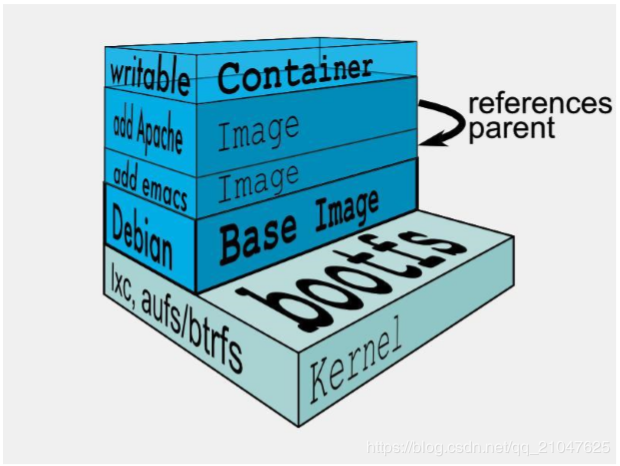
Docker 容器互联

1  基于Docker Volum的容器互联

1.1 Docker的文件存储

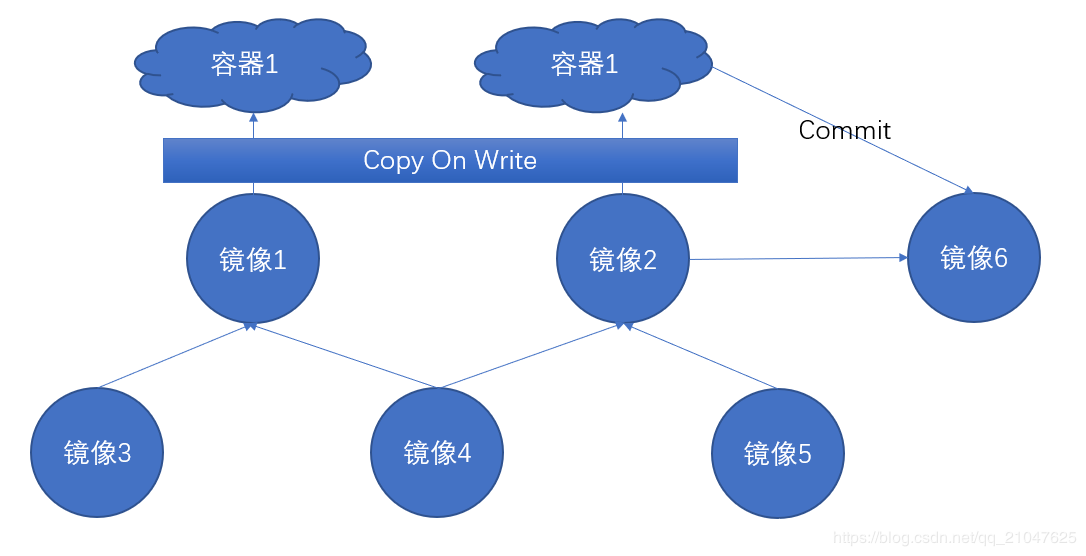
docker的文件系统是copy on write方式的, 文件是一层层往上叠加的, 最下的一层是只读的, 要修改的时候会复制这层只读的覆盖在只读的上面作为一层可写的, 这个时候原来的只读文件依然是存在的这是看不到了

docker 中镜像的存储就是使用的上述方式,分层结构如下



上图Docker的容器就是通过只读的镜像上面覆盖的可写层, 要想把这个可写层持久化, 就要用到之前的讲的 docker commit 了

但是镜像的层次又可以共享:

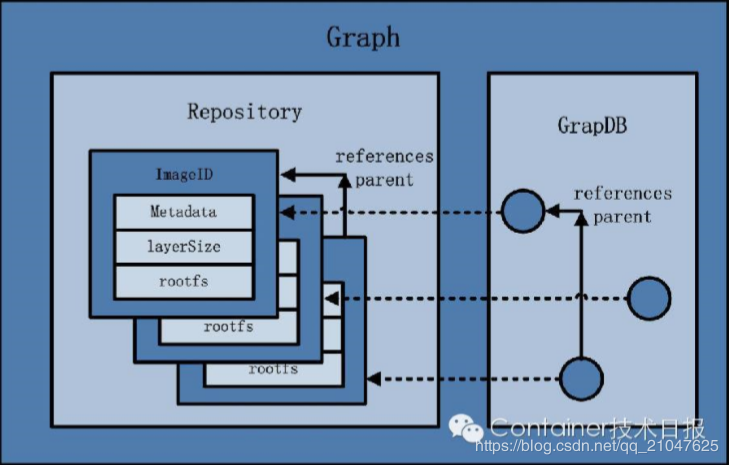


镜像是分层叠加的, 顶层的镜像依赖于底层的镜像, 容器是通过镜像 Copy On Write 一个可写层, 这个可写层 Commit 就完成了持久化又成了一个新的镜像.

Docker镜像的存储位置:

docker的镜像是分层的,一个镜像往往是多个镜像分层叠加出来的, 镜像的分层信息是存储在 Docker Graph 里面

这里记录了镜像有些那些层,   每一层的父层和大小等信息

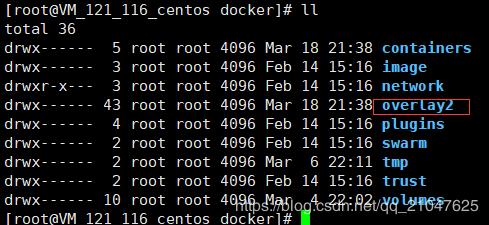


其中 GrapDB 存储的是分层结构 , Repository 里面存储是镜像的信息, 两者结合得到镜像的分层信息

但是这些里面存储都是镜像的信息, 并不是镜像的实际存储位置,

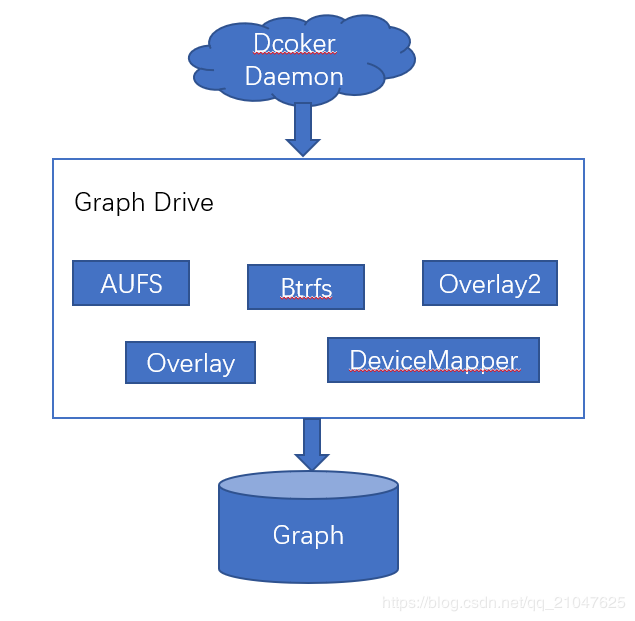
镜像的二进制文件存储位置是. /var/lib/docker/<storage-driver>

<storage-driver> 是  Docker Graph Driver , 不同的 Docker Graph Driver 会存储在不同的目录, 比如本机采用的是 overlay2



Docker Graph Driver 是Docker 使用来管理存储镜像每层内容及可读写的容器的驱动, 目前主要有 DeviceMapper、AUFS、Overlay、Overlay2、Btrfs、ZFS 等，不同的存储驱动实现方式有差异，镜像组织形式可能也稍有不同，但都采用栈式存储，并采用 Copy-on-Write策略。且存储驱动采用热插拔架构，可动态调整。

Graph Driver 就类似于Java连接数据库的 JDBC, 只不过这个驱动是用于 docker Deamon 来对镜像个容器进行读写.



选择策略:

若内核支持多种存储驱动，且没有显式配置，Docker 会根据它内部设置的优先级来选择。优先级为 AUFS > Btrfs/ZFS > Overlay2 > Overlay > DeviceMapper。若使用 DeviceMapper 的话，在生产环境，一定要选择 direct-lvm, loopback-lvm 性能非常差。

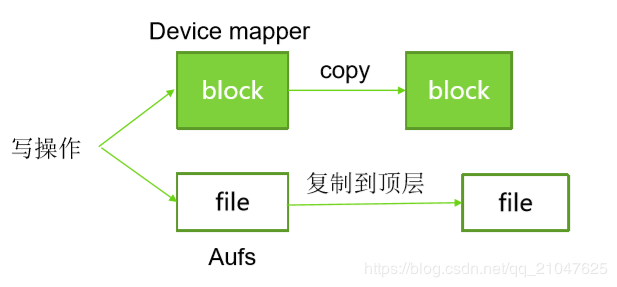
选择会受限于 Docker 版本、操作系统、系统版本等。例如，AUFS 只能用于 Ubuntu 或 Debian 系统，Btrfs 只能用于 SLES （SUSE Linux Enterprise Server, 仅 Docker EE 支持）。

有些存储驱动依赖于后端的文件系统。例如，Btrfs 只能运行于后端文件系统 Btrfs 上。

不同的存储驱动在不同的应用场景下性能不同。例如，AUFS、Overlay、Overlay2 操作在文件级别，内存使用相对更高效，但大文件读写时，容器层会变得很大；DeviceMapper、Btrfs、ZFS 操作在块级别，适合工作在写负载高的场景；容器层数多，且写小文件频繁时，Overlay 效率比 Overlay2 更高；Btrfs、ZFS 更耗内存。

1.2Docker Volume

Docker Copy on Write



和刚刚说的一样, Docker 的写操作主要分为两种, 一个是 块级别的, 一个是文件级别的,  块级别的在修改文件的时候会把修改部分所在的块 copy 到 write层用于写, 文件级别则是直接复制真个文件到 write 层,   所以块级别能够节省空间, 但是块级别 copy的操作次数多, 文件级别直接copy 文件就会是的容器变大, 但是直接copy 文件就会大大减少copy操作的次数.

以上操作对容器中中需要频繁读写的大文件就很不利,  块级别的 COW 操作太频繁,效率低, 文件级别的又会使得容器变得很大, 比如mysql数据库的数据库存储文件.

这个时候就引入了 Volume, Volme 可以将宿主机的文件映射到容器直接操作, 不必经过 COW, 常见的高频写文件有日志系统和数据存储文件.

Vloume使用是通过docker run  的 -v 参数来实现的

[docker@VM\_121\_116\_centos ~]$ docker run -it -p 6379:6379 -v ~/redis/date:/data redis /bin/bash

root@b385ca73cad3:/data# ls

appendonly.aof root tomcat

root@b385ca73cad3:/data# mkdir testdir

root@b385ca73cad3:/data# exit

exit

[docker@VM\_121\_116\_centos ~]$ ll

total 580364

drwxrwxr-x 2 docker docker 4096 Mar 6 22:00 dockerFile

-rw-rw-r-- 1 docker docker 594276824 Mar 6 17:31 ideaIU-191.6014.8.exe

drwxr-xr-x 3 root root 4096 Mar 19 21:26 redis

[docker@VM\_121\_116\_centos ~]$ cd redis/date/

[docker@VM\_121\_116\_centos date]$ ll

total 16

-rw-r--r-- 1 polkitd ssh\_keys 443 Mar 19 21:27 appendonly.aof

-rw-r--r-- 1 polkitd ssh\_keys 249 Mar 19 21:27 root

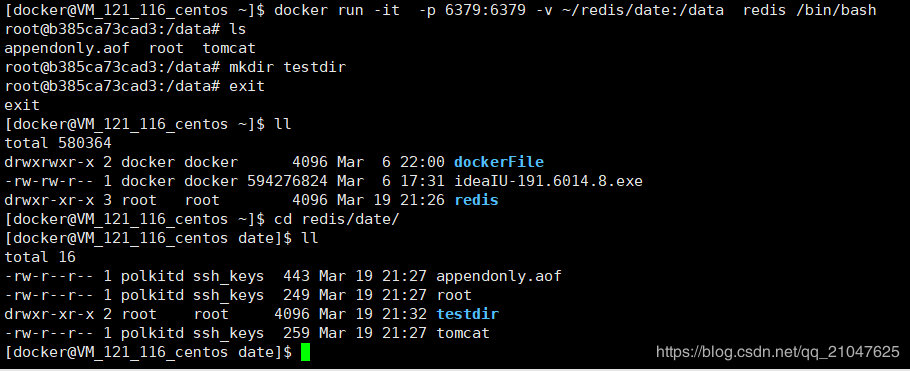
drwxr-xr-x 2 root root 4096 Mar 19 21:32 testdir

-rw-r--r-- 1 polkitd ssh\_keys 259 Mar 19 21:27 tomcat

[docker@VM\_121\_116\_centos date]$

 如命令所示 使用 -v 把宿主机的  ~/redis/date 挂载到 容器的 /data , 之后在容器的 /data  建立一个  testdir 文件夹. 之后在宿主机的 ~/redis/data 里面看见了刚刚建的文件夹. 如不指定宿主机目录, 那么docker就会在docker得volume目录中创建一个目录挂载到容器内, 当容器删除的时候, 这个目录也就随之删除了.

这样容器的 /data 目录就能避免大量的 COW 操作了, 直接进行读写.



在次看看容器信息 使用 docker inspect

[docker@VM\_121\_116\_centos date]$ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

b385ca73cad3 redis "docker-entrypoint..." 8 minutes ago Exited (0) 7 minutes ago sad\_murdock

e8abf8c09a26 mysql:5.6 "docker-entrypoint..." 3 weeks ago Exited (0) 2 weeks ago mysql5.6

[docker@VM\_121\_116\_centos date]$ docker inspect b385ca73cad3

[

{

"Id": "b385ca73cad3607f78fc87214dadf23435eebeaf5ee5551a17f160a432b08264",

"Created": "2019-03-19T13:31:55.437267785Z",

"Path": "docker-entrypoint.sh",

"Args": [

"/bin/bash"

],

"State": {

"Status": "exited",

"Running": false,

"Paused": false,

"Restarting": false,

"OOMKilled": false,

"Dead": false,

"Pid": 0,

"ExitCode": 0,

"Error": "",

"StartedAt": "2019-03-19T13:31:55.783433554Z",

"FinishedAt": "2019-03-19T13:32:42.591792526Z"

},

====================省略==================================

"GraphDriver": {

"Name": "overlay2",

"Data": {

"LowerDir": "/var/lib/docker/overlay2/f2806e52d83beca84bd0bb5a137a4a70974fcecdede03e5bc1b3f5c3711619c9-init/diff:/var/lib/docker/overlay2/61d911e09963bc0e77b7020b7bdbf798a5c35dbd90514c810f3aebb6875610f7/diff:/var/lib/docker/overlay2/fdddcb2b85492183f28660470d17c554e5bdb3a01cc8089e9c2112da7eefb4d5/diff:/var/lib/docker/overlay2/9476ee9fd3b0ccad81bdfa469c4188bda49230dff3598e9018cb763a0ba683c1/diff:/var/lib/docker/overlay2/65cfb70d00faeb7477439d635dc0a6f78f4f3d7bb6c5d28b29c6428744a58ee3/diff:/var/lib/docker/overlay2/55e18ec0a5caafed7907512ee5f2dd3e4db5cd701a1adfe5f4fe3bec2911e922/diff:/var/lib/docker/overlay2/ed7f62d6e611f876fb9eb55b06653b23198a915971fd1efdffbb61c33626f759/diff",

"MergedDir": "/var/lib/docker/overlay2/f2806e52d83beca84bd0bb5a137a4a70974fcecdede03e5bc1b3f5c3711619c9/merged",

"UpperDir": "/var/lib/docker/overlay2/f2806e52d83beca84bd0bb5a137a4a70974fcecdede03e5bc1b3f5c3711619c9/diff",

"WorkDir": "/var/lib/docker/overlay2/f2806e52d83beca84bd0bb5a137a4a70974fcecdede03e5bc1b3f5c3711619c9/work"

}

},

"Mounts": [

{

"Type": "bind",

"Source": "/home/docker/redis/date",

"Destination": "/data",

"Mode": "",

"RW": true,

"Propagation": "rprivate"

}

],

=============================省略================================

}

}

}

]

[docker@VM\_121\_116\_centos date]$



可以看到 将  /home/docker/redis/date 挂载到 /data

讲了这么多怎么基于 Volume 互联呢?  我们刚刚把宿主机一个指定的目录挂载到了容器内部, 那么要是我们把这个目录挂载到多个容器有什么效果呢?

我们再次创建一个容器还是挂在这个目录

[docker@VM\_121\_116\_centos date]$ docker run -it -p 26379:6379 -v ~/redis/date:/data --name redistwo redis /bin/bash

root@cf0ca7e833bb:/data# ls

appendonly.aof root testdir tomcat

root@cf0ca7e833bb:/data# mkdir testdirtwo

root@cf0ca7e833bb:/data# exit

exit

为了区别刚刚那个, 我们加了 --name 参数, 进入容器发现有刚刚的建的  testdir, 我们再次建一个 testdirtwo, 再次回到上个容器看看

[docker@VM\_121\_116\_centos date]$ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

cf0ca7e833bb redis "docker-entrypoint..." 3 minutes ago Exited (0) 2 minutes ago redistwo

b385ca73cad3 redis "docker-entrypoint..." 26 minutes ago Exited (0) 25 minutes ago sad\_murdock

e8abf8c09a26 mysql:5.6 "docker-entrypoint..." 3 weeks ago Exited (0) 2 weeks ago mysql5.6

[docker@VM\_121\_116\_centos date]$ docker start b385ca73cad3

b385ca73cad3

[docker@VM\_121\_116\_centos date]$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

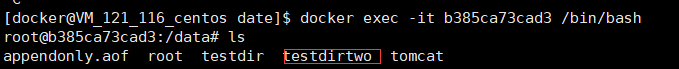
b385ca73cad3 redis "docker-entrypoint..." 28 minutes ago Up 7 seconds 0.0.0.0:6379->6379/tcp sad\_murdock

[docker@VM\_121\_116\_centos date]$ docker exec -it b385ca73cad3 /bin/bash

root@b385ca73cad3:/data# ls

appendonly.aof root testdir testdirtwo tomcat

root@b385ca73cad3:/data#



有第二个容器创建的文件夹, 这样就容器之间的互联

对于跨越宿主机的容器互联, 可是在linux采取共享目录等技术, 或者分布式文件系统

有 iscsi nfs ceph 等

1.3 数据容器共享解决方案(volumes-form)

将本地目录挂载到容器, 在 DockerFile 中是不推荐使用的, 因为这样的容器就不可移植了.

就要不指定宿主机的目录, 使用容器存储位置的目录.

如下:

[docker@VM\_121\_116\_centos date]$ docker run -it -p 6379:6379 -v /data --name oneredis redis /bin/bash

root@6636fb7d293c:/data# mkdir redis

root@6636fb7d293c:/data# ls

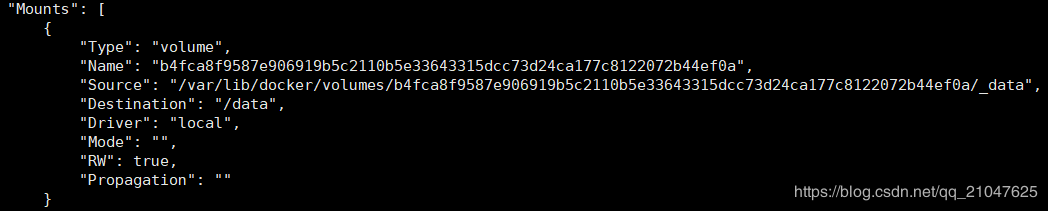
redis

root@6636fb7d293c:/data# exit

exit

[docker@VM\_121\_116\_centos date]$ docker inspect oneredis

查看挂载信息结果: 将 /var/lib/docker/volumes/b4fca8f9587e906919b5c2110b5e33643315dcc73d24ca177c8122072b44ef0a/\_data 挂载到了 /data



我们没有实现约定好共享目录,就不能体检知道挂载的目录,  其他容器怎么从这里读取呢?

这个时候要用 --volumes-from

示例:

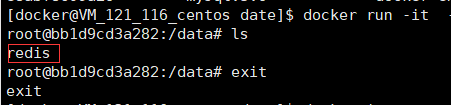
[docker@VM\_121\_116\_centos date]$ docker run -it -p 26379:6379 --volumes-from=oneredis --name tworedis redis /bin/bash

root@bb1d9cd3a282:/data# ls

redis

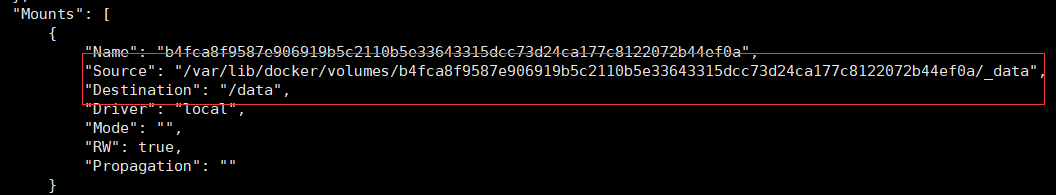
root@bb1d9cd3a282:/data# exit

exit



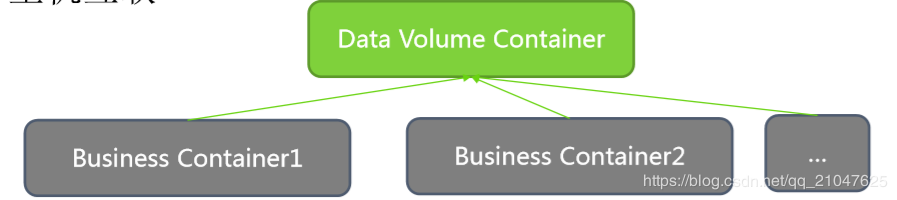
能看到第一个容器创建的目录

再看看容器挂载



和第一个一模一样

这样就解决的移植性的问题



一般的解决方案就是 建议一个 数据共享的管理容器, 其他业务容器 volumes-from 数据管理容器, 这样实现业务容器互联