LU05a - Einführung in Storage mit Containern

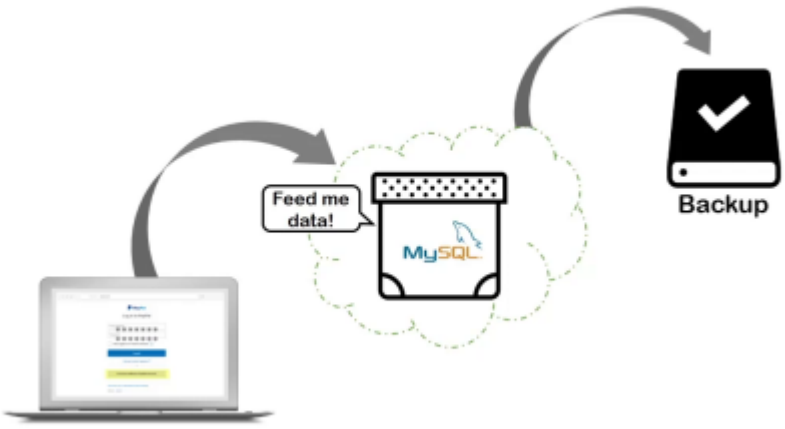
Ziele

1. Ich kann erklären, warum ich explizit Storage (Speicher) anlegen muss.
2. Ich kann Storage (Speicher) anlegen und verwalten.
3. Ich kann mögliche Fehlermeldung lesen und interpretieren.
4. Ich kann mindestens eine Massnahme bei Fehlermeldung vorschlagen und umsetzen.

How to use storage with container

Introduction

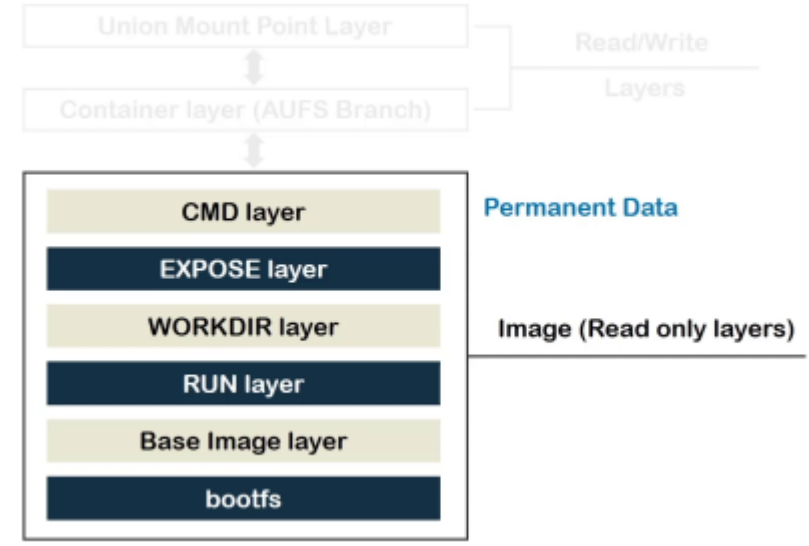
An efficient storage solution in container environment is as important as all topics mentioned till now. With containers the logic is pretty simple: containers data needs to be backed up somewhere as a permanent storage.



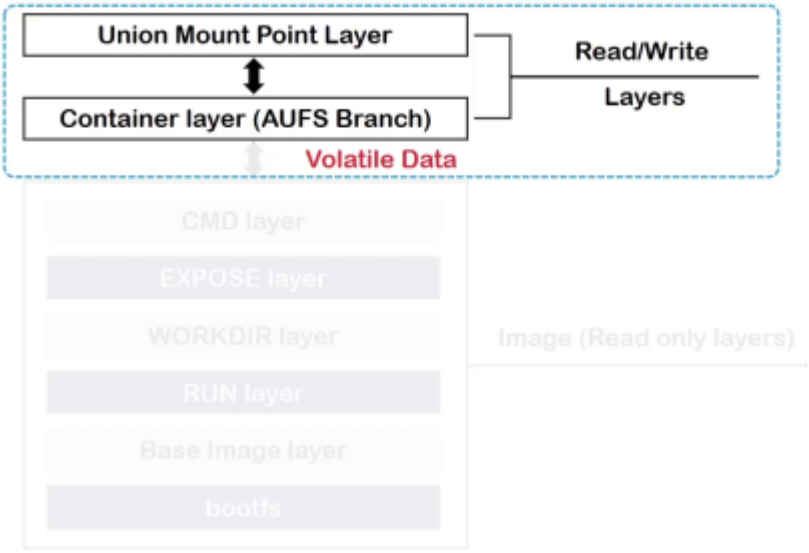
Read-Only vs. Read-Write

Take a look at layered structure of docker image and container data. There are two types of layers:

- read only layers which hold permanent data and is never modified due to copy on write policy and
- read write layers which hold temporary or volatile data.



If a container stops or dies, the volatile data vanishes.



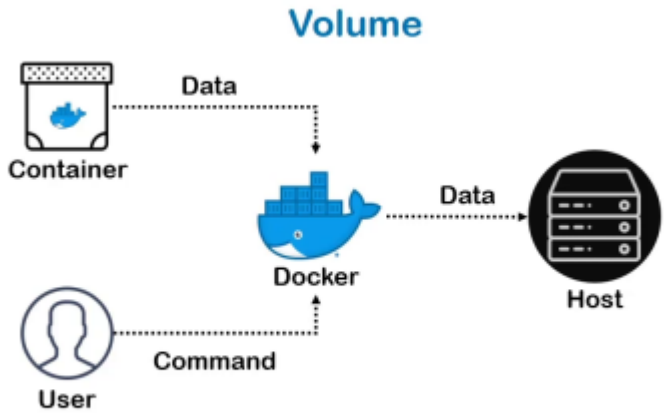
We need to back up the important data from the volatile read write layer of the container. Now the next question is where to store the data? Well, just anywhere. Do you want to store it on some machine which hosts docker? Go ahead. Do you want to store it to another server? Go ahead. Do you want to store it on cloud? Go ahead as well. And the last and genuine question which comes to my mind is.

Type of storage

Volumes

Most commonly used storage type is called a `volume`. In a `volume`, the container storage is completely isolated from the host file system. Although the data of `volume` is sorted in a specific directory of the host, they are controlled and managed by docker command line. Compared to other options of storage which we will visit soon enough, `volumes` are more secure to ship and more reliable to operate. `Volumes` are storage objects of docker which are mounted to containers. In terms of implementation, `volumes` are dedicated directories on hosts file system. If a containerized app is

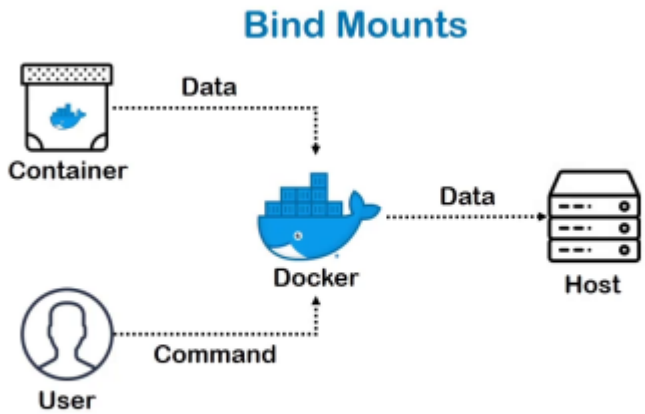
shipped along with the `volume`, people apart from the developer himself using the app will end up creating such a directory on their own docker hosts.



Container provides data to docker engine and user provides commands to store the data in the `volume`or to manage the data in the same. Although what container knows is just the name of the `volume`, not the path on the host. The translation takes place on docker machines end, so external applications having access to containers will have no means to access `volumes` directly.This isolation maintains the integrity and security of hosts and containers.

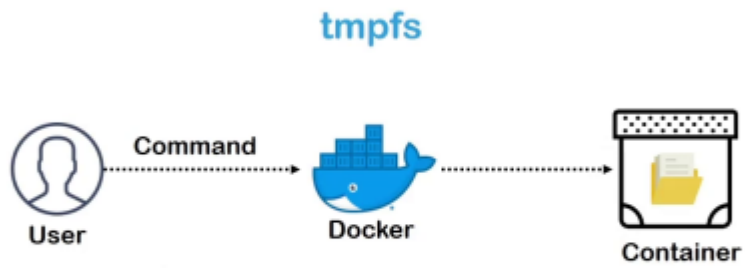
Bind mount

Second option is `bind mount` that allows to use any directory on docker host to store the data. While this might be convenient in some cases, it also exposes the storage location of the container which can have negative impact on the overall security of the application and the host itself. Apart from that other users may not have such path on their host and creating so may not be under their privileges.



Temporary file system (tempfs)

Finally we have `tempfs` or temporary file system. Volumes and bind mount let you share the files between host machine and container so that you can persist the data, even after the container is stopped. If you are running docker on Linux you have 3rd option: `tempfs` mounts. When you create a container with `tempfs` mount the container can create files outside the containers writable layer. As opposed to volumes and bind mounts, a `tempfs` mount is temporary persisted in the host memory, not in storage.



When the container stops, the `tempfs` mount is removed and the files written there won't be persisted. The only sensible use case which comes to my mind for `tempfs` is to store sensitive files which you don't want to persist once the application gets deleted. Something like the browsing history which gets deleted if we use the Incognito tab. Tempfs mounts have their limitations.

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m347/learningunits/lu05/lu05a?rev=1748962676

Last update: **2025/06/03 16:57**