Post-installation steps for Linux

Estimated reading time: 16 minutes

This section contains optional procedures for configuring Linux hosts to work better with Docker.

Manage Docker as a non-root user

The Docker daemon binds to a Unix socket instead of a TCP port. By default that Unix socket is owned by the user `root` and other users can only access it using `sudo`. The Docker daemon always runs as the `root` user.

If you don't want to preface the `docker` command with `sudo`, create a Unix group called `docker` and add users to it. When the Docker daemon starts, it creates a Unix socket accessible by members of the `docker` group.

Warning

The `docker` group grants privileges equivalent to the `root` user. For details on how this impacts security in your system, see *Docker Daemon Attack Surface* (/engine/security/security/#docker-daemon-attack-surface).

Note:

To run Docker without root privileges, see Run the Docker daemon as a non-root user (Rootless mode) (/engine/security/rootless/).

Rootless mode is currently available as an experimental feature.

To create the `docker` group and add your user:

1. Create the `docker` group.

```
$ sudo groupadd docker
```

2. Add your user to the `docker` group.

```
$ sudo usermod -aG docker $USER
```

3. Log out and log back in so that your group membership is re-evaluated.

   If testing on a virtual machine, it may be necessary to restart the virtual machine for changes to take effect.

On a desktop Linux environment such as X Windows, log out of your session completely and then log back in.

On Linux, you can also run the following command to activate the changes to groups:

```
$ newgrp docker
```

4. Verify that you can run `docker` commands without `sudo` .

```
$ docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

If you initially ran Docker CLI commands using `sudo` before adding your user to the `docker` group, you may see the following error, which indicates that your `~/.docker/` directory was created with incorrect permissions due to the `sudo` commands.

```
WARNING: Error loading config file: /home/user/.docker/config.json -
stat /home/user/.docker/config.json: permission denied
```

To fix this problem, either remove the `~/.docker/` directory (it is recreated automatically, but any custom settings are lost), or change its ownership and permissions using the following commands:

```
$ sudo chown "$USER":"$USER" /home/"$USER"/.docker -R
$ sudo chmod g+rwx "$HOME/.docker" -R
```

Configure Docker to start on boot

Most current Linux distributions (RHEL, CentOS, Fedora, Ubuntu 16.04 and higher) use `systemd` to manage which services start when the system boots. Ubuntu 14.10 and below use `upstart` .

`systemd`

```
$ sudo systemctl enable docker
```

To disable this behavior, use `disable` instead.

```
$ sudo systemctl disable docker
```

If you need to add an HTTP Proxy, set a different directory or partition for the Docker runtime files, or make other customizations, see customize your systemd Docker daemon options (/config/daemon /systemd/).

`upstart`

Docker is automatically configured to start on boot using `upstart`. To disable this behavior, use the following command:

```
$ echo manual | sudo tee /etc/init/docker.override
```

`chkconfig`

```
$ sudo chkconfig docker on
```

Use a different storage engine

For information about the different storage engines, see Storage drivers (/storage/storagedriver/). The default storage engine and the list of supported storage engines depend on your host's Linux distribution and available kernel drivers.

Configure default logging driver

Docker provides the capability (/config/containers/logging/) to collect and view log data from all containers running on a host via a series of logging drivers. The default logging driver, `json-file`, writes log data to JSON-formatted files on the host filesystem. Over time, these log files expand in size, leading to potential exhaustion of disk resources. To alleviate such issues, either configure an alternative logging driver such as Splunk or Syslog, or set up log rotation (/config/containers/logging/configure/#configure-the-default-logging-driver) for the default driver. If you configure an alternative logging driver, see Use `docker logs` to read container logs for remote logging drivers (/config/containers/logging/dual-logging/).

Configure where the Docker daemon listens for connections

By default, the Docker daemon listens for connections on a UNIX socket to accept requests from local clients. It is possible to allow Docker to accept requests from remote hosts by configuring it to listen on an IP address and port as well as the UNIX socket. For more detailed information on this configuration option take a look at "Bind Docker to another host/port or a unix socket" section of the Docker CLI Reference (/engine/reference/commandline/dockerd/) article.

Secure your connection

Before configuring Docker to accept connections from remote hosts it is critically important that you understand the security implications of opening docker to the network. If steps are not taken to secure the connection, it is possible for remote non-root users to gain root access on the host. For more information on how to use TLS certificates to secure this connection, check this article on how to protect the Docker daemon socket (/engine/security/https/).

Configuring Docker to accept remote connections can be done with the `docker.service` systemd unit file for Linux distributions using systemd, such as recent versions of RedHat, CentOS, Ubuntu and SLES, or with the `daemon.json` file which is recommended for Linux distributions that do not use systemd.

systemd vs daemon.json

Configuring Docker to listen for connections using both the `systemd` unit file and the `daemon.json` file causes a conflict that prevents Docker from starting.

Configuring remote access with `systemd` unit file

1. Use the command `sudo systemctl edit docker.service` to open an override file for `docker.service` in a text editor.
2. Add or modify the following lines, substituting your own values.

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://127.0.0.1:2375
```

3. Save the file.
4. Reload the `systemctl` configuration.

```
 $ sudo systemctl daemon-reload
```

5. Restart Docker.

```
$ sudo systemctl restart docker.service
```

6. Check to see whether the change was honored by reviewing the output of `netstat` to confirm `dockerd` is listening on the configured port.

```
$ sudo netstat -lntp | grep dockerd
tcp        0      0 127.0.0.1:2375          0.0.0.0:*               LISTEN      3
```

Configuring remote access with `daemon.json`

1. Set the `hosts` array in the `/etc/docker/daemon.json` to connect to the UNIX socket and an IP address, as follows:

```
{
"hosts": ["unix:///var/run/docker.sock", "tcp://127.0.0.1:2375"]
}
```

2. Restart Docker.

3. Check to see whether the change was honored by reviewing the output of `netstat` to confirm `dockerd` is listening on the configured port.

```
$ sudo netstat -lntp | grep dockerd
tcp        0      0 127.0.0.1:2375          0.0.0.0:*               LISTEN      3
```

Enable IPv6 on the Docker daemon

To enable IPv6 on the Docker daemon, see Enable IPv6 support (/config/daemon/ipv6/).

Troubleshooting

Kernel compatibility

Docker cannot run correctly if your kernel is older than version 3.10 or if it is missing some modules. To check kernel compatibility, you can download and run the `check-config.sh` (https://raw.githubusercontent.com/docker/docker/master/contrib/check-config.sh) script.

```
$ curl https://raw.githubusercontent.com/docker/docker/master/contrib/check-config.sh

$ bash ./check-config.sh
```

The script only works on Linux, not macOS.

`Cannot connect to the Docker daemon`

If you see an error such as the following, your Docker client may be configured to connect to a Docker daemon on a different host, and that host may not be reachable.

```
Cannot connect to the Docker daemon. Is 'docker daemon' running on this host?
```

To see which host your client is configured to connect to, check the value of the `DOCKER_HOST` variable in your environment.

```
$ env | grep DOCKER_HOST
```

If this command returns a value, the Docker client is set to connect to a Docker daemon running on that host. If it is unset, the Docker client is set to connect to the Docker daemon running on the local host. If it is set in error, use the following command to unset it:

```
$ unset DOCKER_HOST
```

You may need to edit your environment in files such as `~/.bashrc` or `~/.profile` to prevent the `DOCKER_HOST` variable from being set erroneously.

If `DOCKER_HOST` is set as intended, verify that the Docker daemon is running on the remote host and that a firewall or network outage is not preventing you from connecting.

IP forwarding problems

If you manually configure your network using `systemd-network` with `systemd` version 219 or higher, Docker containers may not be able to access your network. Beginning with `systemd` version 220, the forwarding setting for a given network (`net.ipv4.conf.<interface>.forwarding`) defaults to *off*. This setting prevents IP forwarding. It also conflicts with Docker's behavior of enabling the `net.ipv4.conf.all.forwarding` setting within containers.

To work around this on RHEL, CentOS, or Fedora, edit the `<interface>.network` file in `/usr/lib/systemd/network/` on your Docker host (ex: `/usr/lib/systemd/network/80-container-host0.network`) and add the following block within the `[Network]` section.

```
[Network]
...
IPForward=kernel
# OR
IPForward=true
...
```

This configuration allows IP forwarding from the container as expected.

`DNS resolver found in resolv.conf and containers can't use it`

Linux systems which use a GUI often have a network manager running, which uses a `dnsmasq` instance running on a loopback address such as `127.0.0.1` or `127.0.1.1` to cache DNS requests, and adds this entry to `/etc/resolv.conf` . The `dnsmasq` service speeds up DNS look-ups and also provides DHCP services. This configuration does not work within a Docker container which has its own network namespace, because the Docker container resolves loopback addresses such as `127.0.0.1` to itself, and it is very unlikely to be running a DNS server on its own loopback address.

If Docker detects that no DNS server referenced in `/etc/resolv.conf` is a fully functional DNS server, the following warning occurs and Docker uses the public DNS servers provided by Google at `8.8.8.8` and `8.8.4.4` for DNS resolution.

```
WARNING: Local (127.0.0.1) DNS resolver found in resolv.conf and containers
can't use it. Using default external servers : [8.8.8.8 8.8.4.4]
```

If you see this warning, first check to see if you use `dnsmasq` :

```
$ ps aux |grep dnsmasq
```

If your container needs to resolve hosts which are internal to your network, the public nameservers are not adequate. You have two choices:

- You can specify a DNS server for Docker to use, or
- You can disable `dnsmasq` in NetworkManager. If you do this, NetworkManager adds your true DNS nameserver to `/etc/resolv.conf` , but you lose the possible benefits of `dnsmasq` .

You only need to use one of these methods.

Specify DNS servers for Docker

The default location of the configuration file is `/etc/docker/daemon.json` . You can change the location of the configuration file using the `--config-file` daemon flag. The documentation below assumes the configuration file is located at `/etc/docker/daemon.json` .

1. Create or edit the Docker daemon configuration file, which defaults to `/etc/docker/daemon.json` file, which controls the Docker daemon configuration.

   ```
   $ sudo nano /etc/docker/daemon.json
   ```

2. Add a `dns` key with one or more IP addresses as values. If the file has existing contents, you only need to add or edit the `dns` line.

   ```
   {
   	"dns": ["8.8.8.8", "8.8.4.4"]
   }
   ```

   If your internal DNS server cannot resolve public IP addresses, include at least one DNS server which can, so that you can connect to Docker Hub and so that your containers can resolve internet domain names.

   Save and close the file.

3. Restart the Docker daemon.

```
$ sudo service docker restart
```

4. Verify that Docker can resolve external IP addresses by trying to pull an image:

```
$ docker pull hello-world
```

5. If necessary, verify that Docker containers can resolve an internal hostname by pinging it.

```
$ docker run --rm -it alpine ping -c4 <my_internal_host>

PING google.com (192.168.1.2): 56 data bytes
64 bytes from 192.168.1.2: seq=0 ttl=41 time=7.597 ms
64 bytes from 192.168.1.2: seq=1 ttl=41 time=7.635 ms
64 bytes from 192.168.1.2: seq=2 ttl=41 time=7.660 ms
64 bytes from 192.168.1.2: seq=3 ttl=41 time=7.677 ms
```

DISABLE `DNSMASQ`

Ubuntu

If you prefer not to change the Docker daemon's configuration to use a specific IP address, follow these instructions to disable `dnsmasq` in NetworkManager.

1. Edit the `/etc/NetworkManager/NetworkManager.conf` file.
2. Comment out the `dns=dnsmasq` line by adding a `#` character to the beginning of the line.

```
# dns=dnsmasq
```

   Save and close the file.

3. Restart both NetworkManager and Docker. As an alternative, you can reboot your system.

```
$ sudo restart network-manager
$ sudo restart docker
```

RHEL, CentOS, or Fedora

To disable `dnsmasq` on RHEL, CentOS, or Fedora:

1. Disable the `dnsmasq` service:

```
$ sudo service dnsmasq stop

$ sudo systemctl disable dnsmasq
```

2. Configure the DNS servers manually using the Red Hat documentation (https://access.redhat.com /documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s1-networkscripts-interfaces.html).

Allow access to the remote API through a firewall

If you run a firewall on the same host as you run Docker and you want to access the Docker Remote API from another host and remote access is enabled, you need to configure your firewall to allow incoming connections on the Docker port, which defaults to `2376` if TLS encrypted transport is enabled or `2375` otherwise.

Two common firewall daemons are UFW (Uncomplicated Firewall) (https://help.ubuntu.com /community/UFW) (often used for Ubuntu systems) and firewalld (http://www.firewalld.org/) (often used for RPM-based systems). Consult the documentation for your OS and firewall, but the following information might help you get started. These options are fairly permissive and you may want to use a different configuration that locks your system down more.

- UFW: Set `DEFAULT_FORWARD_POLICY="ACCEPT"` in your configuration.
- firewalld: Add rules similar to the following to your policy (one for incoming requests and one for outgoing requests). Be sure the interface names and chain names are correct.

```
<direct>
  [ <rule ipv="ipv6" table="filter" chain="FORWARD_direct" priority="0"> -i zt0 -
  [ <rule ipv="ipv6" table="filter" chain="FORWARD_direct" priority="0"> -o zt0 -
</direct>
```

`Your kernel does not support cgroup swap limit capabilities`

On Ubuntu or Debian hosts, You may see messages similar to the following when working with an image.

```
WARNING: Your kernel does not support swap limit capabilities. Limitation discarded.
```

This warning does not occur on RPM-based systems, which enable these capabilities by default.

If you don't need these capabilities, you can ignore the warning. You can enable these capabilities on Ubuntu or Debian by following these instructions. Memory and swap accounting incur an overhead of about 1% of the total available memory and a 10% overall performance degradation, even if Docker is not running.

1. Log into the Ubuntu or Debian host as a user with `sudo` privileges.
2. Edit the `/etc/default/grub` file. Add or edit the `GRUB_CMDLINE_LINUX` line to add the following two key-value pairs:

   ```
   GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"
   ```

   Save and close the file.

3. Update GRUB.

   ```
   $ sudo update-grub
   ```

   If your GRUB configuration file has incorrect syntax, an error occurs. In this case, repeat steps 2 and 3.

   The changes take effect when the system is rebooted.

Next steps

- Take a look at the Get started (/get-started/) training modules to learn how to build an image and run it as a containerized application.
- Review the topics in Develop with Docker (/develop/) to learn how to build new applications using Docker.

Docker (/search/?q=Docker), Docker documentation (/search/?q=Docker documentation), requirements (/search/?q=requirements), apt (/search/?q=apt), installation (/search/?q=installation), ubuntu (/search /?q=ubuntu), install (/search/?q=install), uninstall (/search/?q=uninstall), upgrade (/search/?q=upgrade), update (/search/?q=update)