# 镜像

UnionFS （联合文件系统）是一种轻量级分层的文件系统，支持对文件系统的修改作为一次提交来一层一层的叠加，可以同时将不同目录挂载到同一个虚拟文件系统下，Union文件系统是Docker镜像的基础，镜像可以通过分层来进行集成，基于基础镜像，可以制作各种具体的应用镜像。

特性：一次加载多个文件系统，但是从外面来看，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统包含所有底层的文件和目录

bootfs： 只要包含kernel，bootfs主要是引导加载kernel， linux刚启动的时候回架子啊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系统是一样的，包含boot加载器和内核，当boot加载完成之后整个内核就在内存最后哦哦那个了，此时内存的使用权已经有bootfs转交给内核，bootfs会被卸载

rootfs 在bootfs纸上，包含的就是典型linux系统中的 /dev /proc /etc 等标准文件和目录，rootfs就是各种不同的操作系统的发行版，如 ubuntu Centos

当你在Docker 下载Ubuntu系统的时候，实际上下载的是那些，底层的kernel都是都是公用的，在加上rootfs（包括最基本的命令、工具和程序库）就是一个系统了。不同的linux发行版，kernel都是一样的，只有rootfs 的区别

一个镜像最终给你干活的、最外面暴露的哪个镜像

Docker 使用这种分层的好处是什么?
资源共享，会有其他的工具也会采用某一个资源，而这个资源也会被其他资源作为依赖，此时这个资源就只需要准备一份，例如，多个镜像都从相同的base镜像构建而来，那么在宿主机器只需要硬盘上保存一封base镜像，同时内存中也只需要加载一份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

镜像都是只读的，当容器启动的时候，一个新的可写层被加载到镜像的顶部，当着一层通常被称作是“容器层”，“容器层”之下的都是被叫做“”镜像层。

提交容器副本，使之成为一个新的镜像，提交的镜像是我们定义的，符合我们需求的一个镜像

```
docker commit -m “message” -a “作者” 命名空间/镜像:[标签]

docker run -it -p 8888:8080 tomcat (-P 是随机分配端口， -p 新端口:默认端口 完成端口映射)
```

后台访问：

```
docker run -d -p 6666:8080 tomcat 如果只是这样，在启动的时候，是没有打印任何日志的，你只能在输入          localhost:6666等可以访问到tomcat。
```

# 容器数据卷

将容器中运算结果数据，长期保存下来，也就是将数据持久化，当容器中运行结束的时候，运行结果就会丢失，所以在关闭之前需要将数据持久化，保存的时候，应该是保存到硬盘的，在容器中的数据，好比是在内存中的数据，

我们希望对容器中的数据是持久化的
容器中间希望是可能共享数据的

容器数据卷：
做持久化的东东，数据共享和数据持久化工作，

- 数据卷可在容器之间共享数据，包括主机到容器、容器到主机的共享
- 卷中的修改可以直接生效
- 数据卷中的更爱不会包含在镜像的跟新中
- 数据卷的生命周期一直持续到没有容器使用为止

## 添加容器卷

直接命令添加

```
docker run -it -v /宿主机绝对路径目录:/容器内路径目录 镜像名    v是colume的缩写

docker run -it -v /Users/xuxliu/Desktop/mydataDockerVolume:/mydataDockerContainer ubuntu
在主机桌面创建文件夹 mydataDockerVolume 和 docker 中的ubuntu容器创建 mydataDockerContainer文件
实现二者的数据共享，相当于在容器中插进去 U盘，这里你并不需要在执行命令之前新建号这些文件夹
docker 命令会为你新建的
```

查看是否挂载成功

```
docker inspect 你启动的容器ID
查看输出的内容中的 Mounts 内容，是否和你刚才设置的路径一致
"Mounts": [
            {
                "Type": "bind",
                "Source": "/Users/xuxliu/Desktop/mydataDockerVolume",
                "Destination": "/mydataDockerContainer",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            }
R: 读、W: 写  是否可以文件夹进行这些操作
```

不管是在容器里面修改文件，还是在主机中修改文件，二者都会是同步的。

当容器中的东西修改，都会保存到主机上，你退出容器后，之前修改的东西都会在主机哪个文件中存在

当容器退出之后，继续在主机上对文件进行修改，此时你在进入容器，那么在容器退出后宿主机对文件的修改还是在容器内的，二者之间是完全同步的。

需要注意的是，此时进入容器应该是使用attach 命令，而不是exec

```
docker start 647adb0d8024
docker attach 647adb0d8024
```

# 修改权限

假如你在执行

```
docker inspect 你启动的容器ID
```

时候发现，读写命令式False，那么你就应该去修改权限

执行以下命令的时候，你在创建卷的时候设定读写权限

```
docker run -it -v /Users/xuxliu/Desktop/mydataDockerVolume:/mydataDockerContainer:ro ubuntu
```

:ro == read only
没有写权限

然后在执行命令

```
docker inspect 你启动的容器ID
```

写权限是false 的

使用dockerfile添加数据卷

dockerfile 是对 镜像的一种描述

```
From :  镜像来自原哪
env :
Run:
EXPOSE: 暴露端口
cmd:  命令
```

在本机建立dockerfile文件
在dockerfile中使用volume指令来给镜像添加一个或者多个数据卷
VOLUME[“/目录1”， “目录2”….]
-v 创建容器目录这种方法不能直接在Dockerfile中实现，由于宿主机目录是依赖于特定宿主机的，并不能够保证在所有的宿主机都存在这一的特定目录
dockerfile 文件里面内容

```
from ubuntu
volume ["/dataVolume1","/dataVolume2","/dataVolume3"]
cmd echo "finish ---------- successful!"
cmd /bin/bash
```

翻译过来等同于以下

```
Docker run -it -v /dataVolume1 -v /dataVolume2 -v /dataVolume3 ubuntu /bin/bash
```

执行dockerfile， 执行后获得一个新的镜像

```
docker build -f Dockerfile -t zzyy/ubuntu .
-f 执行dockerfile的文件目录
-t 给添加别名
最后不要忘记最后面的哪个 .
```

输出的日志

```
Sending build context to Docker daemon  2.048kB          开始和守护进程通信
Step 1/4 : from ubuntu
---> ea4c82dcd15a
Step 2/4 : volume ["/dataVolume1","/dataVolume2","/dataVolume3"]
---> Running in 4415a4c6ed93
Removing intermediate container 4415a4c6ed93
---> 527f651eb405
Step 3/4 : cmd echo "finish ---------- successful!"
---> Running in bb379f101977
Removing intermediate container bb379f101977
---> daae9b2834b9
Step 4/4 : cmd /bin/bash
---> Running in bb575a3db31f
Removing intermediate container bb575a3db31f
---> 93c8ef07e9a1
Successfully built 93c8ef07e9a1                     执行结束后新的镜像ID
Successfully tagged zzyy/ubuntu:latest              执行结束后新的镜像别名
```

执行

```
docker images
```

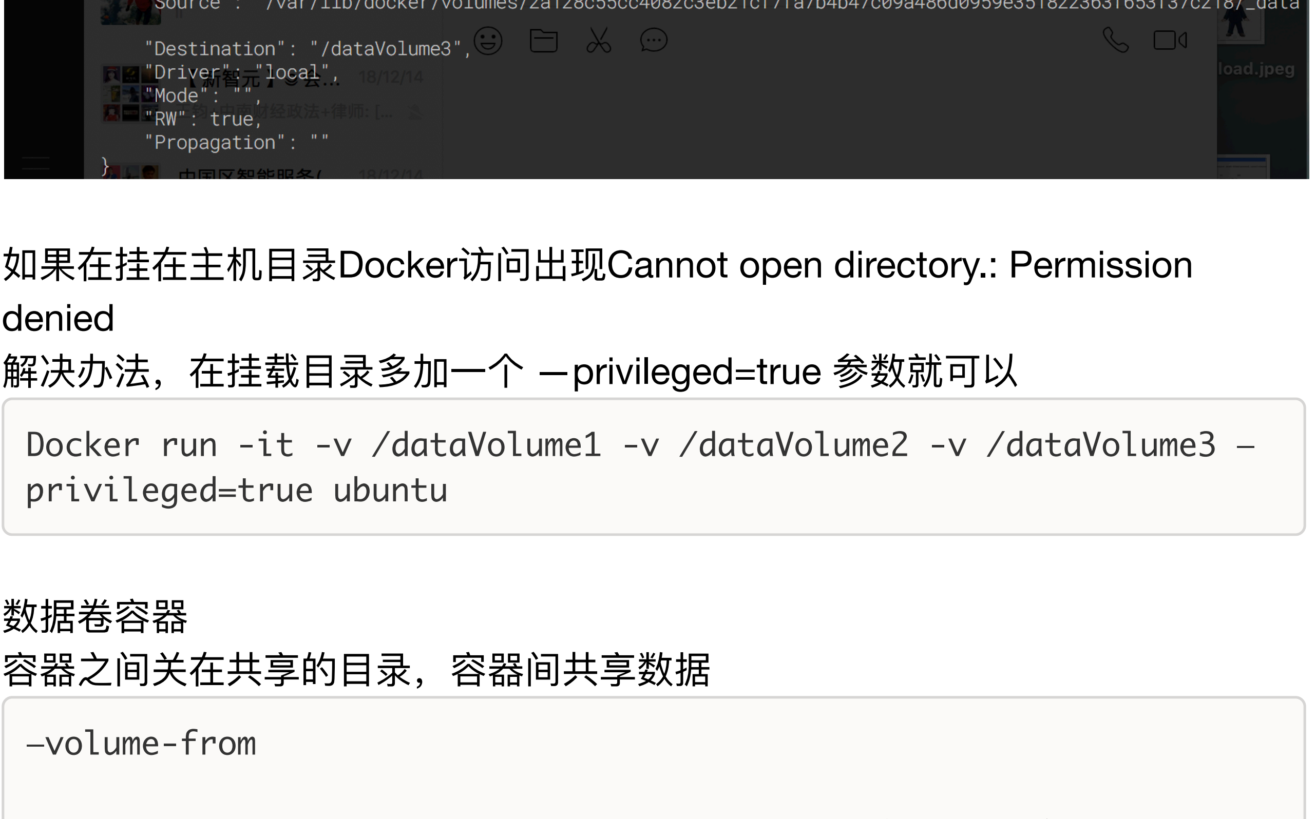
也可以看见新的那个镜像，

运行新建的镜像容器，

```
docker run -it zzyy/ubuntu
```

看见dockerfile的那几个目录，至此完成了，容器内的目录

而宿主机的文件，docker是自动创建的



```
"Mounts": [
    {
        "Type": "volume",
        "Name": "a20882aa5b228becd19a305ac75d4112c3eea8fa8168cec73479deb0dee7bfd0",
        "Source": "/var/lib/docker/volumes/a20882aa5b228becd19a305ac75d4112c3eea8fa8168cec73479deb0dee7bfd0/_data",
        "Destination": "/dataVolume1",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    },
    {
        "Type": "volume",
        "Name": "0b8a0b4f0fd8d70c210e9de84b23e86205746c77b6e57207f469a1b7a8790e86",
        "Source": "/var/lib/docker/volumes/0b8a0b4f0fd8d70c210e9de84b23e86205746c77b6e57207f469a1b7a8790e86/_data",
        "Destination": "/dataVolume2",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    },
    {
        "Type": "volume",
        "Name": "2a128c55cc4082c3eb21cf7fa7b4b47c09a486d0959e351822363f653f37c218",
        "Source": "/var/lib/docker/volumes/2a128c55cc4082c3eb21cf7fa7b4b47c09a486d0959e351822363f653f37c218/_data",
        "Destination": "/dataVolume3",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
```

如果在挂在主机目录Docker访问出现Cannot open directory.: Permission denied
解决办法，在挂载目录多加一个 —privileged=true 参数就可以

```
Docker run -it -v /dataVolume1 -v /dataVolume2 -v /dataVolume3 –privileged=true ubuntu
```

数据卷容器
容器之间关在共享的目录，容器间共享数据

```
–volume-from

docker run -it --name dc01 zzyy/ubuntu 给这个容器一个名字 dc01

在新起一个容器
docker run -it --name dc02 --volumes-from dc01 zzyy/ubuntu

在新起一个容器
docker run -it --name dc03 --volumes-from dc01 zzyy/ubuntu
```

然后任意一个容器中编辑共享的那些文件，都可以在三个中发现修改，也就是 --volumes-from 命令不仅继承，而且共享
Dc02， dc03都是继承来源于 dc01， 假如现在讲dc01 删除，刚才那些共享文件在dc02， dc03 都还是存在的
此时你在dc02上继续对共享目录做修改，那么dc03 上还会有这些修改，同理 dc03 的修改 在 dc02 还是可见的。

同样的你在继承dc03创建dc04，在dc04 上的修改，在dc02和dc04上都还是可见的，如果你删掉dc03，那么dc02和dc04都还是共享关系的

**所容器卷上的配置信息的传递，数据卷的生命周期一直会持续到没有容器使用它为止，都是可以全量备份的**