Includes

- daemon-cfg-desktop
- desktop-license-update
- desktop-linux-launch
- engine-license
- extensions-form
- hub-categories
- install-script
- root-errors
- sandboxes-local-scope
- swarm-compose-compat
- swarm-mode
- build
- configs
- configure-watch
- deploy
- extension
- fragments
- include
- interpolation
- merge
- networks
- profiles
- secrets
- services-depends-on
- services-develop
- services-env-file
- services-environment

- services-healthcheck
- services-networks
- services-ports
- services-secrets
- services-volumes
- services
- volumes
- watch

daemon-cfg-desktop

> [!NOTE]
>
> If you're using Docker Desktop, edit the daemon configuration through the Docker Desktop Dashboard. Open **Settings** and select **Docker Engine**. For details, see Docker Engine settings.

desktop-license-update

Here’s a summary of the key points:

- Docker Desktop is free for small businesses (fewer than 250 employees AND less than $10 million in annual revenue), personal use, education, and non-commercial open source projects.
- Otherwise, it requires a paid subscription for professional use.
- Paid subscriptions are also required for government entities.
- The Docker Pro, Team, and Business subscriptions include commercial use of Docker Desktop.

desktop-linux-launch

To start Docker Desktop for Linux:

1. Navigate to the Docker Desktop application in your Gnome/KDE Desktop.
2. Select **Docker Desktop** to start Docker.
   The Docker Subscription Service Agreement displays.
3. Select **Accept** to continue. Docker Desktop starts after you accept the terms.
   Note that Docker Desktop won't run if you do not agree to the terms. You can choose to accept the terms at a later date by opening Docker Desktop.

   For more information, see Docker Desktop Subscription Service Agreement. It is recommended that you also read the FAQs.

Alternatively, open a terminal and run:

```
$ systemctl --user start docker-desktop
```

When Docker Desktop starts, it creates a dedicated context that the Docker CLI can use as a target and sets it as the current context in use. This is to avoid a clash with a local Docker Engine that may be running on the Linux host and using the default context. On shutdown, Docker Desktop resets the current context to the previous one.

The Docker Desktop installer updates Docker Compose and the Docker CLI binaries on the host. It installs Docker Compose V2 and gives users the choice to link it as docker-compose from the Settings panel. Docker Desktop installs the new Docker CLI binary that includes cloud-integration capabilities in `/usr/local/bin/com.docker.cli` and creates a symlink to the classic Docker CLI at `/usr/local/bin`.

After you've successfully installed Docker Desktop, you can check the versions of these binaries by running the following commands:

```
$ docker compose version
Docker Compose version v2.39.4

$ docker --version
Docker version 28.4.0, build d8eb465

$ docker version
Client:
 Version:           28.4.0
```

```
 API version:       1.51
 Go version:        go1.24.7
<...>
```

To enable Docker Desktop to start on sign in, from the Docker menu, select **Settings** > **General** > **Start Docker Desktop when you sign in to your computer**.

Alternatively, open a terminal and run:

```
$ systemctl --user enable docker-desktop
```

To stop Docker Desktop, select the Docker menu icon to open the Docker menu and select **Quit Docker Desktop**.

Alternatively, open a terminal and run:

```
$ systemctl --user stop docker-desktop
```

engine-license

Apache License, Version 2.0. See LICENSE for the full license.

extensions-form

> Already built an extension?
>
> Let us know about your experience using the feedback form.

hub-categories

The categories include:

- **API management**: Tools for creating, publishing, analyzing, and securing APIs.
- **Content management system:** Software applications to create and manage digital content through templates, procedures, and standard formats.
- **Data science:** Tools and software to support analyzing data and generating actionable insights.
- **Developer tools:** Software and utilities that assist developers in creating, debugging, maintaining, and supporting applications and systems.
- **Databases & storage:** Systems for storing, retrieving, and managing data.
- **Languages & frameworks:** Programming language runtimes and frameworks.
- **Integration & delivery:** Tools for Continuous Integration (CI) and Continuous Delivery (CD).
- **Internet of things:** Tools supporting Internet of Things (IoT) applications.
- **Machine learning & AI:** Tools and frameworks optimized for artificial intelligence and machine learning projects, such as pre-installed libraries and frameworks for data analysis, model training, and deployment.
- **Message queues:** Message queuing systems optimized for reliable, scalable, and efficient message handling.
- **Monitoring & Observability:** Tools to track software and system performance through metrics, logs, and traces, as well as observability to explore the system’s state and diagnose issues.
- **Networking:** Repositories that support data exchange and connecting computers and other devices to share resources.
- **Operating systems:** Software that manages all other programs on a computer and serves as an intermediary between users and the computer hardware, while overseeing applications and system resources.
- **Security:** Tools to protect a computer system or network from theft, unauthorized access, or damage to their hardware, software, or electronic data, as well as from service disruption.
- **Web servers:** Software to serve web pages, HTML files, and other assets to users or other systems.
- **Web analytics:** Tools to collect, measure, analyze, and report on web data and website visitor engagement.

install-script

Install using the convenience script

Docker provides a convenience script at https://get.docker.com/ to install Docker into development environments non-interactively. The convenience script isn't recommended for production environments, but it's useful for creating a provisioning script tailored to your needs. Also refer to the install using the repository steps to learn about installation steps to install using the package repository. The source code for the script is open source, and you can find it in the `docker-install` repository on GitHub.

Always examine scripts downloaded from the internet before running them locally. Before installing, make yourself familiar with potential risks and limitations of the convenience script:

- The script requires `root` or `sudo` privileges to run.
- The script attempts to detect your Linux distribution and version and configure your package management system for you.
- The script doesn't allow you to customize most installation parameters.
- The script installs dependencies and recommendations without asking for confirmation. This may install a large number of packages, depending on the current configuration of your host machine.
- By default, the script installs the latest stable release of Docker, containerd, and runc. When using this script to provision a machine, this may result in unexpected major version upgrades of Docker. Always test upgrades in a test environment before deploying to your production systems.
- The script isn't designed to upgrade an existing Docker installation. When using the script to update an existing installation, dependencies may not be updated to the expected version, resulting in outdated versions.

> [!TIP]
>
> Preview script steps before running. You can run the script with the `--dry-run` option to learn what steps the script will run when invoked:
>
> ```
> $ curl -fsSL https://get.docker.com -o get-docker.sh
> $ sudo sh ./get-docker.sh --dry-run
> ```

This example downloads the script from https://get.docker.com/ and runs it to install the latest stable release of Docker on Linux:

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
Executing docker install script, commit: 7cae5f8b0decc17d6571f9f52eb840fbc13b2737
<...>
```

You have now successfully installed and started Docker Engine. The `docker` service starts automatically on Debian based distributions. On `RPM` based distributions, such as CentOS, Fedora or RHEL, you need to start it manually using the appropriate `systemctl` or `service` command. As the message indicates, non-root users can't run Docker commands by default.

> **Use Docker as a non-privileged user, or install in rootless mode?**
>
> The installation script requires `root` or `sudo` privileges to install and use Docker. If you want to grant non-root users access to Docker, refer to the post-installation steps for Linux. You can also install Docker without `root` privileges, or configured to run in rootless mode. For instructions on running Docker in rootless mode, refer to run the Docker daemon as a non-root user (rootless mode).

Install pre-releases

Docker also provides a convenience script at https://test.docker.com/ to install pre-releases of Docker on Linux. This script is equal to the script at `get.docker.com`, but configures your package manager to use the test channel of the Docker package repository. The test channel includes both stable and pre-releases (beta versions, release-candidates) of Docker. Use this script to get early access to new releases, and to evaluate them in a testing environment before they're released as stable.

To install the latest version of Docker on Linux from the test channel, run:

```
$ curl -fsSL https://test.docker.com -o test-docker.sh
$ sudo sh test-docker.sh
```

Upgrade Docker after using the convenience script

If you installed Docker using the convenience script, you should upgrade Docker using your package manager directly. There's no advantage to re-running the convenience script. Re-running

it can cause issues if it attempts to re-install repositories which already exist on the host machine.

root-errors

> [!TIP]
>
> Receiving errors when trying to run without root?
>
> The `docker` user group exists but contains no users, which is why you're required to use `sudo` to run Docker commands. Continue to Linux postinstall to allow non-privileged users to run Docker commands and for other optional configuration steps.

sandboxes-local-scope

This page describes local sandboxes. For cloud behavior and limitations, see Compare local and cloud sandboxes.

swarm-compose-compat

> [!NOTE]
>
> The `docker stack deploy` command uses the legacy Compose file version 3 format, used by Compose V1. The latest format, defined by the Compose specification isn't compatible with the `docker stack deploy` command.
>
> For more information about the evolution of Compose, see History of Compose.

swarm-mode

> [!NOTE]
>
> Swarm mode is an advanced feature for managing a cluster of Docker daemons.
>
> Use Swarm mode if you intend to use Swarm as a production runtime environment.
>
> If you're not planning on deploying with Swarm, use Docker Compose instead. If you're developing for a Kubernetes deployment, consider using the integrated Kubernetes feature in Docker Desktop.

build

Build is an optional part of the Compose Specification. It tells Compose how to (re)build an application from source and lets you define the build process within a Compose file in a portable way. `build` can be either specified as a single string defining a context path, or as a detailed build definition.

configs

Configs let services adapt their behaviour without the need to rebuild a Docker image. As with volumes, configs are mounted as files into a container's filesystem. The location of the mount point within the container defaults to `/<config-name>` in Linux containers and `C:\<config-name>` in Windows containers.

configure-watch

1. Add `watch` sections to one or more services in `compose.yaml`.
2. Run `docker compose up --watch` to build and launch a Compose project and start the file watch mode.
3. Edit service source files using your preferred IDE or editor.

deploy

Deploy is an optional part of the Compose Specification. It provides a set of deployment specifications for managing the behavior of containers across different environments.

extension

Extensions can be used to make your Compose file more efficient and easier to maintain.

Use the prefix `x-` as a top-level element to modularize configurations that you want to reuse. Compose ignores any fields that start with `x-`, this is the sole exception where Compose silently ignores unrecognized fields.

fragments

With Compose, you can use built-in YAML features to make your Compose file neater and more efficient. Anchors and aliases let you create reusable blocks. This is useful if you start to find common configurations that span multiple services. Having reusable blocks minimizes potential mistakes.

include

With `include`, you can incorporate a separate `compose.yaml` file directly in your current `compose.yaml` file. This makes it easy to modularize complex applications into sub-Compose files, which in turn enables application configurations to be made simpler and more explicit.

interpolation

Values in a Compose file can be set by variables and interpolated at runtime. Compose files use a Bash-like syntax `${VARIABLE}`. Both `$VARIABLE` and `${VARIABLE}` syntax is supported.

merge

Compose lets you define a Compose application model through multiple Compose files. When doing so, Compose follows certain rules to merge Compose files.

networks

Networks let services communicate with each other. By default Compose sets up a single network for your app. Each container for a service joins the default network and is both reachable by other containers on that network, and discoverable by the service's name. The top-level `networks` element lets you configure named networks that can be reused across multiple services.

profiles

Profiles help you adjust your Compose application for different environments or use cases by selectively activating services. Services can be assigned to one or more profiles; unassigned services start/stop by default, while assigned ones only start/stop when their profile is active. This setup means specific services, like those for debugging or development, to be included in a single `compose.yml` file and activated only as needed.

secrets

Docker Compose provides a way for you to use secrets without having to use environment variables to store information. If you’re injecting passwords and API keys as environment variables, you risk unintentional information exposure. Services can only access secrets when explicitly granted by a `secrets` attribute within the `services` top-level element.

services-depends-on

With the `depends_on` attribute, you can control the order of service startup and shutdown. It is useful if services are closely coupled, and the startup sequence impacts the application's functionality.

services-develop

Develop lets Compose focus on the development use-case of running applications on a local machine. It also supports some development hooks to improve the velocity of your local workflow, also known as your "inner loop".

services-env-file

The `env_file` attribute is used to specify one or more files that contain environment variables to be passed to the containers.

services-environment

The `environment` attribute defines environment variables set in the container. `environment` can use either an array or a map. Any boolean values; true, false, yes, no, should be enclosed in quotes to ensure they are not converted to True or False by the YAML parser.

services-healthcheck

The `healthcheck` attribute declares a check that's run to determine whether or not the service containers are "healthy". It works in the same way, and has the same default values, as the HEALTHCHECK Dockerfile instruction set by the service's Docker image. Your Compose file can override the values set in the Dockerfile.

services-networks

The `networks` attribute defines the networks that service containers are attached to, referencing entries under the `networks` top-level element. The `networks` attribute helps manage the networking aspects of containers, providing control over how services are segmented and interact within the Docker environment. This is used to specify which networks the containers for that service should connect to. This is important for defining how containers communicate with each other and externally.

services-ports

The `ports` is used to define the port mappings between the host machine and the containers. This is crucial for allowing external access to services running inside containers. It can be defined using short syntax for simple port mapping or long syntax, which includes additional options like protocol type and network mode.

services-secrets

The `secrets` attribute grants access to sensitive data defined by the secrets top-level element on a per-service basis. Services can be granted access to multiple secrets.

services-volumes

The `volumes` attribute define mount host paths or named volumes that are accessible by service containers. You can use `volumes` to define multiple types of mounts; `volume`, `bind`, `tmpfs`, or `npipe`.

If the mount is a host path and is only used by a single service, it can be declared as part of the service definition. To reuse a volume across multiple services, a named volume must be declared in the `volumes` top-level element.

services

A service is an abstract definition of a computing resource within an application which can be scaled or replaced independently from other components. Services are backed by a set of containers, run by the platform according to replication requirements and placement constraints. As services are backed by containers, they are defined by a Docker image and set of runtime arguments. All containers within a service are identically created with these arguments.

volumes

Volumes are persistent data stores implemented by the container engine. Compose offers a neutral way for services to mount volumes, and configuration parameters to allocate them to infrastructure. The top-level `volumes` declaration lets you configure named volumes that can be reused across multiple services.

watch

The `watch` attribute automatically updates and previews your running Compose services as you edit and save your code. For many projects, this enables a hands-off development workflow once Compose is running, as services automatically update themselves when you save your work.