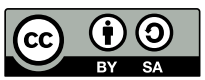
Cloud computing

Volumes: data

Aleix Llusà Serra

Master's degree in Machine Learning and Cybersecurity for Internet Connected Systems
Universitat Politècnica de Catalunya
https://epsem.upc.edu

24 de setembre de 2026



This work is licensed under a Creative Commons "Attribution-ShareAlike 4.0 International" license.

Data

Recall: An image is all except data. When deploying containers all (except data) comes zero from image. So, how to persist data?

Volumes

- Persist data
- Share* data among multiple containers (*data corruption?)
- Easy to backup
- Extensible by volume drivers (remote volumes, encryption...)

Volume in compose

app/compose.yml

```
services:
  db:
    image: mysql:latest
    environment:
      - MYSQL_ROOT_PASSWORD=XXX
    volumes:
    - /in/your/host:/var/lib/mysql
```

Note: there is also a `docker run --volume`.

Volume in Dockerfile

app/web/Dockerfile

```
FROM nginx:latest
[...]
VOLUME /data
VOLUME /var/something
```

In case you forget to define volumes for containers, if they are defined in the Dockerfile at least docker will keep the data in that directories. That is the case for /var/lib/mysql in the mysql image [https://hub.docker.com/_/mysql/].

Volume in compose: there is more

https://docs.docker.com/engine/extend/legacy_plugins/#volume-plugins

app/compose.yml

```
services:
  db:
    image: mysql:latest
    environment:
      - MYSQL_ROOT_PASSWORD=XXX
    volumes:
    - my_db_data:/var/lib/mysql
volumes:
  my_db_data:
    driver_opts:
      type: "nfs"
      o: "addr=10.40.0.199,nolock,soft,rw"
      device: ":/docker/example"
```

About data

What is data? Difficult question.

- Persistent data: Essential/Critical data ⟶ DataBase
- Volatile data: Important or essential data for the application, however recoverable or recomputable ⟶ shared volumes
  - Graphic previews, aggregations computations, image scaling...(the generation can take long time)
  - Files being uploaded: if while uploading it fails, bad luck but you can upload again. Critical once uploaded, so apply DataBase theorems.
  - Shared data between multiple services/containers/processes. E.g. a TLS certificate: if you lose it you can generate it again (even though the generation can take long time)
  - Shared volatile data: caches, proxys...shared between services but can be discarded at any time or when rebooting.
- Logs ⟶ Very critical. You might send to another place: `https://docs.docker.com/engine/logging/configure/`
- Static data ⟶ Are you sure they are not configuration nor code?

About data

What is not data? More difficult question. What we do not need to keep:

- If the service needs data when starting ⟶ Maybe it's configuration, not data
- If the service must reboot when changing data ⟶ Maybe it's configuration, not data
- If data is non-persistent, you should use tmpfs mount (in the host's system memory). Remark: by default all files created inside a container are stored on a writable container layer, which can reduce performance as compared to using data volumes.
- If something is not data, in docker it should be part of the image, an environmental variable, a config, or a secret.

Conclusion: try to design your microservices as stateless as possible. When it's not possible, try to wisely think about whether you have data or another thing. And when it's actual data, think wisely whether it's critical.

Data backup

A copy of data to restore after a data loss.

What is needed in case of a disaster recovery?

- Only critical data + Infrastructure as code
- Other data is recoverable in case of disaster (even though it can take long time)
- Copy critical data regularly to other locations and preserve some historic
- For DBMS you may need specific dumps

Exercise: simple web filemanager

Deploy a simple filemanager, however with an extremely microservices orientation. Deploy three services:

- One http server for viewing in a browser the files in a server directory. Hint: containerise standard Python's `http.server`.
- One http server for uploading in a browser a file to a server directory. Hint: containerise `https://pypi.org/project/uploadserver/`.
- An nginx/Apache or similar to publish the service and to proxy to the other services.

Hints: a) You will need a shared volume between the viewing and the uploading service. b) You can use `working_dir` in the compose. c) You can use `command` in the compose.