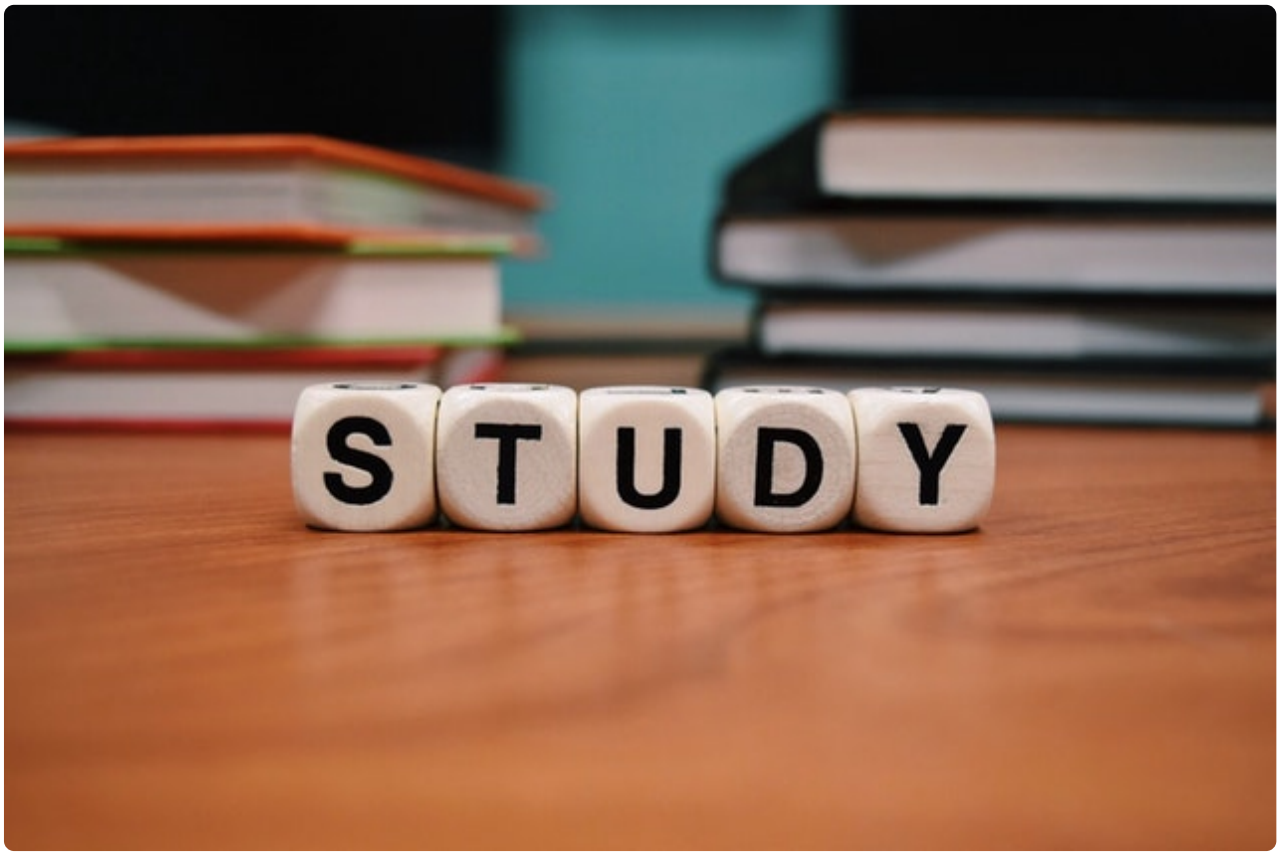
21 数据共享：volume 的使用指南

更新时间：2020-09-02 10:06:20



> 人不可有傲气，但不可无傲骨。——徐悲鸿

在上一篇文章中，我们介绍了 Docker 数据存储的三种模式：***volume***，***bind mounts***，***tmpfs***。主要还是偏理论部门，这篇文章我们就来看一下 volume 的典型使用场景。

1. 背景

我们知道 Docker 的镜像是由多个只读层组合而成的，运行的容器在这些只读层之上加入一个读写层。这种设计主要提高了镜像构建、存储和分发的效率，但是这种设计带来了一些使用上的不便：

- 容器中的文件存储在宿主机上面形式复杂，不能在宿主机上方便地访问和修改。
- 容器运行过程中的数据都存储在读写层，也就意味着容器停止之后，这些数据都将丢失。
- 容器之间的数据无法直接共享使用。

volume 正是诞生于这个背景下。volume 存在于一个或多个容器中的特定文件或者文件夹，这个目录能够以联合文件系统的形式在宿主机中存在，并未数据的共享和持久化提供一下便利，包括：

- volume 在容器创建时会自动初始化，容器运行起来就可以直接使用。当然你也可以自己先创建 volume。
- volume 可以在不同容器之间共享和重用。
- 对 volume 中数据的操作会立刻生效。
- 对 volume 中数据的操作不会影响到镜像本身。

- volume 的生命周期独立于容器的生命周期，也就是说即使删除容器，volume 依然可以使用。

下面我们重点介绍一下 volume 的几种典型使用场景。

2. 先创建 volume

我们可以通过命令 `docker volume create` 创建 volume。

```
[root@docker ~]# docker volume create myvol2
myvol2
```

上面的命令就是创建一个名字叫 ***my-vol*** 的 volume。创建完之后，这个 volume 的数据会被存储在 `/var/lib/docker/volumes/myvol2/` 。其实这个目录 `/var/lib/docker/volumes/` 下面会保存所有 volume，如果 volume 没有名字，则 volume 对应的是一个长字符串，下面是我的 Docker 宿主机上面的 volumes 列表。

```
[root@docker ~]# ls /var/lib/docker/volumes/
0d677566872e112e6792b7dd1e71f4b5c26fec701de4d43fe401fd1d5bd93afd/
12a0226aff0e607425bd2f8ed6544154ec276feda24dee39255e377b978d4014/
22340dc6d144f4f4be30c93afc1186734f8559acb20aeeb861fa929d4c26e30b/
9bfb60bba113c35ab3c973edde821c493ec1b79a6a561d3eb6ca47105b12fba9/
a074cf769c98c44395e8ba9b11f473ccb296ccfef4a8b4de8d3d56632a4bb562/
a7fe694cc0abea99d1e455e31d25a49a523e4ff661f4172d48e3b61ccd00c2c0/
b769fe6cc2c9368299aea8d0a16ebd2911d1a15752d0c8e91706afe2974f70f8/
b801cc75e3485b5d90ed59cc38eeb86d96401c2698fb8598f73969c92fba4e48/
metadata.db
my-vol/
myvol2/
```

创建完 volume 之后，我们就可以按需使用创建出来 volume，使用参数 `-v` 或者 `--mount` 。

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

```
$ docker run -d \
  --name devtest \
  -v myvol2:/app \
  nginx:latest
```

容器启动之后，我们就可以通过命令 `docker inspect` 去查看容器中的 volume 信息如下，我们可以看到其中的 Source 字段即为 volume 在宿主机的目录。

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

3. 容器启动添加 volume

第一种使用方式比较简单，直接在创建容器的时候用 `-v` 的参数（或者使用 `--mount` 参数）创建一个 volume，这时候会创建出来一个匿名的 volume，也就是上一小节的那一串字符串。

```
[root@docker ~]# docker run --rm -it -v /data busybox sh
/ # ls /
bin   data  dev   etc   home  proc  root  sys   tmp   usr   var
```

然后我们使用 `docker inspect` 查看启动的容器的 volume 相关信息。

```
"Mounts": [
    {
        "Type": "volume",
        "Name": "41553e5efcc2ebce3e1ce35df0afa4e0a6b456a2583d2990184e32a2da525c65",
        "Source": "/var/lib/docker/volumes/41553e5efcc2ebce3e1ce35df0afa4e0a6b456a2583d2990184e32a2da525c65/_data",
        "Destination": "/data",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```

4. 从宿主机挂载

从宿主机挂载的应用场景是我们有时候想要将宿主机的文件共享到容器中，但是有不想停止容器，那么我们就可以通过这种方式来实现。使用方式是 `-v` 参数，格式为 `-v /host/dir:/container/dir` 。下面我们演示一下，首先我们在宿主机上面找到一个目录 `/root/demo` 。

```
[root@docker demo]# ls
hello.py  __pycache__  webapp.py  webapp.pyc
[root@docker demo]# pwd
/root/demo
```

启动容器。

```
[root@docker demo]# docker run -it -v /root/demo:/demo busybox sh
/ # ls /demo
__pycache__  hello.py     webapp.py    webapp.pyc
```

从上面的输出我们可以看到宿主机的目录以及映射到容器内部了。我们可以在宿主机上面的目录做一点改动，然后看一下对应的容器目录有没有同步改动。

```
[root@docker demo]# touch 123
[root@docker demo]# ls
123  hello.py  __pycache__  webapp.py  webapp.pyc
[root@docker demo]#
```

下面是容器内部对应的目录。

```
/ # ls /demo
123          __pycache__  hello.py     webapp.py    webapp.pyc
```

5. 使用 Dockerfile 添加 volume

在 dockerfile 的语法中可以通过 `VOLUME` 创建一个 volume 或者多个 volume。

```
#创建一个 volume
VOLUME /data
# 创建多个 volume
VOLUME ["/data1", "/data2"]
```

创建 volume 之后，Docker 会在容器启动时挂载一个 volume 到挂载点 `/data` 。如果镜像中存在目录 `/data` ，则这个文件夹中的文件都将全部被复制到宿主机中 volume 对应的文件夹中，一般位于目录 `/var/lib/docker/volumes/` 。

我们下面演示一下通过 busybox 作为 base 镜像构建自己的镜像，并创建一个 volume。

```
FROM busybox:latest
VOLUME /data
```

使用 `docker build` 基于此 Dockerfile 构建新的镜像。

```
[root@docker dockerfile2]# docker build -t volume-image:v1 .
Sending build context to Docker daemon  5.12kB
Step 1/2 : FROM busybox:latest
 ---> 6d5fcfe5ff17
Step 2/2 : VOLUME /data
 ---> Running in c74eeadfbf3f
Removing intermediate container c74eeadfbf3f
 ---> 71d5c091d1c0
Successfully built 71d5c091d1c0
Successfully tagged volume-image:v1
[root@docker dockerfile2]# docker images | grep volume-image
volume-image        v1                  71d5c091d1c0        13 seconds ago      1.22MB
[root@docker dockerfile2]#
```

启动容器，然后我们可以发现在根目录确实多了一个 /data 目录，这个挂载点挂载的就是我们新建的 volume。

```
[root@docker dockerfile2]# docker run -ti volume-image:v1 sh
/ # ls /
bin   data  dev   etc   home  proc  root  sys   tmp   usr   var
/ #
```

我们可以通过 `docker inspect <containerId>` 来查看其中的 volume 信息。其中的 Source 就是 volume 所在宿主机的位置 `/var/lib/docker/volumes/` 。

```
...
"Mounts": [
    {
        "Type": "volume",
        "Name": "b769fe6cc2c9368299aea8d0a16ebd2911d1a15752d0c8e91706afe2974f70f8",
        "Source": "/var/lib/docker/volumes/b769fe6cc2c9368299aea8d0a16ebd2911d1a15752d0c8e91706afe2974f70f8/_data",
        "Destination": "/data",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```

在 Dockerfile 中使用 volume 有一点需要注意的是，由于 volume 只有在容器创建的时候才会挂载进来，所以如果我们在 Dockerfile 中尝试将 volume 作为一个目录并做一下操作，这个是不会生效的，原因就是在当前镜像中这个文件夹还不存在。比如如下的操作：

```
VOLUME /data
RUN touch /data/file
```

正确做法是先创建该目录，执行我们需要的操作，最后添加 volume。

```
RUN mkdir /data
RUN touch /data/file
VOLUME /data
```

6. volume 共享

volume 共享可以通过参数 `--volumes-from` 实现。我们在上面启动了一个带有 volume 的容器，我们下面启动一个新的容器共享之前的 volume。

```
[root@docker ~]# docker ps | grep volume-image:v1
12be76fd253e        volume-image:v1     "sh"                20 minutes ago      Up 20 minutes                           modest_lumiere
[root@docker ~]# docker run -ti --volumes-from modest_lumiere busybox:latest sh
/ # ls
bin   data  dev   etc   home  proc  root  sys   tmp   usr   var
/ #
```

从上面的输出，我们可以看到新创建出来的 Docker 容器也有了目录 `/data` ，但是和之前的容器中的 volume 是不是同一个呢？我在之前启动的容器中的 `/data` 目录下创建一个临时文件。

```
/data # touch 1
/data #
```

然后我们在新的 Docker 容器中检查一下该目录下的文件内容。

```
/ # ls /data
1
```

当然你可以通过 `docker inspect` 查看这两个 Docker 容器中的 Mount 中的 volume 信息是不是相同即可。

我们上面的例子中是共享了一个 volume，这里有一个小问题，如果第一个容器中使用了多个 volume，那个通过 `--volumes-from` 共享过来的是不是所有的 volume 呢？答案是肯定的。那么如果要共享其中一个，可以做到吗？通过容器间共享的方式确实是做不到的，我们可以通过手动创建多个 volume，各个容器使用的按需挂载。

7. volume 备份与迁移

我们现在知道了 volume 的数据的存储位置：

- 容器内的指定挂载点
- 宿主机的 `/var/lib/docker/volumes`

这两者的关联可以通过 `docker inspcet` 来查看，那么我们备份的话是备份那部分数据呢？其实都可以。官方建议备份容器内的指定挂载点的数据，好处是可以通过自动化或者说程序化的方式来备份，也就是下面的命令。

```
$ docker run --rm --volumes-from dbstore -v $(pwd):/backup busybox tar cvf /backup/backup.tar /data
```

这行命令有以下几个作用：

1. 启动一个工具容器 busybox，和目标容器共享 volume
2. 将宿主机的当前目录和容器的目录 `/backup` 做映射
3. 将共享的 volume 的挂载点的数据压缩拷贝到容器内部的目录 `/backup`

这样通过三步操作就将原容器中的 volume 数据备份到宿主机上了。对应的我们可以通过下面的命令将备份的数据进行恢复，这里就不再解释了。

```
$ docker run --rm --volumes-from dbstore2 -v $(pwd):/backup ubuntu bash -c "cd /dbdata && tar xvf /backup/backup.tar --strip 1"
```

8. 总结

Volume 是 Docker 中非常有用的技术，它提供了一种将数据和镜像容器解耦的方式。本文介绍了 volume 的六种使用方式：

- 先创建 volume；
- 容器启动添加 volume；
- 从宿主机挂载；
- 在 Dockerfile 中使用 volume；
- volume 共享；
- volume 数据迁移和备份。

希望各位同学可以按自己的场景选择合适的使用方式。

}

20 数据存储：Docker 数据存储的三种模式

22 Dockerfile 你真的会用吗？