Storage in Docker

Create and manage volumes

Unlike a bind mount, you can create and manage volumes outside the scope of any container.

**Create a volume**:

$ docker volume create my-vol

**List volumes**:

$ docker volume ls

local my-vol

**Inspect a volume**:

$ docker volume inspect my-vol

[

{

"Driver": "local",

"Labels": {},

"Mountpoint": "/var/lib/docker/volumes/my-vol/\_data",

"Name": "my-vol",

"Options": {},

"Scope": "local"

}

]

**Remove a volume**:

$ docker volume rm my-vol

Start a container with a volume

If you start a container with a volume that does not yet exist, Docker creates the volume for you. The following example mounts the volume myvol2 into /app/ in the container.

The -v and --mount examples below produce the same result. You can’t run them both unless you remove the devtest container and the myvol2 volume after running the first one.

* --mount

$ docker run -d \

-it \

--name devtest \

--mount source=myvol2,target=/app \

nginx:latest

-v

$ docker run -d \

-it \

--name devtest \

-v myvol2:/app \

nginx:latest

Use docker inspect devtest to verify that the volume was created and mounted correctly. Look for the Mounts section:

"Mounts": [

{

"Type": "volume",

"Name": "myvol2",

"Source": "/var/lib/docker/volumes/myvol2/\_data",

"Destination": "/app",

"Driver": "local",

"Mode": "",

"RW": true,

"Propagation": ""

}

],

This shows that the mount is a volume, it shows the correct source and destination, and that the mount is read-write.

Stop the container and remove the volume.

$ docker container stop devtest

$ docker container rm devtest

$ docker volume rm myvol2

### **Populate a volume using a container**

If you start a container which creates a new volume, as above, and the container has files or directories in the directory to be mounted (such as /app/ above), the directory’s contents will be copied into the volume. The container will then mount and use the volume, and other containers which use the volume will also have access to the pre-populated content.

To illustrate this, this example starts an nginx container and populates the new volume nginx-vol with the contents of the container’s /usr/share/nginx/html directory, which is where Nginx stores its default HTML content.

--mount

$ docker run -d \

-it \

--name=nginxtest \

--mount source=nginx-vol,destination=/usr/share/nginx/html \

nginx:latest

-v

$ docker run -d \

-it \

--name=nginxtest \

-v nginx-vol:/usr/share/nginx/html \

nginx:latest

After running either of these examples, run the following commands to clean up the containers and volumes.

$ docker container stop nginxtest

$ docker container rm nginxtest

$ docker volume rm nginx-vol

Use a read-only volume

For some development applications, it is useful for the container to be able to write into the bind mount, in order for changes to be propagated back to the Docker host. At other times, the container should only be able to read the data and not modify it. Remember that multiple containers can mount the same volume, and it can be mounted read-write for some of them and read-only for others, simultaneously.

This example modifies the one above but mounts the directory as a read-only volume, 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.

* -v

$ docker run -d \

-it \

--name=nginxtest \

-v nginx-vol:/usr/share/nginx/html:ro \

nginx:latest

mount

$ docker run -d \

-it \

--name=nginxtest \

--mount source=nginx-vol,destination=/usr/share/nginx/html \

nginx:latest

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

"Mounts": [

{

"Type": "volume",

"Name": "nginx-vol",

"Source": "/var/lib/docker/volumes/nginx-vol/\_data",

"Destination": "/usr/share/nginx/html",

"Driver": "local",

"Mode": "",

"RW": false,

"Propagation": ""

}

],

Stop and remove the container, and remove the volume:

$ docker container stop nginxtest

$ docker container rm nginxtest

$ docker volume rm nginx-vol

## Start a container with a bind mount

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 thesource directory. The $(pwd) sub-command expands to the current working directory on Linux or macOS hosts.

--mount

$ docker run -d \

-it \

--name devtest \

--mount type=bind,source="$(pwd)"/target,target=/app \

nginx:latest

-v

$ docker run -d \

-it \

--name devtest \

-v "$(pwd)"/target:/app \

nginx:latest

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

"Mounts": [

{

"Type": "bind",

"Source": "/tmp/source/target",

"Destination": "/app",

"Mode": "",

"RW": true,

"Propagation": "rprivate"

}

],

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 the container:

$ docker container stop devtest

$ docker container rm devtest

### **Mounting into a non-empty directory on the container**

If you bind-mount into a non-empty directory on the container, the directory’s existing contents will be 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 [docker volumes](https://docs.docker.com/engine/admin/volumes/volumes/).

--mount

$ docker run -d \

-it \

--name broken-container \

--mount type=bind,source=/tmp,target=/usr \

nginx:latest

docker: Error response from daemon: oci runtime error: container\_linux.go:262:

starting container process caused "exec: \"nginx\": executable file not found in $PATH".

-v

$ docker run -d \

-it \

--name broken-container \

-v /tmp:/usr \

nginx:latest

docker: Error response from daemon: oci runtime error: container\_linux.go:262:

starting container process caused "exec: \"nginx\": executable file not found in $PATH".

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

$ docker container rm broken-container

## Use a read-only bind mount

For some development applications, it is useful for the container to be able to write into the bind mount, in order for changes to be propagated back to the Docker host. At other times, the container should only be able to read the data and not modify it.

This example modifies the one above 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.

--mount

$ docker run -d \

-it \

--name devtest \

--mount type=bind,source="$(pwd)"/target,target=/app,readonly \

nginx:latest

-v

$ docker run -d \

-it \

--name devtest \

-v "$(pwd)"/target:/app:ro \

nginx:latest

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

"Mounts": [

{

"Type": "bind",

"Source": "/tmp/source/target",

"Destination": "/app",

"Mode": "ro",

"RW": false,

"Propagation": "rprivate"

}

],

Stop the container:

$ docker container stop devtest

## Use a tmpfs mount in a container

To use a tmpfs mount in a container, use the --tmpfs flag, or use the --mount flag with type=tmpfs and destination options. There is no source for tmpfs mounts. The following example creates a tmpfs mount at/app in a Nginx container. The first example uses the --mount flag and the second uses the --tmpfs flag.

--mount

$ docker run -d \

-it \

--name tmptest \

--mount type=tmpfs,destination=/app \

nginx:latest

--tmpfs

$ docker run -d \

-it \

--name tmptest \

--tmpfs /app \

nginx:latest

Verify that the mount is a tmpfs mount by running docker container inspect tmptest and looking for the Mountssection:

"Tmpfs": {

"/app": ""

},

Remove the container:

$ docker container stop tmptest

$ Docker container rm tmptest

### **Specify tmpfs options**

tmpfs mounts allow for two configuration options, neither of which is required. If you need to specify these options, you must use the --mount flag, as the --tmpfs flag does not support them.

| **Option** | **Description** |
| --- | --- |
| tmpfs-size | Size of the tmpfs mount in bytes. Unlimited by default. |
| tmpfs-mode | File mode of the tmpfs in octal. For instance, 700 or 0770. Defaults to 1777 or world-writable. |

The following example sets the tmpfs-mode to 1770, so that it is not world-readable within the container.

docker run -d \

-it \

--name tmptest \

--mount type=tmpfs,destination=/app,tmpfs-mode=1770 \

nginx:latest

Sharing Data between Containers

* If you have some persistent data that you want to share between containers, or want to use from non-persistent containers, it’s best to create a named Data Volume Container, and then to mount the data from it.
* $ docker **create** -v /dbdata *--name dbstore mysql /bin/true*
* *$ docker run --name mysql01 --volumes-from dbstore -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql*
* $docker exec -it mysql01 bash
* $df –h
* docker run --name mysql02 --volumes-from dbstore -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql

Backup and Restore from Volume

* Another useful function we can perform with volumes is use them for backups, restores or migrations. We do this by using the --volumes-from flag to create a new container that mounts that volume
* Backup
  + $ docker run --rm --volumes-from dbstore -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /dbdata
* Restore
  + $ docker **run** -v /dbdata --name dbstore2 ubuntu /bin/bash
  + $ docker run --rm --volumes-from dbstore2 -v $(pwd):/backup ubuntu bash -c "cd /dbdata && tar xvf /backup/backup.tar --strip 1"