd-snapshotter": true
  },
  "firewall-backend": "",
  "fixed-cidr": "",
  "fixed-cidr-v6": "",
  "group": "",
  "host-gateway-ip": "",
  "hosts": [],
  "proxies": {
    "http-proxy": "http://proxy.example.com:80",
    "https-proxy": "https://proxy.example.com:443",
    "no-proxy": "*.test.example.com,.example.org"
  },
  "icc": false,
  "init": false,
  "init-path": "/usr/libexec/docker-init",
  "insecure-registries": [],
  "ip": "0.0.0.0",
  "ip-forward": false,
  "ip-masq": false,
  "iptables": false,
  "ip6tables": false,
  "ipv6": false,
  "labels": [],
  "live-restore": true,
```

```
  "log-driver": "json-file",
  "log-format": "text",
  "log-level": "",
  "log-opts": {
    "cache-disabled": "false",
    "cache-max-file": "5",
    "cache-max-size": "20m",
    "cache-compress": "true",
    "env": "os,customer",
    "labels": "somelabel",
    "max-file": "5",
    "max-size": "10m"
  },
  "max-concurrent-downloads": 3,
  "max-concurrent-uploads": 5,
  "max-download-attempts": 5,
  "mtu": 0,
  "no-new-privileges": false,
  "node-generic-resources": [
    "NVIDIA-GPU=UUID1",
    "NVIDIA-GPU=UUID2"
  ],
  "pidfile": "",
  "raw-logs": false,
  "registry-mirrors": [],
  "runtimes": {
    "cc-runtime": {
      "path": "/usr/bin/cc-runtime"
    },
```

```
    "custom": {
      "path": "/usr/local/bin/my-runc-replacement",
      "runtimeArgs": [
        "--debug"
      ]
    }
  },
  "seccomp-profile": "",
  "selinux-enabled": false,
  "shutdown-timeout": 15,
  "storage-driver": "",
  "storage-opts": [],
  "swarm-default-advertise-addr": "",
  "tls": true,
  "tlscacert": "",
  "tlscert": "",
  "tlskey": "",
  "tlsverify": true,
  "userland-proxy": false,
  "userland-proxy-path": "/usr/libexec/docker-proxy",
  "userns-remap": ""
}
```

Note

You can't 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 can't use the

`hosts` key in `daemon.json` to add listening addresses. See custom Docker daemon options for an example on how to configure the daemon using systemd drop-in files.

On Windows

The default location of the configuration file on Windows is `%programdata%\docker\config\daemon.json`. Use the `--config-file` flag to specify a non-default location.

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

```
{
  "authorization-plugins": [],
  "bridge": "",
  "containerd": "\\\\.\\pipe\\containerd-containerd",
  "containerd-namespace": "docker",
  "containerd-plugins-namespace": "docker-plugins",
  "data-root": "",
  "debug": true,
  "default-network-opts": {},
  "default-runtime": "",
  "default-ulimits": {},
  "dns": [],
  "dns-opts": [],
  "dns-search": [],
```

```
  "exec-opts": [],
  "experimental": false,
  "features": {},
  "fixed-cidr": "",
  "group": "",
  "host-gateway-ip": "",
  "hosts": [],
  "insecure-registries": [],
  "labels": [],
  "log-driver": "",
  "log-format": "text",
  "log-level": "",
  "max-concurrent-downloads": 3,
  "max-concurrent-uploads": 5,
  "max-download-attempts": 5,
  "mtu": 0,
  "pidfile": "",
  "raw-logs": false,
  "registry-mirrors": [],
  "shutdown-timeout": 15,
  "storage-driver": "",
  "storage-opts": [],
  "swarm-default-advertise-addr": "",
  "tlscacert": "",
  "tlscert": "",
  "tlskey": "",
  "tlsverify": true
}
```

The `default-runtime` option is by default unset, in which case dockerd automatically detects the runtime. This detection is based on if the `containerd` flag is set.

Accepted values:

- `com.docker.hcsshim.v1` - This is the built-in runtime that Docker has used since Windows supported was first added and uses the v1 HCS API's in Windows.
- `io.containerd.runhcs.v1` - This is uses the containerd `runhcs` shim to run the container and uses the v2 HCS API's in Windows.

Feature options

The optional field `features` in `daemon.json` lets you enable or disable specific daemon features.

```
{
  "features": {
    "some-feature": true,
    "some-disabled-feature-enabled-by-default": false
  }
}
```

The list of feature options include:

- `containerd-snapshotter`: when set to `true`, the daemon uses containerd snapshotters instead of the classic storage drivers for

storing image and container data. For more information, see containerd storage.

- `windows-dns-proxy`: when set to `true`, the daemon's internal DNS resolver will forward requests to external servers. Without this, most applications running in the container will still be able to use secondary DNS servers configured in the container itself, but `nslookup` won't be able to resolve external names. The current default is `false`, it will change to `true` in a future release. This option is only allowed on Windows.

  > ⚠ **Warning**
  >
  > The `windows-dns-proxy` feature flag will be removed in a future release.

Configuration reload behavior

Some options can be reconfigured when the daemon is running without requiring to restart the process. The daemon uses the `SIGHUP` signal in Linux to reload, and a global event in Windows with the key `Global\docker-daemon-config-$PID`. You can modify the options in the configuration file, but the daemon still checks for conflicting settings with the specified CLI 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:

Option	Description
`debug`	Toggles debug mode of the daemon.
`labels`	Replaces the daemon labels with a new set of labels.
`live-restore`	Toggles live restore.
`max-concurrent-downloads`	Configures the max concurrent downloads for each pull.
`max-concurrent-uploads`	Configures the max concurrent uploads for each push.
`max-download-attempts`	Configures the max download attempts for each pull.
`default-runtime`	Configures the runtime to be used if not is specified at container creation.
`runtimes`	Configures the list of available OCI runtimes that can be used to run containers.
`authorization-plugin`	Specifies the authorization plugins to use.
`insecure-registries`	Specifies a list of registries that the daemon should consider insecure.
`registry-mirrors`	Specifies a list of registry mirrors.
`shutdown-timeout`	Configures the daemon's existing configuration timeout with a new timeout for shutting down all

Option	Description
	containers.
`features`	Enables or disables specific features.

Run multiple daemons

> **Note**
>
> Running multiple daemons on a single host is considered experimental. You may encounter unsolved problems, and things may not work as expected in some cases.

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 doesn't 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.

The following daemon options must be configured for each daemon:

```
-b, --bridge=                          Attach containers to a
--exec-root=/var/run/docker            Root of the Docker exe
--data-root=/var/lib/docker            Root of persisted Dock
-p, --pidfile=/var/run/docker.pid      Path to use for daemor
```

```
-H, --host=[]                                Daemon socket(s) to cc
--iptables=true                              Enable addition of ipt
--config-file=/etc/docker/daemon.json  Daemon configuration f
--tlscacert="~/.docker/ca.pem"               Trust certs signed onl
--tlscert="~/.docker/cert.pem"               Path to TLS certificat
--tlskey="~/.docker/key.pem"                 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 important that you understand the meaning of these 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 aren't using the default, you must create and configure the bridge manually, or 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 listens 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 also doesn't add IP masquerading rules, even if you set `--ip-masq` to `true`. Without IP masquerading rules, Docker containers can't 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
```

Default network options

The `default-network-opts` key in the `daemon.json` configuration file, and the equivalent `--default-network-opt` CLI flag, let you specify default values for driver network driver options for new networks.

The following example shows how to configure options for the `bridge` driver using the `daemon.json` file.

```
{
  "default-network-opts": {
    "bridge": {
      "com.docker.network.bridge.host_binding_ipv4": "127.0.0
      "com.docker.network.driver.mtu": "1234"
    }
  }
}
```

This example uses the `bridge` network driver. Refer to the bridge network driver page for an overview of available driver options.

After changing the configuration and restarting the daemon, new networks that you create use these option configurations as defaults.

```
$ docker network create mynet
$ docker network inspect mynet --format "{{json .Options}}"
{"com.docker.network.bridge.host_binding_ipv4":"127.0.0.1","c
```

Note that changing this daemon configuration doesn't affect pre-existing networks.

Using the `--default-network-opt` CLI flag is useful for testing and debugging purposes, but you should prefer using the `daemon.json` file for persistent daemon configuration. The CLI flag expects a value with the following format: `driver=opt=value`, for example:

```
$ sudo dockerd \
  --default-network-opt bridge=com.docker.network.bridge.host
  --default-network-opt bridge=com.docker.network.driver.mtu=
```