Configure and troubleshoot the Docker daemon

Estimated reading time: 11 minutes

After successfully installing and starting Docker, the `dockerd` daemon runs with its default configuration. This topic shows how to customize the configuration, start the daemon manually, and troubleshoot and debug the daemon if you run into issues.

Start the daemon using operating system utilities

On a typical installation the Docker daemon is started by a system utility, not manually by a user. This makes it easier to automatically start Docker when the machine reboots.

The command to start Docker depends on your operating system. Check the correct page under Install Docker (https://docs.docker.com/install/). To configure Docker to start automatically at system boot, see Configure Docker to start on boot (https://docs.docker.com/install/linux/linux-postinstall/#configure-docker-to-start-on-boot).

Start the daemon manually

If you don't want to use a system utility to manage the Docker daemon, or just want to test things out, you can manually run it using the `dockerd` command. You may need to use `sudo`, depending on your operating system configuration.

When you start Docker this way, it runs in the foreground and sends its logs directly to your terminal.

```
$ dockerd

INFO[0000] +job init_networkdriver()
INFO[0000] +job serveapi(unix:///var/run/docker.sock)
INFO[0000] Listening for HTTP on unix (/var/run/docker.sock)
```

To stop Docker when you have started it manually, issue a `Ctrl+C` in your terminal.

Configure the Docker daemon

There are two ways to configure the Docker daemon:

- Use a JSON configuration file. This is the preferred option, since it keeps all configurations in a single place.
- Use flags when starting `dockerd`.

You can use both of these options together as long as you don't specify the same option both as a flag and in the JSON file. If that happens, the Docker daemon won't start and prints an error message.

To configure the Docker daemon using a JSON file, create a file at `/etc/docker/daemon.json` on Linux systems, or `C:\ProgramData\docker\config\daemon.json` on Windows. On MacOS go to the whale in the taskbar > Preferences > Daemon > Advanced.

Here's what the configuration file looks like:

```
{
  "debug": true,
  "tls": true,
  "tlscert": "/var/docker/server.pem",
  "tlskey": "/var/docker/serverkey.pem",
  "hosts": ["tcp://192.168.59.3:2376"]
}
```

With this configuration the Docker daemon runs in debug mode, uses TLS, and listens for traffic routed to `192.168.59.3` on port `2376`. You can learn what configuration options are available in the dockerd reference docs (https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file)

You can also start the Docker daemon manually and configure it using flags. This can be useful for troubleshooting problems.

Here's an example of how to manually start the Docker daemon, using the same configurations as above:

```
dockerd --debug \
  --tls=true \
  --tlscert=/var/docker/server.pem \
  --tlskey=/var/docker/serverkey.pem \
  --host tcp://192.168.59.3:2376
```

You can learn what configuration options are available in the dockerd reference docs (https://docs.docker.com/engine/reference/commandline/dockerd/), or by running:

```
dockerd --help
```

Many specific configuration options are discussed throughout the Docker documentation. Some places to go next include:

- Automatically start containers (https://docs.docker.com/engine/admin/host_integration/)
- Limit a container's resources (https://docs.docker.com/engine/admin/resource_constraints/)
- Configure storage drivers (https://docs.docker.com/engine/userguide/storagedriver/)
- Container security (https://docs.docker.com/engine/security/)

Docker daemon directory

The Docker daemon persists all data in a single directory. This tracks everything related to Docker, including containers, images, volumes, service definition, and secrets.

By default this directory is:

- `/var/lib/docker` on Linux.
- `C:\ProgramData\docker` on Windows.

You can configure the Docker daemon to use a different directory, using the `data-root` configuration option.

Since the state of a Docker daemon is kept on this directory, make sure you use a dedicated directory for each daemon. If two daemons share the same directory, for example, an NFS share, you are going to experience errors that are difficult to troubleshoot.

Troubleshoot the daemon

You can enable debugging on the daemon to learn about the runtime activity of the daemon and to aid in troubleshooting. If the daemon is completely non-responsive, you can also force a full stack trace (/config/daemon/#force-a-full-stack-trace-to-be-logged) of all threads to be added to the daemon log by sending the `SIGUSR` signal to the Docker daemon.

Troubleshoot conflicts between the `daemon.json` and startup scripts

If you use a `daemon.json` file and also pass options to the `dockerd` command manually or using start-up scripts, and these options conflict, Docker fails to start with an error such as:

```
unable to configure the Docker daemon with file /etc/docker/daemon.json:
the following directives are specified both as a flag and in the configuration
file: hosts: (from flag: [unix:///var/run/docker.sock], from file: [tcp://127.0.0.1:2376])
```

If you see an error similar to this one and you are starting the daemon manually with flags, you may need to adjust your flags or the `daemon.json` to remove the conflict.

> **Note**: If you see this specific error, continue to the next section (/config/daemon/#use-the-hosts-key-in-daemon-json-with-systemd) for a workaround.

If you are starting Docker using your operating system's init scripts, you may need to override the defaults in these scripts in ways that are specific to the operating system.

USE THE HOSTS KEY IN DAEMON.JSON WITH SYSTEMD

One notable example of a configuration conflict that is difficult to troubleshoot is when you want to specify a different daemon address from the default. Docker listens on a socket by default. On Debian and Ubuntu systems using `systemd`, this means that a host flag `-H` is always used when starting `dockerd`. If you specify a `hosts` entry in the `daemon.json`, this causes a configuration conflict (as in the above message) and Docker fails to start.

To work around this problem, create a new file `/etc/systemd/system/docker.service.d/docker.conf` with the following contents, to remove the `-H` argument that is used when starting the daemon by default.

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd
```

There are other times when you might need to configure `systemd` with Docker, such as configuring a HTTP or HTTPS proxy (https://docs.docker.com/engine/admin/systemd/#httphttps-proxy).

> **Note**: If you override this option and then do not specify a `hosts` entry in the `daemon.json` or a `-H` flag when starting Docker manually, Docker fails to start.

Run `sudo systemctl daemon-reload` before attempting to start Docker. If Docker starts successfully, it is now listening on the IP address specified in the `hosts` key of the `daemon.json` instead of a socket.

> **Important**: Setting `hosts` in the `daemon.json` is not supported on Docker Desktop for Windows or Docker Desktop for Mac.

Out Of Memory Exceptions (OOME)

If your containers attempt to use more memory than the system has available, you may experience an Out Of Memory Exception (OOME) and a container, or the Docker daemon, might be killed by the kernel OOM killer. To prevent this from happening, ensure that your application runs on hosts with adequate memory and see Understand the risks of running out of memory (https://docs.docker.com/engine/admin/resource_constraints/#understand-the-risks-of-running-out-of-memory).

Read the logs

The daemon logs may help you diagnose problems. The logs may be saved in one of a few locations, depending on the operating system configuration and the logging subsystem used:

Operating system	Location
RHEL, Oracle Linux	`/var/log/messages`
Debian	`/var/log/daemon.log`
Ubuntu 16.04+, CentOS	Use the command `journalctl -u docker.service`
Ubuntu 14.10-	`/var/log/upstart/docker.log`
macOS (Docker 18.01+)	`~/Library/Containers/com.docker.docker/Data/vms/0/console-ring`
macOS (Docker <18.01)	`~/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/console-ring`
Windows	`AppData\Local`

Enable debugging

There are two ways to enable debugging. The recommended approach is to set the `debug` key to `true` in the `daemon.json` file. This method works for every Docker platform.

1. Edit the `daemon.json` file, which is usually located in `/etc/docker/`. You may need to create this file, if it does not yet exist. On macOS or Windows, do not edit the file directly. Instead, go to **Preferences** / **Daemon** / **Advanced**.

2. If the file is empty, add the following:

```
{
  "debug": true
}
```

If the file already contains JSON, just add the key `"debug": true`, being careful to add a comma to the end of the line if it is not the last line before the closing bracket. Also verify that if the `log-level` key is set, it is set to either `info` or `debug`. `info` is the default, and possible values are `debug`, `info`, `warn`, `error`, `fatal`.

3. Send a `HUP` signal to the daemon to cause it to reload its configuration. On Linux hosts, use the following command.

   ```
   $ sudo kill -SIGHUP $(pidof dockerd)
   ```

   On Windows hosts, restart Docker.

Instead of following this procedure, you can also stop the Docker daemon and restart it manually with the debug flag `-D`. However, this may result in Docker restarting with a different environment than the one the hosts' startup scripts create, and this may make debugging more difficult.

Force a stack trace to be logged

If the daemon is unresponsive, you can force a full stack trace to be logged by sending a `SIGUSR1` signal to the daemon.

- **Linux**:

  ```
  $ sudo kill -SIGUSR1 $(pidof dockerd)
  ```

- **Windows Server**:

  Download docker-signal (https://github.com/jhowardmsft/docker-signal).

  Get the process ID of dockerd `Get-Process dockerd`.

  Run the executable with the flag `--pid=<PID of daemon>`.

This forces a stack trace to be logged but does not stop the daemon. Daemon logs show the stack trace or the path to a file containing the stack trace if it was logged to a file.

The daemon continues operating after handling the `SIGUSR1` signal and dumping the stack traces to the log. The stack traces can be used to determine the state of all goroutines and threads within the daemon.

View stack traces

The Docker daemon log can be viewed by using one of the following methods:

- By running `journalctl -u docker.service` on Linux systems using `systemctl`
- `/var/log/messages`, `/var/log/daemon.log`, or `/var/log/docker.log` on older Linux systems
- By running `Get-EventLog -LogName Application -Source Docker -After (Get-Date).AddMinutes(-5) | Sort-Object Time |` on Docker EE for Windows Server

> **Note**: It is not possible to manually generate a stack trace on Docker Desktop for Mac or Docker Desktop for Windows. However, you can click the Docker taskbar icon and choose **Diagnose and feedback** to send information to Docker if you run into issues.

Look in the Docker logs for a message like the following:

```
...goroutine stacks written to /var/run/docker/goroutine-stacks-2017-06-02T193336z.log
...daemon datastructure dump written to /var/run/docker/daemon-data-2017-06-02T193336z.log
```

The locations where Docker saves these stack traces and dumps depends on your operating system and configuration. You can sometimes get useful diagnostic information straight from the stack traces and dumps. Otherwise, you can provide this information to Docker for help diagnosing the problem.

Check whether Docker is running

The operating-system independent way to check whether Docker is running is to ask Docker, using the `docker info` command.

You can also use operating system utilities, such as `sudo systemctl is-active docker` or `sudo status docker` or `sudo service docker status` , or checking the service status using Windows utilities.

Finally, you can check in the process list for the `dockerd` process, using commands like `ps` or `top` .

docker (https://docs.docker.com/glossary/?term=docker), daemon (https://docs.docker.com/glossary/?term=daemon), configuration (https://docs.docker.com/glossary/?term=configuration), troubleshooting (https://docs.docker.com/glossary/?term=troubleshooting)