Volumes top-level element

Table of contents

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.

To use a volume across multiple services, you must explicitly grant each service access by using the volumes attribute within the `services` top-level element. The `volumes` attribute has additional syntax that provides more granular control.

> **Tip**
>
> Working with large repositories or monorepos, or with virtual file systems that are no longer scaling with your codebase? Compose now takes advantage of Synchronized file shares and automatically creates file shares for bind mounts. Ensure you're signed in to Docker with a paid subscription and have enabled both **Access experimental features** and **Manage Synchronized file shares with Compose** in Docker Desktop's settings.

Example

The following example shows a two-service setup where a database's data directory is shared with another service as a volume, named `db-data`, so that it can be periodically backed up.

```
services:
 backend:
   image: example/database
   volumes:
     - db-data:/etc/data

 backup:
   image: backup-service
   volumes:
     - db-data:/var/lib/backup/data

volumes:
 db-data:
```

The `db-data` volume is mounted at the `/var/lib/backup/data` and `/etc/data` container paths for backup and backend respectively.

Running `docker compose up` creates the volume if it doesn't already exist. Otherwise, the existing volume is used and is recreated if it's manually deleted outside of Compose.

Attributes

An entry under the top-level `volumes` section can be empty, in which case it uses the container engine's default configuration for creating a volume. Optionally, you can configure it with the following keys:

driver

Specifies which volume driver should be used. If the driver is not available, Compose returns an error and doesn't deploy the application.

```
volumes:
 db-data:
   driver: foobar
```

driver_opts

`driver_opts` specifies a list of options as key-value pairs to pass to the driver for this volume. The options are driver-dependent.

```
volumes:
 example:
   driver_opts:
     type: "nfs"
```

```
    o: "addr=10.40.0.199,nolock,soft,rw"
    device: ":/docker/example"
```

external

If set to `true`:

- `external` specifies that this volume already exists on the platform and its lifecycle is managed outside of that of the application. Compose doesn't then create the volume, and returns an error if the volume doesn't exist.
- All other attributes apart from `name` are irrelevant. If Compose detects any other attribute, it rejects the Compose file as invalid.

In the example below, instead of attempting to create a volume called `{project_name}_db-data`, Compose looks for an existing volume simply called `db-data` and mounts it into the `backend` service's containers.

```
services:
 backend:
   image: example/database
   volumes:
     - db-data:/etc/data

volumes:
 db-data:
   external: true
```

labels

`labels` are used to add metadata to volumes. You can use either an array or a dictionary.

It's recommended that you use reverse-DNS notation to prevent your labels from conflicting with those used by other software.

```
volumes:
 db-data:
   labels:
     com.example.description: "Database volume"
     com.example.department: "IT/Ops"
     com.example.label-with-empty-value: ""

volumes:
 db-data:
   labels:
     - "com.example.description=Database volume"
     - "com.example.department=IT/Ops"
     - "com.example.label-with-empty-value"
```

Compose sets `com.docker.compose.project` and `com.docker.compose.volume` labels.

name

`name` sets a custom name for a volume. The name field can be used to reference volumes that contain special characters. The name is used as is and is not scoped with the stack name.

```
volumes:
 db-data:
   name: "my-app-data"
```

This makes it possible to make this lookup name a parameter of the Compose file, so that the model ID for the volume is hard-coded but the actual volume ID on the platform is set at runtime during deployment.

For example, if `DATABASE_VOLUME=my_volume_001` in your `.env` file:

```
volumes:
 db-data:
   name: ${DATABASE_VOLUME}
```

Running `docker compose up` uses the volume called `my_volume_001`.

It can also be used in conjunction with the `external` property. This means the name of the volume used to lookup the actual volume on the platform is set separately from the name used to refer to it within the Compose file:

```
volumes:
 db-data:
```

```
external:
  name: actual-name-of-volume
```