Linux post-installation steps for Docker Engine

- Manage Docker as a non-root user
  - Add your user to the docker group
  - Access the docker group on demand
    - Disable password-based entry
- Configure Docker to start on boot with systemd
- Configure default logging driver
- Next steps

These optional post-installation procedures describe how to configure your Linux host machine to work better with Docker.

Manage Docker as a non-root user

The Docker daemon binds to a Unix socket, not a TCP port. By default it's the `root` user that owns the Unix socket, and other users can only access it using `sudo`. The Docker daemon always runs as the `root` user.

When the Docker daemon starts, it creates a Unix socket accessible by members of the `docker` group. On some Linux distributions, the system automatically creates this group when installing Docker Engine using a package manager. In that case, you don't need to create the group manually.

There are two ways to run `docker` commands without `sudo` while the Docker daemon runs as `root`:

- Add your user to the `docker` group for access throughout your login session.
- Access the `docker` group on demand by entering a password-protected, Docker-enabled shell.

> [!WARNING]
>
> The `docker` group grants root-level privileges to the user. For details on how this impacts security in your system, see Docker Daemon Attack Surface.

> [!NOTE]
>
> To run Docker without root privileges, see Run the Docker daemon as a non-root user (Rootless mode).

Add your user to the `docker` group

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 you're running Linux in a virtual machine, it may be necessary to restart the virtual machine for changes to take effect.

   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 a 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:

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

This error indicates that the permission settings for the `~/.docker/` directory are incorrect, due to having used the `sudo` command earlier.
To fix this problem, either remove the `~/.docker/` directory (it's 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
```

Access the `docker` group on demand

Group passwords are a legacy Unix access-control mechanism, but they can be useful for gating Docker access on a single-user workstation. Like `sudo`, this method adds an explicit password step before privileged access. Unlike running `sudo docker`, `newgrp` keeps the Docker CLI running under your user ID and grants access through the shell's primary group, so the CLI doesn't access its configuration as `root`.

Permanent membership in the `docker` group gives every process in your login session access to the Docker socket. The Docker-enabled shell and its descendants inherit access to the Docker socket. This reduces ambient access from applications running elsewhere in your login session. To configure this access, keep your user out of the group, set a group password, and use `newgrp` to start a Docker-enabled shell.

> [!WARNING]
>
> A group password reduces ambient access to the Docker socket, but it doesn't reduce the root-level privileges granted after access is authorized. Group passwords are also shared secrets and don't provide per-user accountability. This method isn't a security boundary against malicious code running as your user. Such code can modify user-writable shell configuration, commands, or scripts that you later use from the Docker-enabled shell and gain Docker access after you authenticate. Don't rely on a group password to contain untrusted code or protect a compromised login session. This configuration is most suitable for a single-user workstation. For stronger isolation, use Rootless mode or run Docker in a virtual machine.

This procedure requires `gpasswd` and `newgrp`. The package names for these commands vary by Linux distribution. Verify that both commands are available:

```
$ command -v gpasswd newgrp
/usr/bin/gpasswd
/usr/bin/newgrp
```

To require a password for Docker access:

1. Create the `docker` group if it doesn't exist:

   ```
   $ sudo groupadd --force docker
   ```

   The `--force` option makes the command succeed when the group already exists.
2. If your user is a member of the `docker` group, remove the membership:

   ```
   $ sudo gpasswd --delete "$USER" docker
   ```

   Sign out of the desktop or SSH session completely, then sign back in. Group membership remains in the credentials of existing processes, so opening a new terminal isn't sufficient. Verify that `docker` is absent from the group list before continuing:

   ```
   $ id -nG
   user wheel
   ```

   Your group list varies by system, but it must not include `docker`.
3. Set a dedicated password for the `docker` group:

   ```
   $ sudo gpasswd docker
   Changing the password for group docker
   New Password:
   Re-enter new password:
   ```

   Don't add your user back to the group. Users configured as group members can use `newgrp` without entering the group password.
4. Start a child shell with `docker` as its primary group:

   ```
   $ newgrp docker
   Password:
   ```

   Verify that the shell still uses your user ID and has `docker` as its primary group, then test Docker access:

   ```
   $ id -un
   user
   $ id -gn
   docker
   $ docker run --rm hello-world
   ```

Commands and applications started from this shell inherit access to the Docker socket. Applications that were already running outside the shell don't gain access.

> [!CAUTION]
> Files and directories created from this shell normally have `docker` as their group owner. Use this shell only for Docker-related commands, or verify the group ownership of files you create.

5. Exit the Docker-enabled shell when you finish:

```
$ exit
```

Verify that `docker` is no longer in the original shell's group list:

```
$ id -nG
user wheel
```

> [!CAUTION]
>
> Exiting the shell doesn't revoke access from background, detached, or daemonized processes started inside it. Those processes retain the `docker` group until they exit.

To change the group password, run `sudo gpasswd docker` again.

Disable password-based entry

To stop using the shared password and require configured group membership, run:

```
$ sudo gpasswd --restrict docker
```

This reverses the password-gated setup. Afterward, only users configured as members of the `docker` group can enter it. The change affects new authorization attempts; it doesn't terminate existing Docker-enabled shells or their descendant processes.

`gpasswd` manages local `/etc/group` and `/etc/gshadow` files. Systems using LDAP, NIS, or another identity service require that service's group-management mechanism.

Configure Docker to start on boot with systemd

Many modern Linux distributions use systemd to manage which services start when the system boots. On Debian and Ubuntu, the Docker service starts on boot by default. To automatically start Docker and containerd on boot for other Linux distributions using systemd, run the following commands:

```
$ sudo systemctl enable docker.service
$ sudo systemctl enable containerd.service
```

To stop this behavior, use `disable` instead.

```
$ sudo systemctl disable docker.service
$ sudo systemctl disable containerd.service
```

You can use systemd unit files to configure the Docker service on startup, for example to add an HTTP proxy, set a different directory or partition for the Docker runtime files, or other customizations. For an example, see Configure the daemon to use a proxy.

Configure default logging driver

Docker provides logging drivers for collecting and viewing log data from all containers running on a host. 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 avoid issues with overusing disk for log data, consider one of the following options:

- Configure the `json-file` logging driver to turn on log rotation.
- Use an alternative logging driver such as the "local" logging driver that performs log rotation by default.
- Use a logging driver that sends logs to a remote logging aggregator.

Next steps

- Take a look at Get started with Docker to learn how to build an image and run it as a containerized application.

□□□ #2
□ □□□ □□□ 2026-10-31 19:23:49 UTC
□ □□□ □□□ 2026-10-31 19:23:50 UTC