Bind mounts

By Docker Inc

Published: 2026-03-23 · Archived: 2026-04-05 17:19:17 UTC

When you use a bind mount, a file or directory on the host machine is mounted from the host into a container. By contrast, when you use a volume, a new directory is created within Docker's storage directory on the host machine. Docker creates and maintains this storage location, but containers access it directly using standard filesystem operations.

Bind mounts are appropriate for the following types of use case:

- Sharing source code or build artifacts between a development environment on the Docker host and a container.
- When you want to create or generate files in a container and persist the files onto the host's filesystem.
- Sharing configuration files from the host machine to containers. This is how Docker provides DNS resolution to containers by default, by mounting `/etc/resolv.conf` from the host machine into each container.

Bind mounts are also available for builds: you can bind mount source code from the host into the build container to test, lint, or compile a project.

If you bind mount file or directory into a directory in the container in which files or directories exist, the pre-existing files are obscured by the mount. This is similar to if you were to save files into `/mnt` on a Linux host, and then mounted a USB drive into `/mnt`. The contents of `/mnt` would be obscured by the contents of the USB drive until the USB drive was unmounted.

With containers, there's no straightforward way of removing a mount to reveal the obscured files again. Your best option is to recreate the container without the mount.

- Bind mounts have write access to files on the host by default.

  One side effect of using bind mounts is that you can change the host filesystem via processes running in a container, including creating, modifying, or deleting important system files or directories. This capability can have security implications. For example, it may affect non-Docker processes on the host system.

  You can use the `readonly` or `ro` option to prevent the container from writing to the mount.

- Bind mounts are created to the Docker daemon host, not the client.

  If you're using a remote Docker daemon, you can't create a bind mount to access files on the client machine in a container.

For Docker Desktop, the daemon runs inside a Linux VM, not directly on the native host. Docker Desktop has built-in mechanisms that transparently handle bind mounts, allowing you to share native host filesystem paths with containers running in the virtual machine.

- Containers with bind mounts are strongly tied to the host.

  Bind mounts rely on the host machine's filesystem having a specific directory structure available. This reliance means that containers with bind mounts may fail if run on a different host without the same directory structure.

To create a bind mount, you can use either the `--mount` or `--volume` flag.

In general, `--mount` is preferred. The main difference is that the `--mount` flag is more explicit and supports all the available options.

If you use `--volume` to bind-mount a file or directory that does not yet exist on the Docker host, Docker automatically creates the directory on the host for you. It's always created as a directory.

`--mount` does not automatically create a directory if the specified mount path does not exist on the host. Instead, it produces an error:

Options for --mount

The `--mount` flag consists of multiple key-value pairs, separated by commas and each consisting of a `<key>=<value>` tuple. The order of the keys isn't significant.

Valid options for `--mount type=bind` include:

Option	Description
`source`, `src`	The location of the file or directory on the host. This can be an absolute or relative path.
`destination`, `dst`, `target`	The path where the file or directory is mounted in the container. Must be an absolute path.
`readonly`, `ro`	If present, causes the bind mount to be mounted into the container as read-only.
`bind-propagation`	If present, changes the bind propagation.

Options for --volume

The `--volume` or `-v` flag consists of three fields, separated by colon characters (`:`). The fields must be in the correct order.

The first field is the path on the host to bind mount into the container. The second field is the path where the file or directory is mounted in the container.

The third field is optional, and is a comma-separated list of options. Valid options for `--volume` with a bind mount include:

Option	Description
`readonly`, `ro`	If present, causes the bind mount to be mounted into the container as read-only.
`z`, `Z`	Configures SELinux labeling. See Configure the SELinux label
`rprivate` (default)	Sets bind propagation to `rprivate` for this mount. See Configure bind propagation.
`private`	Sets bind propagation to `private` for this mount. See Configure bind propagation.
`rshared`	Sets bind propagation to `rshared` for this mount. See Configure bind propagation.
`shared`	Sets bind propagation to `shared` for this mount. See Configure bind propagation.
`rslave`	Sets bind propagation to `rslave` for this mount. See Configure bind propagation.
`slave`	Sets bind propagation to `slave` for this mount. See Configure bind propagation.

Consider a case where you have a directory `source` and that when you build the source code, the artifacts are saved into another directory, `source/target/`. You want the artifacts to be available to the container at `/app/`, and you want the container to get access to a new build each time you build the source on your development host. Use the following command to bind-mount the `target/` directory into your container at `/app/`. Run the command from within the `source` directory. The `$(pwd)` sub-command expands to the current working directory on Linux or macOS hosts. If you're on Windows, see also Path conversions on Windows.

The following `--mount` and `-v` examples produce the same result. You can't run them both unless you remove the `devtest` container after running the first one.

Use `docker inspect devtest` to verify that the bind mount was created correctly. Look for the `Mounts` section:

This shows that the mount is a `bind` mount, it shows the correct source and destination, it shows that the mount is read-write, and that the propagation is set to `rprivate`.

Stop and remove the container:

Mount into a non-empty directory on the container

If you bind-mount a directory into a non-empty directory on the container, the directory's existing contents are obscured by the bind mount. This can be beneficial, such as when you want to test a new version of your application without building a new image. However, it can also be surprising and this behavior differs from that of volumes.

This example is contrived to be extreme, but replaces the contents of the container's `/usr/` directory with the `/tmp/` directory on the host machine. In most cases, this would result in a non-functioning container.

The `--mount` and `-v` examples have the same end result.

The container is created but does not start. Remove it:

For some development applications, the container needs to write into the bind mount, so changes are propagated back to the Docker host. At other times, the container only needs read access.

This example modifies the previous one, but mounts the directory as a read-only bind mount, by adding `ro` to the (empty by default) list of options, after the mount point within the container. Where multiple options are present, separate them by commas.

The `--mount` and `-v` examples have the same result.

Use `docker inspect devtest` to verify that the bind mount was created correctly. Look for the `Mounts` section:

Stop and remove the container:

When you bind mount a path that itself contains mounts, those submounts are also included in the bind mount by default. This behavior is configurable, using the `bind-recursive` option for `--mount`. This option is only supported with the `--mount` flag, not with `-v` or `--volume`.

If the bind mount is read-only, the Docker Engine makes a best-effort attempt at making the submounts read-only as well. This is referred to as recursive read-only mounts. Recursive read-only mounts require Linux kernel version 5.12 or later. If you're running an older kernel version, submounts are automatically mounted as read-write by default. Attempting to set submounts to be read-only on a kernel version earlier than 5.12, using the `bind-recursive=readonly` option, results in an error.

Supported values for the `bind-recursive` option are:

Value	Description
`enabled` (default)	Read-only mounts are made recursively read-only if kernel is v5.12 or later. Otherwise, submounts are read-write.
`disabled`	Submounts are ignored (not included in the bind mount).
`writable`	Submounts are read-write.
`readonly`	Submounts are read-only. Requires kernel v5.12 or later.

Bind propagation defaults to `rprivate` for both bind mounts and volumes. It is only configurable for bind mounts, and only on Linux host machines. Bind propagation is an advanced topic and many users never need to configure it.

Bind propagation refers to whether or not mounts created within a given bind-mount can be propagated to replicas of that mount. Consider a mount point `/mnt`, which is also mounted on `/tmp`. The propagation settings control whether a mount on `/tmp/a` would also be available on `/mnt/a`. Each propagation setting has a recursive

counterpoint. In the case of recursion, consider that `/tmp/a` is also mounted as `/foo`. The propagation settings control whether `/mnt/a` and/or `/tmp/a` would exist.

> Mount propagation doesn't work with Docker Desktop.

Propagation setting	Description
`shared`	Sub-mounts of the original mount are exposed to replica mounts, and sub-mounts of replica mounts are also propagated to the original mount.
`slave`	similar to a shared mount, but only in one direction. If the original mount exposes a sub-mount, the replica mount can see it. However, if the replica mount exposes a sub-mount, the original mount cannot see it.
`private`	The mount is private. Sub-mounts within it are not exposed to replica mounts, and sub-mounts of replica mounts are not exposed to the original mount.
`rshared`	The same as shared, but the propagation also extends to and from mount points nested within any of the original or replica mount points.
`rslave`	The same as slave, but the propagation also extends to and from mount points nested within any of the original or replica mount points.
`rprivate`	The default. The same as private, meaning that no mount points anywhere within the original or replica mount points propagate in either direction.

Before you can set bind propagation on a mount point, the host filesystem needs to already support bind propagation.

For more information about bind propagation, see the Linux kernel documentation for shared subtree.

The following example mounts the `target/` directory into the container twice, and the second mount sets both the `ro` option and the `rslave` bind propagation option.

The `--mount` and `-v` examples have the same result.

Now if you create `/app/foo/`, `/app2/foo/` also exists.

If you use SELinux, you can add the `z` or `Z` options to modify the SELinux label of the host file or directory being mounted into the container. This affects the file or directory on the host machine itself and can have consequences outside of the scope of Docker.

- The `z` option indicates that the bind mount content is shared among multiple containers.
- The `Z` option indicates that the bind mount content is private and unshared.

Use extreme caution with these options. Bind-mounting a system directory such as `/home` or `/usr` with the `Z` option renders your host machine inoperable and you may need to relabel the host machine files by hand.

> When using bind mounts with services, SELinux labels (`:Z` and `:z`), as well as `:ro` are ignored. See moby/moby #32579 for details.

This example sets the `z` option to specify that multiple containers can share the bind mount's contents:

It is not possible to modify the SELinux label using the `--mount` flag.

A single Docker Compose service with a bind mount looks like this:

For more information about using volumes of the `bind` type with Compose, see Compose reference on the volumes top-level element. and Compose reference on the volume attribute.

- Learn about volumes.
- Learn about tmpfs mounts.
- Learn about storage drivers.
