链滴

Docker 镜像与数据容器卷

作者：zouchanglin

原文链接：https://ld246.com/article/1564804078766

来源网站：链滴

Docker镜像

镜像是一种轻量级、可执行的独立软件包，用来打包软件运行环境和基于运行环境开发的软件，它包运行某个软件所需的所有内容，包括代码、运行时、库、环境变量和配置文件。

UnionFS（联合文件系统）

UnionFS（联合文件系统）：Union文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文系统下(unite several directories into a single virtual filesystem)。Union 文件系统是 Docker 镜的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜。

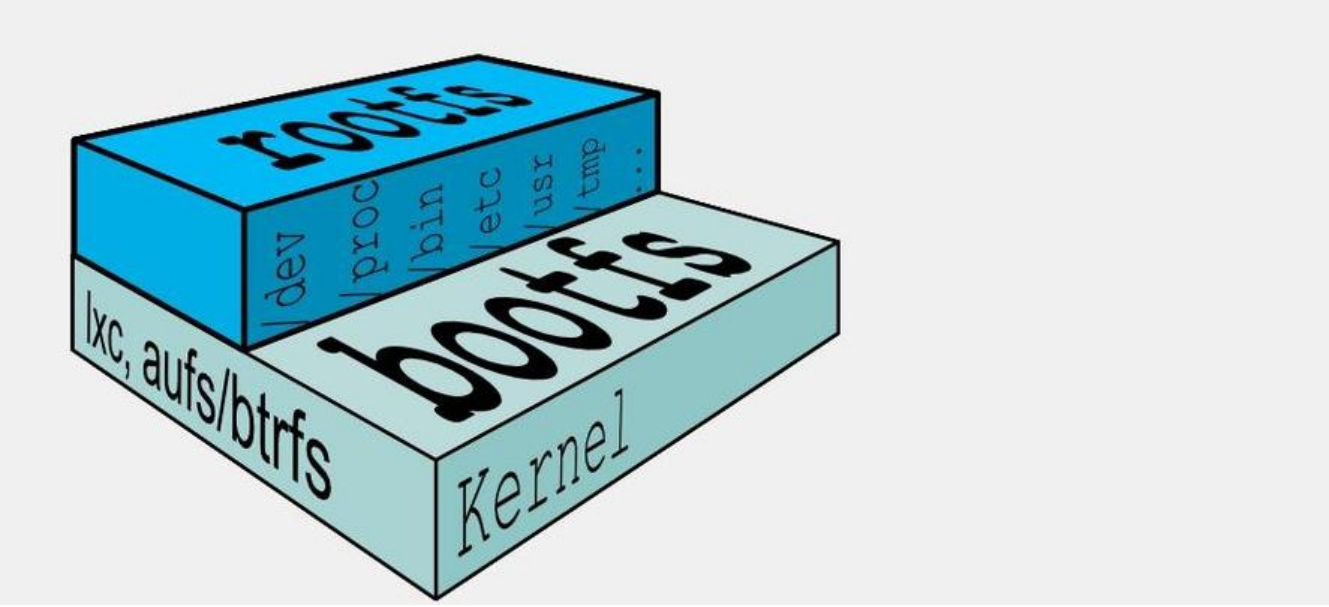
特性：一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录

Docker镜像加载原理

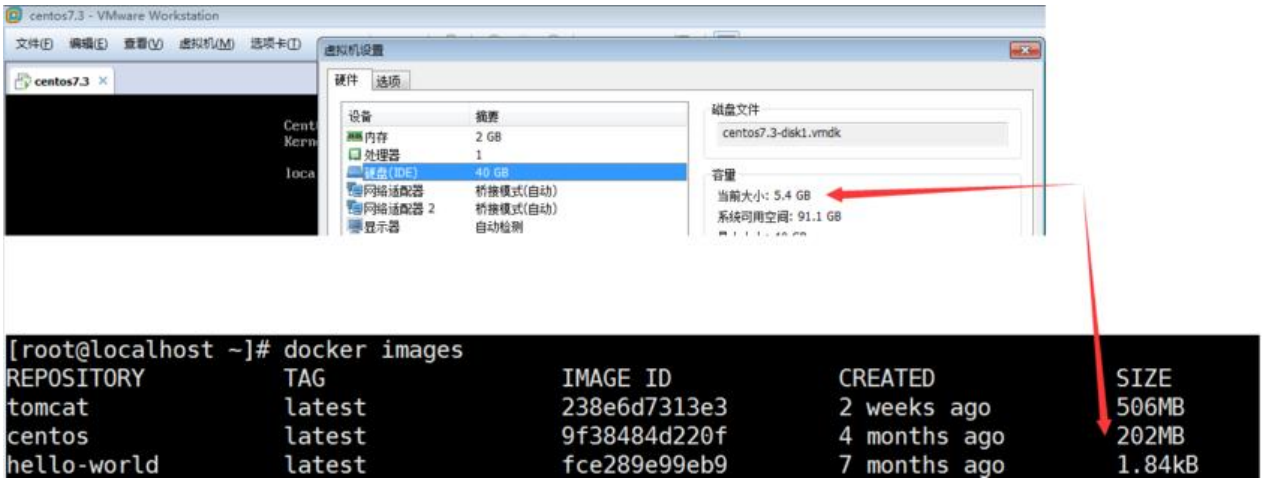
docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统UnionFS。

bootfs(boot file system)主要包含bootloader和kernel, bootloader主要是引导加载kernel, Linux启动时会加载bootfs文件系统，在Docker镜像的最底层是bootfs。这一层与我们典型的Linux/Unix统是一样的，包含boot加载器和内核。当boot加载完成之后整个内核就都在内存中了，此时内存的用权已由bootfs转交给内核，此时系统也会卸载bootfs。

rootfs (root file system)，在bootfs之上。包含的就是典型 Linux 系统中的 /dev, /proc, /bin, /etc等标准目录和文件。rootfs就是各种不同的操作系统发行版，比如Ubuntu，Centos等



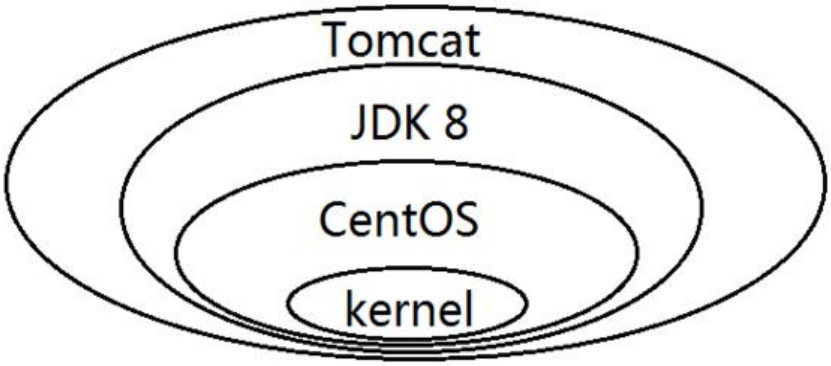
这也就解释了为什么我们开的虚拟机很大，但是Docker里面的镜像却很小：



对于一个精简的OS，rootfs可以很小，只需要包括最基本的命令、工具和程序库就可以了，因为底层接用Host的kernel，自己只需要提供 rootfs 就行了。由此可见对于不同的linux发行版, bootfs基本一致的, rootfs会有差别, 因此不同的发行版可以公用bootfs。

分层的镜像

以pull为例，在下载的过程中可以看到docker的镜像好像是在一层一层的在下载，这也就解释了为什Tomcat的镜像那么大



最大的一个好处就是 - **共享资源**

比如：有多个镜像都从相同的 base 镜像构建而来，那么宿主机只需在磁盘上保存一份base镜像，同时内存中也只需加载一份 base 镜像，就可以为所有容器服务了，而且镜像的每一层都可以被共享。

镜像的特点

Docker镜像都是只读的，当容器启动时，一个新的可写层被加载到镜像的顶部。这一层通常被称作容器层”，“容器层”之下的都叫“镜像层”。

Docker镜像commit操作

docker commit 提交容器副本使之成为一个新的镜像

docker commit -m="提交的描述信息" -a="作者" 容器ID 要创建的目标镜像名:[标签名]

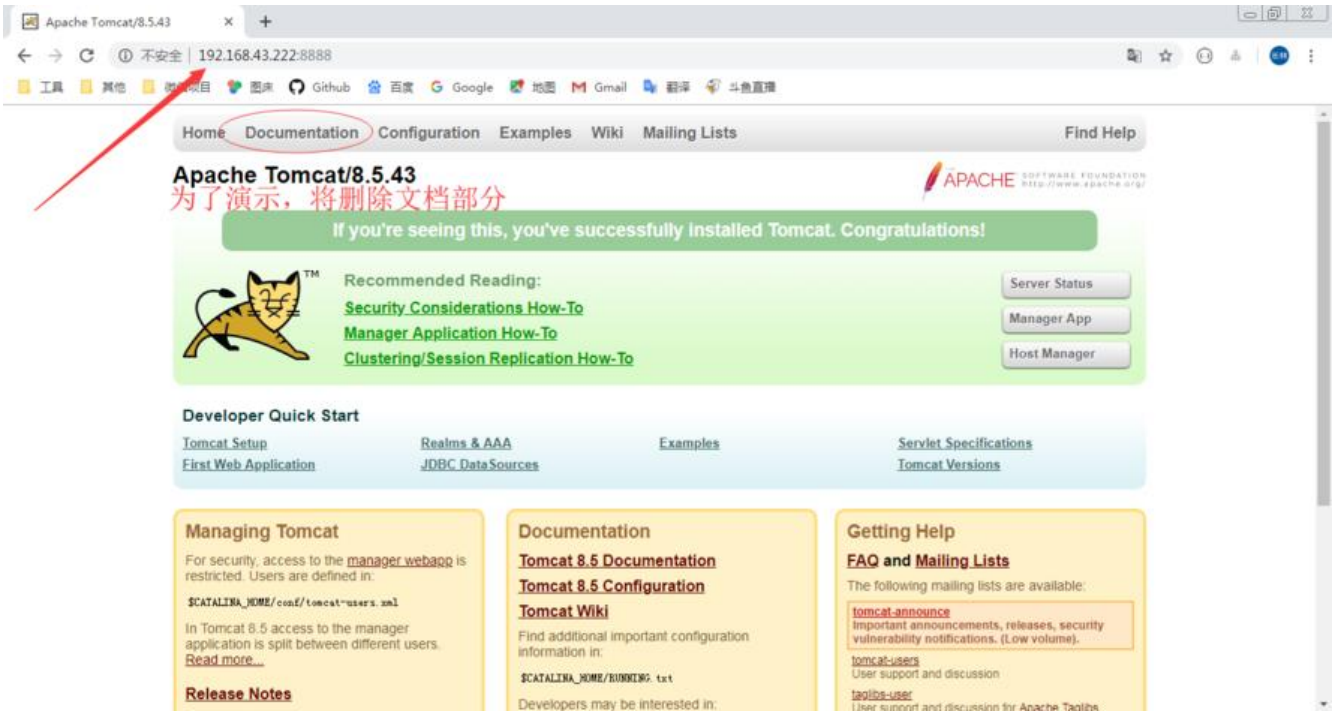
演示操作

1、先下载一个Tomcat镜像，并且运行Tomcat

docker pull tomcat

docker run -it -p 8888:8080 tomcat (使用-P是随机分配端口，分配的是Docker对外暴露的端口)

```
[root@localhost ~]# docker run -it -p 8888:8080 tomcat
Using CATALINA_BASE:   Docker对外暴露端口  Tomcat端口
Using CATALINA_HOME:   /usr/local/tomcat
Using CATALINA_TMPDIR: /usr/local/tomcat/temp
Using JRE_HOME:        /usr/local/openjdk-8
Using CLASSPATH:       /usr/local/tomcat/bin/bootstrap.jar:/usr/local/tomcat/bin/tomcat
```

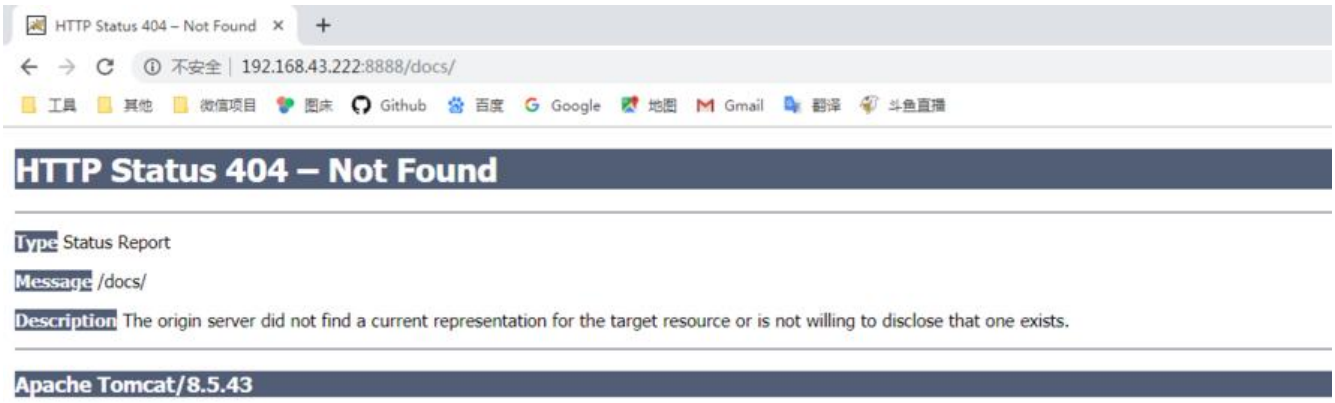


2、删除Tomcat的docs部分

进入到这个容器中，删除webapps下的docs文件夹

```
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
      PORTS                    NAMES
4f2161bbdb42        tomcat              "catalina.sh run"   11 minutes ago      Up 11 minutes
      0.0.0.0:8888->8080/tcp   gracious_mirzakhani
02cb0e08a3cd        centos              "/bin/bash"         3 hours ago         Up 3 hours
                               happy_volhard
[root@localhost ~]# docker exec -it 4f2161bbdb42 /bin/bash
root@4f2161bbdb42:/usr/local/tomcat# ls -ahl
total 124K
drwxr-sr-x. 1 root staff   42 Jul 18 02:55 .
drwxrwsr-x. 1 root staff   20 Jul 18 02:47 ..
-rw-r--r--. 1 root root   20K Jul  4 20:56 BUILDING.txt
-rw-r--r--. 1 root root  5.3K Jul  4 20:56 CONTRIBUTING.md
-rw-r--r--. 1 root root   56K Jul  4 20:56 LICENSE
-rw-r--r--. 1 root root  1.7K Jul  4 20:56 NOTICE
-rw-r--r--. 1 root root  3.2K Jul  4 20:56 README.md
-rw-r--r--. 1 root root  7.0K Jul  4 20:56 RELEASE-NOTES
-rw-r--r--. 1 root root   16K Jul  4 20:56 RUNNING.txt
```

此时访问docs就会404，说明我们删除成功



也即当前的tomcat运行实例是一个没有文档内容的容器，以它为模板commit一个没有doc的tomcat镜像xpu/tomcat_02 （xpu是命名空间，就相当于类的包名）

3、命令打包

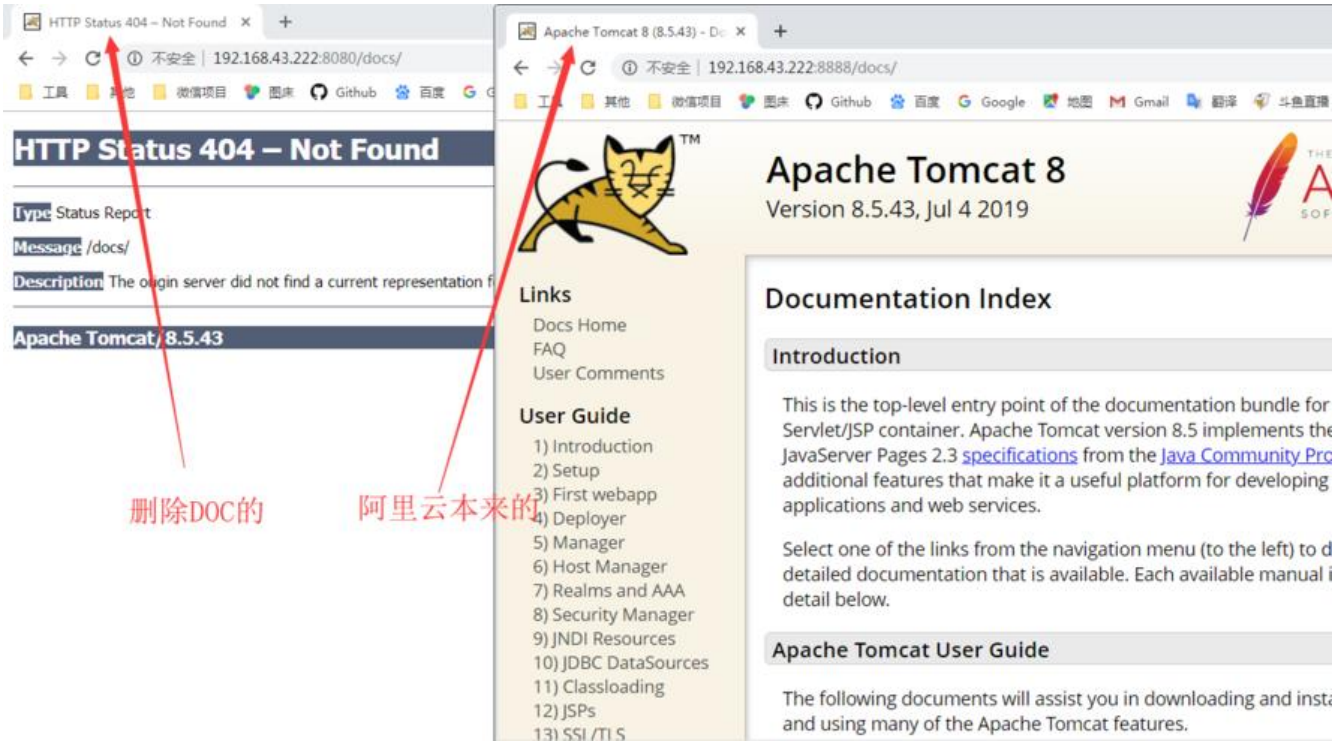
```
[root@localhost ~]# docker commit -m="This is my Tomcat" -a="Tim" 4f2161bbdb42 xpu/tomcat_02:1.2
sha256:b29a3c2a5e09af550d3bee9b6ff3cd9cf8e2b2b2397dc53278f4c495607f748f
[root@localhost ~]#
```

```
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
xpu/tomcat_02       1.2                 b29a3c2a5e09        50 seconds ago      506MB
tomcat              latest              238e6d7313e3        2 weeks ago         506MB
centos              latest              9f38484d220f        4 months ago        202MB
hello-world         latest              fce289e99eb9        7 months ago        1.84kB
```

docker rm -f $(docker ps -q) 删除正在运行的所有容器

4、后台方式启动Tomcat

```
docker run -d -p 8080:8080 tomcat
```



Docker容器数据卷

先来看看Docker的理念：

1、将运用与运行的环境打包形成容器运行 ， 运行可以伴随着容器，但是我们对数据的要求希望是持化的

2、容器之间希望有可能共享数据

Docker容器产生的数据，如果不通过docker commit生成新的镜像，使得数据做为镜像的一部分保下来，

那么当容器删除后，数据自然也就没有了。

为了能保存数据在Docker中我们使用卷，也就是容器数据卷！

Docker容器数据卷有点类似我们Redis里面的rdb和aof文件，也就是把运行时的数据持久化在硬盘上

卷就是目录或文件，存在于一个或多个容器中，由docker挂载到容器，但不属于联合文件系统，因此够绕过Union File System提供一些用于持续存储或共享数据的特性：

卷的设计目的就是数据的持久化，完全独立于容器的生存周期，因此Docker不会在容器删除时删除挂载的数据卷

特点：

1：数据卷可在容器之间共享或重用数据

2：卷中的更改可以直接生效

3：数据卷中的更改不会包含在镜像的更新中

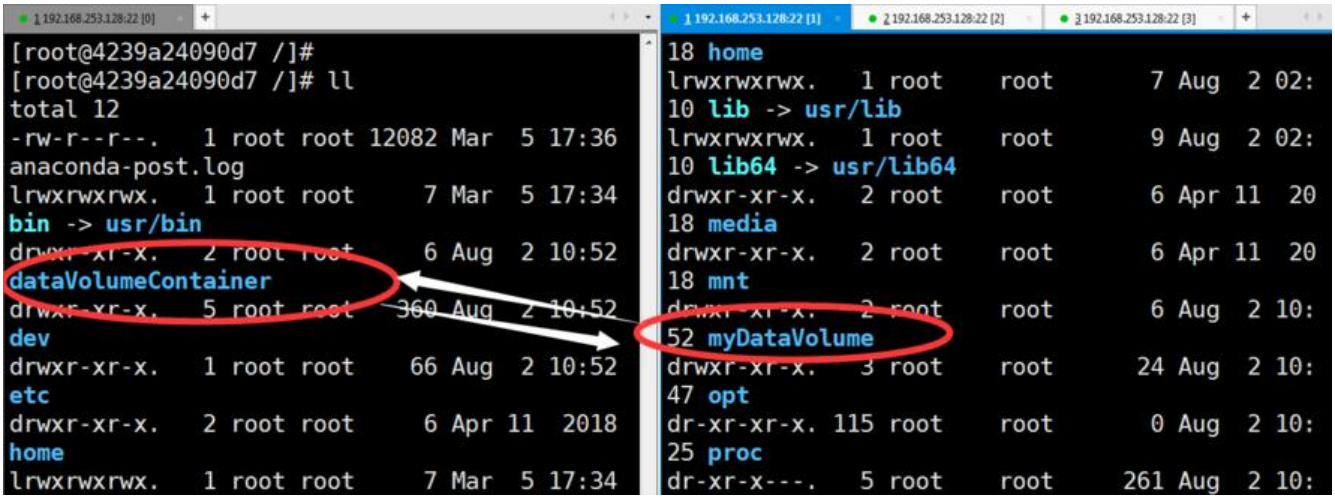
4：数据卷的生命周期一直持续到没有容器使用它为止

添加数据卷 _使用-v命令

docker run -it -v /宿主机绝对路径目录:/容器内目录　　镜像名

docker run -it -v /myDataVolume:/dataVolumeContainer centos

这样便添加了数据卷

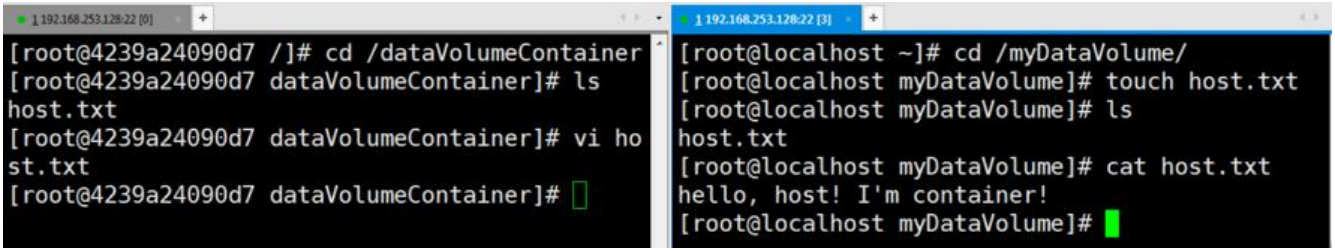


同样的，使用docker inspect 容器名称 便可以查看JSON形式描述的容器：

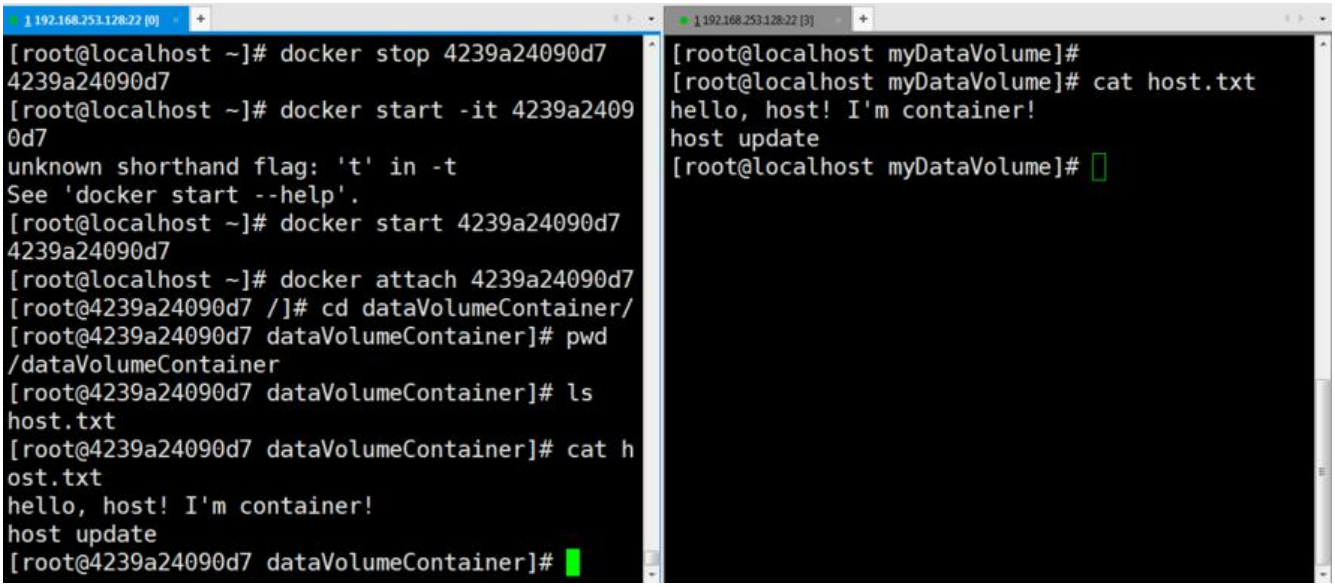
```
[root@localhost ~]# docker inspect 4239a24090d
```

```
        "Mounts": [
            {
                "Type": "bind",
                "Source": "/myDataVolume",
                "Destination": "/dataVolumeCon
tainer",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            }
        ],
```

测试一下两者的文件通信共享(容器和宿主机之间数据共享)



容器停止退出后，主机修改后数据是否同步



带有写保护权限的数据卷

docker run -it -v /宿主机绝对路径目录:/容器内目录:ro 镜像名

ro即是ReadOnly，只读，不允许容器修改数据卷，而宿主机才可以！

```
[root@localhost ~]# docker run -it -v /myDataV
olume:/dataVolumeContainer:ro centos
[root@add72072f2c5 /]#
[root@add72072f2c5 /]#
[root@add72072f2c5 /]#
[root@add72072f2c5 /]# cd /
[root@add72072f2c5 /]# ls
anaconda-post.log    etc    media  root  sys
bin                  home   mnt    run   tmp
dataVolumeContainer  lib    opt    sbin  usr
dev                  lib64  proc   srv   var
[root@add72072f2c5 /]# cd d
bash: cd: d: No such file or directory
[root@add72072f2c5 /]# cd dataVolumeContainer/
[root@add72072f2c5 dataVolumeContainer]# touch newfile
touch: cannot touch 'newfile': Read-only file system
[root@add72072f2c5 dataVolumeContainer]#
```

添加数据卷_使用Dockerfile

在Linux下写项目很多时候用到makefile来构建工程，或者是通过Shell脚本把一系列的操作封装起来
所以理解Dockerfile就不难，之前说过镜像是层层包裹的，就比如：Tomcat镜像肯定是依赖于JDK镜的，所以Dockerfile还是很重要的

先编写Dockerfile (注意VOLUME后面有空格)

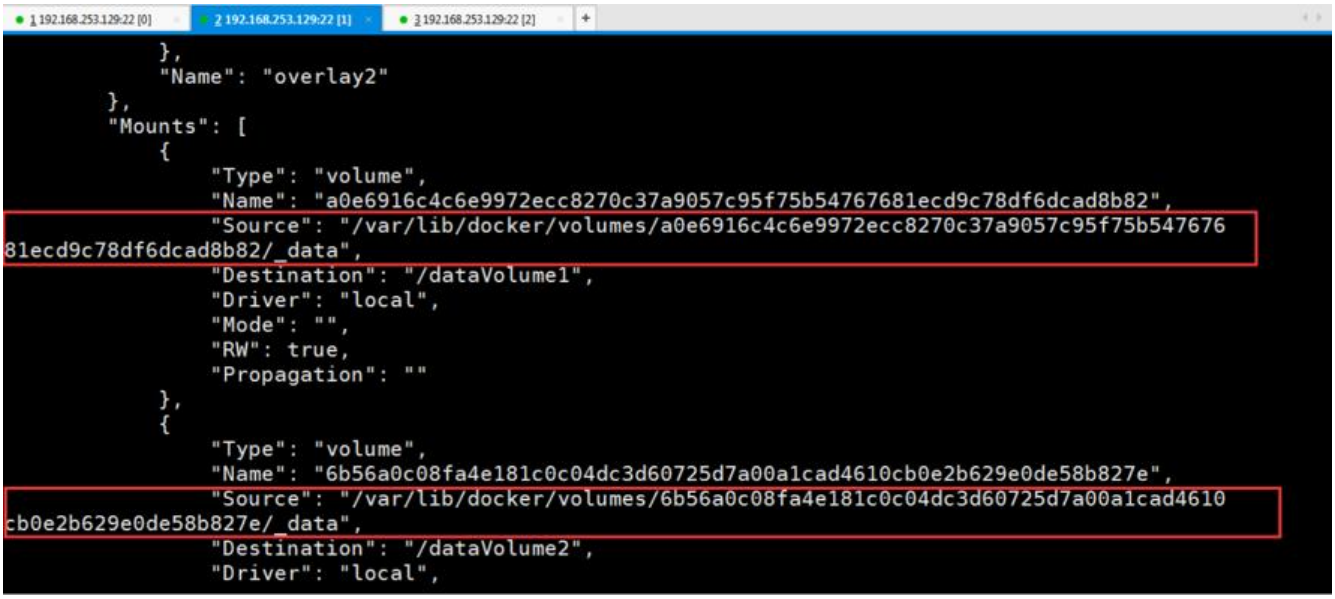
```
[root@localhost mydocker]# cat Dockerfile
# volume test
FROM centos
VOLUME ["/dataVolume1","/dataVolume2"]
CMD echo "finished, ------success"
CMD /bin/bash
[root@localhost mydocker]#
```

```
[root@localhost mydocker]# docker build -f /mydocker/Dockerfile -t tim/centos .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM centos
 ---> 9f38484d220f
Step 2/4 : VOLUME ["/dataVolume1","/dataVolume2"]
 ---> Running in bf1b626d403c
Removing intermediate container bf1b626d403c
 ---> cf233448eb04
Step 3/4 : CMD echo "finished, ------success"
 ---> Running in 2039cf5c9319
Removing intermediate container 2039cf5c9319
 ---> 79cddcbc5c3c
Step 4/4 : CMD /bin/bash
 ---> Running in 96e833541cf0
Removing intermediate container 96e833541cf0
 ---> c25492a92acb
Successfully built c25492a92acb
Successfully tagged tim/centos:latest
[root@localhost mydocker]#
```

由上图输出也可以看出，这个镜像是分层构建的！接下来检测一下构建结果：

```
[root@localhost mydocker]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
tim/centos          latest              c25492a92acb        41 minutes ago      202MB
xpu/tomcat_02       1.2                 b29a3c2a5e09        3 hours ago         506MB
tomcat              latest              238e6d7313e3        2 weeks ago         506MB
centos              latest              9f38484d220f        4 months ago        202MB
hello-world         latest              fce289e99eb9        7 months ago        1.84kB
[root@localhost mydocker]# docker run -it c25492a92acb
[root@c812194c32b8 /]# ls / |grep dataVolume
dataVolume1
dataVolume2
[root@c812194c32b8 /]#
```

那么对应的宿主机器目录在哪呢？虽然我们没有手动指定，但是dockerfile有默认的路径，使用docker inspect 容器ID查看容器细节：



如果Docker挂载主机目录Docker访问出现cannot open directory .: Permission denied，解决办法在挂载目录后多加一个--privileged=true参数即可

所以到目前为止可以吧数据卷简单的理解为虚拟机和物理机的共享文件夹，上面讲述了两种创建容器据卷的方式，方式一不是很常用，使用Dockerfile的方式是更适合使用的！

数据卷容器

命名的容器挂载数据卷，其它容器通过挂载这个(父容器)实现数据共享，挂载数据卷的容器，称之为据卷容器

```
[root@localhost ~]# docker run -it --name dc01 tim/centos
[root@cab0c16b8b27 /]# ll
total 12
-rw-r--r--.   1 root root 12082 Mar  5 17:36 anaconda-post.log
lrwxrwxrwx.   1 root root     7 Mar  5 17:34 bin -> usr/bin
drwxr-xr-x.   2 root root     6 Aug  2 14:42 dataVolume1
drwxr-xr-x.   2 root root     6 Aug  2 14:42 dataVolume2
drwxr-xr-x.   5 root root   360 Aug  2 14:42 dev
drwxr-xr-x.   2 root root     6 Apr 11  2018 mnt
[root@cab0c16b8b27 /]# cd dataVolume2/
[root@cab0c16b8b27 dataVolume2]# pwd
/dataVolume2
[root@cab0c16b8b27 dataVolume2]# touch dc01.txt
```

```
[root@localhost ~]# docker run -it --name dc02 --volumes-from dc01 tim/centos
[root@79bbca181e72 /]# ll
total 12
-rw-r--r--.   1 root root 12082 Mar  5 17:36 anaconda-post.log
lrwxrwxrwx.   1 root root     7 Mar  5 17:34 bin -> usr/bin
drwxr-xr-x.   2 root root     6 Aug  2 14:42 dataVolume1
drwxr-xr-x.   2 root root    22 Aug  2 14:42 dataVolume2
drwxr-xr-x.   5 root root   360 Aug  2 14:44 dev
drwxr-xr-x.   1 root root    66 Aug  2 14:44 etc
drwxr-xr-x.   2 root root     6 Apr 11  2018 home
drwxr-xr-x.  18 root root   238 Mar  5 17:34 var
[root@79bbca181e72 /]# cd dataVolume2/
[root@79bbca181e72 dataVolume2]# ls
dc01.txt
[root@79bbca181e72 dataVolume2]# touch dc02.txt
```

```
[root@localhost _data]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
          PORTS               NAMES
79bbca181e72        tim/centos          "/bin/sh -c /bin/bash"   50 seconds ago      Up 49 seco
nds                             dc02
cab0c16b8b27        tim/centos          "/bin/sh -c /bin/bash"   3 minutes ago       Up 3 minut
es                              dc01
[root@localhost _data]# docker run -it --name dc03 --volumes-from dc01 tim/centos
[root@93fb17e370f7 /]# cd /dataVolume2/
[root@93fb17e370f7 dataVolume2]# ls
dc01.txt  dc02.txt
[root@93fb17e370f7 dataVolume2]# touch dc03.txt
```

先见了三个容器，dc01是之前建好的，里面有容器卷dataVolume1，dataVolume2，剩下两个容器别是dc02，dc03，然后通过

docker run -it --name dc02 --volumes-from dc01 tim/centos

这样的命令去新建一个dc02容器继承父容器dc01，同样的，使用此命令新建dc03容器继承父容器dc0，于是dc02和dc03都含有一两个容器卷dataVolume1，dataVolume2，于是dc01、dc02、dc03都数据共享的：

```
[root@cab0c16b8b27 dataVolume2]# pwd
/dataVolume2
[root@cab0c16b8b27 dataVolume2]# touch dc01.txt
[root@cab0c16b8b27 dataVolume2]# pwd
/dataVolume2
[root@cab0c16b8b27 dataVolume2]# ls
dc01.txt  dc02.txt  dc03.txt
```

dc01 dc02.txt 、dc03.txt都是子容器新建的文件

接下来删除dc01，然后dc02新建一个dc02_update.txt，虽然dc03是继承自dc01的，但是dc03仍然以看到dc02_update.txt，看下图：

```
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
  PORTS               NAMES
93fb17e370f7        tim/centos          "/bin/sh -c /bin/bash"   About an hour ago   Up About an hour
                      dc03
79bbca181e72        tim/centos          "/bin/sh -c /bin/bash"   About an hour ago   Up About an hour
                      dc02
cab0c16b8b27        tim/centos          "/bin/sh -c /bin/bash"   About an hour ago   Up About an hour
                      dc01
[root@localhost ~]# docker stop dc01
dc01
[root@localhost ~]# docker rm -f dc01
dc01
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
  PORTS               NAMES
93fb17e370f7        tim/centos          "/bin/sh -c /bin/bash"   About an hour ago   Up About an hour
                      dc03
79bbca181e72        tim/centos          "/bin/sh -c /bin/bash"   About an hour ago   Up About an hour
                      dc02
[root@localhost ~]# docker attach dc02
[root@79bbca181e72 dataVolume2]# pwd
/dataVolume2
[root@79bbca181e72 dataVolume2]# cd ..
[root@79bbca181e72 /]# ls
anaconda-post.log  dataVolume1  dev  home  lib64  mnt  proc  run   srv  tmp  var
bin                dataVolume2  etc  lib   media  opt  root  sbin  sys  usr
[root@79bbca181e72 /]# cd dataVolume2
[root@79bbca181e72 dataVolume2]# ls
dc01.txt  dc02.txt  dc03.txt
[root@79bbca181e72 dataVolume2]# touch dc02_update.txt
```

删除dc01，并让dc02新建dc02_update.txt

```
[root@93fb17e370f7 dataVolume2]# touch dc03.txt
[root@93fb17e370f7 dataVolume2]# ls
dc01.txt  dc02.txt  dc02_update.txt  dc03.txt
```

dc03容器卷中的数据

由此得出一个重要结论：

容器之间配置信息的传递，数据卷的生命周期一直持续到没有容器使用它为止