

# Contents

- Manage data in Docker
  - Volumes
  - Bind mounts
  - tmpfs mounts



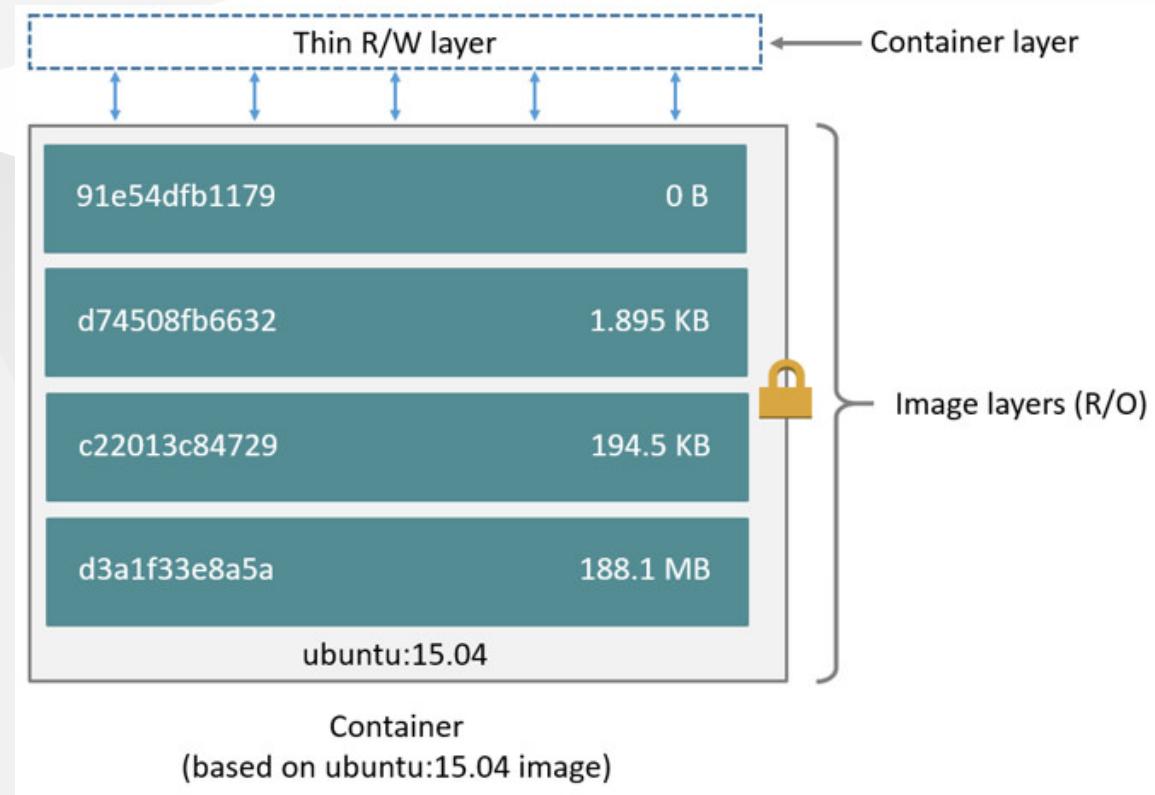
### Storage overview

이번 장은 Docker에서 데이터를 관리하는 방법에 관한 내용입니다.

컨테이너를 실행한 후 컨테이너 내부에서 생성된 모든 파일은 Writable container layer에 저장됩니다.

이전에 본 내용을 다시 떠올려 보겠습니다.

Writable container layer는 아래 그림에서 Thin R/W layer (Container layer)라고 표시된 부분입니다.

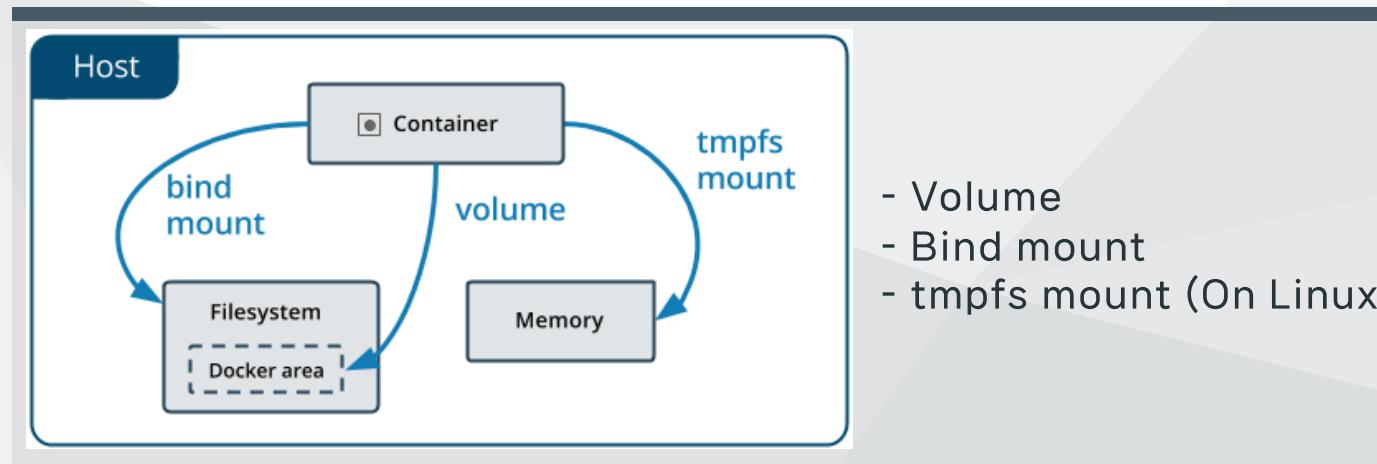


### Storage overview

이런 데이터 저장방법은 다음과 같은 문제점들을 가지고 있습니다.

- 컨테이너가 중지되면 Container layer의 데이터는 컨테이너와 함께 제거되고, 유지되지 않습니다.
- 다른 프로세스 또는 애플리케이션에서 컨테이너의 데이터에 접근하기가 쉽지 않습니다.
- 컨테이너의 Writable layer에 데이터를 기록하기 위해서는 파일시스템을 관리할 **Storage driver** 가 필요합니다.  
Storage driver는 리눅스의 커널을 이용해서 union filesystem을 제공하기 때문에, 이러한 추가적인 추상화로 인해 Host 파일시스템에 직접 데이터를 기록하는 것과 비교해보면 성능면에서 약점으로 작용합니다.

위에서 나열한 문제들을 해결하기 위해서, Docker는 Host 머신에 파일을 저장하는 방법을 제공합니다.



### Manage data in Docker

다음은 대표적인 데이터 저장 방법(마운트 유형)입니다.

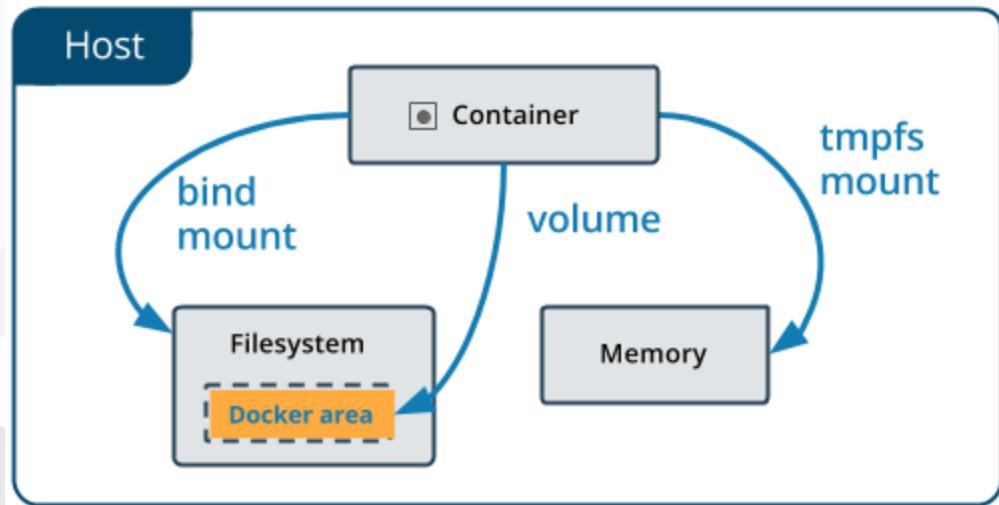
사용하기로 선택한 마운트 유형에 관계없이 데이터는 컨테이너 내에서 동일하게 보여지지만, 유형별 차이는 데이터가 저장되는 Host 머신의 위치에 있습니다.

- **Volumes** : Docker에 의해 관리되는 Host 파일시스템의 정해진 영역(e.g. `/var/lib/docker/volumes/` on Linux)에 데이터를 저장합니다.  
Volume은 Docker에서 영구 데이터를 관리하는 가장 좋은 방법입니다.
- **Bind mounts** : Host 머신의 파일시스템 어디에나 파일을 저장할 수 있는 방법입니다. Docker외의 다른 프로세스에서 데이터의 접근과 수정이 가능합니다.
- **tmpfs mounts** : Host의 메모리에 데이터를 저장하는 방법입니다. (파일시스템에는 저장되지 않음.)

Type of mount	Use cases
Volumes	<ul style="list-style-type: none"><li>- 컨테이너간 데이터 공유</li><li>- 데이터의 원격저장 (e.g. remote Host, Cloud provider)</li><li>- 고성능 I/O</li></ul>
Bind mounts	<ul style="list-style-type: none"><li>- Host 머신의 config.정보 공유(e.g. <code>/etc/resolve.conf</code>)</li><li>- Host 머신의 Source code나 Build artifact 공유</li></ul>
tmpfs mounts	<ul style="list-style-type: none"><li>- 민감정보(e.g. <code>secrets</code>)를 컨테이너 lifecycle동안 저장</li><li>- 영구저장이 필요없는 대용량 데이터의 처리 시 컨테이너의 성능보장</li></ul>

 [Manage data in Docker](#)

## Volumes



Docker에 의해 만들어지고 관리되는 Volume은 `docker volume create` 명령어로 명시적으로 생성할 수 있습니다.

```
ubuntu@ip-10-0-1-14:~$ docker volume create my-volume
volume1
ubuntu@ip-10-0-1-14:~$ docker volume ls
DRIVER      VOLUME NAME
local      my-volume
```

## Docker & Kubernetes - 03. Docker storage

Volume을 생성하면 Docker Host 머신에서 Docker에서 관리하는 특정영역에 디렉토리가 생성되고 사용됩니다.

```
ubuntu@ip-10-0-1-14:~$ docker volume inspect my-volume
[
  {
    "CreatedAt": "2022-06-26T05:40:29Z",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/my-volume/_data",
    "Name": "my-volume",
    "Options": {},
    "Scope": "local"
  }
]
ubuntu@ip-10-0-1-14:~$ sudo ls /var/lib/docker/volumes/
metadata.db      my-volume
```

/var/lib/docker/volumes/ 은 root:root ownership를 가지고 있으므로  
적절한 조회 명령을 사용해야 함. (e.g. sudo ls /var/lib/docker/volumes/ )

## Docker & Kubernetes - 03. Docker storage

Volume을 컨테이너 실행 시 마운트하면 이 디렉토리가 컨테이너의 지정된 디렉토리로 마운트됩니다.

```
ubuntu@ip-10-0-1-14:~$ docker run -it --name myubuntu --mount source=my-volume,target=/volumedata ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
405f018f9d1d: Pull complete
Digest: sha256:b6b83d3c331794420340093eb706a6f152d9c1fa51b262d9bf34594887c2c7ac
Status: Downloaded newer image for ubuntu:latest

root@20106b447a75:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr volumedata
root@20106b447a75:/# cd volumedata/
root@20106b447a75:/volumedata# touch hellovolume
root@20106b447a75:/volumedata# ls
hellovolume
root@20106b447a75:/volumedata# exit
exit

ubuntu@ip-10-0-1-14:~$ sudo ls /var/lib/docker/volumes/my-volume/_data
hellovolume
```

위의 예제는 my-volume이라는 Volume이 ubuntu 컨테이너의 /volumedata에 마운트되어 사용되는 예제입니다.

`--mount source=my-volume,target=/volumedata` 는 `--volume my-volume:/volumedata` 과 같이 사용할 수도 있습니다.

둘의 차이는 [Choose the -v or --mount flag](#) 를 참고하세요.

## Docker & Kubernetes - 03. Docker storage

### Volumes

Container 정보를 살펴보면 기본설정으로 Volume의 읽기쓰기모드가 RW(읽기쓰기) 인 것을 알 수 있습니다.

```
ubuntu@ip-10-0-1-14:~$ docker inspect myubuntu
[
  {
    "Id": "20106b447a758e6c263d646c1852e9e048919d9d8f168926d9f257c32442a810",
    "Created": "2022-06-26T05:43:23.834782021Z",
    "Path": "bash",
    "Args": [],
    ... 생략 ...
    "Mounts": [
      {
        "Type": "volume",
        "Name": "my-volume",
        "Source": "/var/lib/docker/volumes/my-volume/_data",
        "Destination": "/volumedata",
        "Driver": "local",
        "Mode": "z",
        "RW": true,
        "Propagation": ""
      }
    ],
    ... 생략 ...
  }
]
```

"RW": true, -> RW(읽기쓰기) 모드

## Docker & Kubernetes - 03. Docker storage

### Volumes

--mount 또는 --volume 옵션에 readonly 옵션을 추가해서 읽기쓰기 모드를 변경 할 수도 있습니다.

```
ubuntu@ip-10-0-1-14:~$ docker run -it --name myubuntu_ro --volume my-volume:/volumedata:ro ubuntu
root@9a16f5ab69b1:/# touch /volumedata/test
touch: cannot touch '/volumedata/test': Read-only file system
root@9a16f5ab69b1:/# exit
exit
ubuntu@ip-10-0-1-14:~$ docker inspect myubuntu_ro
[
  {
    "Id": "9a16f5ab69b1b2e887dcf28b397e4338d00786cec4a220b990079d37fbc3fb60",
    ... 생략 ...
    "Mounts": [
      {
        "Type": "volume",
        "Name": "my-volume",
        "Source": "/var/lib/docker/volumes/my-volume/_data",
        "Destination": "/volumedata",
        "Driver": "local",
        "Mode": "z",
        "RW": false,
        "Propagation": ""
      }
    ],
    ... 생략 ...
  }
]
```

"RW": false, -> Readonly 모드

## Docker & Kubernetes - 03. Docker storage

### Volumes

Volume은 동시에 여러 컨테이너에 마운트 할 수 있으며, 자동으로 제거되지 않습니다. 제거하기 위해서는 `docker volume rm`, `docker volume prune` 명령어로 제거 할 수 있습니다.

```
ubuntu@ip-10-0-1-14:~$ docker volume rm my-volume  
my-volume
```

사용하지 않는 모든 Volume을 제거하려면 `docker volume prune` 명령어를 사용하면 됩니다.

```
ubuntu@ip-10-0-1-14:~$ docker volume prune  
WARNING! This will remove all local volumes not used by at least one container.  
Are you sure you want to continue? [y/N] y
```

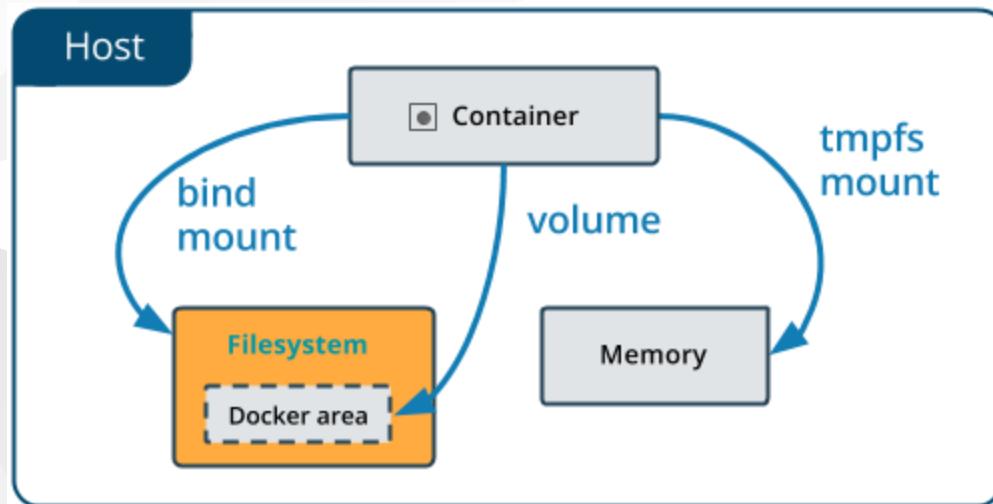
### volume 관련 명령어

Command	Description
<code>docker volume create</code>	Create a volume
<code>docker volume inspect</code>	Display detailed information on one or more volumes
<code>docker volume ls</code>	List volumes
<code>docker volume rm</code>	Remove one or more volumes
<code>docker volume prune</code>	Remove all unused local volumes

### Bind mount

Bind mount는 Volume에 비해 기능이 제한되어 있습니다.

Bind mount를 사용하면 Host 머신의 특정 파일이나 디렉토리가 컨테이너에 마운트되고, Host 머신의 마운트 경로는 절대경로로 참조됩니다.



### Bind mount

bind mount를 사용해서 컨테이너를 실행해보겠습니다.

```
ubuntu@ip-10-0-1-14:~$ docker run -it -v /volume/bindmount:/data/bindmount ubuntu
root@53bd95362964:/# cd /data/bindmount
root@53bd95362964:/data/bindmount# touch testfile
root@53bd95362964:/data/bindmount# ls
testfile
root@53bd95362964:/data/bindmount# exit
exit
ubuntu@ip-10-0-1-14:~$ ls /volume/bindmount/
testfile
```

`-v /volume/bindmount:/data/bindmount` 는 `--mount type=bind,source=/volume/bindmount,target=/data/bindmount` 과 같이 사용할 수도 있습니다.  
`-v` 는 Host 머신에 디렉토리가 존재하지 않는경우 자동으로 생성해줍니다. 반면 `--mount` 는 오류만 생성합니다.(디렉토리가 있어야 합니다.)  
둘의 차이는 [Choose the -v or --mount flag](#) 를 참고하세요.

### Bind mount

Bind mount는 Volume에 비해 아래와 같은 불리한 점이 있기 때문에, 가능하다면 Volume을 사용하는 것이 권장됩니다.

- Bind mount는 Host의 디렉토리 구조에 의존적입니다.
- Docker에 의해 관리되어지지 않습니다.
- Container의 프로세스가 Host의 파일시스템을 변경할 수 있으므로 보안상 위협이 될 수 있습니다.

### Summary

- Manage data in Docker
  - Volumes : Docker에 의해 관리되는 영역에 데이터를 저장
  - Bind mounts : Host 머신의 어느곳이나 지정하여 데이터를 저장
  - tmpfs mounts : Host 머신의 메모리에 데이터를 저장
- Mount 방법
  - `--mount` flag를 사용하는 방법
    - 사용 예시 : `--mount type=volume,source=my-volume,target=/volumedata,readonly`
  - `--volume` (`-v`) flag를 사용하는 방법
    - 사용 예시 : `--volume my-volume:/volumedata:ro`