Image mounts

- When to use image mounts
- Mounting over existing data
- Considerations and constraints
- Syntax
  - Options for --mount
- Use an image mount in a container
- Mount a subpath of an image
- Use an image mount with Docker Compose
- Next steps

Volumes, bind mounts, and tmpfs mounts all give a container a place to read and write data. An image mount is different: instead of mounting a directory or a memory-backed filesystem, it mounts the contents of another image into the container.

When you use an image mount, the filesystem of a second image is mounted into the container at a path you choose. The container can read the files from that image alongside its own filesystem, without those files being part of the container's own image. This is useful when you want to bring the tools or assets from one image into a container that's running a different image.

Image mounts are read-only. The mounted image is never modified, and the container can't write to the mount.

> [!NOTE] Image mounts require the containerd image store.

When to use image mounts

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

- Debugging a minimal or hardened image that doesn't include a shell or common utilities. You can mount a tool-rich image, such as `busybox`, into the running container's namespace and run those tools without changing the original image. For a worked example, see Debug with Docker Hardened Images.
- Sharing read-only assets, such as datasets, models, or static content, that are distributed as an image and consumed by containers running a different image.
- Keeping application images small by packaging optional tooling in a separate image and mounting it only when needed.

Mounting over existing data

If you mount an image 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.

Considerations and constraints

- Image mounts are always read-only. The container can't modify the mounted image, and changes aren't persisted anywhere.
- The source image must already exist in the daemon's image store. Docker doesn't pull the source image automatically when you create the mount. If the image isn't present, the command fails:

  ```
  $ docker run --mount type=image,source=busybox:musl,destination=/dbg alpine
  docker: Error response from daemon: No such image: busybox:musl
  ```

  Pull the image first with `docker pull`, then create the mount.
- Image mounts require the containerd image store. They aren't available when the daemon uses the classic storage drivers.
- You can only create an image mount with the `--mount` flag. There is no `--volume` (`-v`) equivalent.
- Running an executable from a mounted image requires a compatible runtime in the container. A dynamically linked binary only runs if the container provides a matching dynamic linker and shared libraries. For example, a glibc-based binary fails in a musl-based image such as Alpine. Statically linked binaries, or mounting only data from an image, avoid this constraint.

Syntax

To mount an image with the `docker run` command, use the `--mount` flag with `type=image`.

```
$ docker run --mount type=image,src=<image-reference>,dst=<container-path>
```

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.

```
$ docker run --mount type=image,src=<image-reference>,dst=<container-path>[,<key>=<value>...]
```

Options for --mount

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

Option	Description
`source`, `src`	The reference of the image to mount, for example `busybox` or `busybox:musl`. The image must exist locally.
`destination`, `dst`, `target`	The path where the image is mounted in the container. Must be an absolute path.
`image-subpath`	Path inside the source image to mount instead of the image root. See Mount a subpath of an image.

```
$ docker run --mount type=image,src=busybox,dst=/dbg,image-subpath=bin
```

Use an image mount in a container

The following example runs an Alpine container and mounts the `busybox:musl` image at `/dbg`. Pull the source image first, since Docker doesn't pull it for you when creating the mount. This example uses the musl-based BusyBox image so its binaries are compatible with the musl-based Alpine container.

```
$ docker pull busybox:musl
$ docker run -d \
  -it \
  --name imgtest \
  --mount type=image,source=busybox:musl,destination=/dbg \
```

```
  alpine:latest
```

The container can now read the BusyBox tools from `/dbg` while running the Alpine image:

```
$ docker exec imgtest /dbg/bin/echo "hello from busybox"
hello from busybox
```

Verify that the mount is an `image` mount by looking in the `Mounts` section of the `docker inspect` output:

```
$ docker inspect imgtest --format '{{ json .Mounts }}'
[{"Type":"image","Name":"busybox:musl","Source":"/var/lib/docker/rootfs/overlayfs/...","Destin
ation":"/dbg","Mode":"","RW":false,"Propagation":"rprivate"}]
```

This shows that the mount is an `image` mount, that its source is the `busybox:musl` image, and that it's read-only (`"RW":false`).

Stop and remove the container:

```
$ docker container rm -fv imgtest
```

Mount a subpath of an image

Use the `image-subpath` option to mount a specific directory from the source image instead of its root. For example, to mount only the `bin` directory of the `busybox` image at `/tools`:

```
$ docker run -d \
  -it \
  --name imgtest \
  --mount type=image,source=busybox,destination=/tools,image-subpath=bin \
  alpine:latest
```

The container sees the contents of the image's `bin` directory at `/tools`.

Use an image mount with Docker Compose

A single Docker Compose service with an image mount looks like this:

```
services:
  app:
    image: alpine:latest
    volumes:
      - type: image
        source: busybox
        target: /dbg
```

To mount a subpath of the image, use the `subpath` option under `image`:

```
services:
  app:
    image: alpine:latest
    volumes:
      - type: image
        source: busybox
        target: /tools
        image:
          subpath: bin
```

The `image.subpath` option is available in Docker Compose version 2.35.0 and later. For more information about using mounts of the `image` type with Compose, see the Compose reference on the volume attribute.

Next steps

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

□□□ #2
□ □□□ □□□ 2026-10-31 19:31:02 UTC
□ □□□ □□□ 2026-10-31 19:31:03 UTC