[mydocker]---通过例子理解存储驱动AUFS

nicktming 关注
2019.05.11 09:55:51 字数 1,745 阅读 619

前言

> Docker 内置多种存储驱动. 最开始采用 AUFS 作为文件系统, 其分层概念实现了多个 Container 可以共享同一个 image . 由于 AUFS 未并入 Linux 内核, 且只支持 Ubuntu , 因此 Docker 0.7 版本中引入了存储驱动, 目前 Docker 支持5中存储驱动.

文件系统/存储	存储驱动名称
OverlayFs	overlay/overlay2
AUFS	aufs
Btrfs	btrfs
Device Mapper	devicemapper
VFS	vfs
ZFS	zfs

> 本文主要介绍 AUFS , 会先通过一个例子来理解 AUFS 是如何工作的, 最后会解释其原理.

AUFS

> AUFS 能透明覆盖一或多个现有文件系统的层状文件系统,把多层合并成文件系统的单层表示.

例子

```
root@nicktming:~/aufs# pwd
/root/aufs
root@nicktming:~/aufs# ls
run.sh
root@nicktming:~/aufs# cat run.sh
mkdir container-layer
echo "I am container-layer" > container-layer/container-layer.txt

mkdir mnt

for i in {1..3}
do
mkdir -p image-layer$i/subdir$i
echo "I am image layer $i" > image-layer$i/image-layer$i.txt
echo "subdir $i" > image-layer$i/subdir$i/subdir$i.txt
done
```


nicktming 关注
总资产98 (约9.01元)

[istio源码分析][citadel] citadel之node_agent_k8s(ingressgateway
阅读 140

[istio源码分析][citadel] citadel之istio_ca(grpc server)
阅读 57

```
root@nicktming:~/aufs# ./run.sh
root@nicktming:~/aufs# tree
.
|-- container-layer
|   `-- container-layer.txt
|-- image-layer1
|   |-- image-layer1.txt
|   `-- subdir1
|       `-- subdir1.txt
|-- image-layer2
|   |-- image-layer2.txt
|   `-- subdir2
|       `-- subdir2.txt
|-- image-layer3
|   |-- image-layer3.txt
|   `-- subdir3
|       `-- subdir3.txt
|-- mnt
`-- run.sh

8 directories, 8 files
```

> 可以看到当前 `aufs` 文件夹的目录结构. 并且每个文件的内容如下:

```
root@nicktming:~/aufs# cat container-layer/container-layer.txt
I am container-layer
root@nicktming:~/aufs# cat image-layer1/image-layer1.txt
I am image layer 1
root@nicktming:~/aufs# cat image-layer1/subdir1/subdir1.txt
subdir 1
root@nicktming:~/aufs# cat image-layer2/image-layer2.txt
I am image layer 2
root@nicktming:~/aufs# cat image-layer2/subdir2/subdir2.txt
subdir 2
root@nicktming:~/aufs# cat image-layer3/image-layer3.txt
I am image layer 3
root@nicktming:~/aufs# cat image-layer3/subdir3/subdir3.txt
subdir 3
root@nicktming:~/aufs# ls mnt/
root@nicktming:~/aufs#
```

> 可以将这些文件目录联合起来挂载到 `mnt` 目录下.

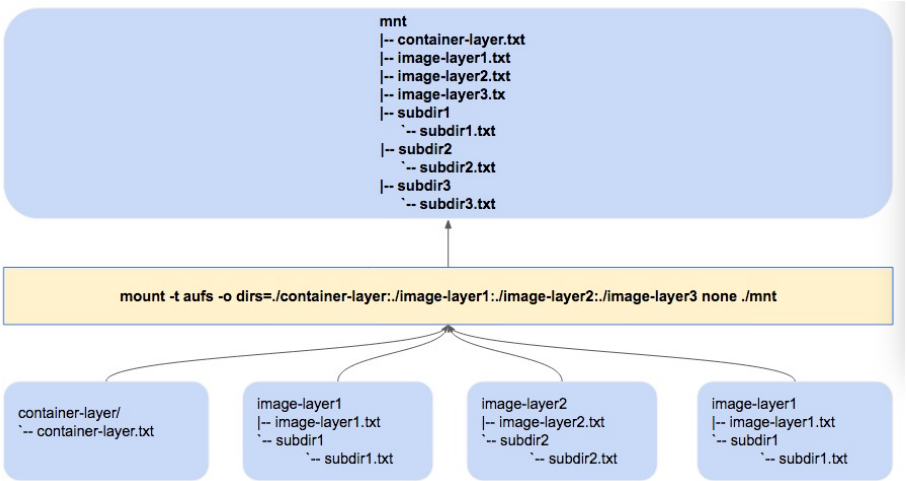
```
root@nicktming:~/aufs# mount -t aufs -o dirs=./container-layer:./image-layer1:./image-lay
root@nicktming:~/aufs# cat /sys/fs/aufs/
config               si_e9a1e7a68699fcc6/
root@nicktming:~/aufs# cat /sys/fs/aufs/si_e9a1e7a68699fcc6/*
/root/aufs/container-layer=rw
/root/aufs/image-layer1=ro
/root/aufs/image-layer2=ro
/root/aufs/image-layer3=ro
64
65
66
67
/root/aufs/container-layer/.aufs.xino
```

> 可以看到 `aufs` 文件系统会默认把 `dirs` 后的第一个文件设置为**可读写**, 其余文件设置为**只读**. 接下来看一下整个目录结构.

```
root@nicktming:~/aufs# tree
.
|-- container-layer
|   `-- container-layer.txt
|-- image-layer1
|   |-- image-layer1.txt
|   `-- subdir1
|       `-- subdir1.txt
|-- image-layer2
|   |-- image-layer2.txt
|   `-- subdir2
|       `-- subdir2.txt
|-- image-layer3
|   |-- image-layer3.txt
|   `-- subdir3
|       `-- subdir3.txt
|-- mnt
|   |-- container-layer.txt
|   |-- image-layer1.txt
|   |-- image-layer2.txt
|   |-- image-layer3.txt
|   |-- subdir1
|   |   `-- subdir1.txt
|   |-- subdir2
|   |   `-- subdir2.txt
|   `-- subdir3
|       `-- subdir3.txt
`-- run.sh

11 directories, 15 files
```

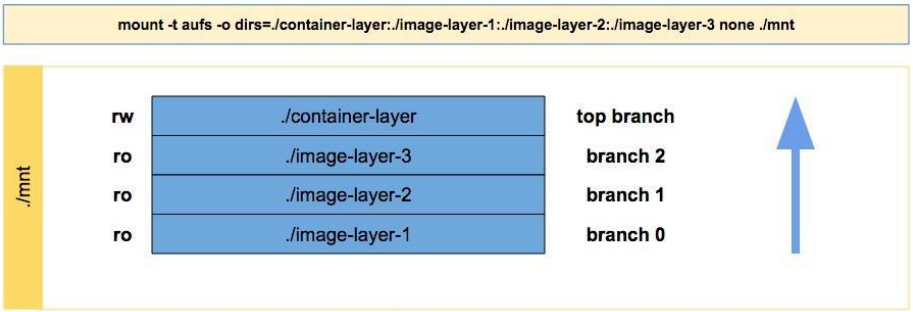
可以看到挂载点 `mnt` 中已经有每个被挂载的文件, 也就是说将不同目录挂载到同一个虚拟文件系统. 无论底下有多少层都是只读的,只有最上层的文件系统是课写的, 也就是这里的 `container-layer` .



layer.png

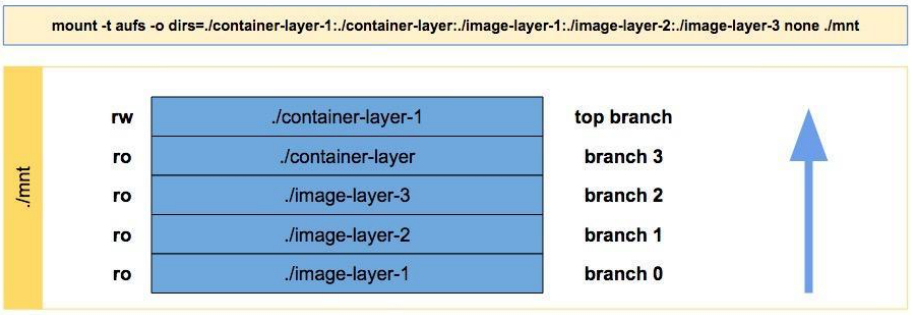
将多个目录合并成一个虚拟文件系统,成员目录称为虚拟文件系统的一个分支(`branch`). 也就是说 `container-layer` , `image-layer1` , `image-layer2` , `image-layer3` 都被称为 `branch` . 每个 `branch` 有3个权限, 只读(`ro`),读写(`rw`),写隐藏(`wo`). 一般情况下, `aufs` 只有最上层的

> branch (在这里是 **container-layer**)有读写权限, 其余 branch 为只读权限.



6791554437422_.pic.jpg

如果增加一层的话,也只有最顶层的文件有可读写权限.



6771554437359_.pic.jpg

> mnt 称为挂载点, 用户做操作是在 mnt 挂载点上做操作. 接下来可以做些简单的操作来看看做些文件是如何变化的.

修改

> 修改 image-layer1.txt 中的内容.

```
root@nicktming:~/aufs# cat mnt/image-layer1.txt
I am image layer 1
root@nicktming:~/aufs# echo "\n write something to image/layer1/image-layer1.txt\n" >> mn
root@nicktming:~/aufs# cat mnt/image-layer1.txt
I am image layer 1
\n write something to image/layer1/image-layer1.txt\n
```

> 可以看到 mnt/image-layer1.txt 中的内容确实发生了变化, 接下来看看 image-layer1/image-

> layer1.txt 中的内容是否发生了改变.

```
root@nicktming:~/aufs# cat image-layer1/image-layer1.txt
I am image layer 1
```

> 可以看到 image-layer1/image-layer1.txt 中的内容没有丝毫改变. 因为 mnt 只是个挂载点, 当取消挂载的时候 mnt 里面的内容都会没有了的,那真正写上去的内容在哪里呢?此时查看一下整个目录的结构.

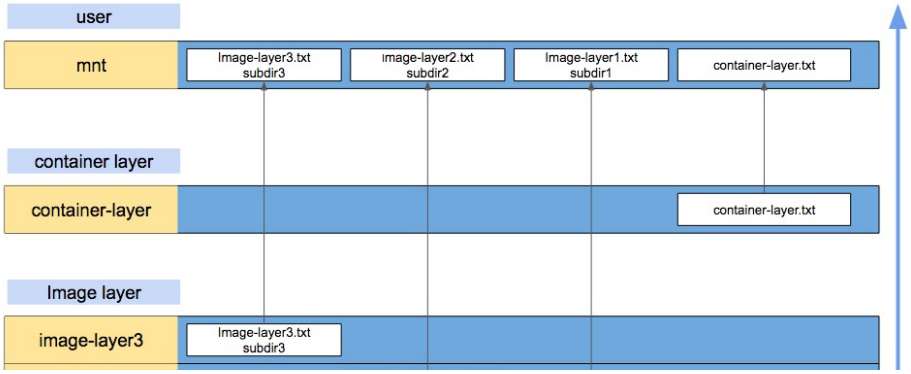
```
root@nicktming:~/aufs# tree
.
|-- container-layer
|   |-- container-layer.txt
|   `-- image-layer1.txt
|-- image-layer1
|   |-- image-layer1.txt
|   `-- subdir1
|       `-- subdir1.txt
|-- image-layer2
|   |-- image-layer2.txt
|   `-- subdir2
|       `-- subdir2.txt
|-- image-layer3
|   |-- image-layer3.txt
|   `-- subdir3
|       `-- subdir3.txt
|-- mnt
|   |-- container-layer.txt
|   |-- image-layer1.txt
|   |-- image-layer2.txt
|   |-- image-layer3.txt
|   |-- subdir1
|   |   `-- subdir1.txt
|   |-- subdir2
|   |   `-- subdir2.txt
|   `-- subdir3
|       `-- subdir3.txt
`-- run.sh

11 directories, 16 files
```

> 可以看到 container-layer 文件夹下面多了一个 image-layer1.txt , 查看里面的内容.

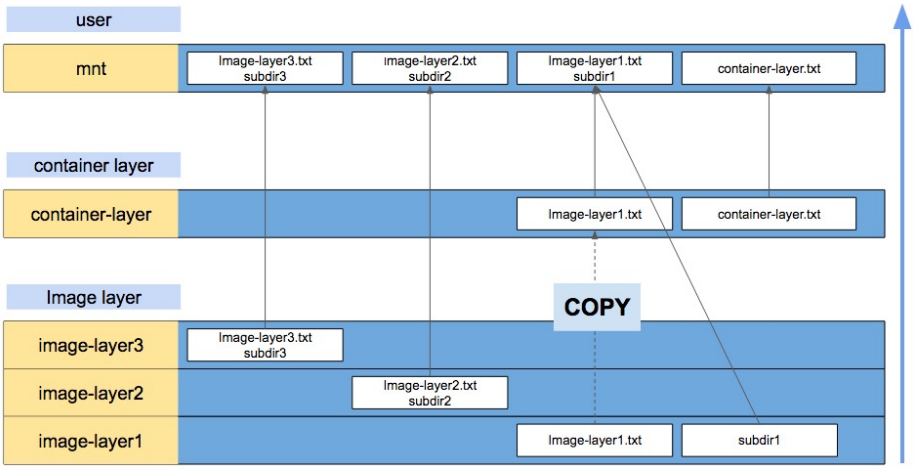
```
root@nicktming:~/aufs# cat container-layer/image-layer1.txt
I am image layer 1
\n write something to image/layer1/image-layer1.txt\n
```

> 发现 container-layer/image-layer1.txt 正是所要的修改. 从这里可以看到 container-layer 确实可读写的, image-layer1 是只读.



mnt.png

从 `user` 的角度可以看到所有被挂载的文件, 当修改 `mnt` 的 `image-layer1.txt` 时, 可读写层也就是 `container-layer` 会从只读层也就是 `image-layer1` 中把对应的 `image-layer1.txt` 复制到可写层 `container-layer` 并且修改, 当 `user` 看到文件的时候读到的 `container-layer` 的 `image-layer1.txt` , 因为此时 `container/image-layer1.txt` 会覆盖 `image-layer1/image-layer1.txt` , 但是 `image-layer1/image-layer1.txt` 并没有做任何改变, 整个 `image-layer1` 层没有做任何改变. 如下图所示:



update.png

删除

删除 `mnt/image-layer2.txt` 看看会有什么改变.

```
root@nicktming:~/aufs# rm mnt/image-layer2.txt
root@nicktming:~/aufs# cat mnt/image-layer2.txt
cat: mnt/image-layer2.txt: No such file or directory
root@nicktming:~/aufs# cat image-layer2/image-layer2.txt
I am image layer 2
```

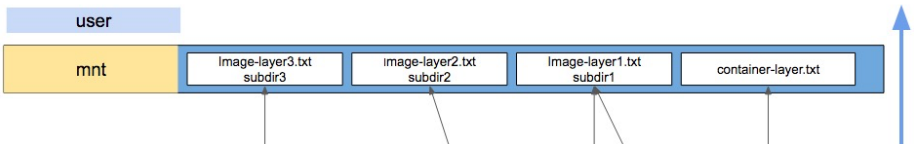
```
root@nicktming:~/aufs# tree
.
|-- container-layer
|   |-- container-layer.txt
|   `-- image-layer1.txt
|-- image-layer1
|   |-- image-layer1.txt
|   `-- subdir1
|       `-- subdir1.txt
|-- image-layer2
|   |-- image-layer2.txt
|   `-- subdir2
|       `-- subdir2.txt
|-- image-layer3
|   |-- image-layer3.txt
|   `-- subdir3
|       `-- subdir3.txt
|-- mnt
|   |-- container-layer.txt
|   |-- image-layer1.txt
|   |-- image-layer3.txt
|   |-- subdir1
|   |   `-- subdir1.txt
|   |-- subdir2
|   |   `-- subdir2.txt
|   `-- subdir3
|       `-- subdir3.txt
`-- run.sh

11 directories, 15 files
```

> 发现 mnt 目录下看不到 image-layer2.txt , 但是 image-layer2 层的 image-layer2.txt 仍然存在, 所以可以说只是逻辑删除了该 image-layer2.txt 文件, 或者是此 aufs 做了什么操作让用户看不到该 image-layer2.txt 文件.

> **原因:** 因为删除的 image-layer2.txt 是属于镜像层文件, 容器层 container-layer 会创建一个 .wh 前缀的隐藏文件, 从而实现对 image-layer2.txt 的隐藏.

```
root@nicktming:~/aufs# ls -la container-layer/
total 24
drwxr-xr-x 4 root root 4096 Apr  5 14:44 .
drwxr-xr-x 7 root root 4096 Apr  5 11:12 ..
-rw-r--r-- 1 root root   21 Apr  5 11:12 container-layer.txt
-rw-r--r-- 1 root root   73 Apr  5 13:44 image-layer1.txt
-r--r--r-- 2 root root    0 Apr  5 11:15 .wh.image-layer2.txt
-r--r--r-- 2 root root    0 Apr  5 11:15 .wh..wh.aufs
drwx------ 2 root root 4096 Apr  5 11:15 .wh..wh.orph
drwx------ 2 root root 4096 Apr  5 11:15 .wh..wh.plnk
```



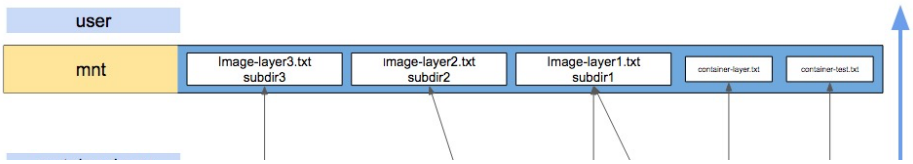
delete.png

增加

在 `mnt` 中增加文件. 基于上面的知识, 可以知道创建的文件肯定会存在于 `container-layer` 中, 并且也是在容器层 `container-layer` 中创建的.

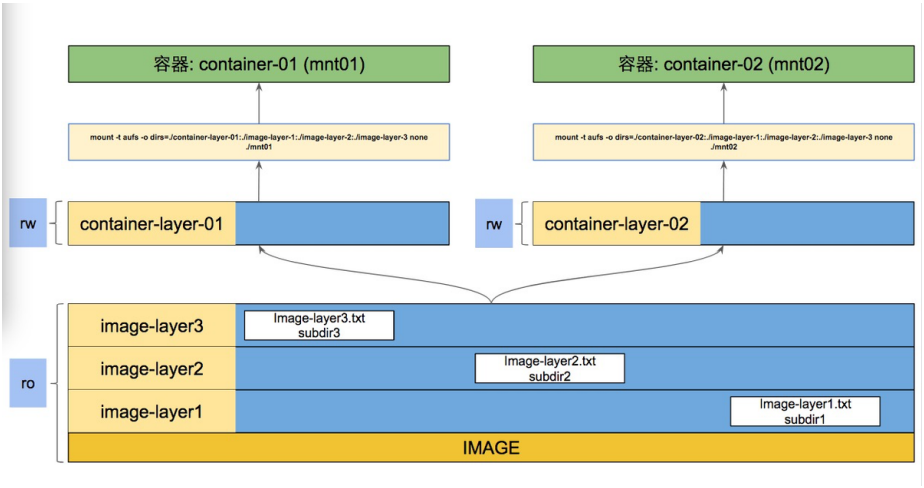
```
root@nicktming:~/aufs# echo "container-01" > mnt/container-test.txt
root@nicktming:~/aufs# cat mnt/container-test.txt
container-01
root@nicktming:~/aufs# cat container-layer/container-test.txt
container-01
root@nicktming:~/aufs# tree
.
|-- container-layer
|   |-- container-layer.txt
|   |-- container-test.txt
|   `-- image-layer1.txt
|-- image-layer1
|   |-- image-layer1.txt
|   `-- subdir1
|       `-- subdir1.txt
|-- image-layer2
|   |-- image-layer2.txt
|   `-- subdir2
|       `-- subdir2.txt
|-- image-layer3
|   |-- image-layer3.txt
|   `-- subdir3
|       `-- subdir3.txt
|-- mnt
|   |-- container-layer.txt
|   |-- container-test.txt
|   |-- image-layer1.txt
|   |-- image-layer3.txt
|   |-- subdir1
|   |   `-- subdir1.txt
|   |-- subdir2
|   |   `-- subdir2.txt
|   `-- subdir3
|       `-- subdir3.txt
`-- run.sh

11 directories, 17 files
```



图片.png

可以看到在容器层 `container-layer` 中增加删除修改容器层的文件是不会影响到镜像层中的任何内容的. 由此可以达到根据一个 `image` 启动多个容器的目的.



containers.png

umount

```
root@nicktming:~/aufs# df -h
df: ‘/tmp/tmp5RAi0E’: No such file or directory
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        50G  2.7G   45G   6% /
none            4.0K     0  4.0K   0% /sys/fs/cgroup
udev            487M   12K  487M   1% /dev
tmpfs           100M  356K  100M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none            497M   24K  497M   1% /run/shm
none            100M     0  100M   0% /run/user
none             50G  2.7G   45G   6% /root/aufs/mnt
root@nicktming:~/aufs# umount /root/aufs/mnt
root@nicktming:~/aufs# tree
.
|-- container-layer
|   |-- container-layer.txt
|   |-- container-test.txt
|   `-- image-layer1.txt
|-- image-layer1
|   |-- image-layer1.txt
|   `-- subdir1
|       `-- subdir1.txt
|-- image-layer2
|   |-- image-layer2.txt
|   `-- subdir2
|       `-- subdir2.txt
```

```
|-- image-layer3
|   |-- image-layer3.txt
|   `-- subdir3
|       `-- subdir3.txt
|-- mnt
`-- run.sh

8 directories, 10 files
```

> 可以看到 umount 之后 mnt 里面没有任何内容了, 所有内容保存在了 container-layer 之中了.

[mydocker]---通过例子理解存储驱动AUFS

> 通过上面的例子基本上可以理解 AUFS 是如何工作的, 其实其涉及到的技术为**写时复制(copy-on-write)**.

写时复制

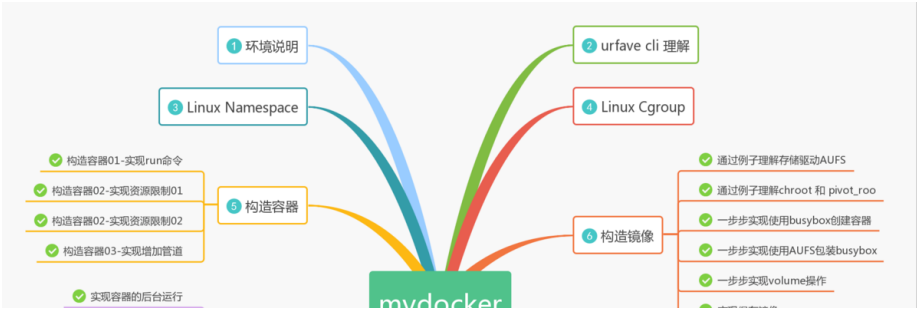
> 是一种对可修改资源实现高效复制的资源管理技术. 思想为**如果一个资源是重复的并且没有任何修改,这时并不需要立即创建一个新的资源, 因为这个资源可以被新旧实例共享. 创建新资源发生第一次写操作, 也就是对资源进行修改的时候.**

> 比如对一个 image 可以启动多个容器, 多个容器可以共享镜像层的文件, 这样可以减少大量的磁盘空间, 但是当某个容器的容器层需要对镜像层的文件进行修改的时候, 此时该容器的容器层会复制一份镜像层中此文件到容器层, 但是别的容器还是可以共享此镜像层的这个文件并不需要创建. **使用CoW可以有效的提高磁盘的利用率.**

参考

> 1. https://blog.csdn.net/yourun_cloud/article/details/62883721
> 2. http://dockone.io/article/1513
> 3. 自己动手写docker.(基本参考此书,加入一些自己的理解,加深对 docker 的理解)

全部内容



mydocker.png

写下你的评论... 评论0 赞 ···

1. [mydocker]---环境说明
2. [mydocker]---urfave cli 理解
3. [mydocker]---Linux Namespace
4. [mydocker]---Linux Cgroup
5. [mydocker]---构造容器01-实现run命令
6. [mydocker]---构造容器02-实现资源限制01
7. [mydocker]---构造容器02-实现资源限制02
8. [mydocker]---构造容器03-实现增加管道
9. [mydocker]---通过例子理解存储驱动AUFS
10. [mydocker]---通过例子理解chroot 和 pivot_root
11. [mydocker]---一步步实现使用busybox创建容器
12. [mydocker]---一步步实现使用AUFS包装busybox
13. [mydocker]---一步步实现volume操作
14. [mydocker]---实现保存镜像
15. [mydocker]---实现容器的后台运行
16. [mydocker]---实现查看运行中容器
17. [mydocker]---实现查看容器日志
18. [mydocker]---实现进入容器Namespace
19. [mydocker]---实现停止容器
20. [mydocker]---实现删除容器
21. [mydocker]---实现容器层隔离
22. [mydocker]---实现通过容器制作镜像
23. [mydocker]---实现cp操作
24. [mydocker]---实现容器指定环境变量
25. [mydocker]---网际协议IP
26. [mydocker]---网络虚拟设备veth bridge iptables
27. [mydocker]---docker的四种网络模型与原理实现(1)
28. [mydocker]---docker的四种网络模型与原理实现(2)
29. [mydocker]---容器地址分配
30. [mydocker]---网络net/netlink api 使用解析
31. [mydocker]---网络实现
32. [mydocker]---网络实现测试

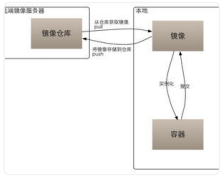
（3）UFS

Union File System Union File System是一种为linux FressBSD和Net...

爱喝咖啡的土拨鼠 阅读 107 评论 0 赞 0

docker的使用及原理

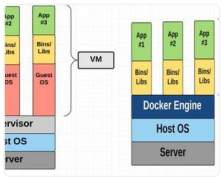
引子 如何在内网搭一个私人笔记 一步步搭建Leanote笔记 使用docker 什么是docker 下面是从wik...



阿里云云栖号 阅读 1,945 评论 0 赞 13

Docker底层技术

Docker容器技术已经发展了好些年，在很多项目都有应用，线上运行也很稳定。整理了部分Docker的学习笔记以及新...



Java大蜗牛 阅读 2,665 评论 2 赞 10

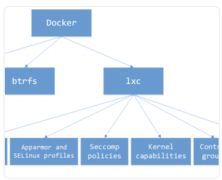
Docker理论与实践（一）

文章作者：Tyan博客：noahsnail.com 1. docker安装及介绍 1.1 docker安装 Mac...

SnailTyan 阅读 670 评论 0 赞 17

Docker核心技术

资料来源：http://blog.csdn.net/andylau00j/article/details/5458...



丹之 阅读 689 评论 0 赞 0