

Docker 数据管理

郭宁@mobike

2018/09/25

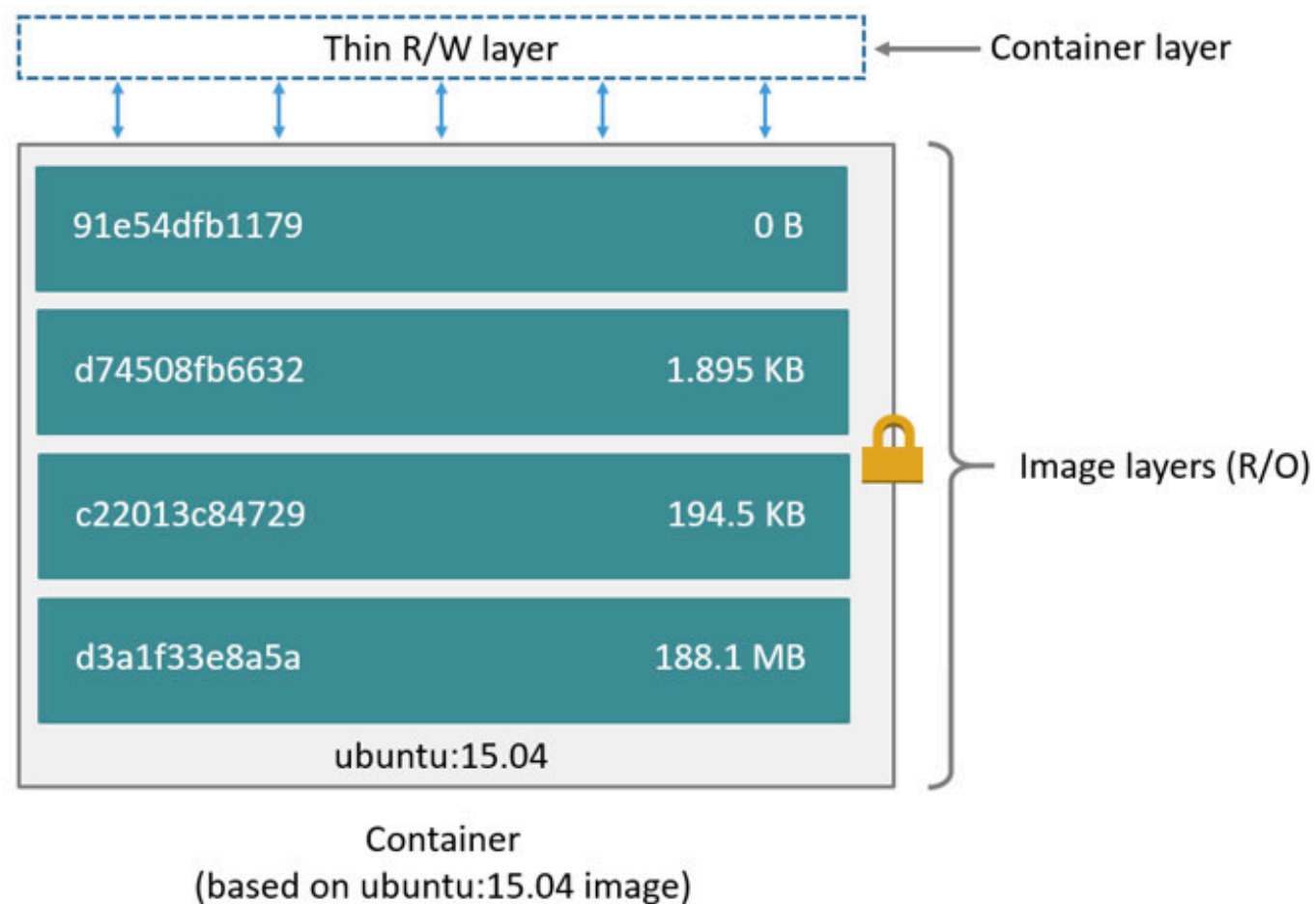
目录

- 现状 & 目标
- 数据管理的几种方式
 - bind mount
 - volume
 - tmpfs mount
- 总结汇总
- 参考资料

现状 & 目标

- 现状:

- 一个**镜像**，由多个层堆叠而成，都为**只读层**
- 当创建**容器**时，会在顶端新建一个**可写层**，被称为**容器层**
- 所有文件更改，都发生在**容器层**
- **容器删除时**，**容器层**也会删除



现状 & 目标

- 问题：

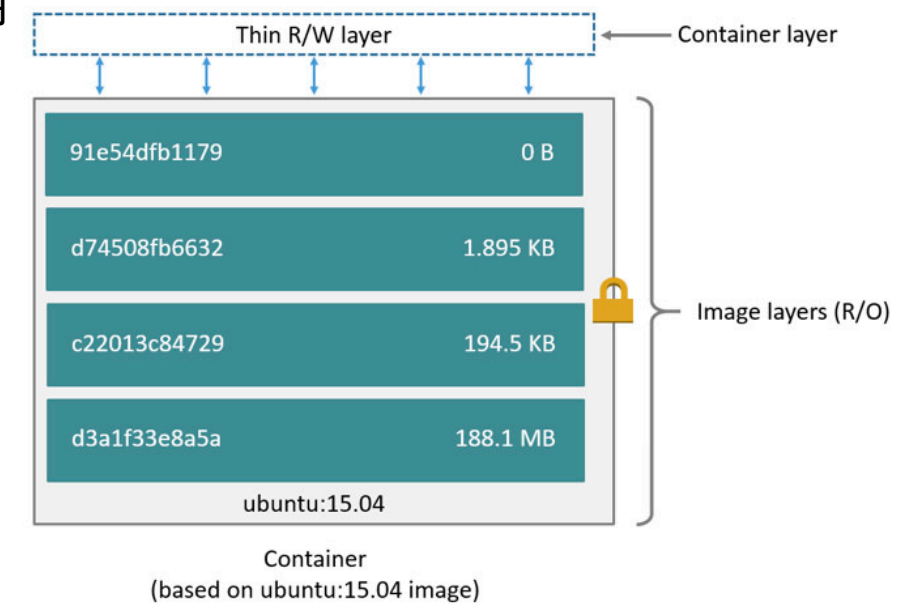
- 持久化：容器终止时，「容器层」数据不持久化
- 紧耦合：
 - 容器的可写层，硬绑定在宿主机上
 - 容器和存储的生命周期，硬绑定，无法单独管理数据的生命周期

- 目标：

- 持久化：容器终止后，数据仍存在，可以被其他容器复用
- 松耦合：
 - 数据不仅可以存储在宿主机，也可以存储在远端；
 - 数据不仅存储在 writeable layer 容器层，也可以写在其他地方；
- 数据共享
 - 容器内：数据持久化，跟宿主机之间数据共享，数据迁移
 - 容器间：数据共享

- 思路：

- 「容器」与「数据」，解耦

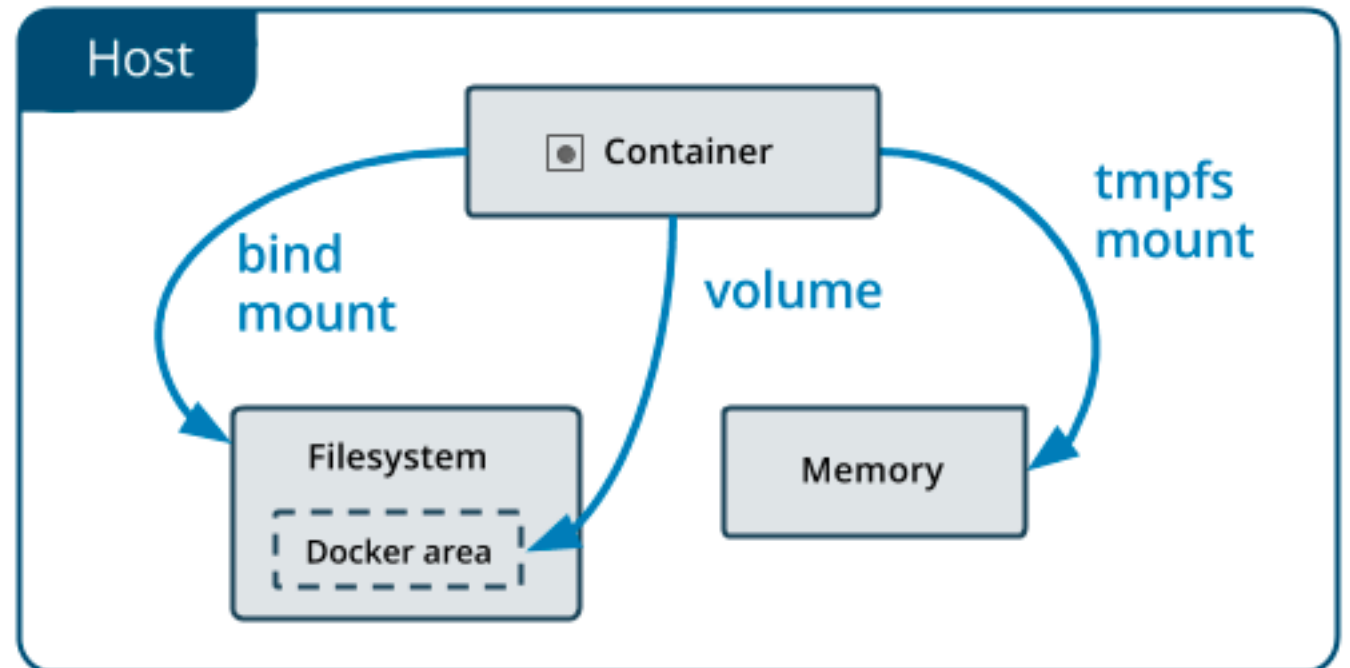


目录

- 现状 & 目标
- 数据管理的几种方式
 - bind mount
 - volume
 - tmpfs mount
- 总结汇总
- 参考资料

数据管理的几种方式

- **bind mount:** (早期方案)
 - 挂载「宿主机」FileSystem 目录
 - 持久化存储
- **volume:** (官方推荐)
 - 数据卷
 - 持久化存储
- **tmpfs mount:**
 - tmpfs 挂载内存 (仅限 Linux 系统)
 - 非持久化存储
 - 生命周期, 跟 container 绑定
 - 容器私有, 无法共享
 - 占用宿主机的内容, 而不是 container 内存



数据管理的几种方式

- 如何创建/挂载 volume、bind mount、tmpfs mount:
 - **Docker 17.06 之前**: -v/--volume/--tmpfs 参数设定
 - **Docker 17.06 之后**: **--mount** 参数设定, 更清晰

关于 --mount 的用法:

```
--mount <key>=<value>,<key>=<value>,<key>=<value>
```

其中, key 的具体取值说明:

key	是否必须	说明	value	备注
type	否	类别	<ul style="list-style-type: none">• volume (默认)• bind: bind mount• tmpfs	
source	否	<ul style="list-style-type: none">• 源地址• 别名: src	<ul style="list-style-type: none">• 已经命名的 volume: volume name• 匿名的 volume: 忽略此字段	
destination	是	<ul style="list-style-type: none">• 目标目录• 别名: dst、target	<ul style="list-style-type: none">• 容器内的目录路径 (目录 or 文件)	
readonly	否	<ul style="list-style-type: none">• 设定为只读• 仅限 bind mount 模式		
volume-opt	否	<ul style="list-style-type: none">• 传递 option 参数• 可以多次使用 volume-opt 以此传递多个• 通过 docker volume inspect 可查看	<ul style="list-style-type: none">• key=value 形式	

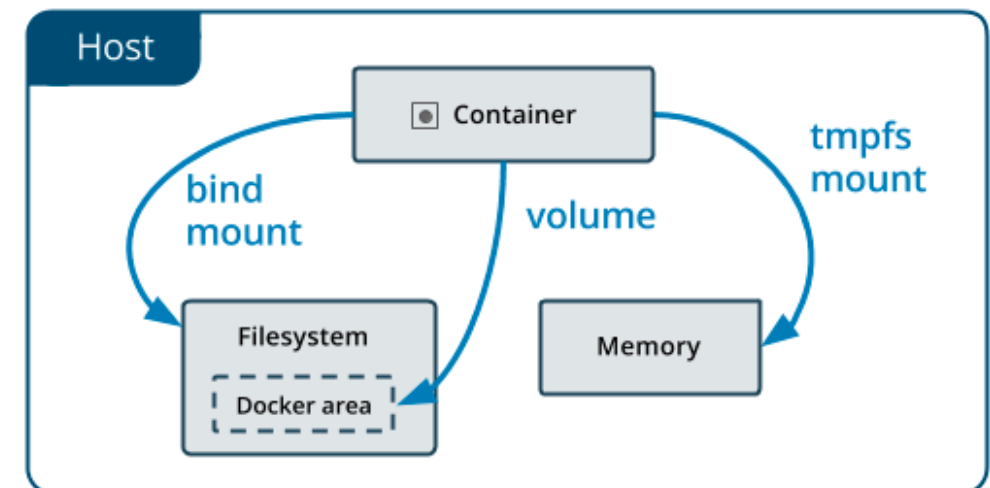
数据管理的几种方式

- 如何分析 volume、bind mount、tmpfs mount 的挂载信息：
 - *docker inspect [containerId]*

分类	参数	含义	备注
使用	Type	类别	• volume
	Name	名称，唯一标志	
	Source	对应「宿主机」的位置	
	Destination	「容器」中挂载的位置	
	Driver	驱动器	• local：本地宿主机 FS
	Mode	模式	• z：多容器之间，在共享 • Z：单个容器独占 • ro：bind mount 下，read only
	RW	可读可写	• true：可读可写 • false：只读
	Propagation	传播规则，不同挂载点之间，是否同步触发挂载	• rprivate：默认 • 使用限制： <ul style="list-style-type: none">• 只在 bind mount + Linux 宿主机时，可配置• 需要宿主机的 FS 支持 • 用法：高级特性，使用的时候，再查询

bind mount


- 关于 Bind mount: (Docker 的早期方案)
 - 作用: 依赖 bind mount, 将「宿主机 FS」上的文件 or 目录, 挂载到容器内
 - 具体用法:
 - 使用宿主机的「完整路径名」(绝对路径) 或「相对路径名」
 - 宿主机的目录或文件, 如果不存在, 则, 在挂载过程中, 会自动创建
 - 仅限于使用 -v 和 --volume 方式, 进行 bind mount, 此时, 自动创建的都是「目录」
 - --mount 方式, 不会自动创建, 而会抛出异常
- 使用建议:
 - 优先使用 **volume**
 - 无法使用 docker CLI 的命令行, 管理 bind mount
 - container 中进程, 直接读写 FS 上重要文件, 很灵活, 但需谨慎



bind mount

1. bind mount: 创建 container 时, 同步 bind mount

```
$ docker run -d \  
  -it \  
  --name devtest \  
  --mount type=bind,source="$(pwd)"/target,target=/app \  
  nginx:latest
```



docker: Error response from daemon: invalid mount config for type "bind": bind source path does not exist.

2. 分析 bind mount

```
$ docker inspect devtest
```


...

```
  "Mounts": [  
    {  
      "Type": "bind",  
      "Source": "/Users/guoning/ningg/github/docker-learn/volume",  
      "Destination": "/app",  
      "Mode": "",  
      "RW": true,  
      "Propagation": "rprivate"  
    }  
  ],  
  ...
```



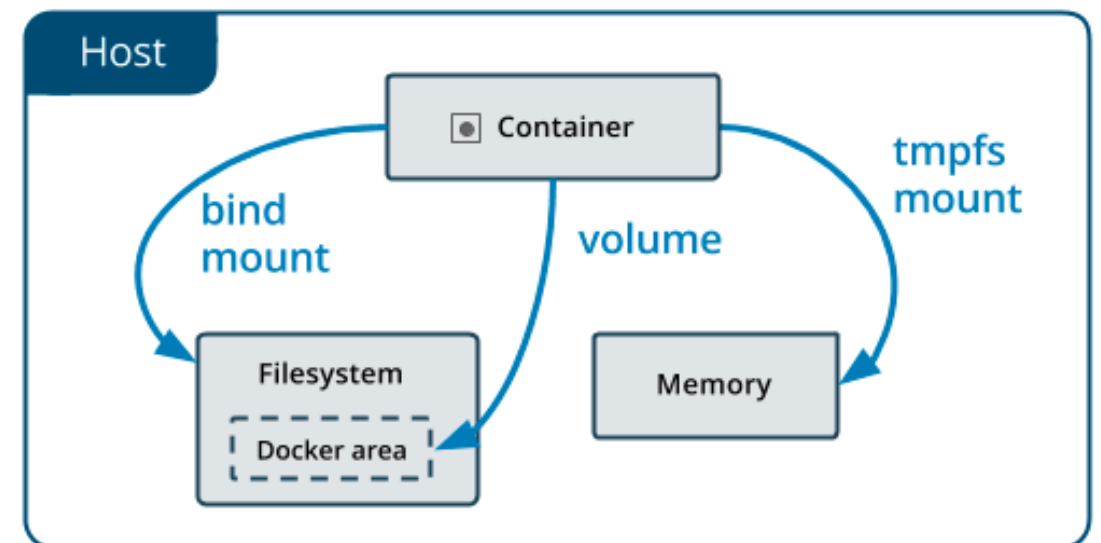
3. 只读模式 bind mount

```
$ docker run -d \  
  -it \  
  --name devtest \  
  --mount type=bind,source="$(pwd)"/target,target=/app,readonly \  
  nginx:latest
```



volume

- 作用：（官方推荐）
 - 数据存储在「宿主机 FS」上，为 **Docker** 独占的空间，非 Docker 进程不应该修改
- 注意事项：
 - 容器之间，数据共享
 - 容器终止，数据仍存在
 - 多容器，同时挂载
 - 远端存储：数据存储到远端
 - 宿主机，无法提前确认「目录已存在」时
 - 首次挂载时，自动创建目录
 - volume 是逻辑视图



volume

a. 创建 volume

```
docker volume create my-vol
```

b. 查看 volume 列表

```
docker volume ls
```

c. 分析 volume 详情

```
docker volume inspect my-vol
```

```
...
```

```
[
```

```
{
```

```
    "CreatedAt": "2018-09-20T06:36:08Z",
```

```
    "Driver": "local",
```

```
    "Labels": {},
```

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

```
    "Name": "my-vol",
```

```
    "Options": {},
```

```
    "Scope": "local"
```

```
}
```

```
]
```

```
...
```

d. 删除

```
docker volume rm my-vol
```

volume

1. 自动创建 volume 绑定 volume, Docker 会自动创建对应 volume

```
$ docker run -d \  
  --name devtest \  
  --mount source=myvol2,target=/app \  
  nginx:latest
```

查看 volume 列表

```
$ docker volume ls
```

DRIVER	VOLUME NAME
local	myvol2

查看 container 对应的挂载点

```
$ docker inspect [containerId]|[containerName]
```

...

```
"Mounts": [  
  {  
    "Type": "volume",  
    "Name": "myvol2",  
    "Source": "/var/lib/docker/volumes/myvol2/_data",  
    "Destination": "/app",  
    "Driver": "local",  
    "Mode": "z",  
    "RW": true,  
    "Propagation": ""  
  }  
],
```

...

2. 创建只读的 volume

```
$ docker run -d \  
  --name=nginxtestReadOnly \  
  --mount source=nginx-vol,destination=/usr/share/nginx/html,readonly \  
  nginx:latest
```


volume（数据共享）

- 分为 2 个方面：
 - 同一宿主机，容器间，数据共享
 - 不同宿主机，容器间，数据共享

volume (数据共享)


1. 创建 volume: 创建一个 container, 并创建 volume

```
$ docker run -d \  
  --name devtest \  
  --mount source=myvol2,target=/app \  
  nginx:latest
```



登录容器, 查看目标目录下, 文件列表

```
$ docker exec -it devtest /bin/bash 同一宿主机, 容器见, volume 共享  
$ cd /app  
$ ls -alh
```



2. 容器间, 共享 volume: 创建另一个 container, 共享上述 volume

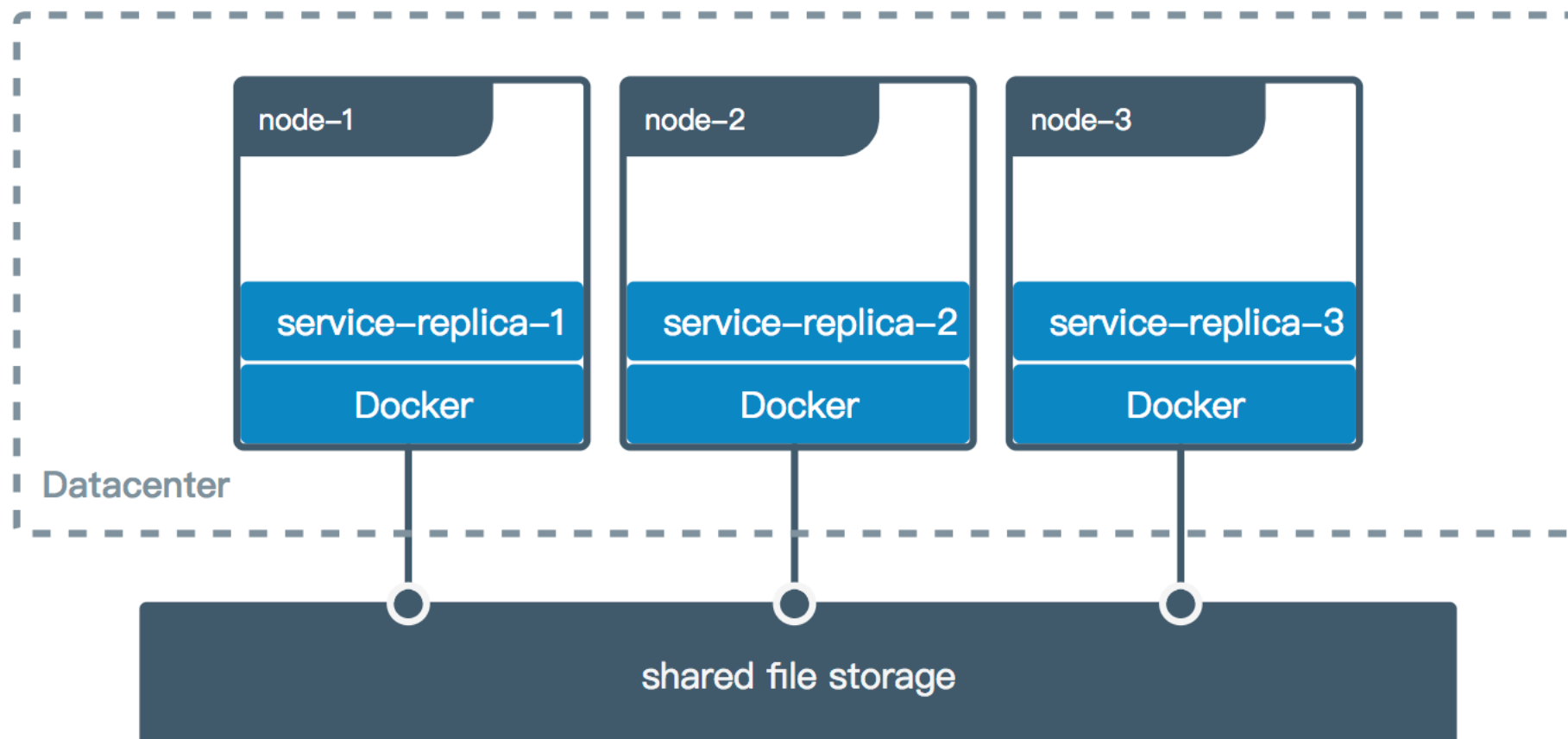
```
$ docker run -d \  
  --name=ngintest \  
  --mount source=myvol2,destination=/usr/share/nginx/html \  
  nginx:latest
```

登录容器, 查看目标目录下, 文件列表

```
$ docker exec -it devtest /bin/bash  
$ cd /app  
$ ls -alh
```

volume（数据共享）

- 不同宿主机，实现「容器间数据共享」，基本上，都是 2 个思路可选：
 - 应用层处理：应用上，将存储抽取为独立的业务逻辑，直接对远端存储，进行读写
 - 驱动层处理：创建 volume 时，使用支持远端存储的驱动，例如，可以直接对 NFS、Amazon S3 等进行读写的驱动；跟应用层解耦了，但依赖于「驱动」



volume（数据共享）

下述示例，涵盖 2 个场景：

1. 独立创建 volume：使用 vieux/sshfs 的 volume 驱动，单独创建一个 volume
2. 伴随创建 volume：创建 container 过程中，创建 volume

具体示例：

场景 A：创建独立的 volume

1. 安装插件

```
$ docker plugin install --grant-all-permissions vieux/sshfs
```

2. 创建 volume

```
$ docker volume create --driver vieux/sshfs \  
-o sshcmd=test@node2:/home/test \  
-o password=testpassword \  
sshvolume
```

场景 B：创建 container 过程中，创建 volume

```
$ docker run -d \  
--name sshfs-container \  
--volume-driver vieux/sshfs \  
--mount src=sshvolume,target=/app,volume-opt=sshcmd=test@node2:/home/test,volume-opt=password=testpassword \  
nginx:latest
```

tmpfs mount

- 关于 tmpfs mount:
 - 使用场景:
 - 非持久化存储
 - 在 Container 存活期间, 使用
 - 「Linux 版本」的宿主机
 - 用于存储:
 - 非持久化的状态 or 敏感信息
 - 实例:
 - swarm 集群管理中, 使用 tmpfs mount 来挂载 secrets (密钥)

tmpfs mount

使用实例：

```
# 1. 创建 tmpfs mount
$ docker run -d \
  -it \
  --name tmptest \
  --mount type=tmpfs,destination=/app \
  nginx:latest

# 2. 分析 mount
$ docker inspect tmptest
...
  "Mounts": [
    {
      "Type": "tmpfs",
      "Source": "",
      "Destination": "/app",
      "Mode": "",
      "RW": true,
      "Propagation": ""
    }
  ],
  ...

# 3. 设置参数: tmpfs 内存大小 和 mode, 默认为物理内存大小, tmpfs-size (单位 Byte) , tmpfs-mode (rwx, 1777)
$ docker run -d \
  -it \
  --name tmptest \
  --mount type=tmpfs,destination=/app,tmpfs-mode=1770 \
  nginx:latest
```

关于 tmpfs-mode 的默认 1777 模式，其中使用了 sticky bit：（约束 删除、移动等特殊的写权限，只有 owner 才有权限）

- 细节，参考：[linux特殊文件权限 suid sgid sticky-bit](#), [Linux文件权限：Sticky bit, SUID, SGID](#)

目录

- 现状 & 目标
- 数据管理的几种方式
 - bind mount
 - volume
 - tmpfs mount
- 总结汇总
- 参考资料

总结

分类	适用场景	相对优势	备注
volume	<ul style="list-style-type: none">容器之间，数据共享<ul style="list-style-type: none">首次挂载时，自动创建目录容器终止，数据仍存在多容器，同时挂载宿主机，无法提前创建目录<ul style="list-style-type: none">volume 是逻辑视图在具体使用时，会自动创建远端存储：数据存储到远端数据备份：直接从磁盘上，进行备份	<p>首选方案。</p> <p>相对 bind mount 的优势：</p> <ul style="list-style-type: none">使用 docker CLI 命令行工具管理Linux 和 Windows 容器，都可使用容器间，更安全的共享数据存储到远端方便备份数据的生命周期，脱离于容器不需要提前创建目录	<ul style="list-style-type: none">完全由 Docker 管理安全：只能被 Docker 进程访问自动创建时，默认参数是什么？volume 的参数，有哪些？
bind mount	<ul style="list-style-type: none">容器和宿主机之间，数据共享<ul style="list-style-type: none">DNS 解析时，容器共用宿主机配置文件容器和宿主机之间，代码共享<ul style="list-style-type: none">本地编译的代码，容器中，立即可见宿主机，提前明确创建目录 or 文件无法使用 docker CLI，管理 bind mount	<ul style="list-style-type: none">直接对 FS 文件读写，很灵活，但需谨慎	
tmpfs mount	<ul style="list-style-type: none">不想持久化的数据<ul style="list-style-type: none">宿主机上，不想持久化容器中，也不想持久化安全数据 or 性能考虑，只存在内存中	<p>使用约束：</p> <ul style="list-style-type: none">只在 Linux 宿主机，可以使用生命周期，跟 container 绑定容器私有，无法共享占用宿主机的内容，而不是 container 内存？	

参考资料

- FinTalk-20180925-Docker: 数据管理
 - <https://wiki.mobike.com/pages/viewpage.action?pageId=34475512>
- Manage data in Docker: <https://docs.docker.com/storage/>

Q & A

