

[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

```
1 root@nicktming:~/aufs# pwd
2 /root/aufs
3 root@nicktming:~/aufs# ls
4 run.sh
5 root@nicktming:~/aufs# cat run.sh
6 mkdir container-layer
7 echo "I am container-layer" > container-layer/container-layer.txt
8
9 mkdir mnt
10
11 for i in {1..3}
12 do
13 mkdir -p image-layer$i/subdir$i
14 echo "I am image layer $i" > image-layer$i/image-layer$i.txt
15 echo "subdir $i" > image-layer$i/subdir$i/subdir$i.txt
16 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

```
17 | root@nicktming:~/aufs# ./run.sh
18 | root@nicktming:~/aufs# tree
19 | .
20 | |-- container-layer
21 | |   |-- container-layer.txt
22 | |-- image-layer1
23 | |   |-- image-layer1.txt
24 | |   |-- subdir1
25 | |       |-- subdir1.txt
26 | |-- image-layer2
27 | |   |-- image-layer2.txt
28 | |   |-- subdir2
29 | |       |-- subdir2.txt
30 | |-- image-layer3
31 | |   |-- image-layer3.txt
32 | |   |-- subdir3
33 | |       |-- subdir3.txt
34 | |-- mnt
35 | |-- run.sh
36 |
37 | 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:

```
1 | root@nicktming:~/aufs# cat container-layer/container-layer.txt
2 | I am container-layer
3 | root@nicktming:~/aufs# cat image-layer1/image-layer1.txt
4 | I am image layer 1
5 | root@nicktming:~/aufs# cat image-layer1/subdir1/subdir1.txt
6 | subdir 1
7 | root@nicktming:~/aufs# cat image-layer2/image-layer2.txt
8 | I am image layer 2
9 | root@nicktming:~/aufs# cat image-layer2/subdir2/subdir2.txt
10 | subdir 2
11 | root@nicktming:~/aufs# cat image-layer3/image-layer3.txt
12 | I am image layer 3
13 | root@nicktming:~/aufs# cat image-layer3/subdir3/subdir3.txt
14 | subdir 3
15 | root@nicktming:~/aufs# ls mnt/
16 | root@nicktming:~/aufs#
```

可以将这些文件目录联合起来挂载到 **mnt** 目录下.

```
1 | root@nicktming:~/aufs# mount -t aufs -o dirs=./container-layer:./image-layer1:./image-layer2:./image-layer3 ./aufs mnt/
2 | root@nicktming:~/aufs# cat /sys/fs/aufs/
3 | config si_e9a1e7a68699fcc6/
4 | root@nicktming:~/aufs# cat /sys/fs/aufs/si_e9a1e7a68699fcc6/*
5 | /root/aufs/container-layer=rw
6 | /root/aufs/image-layer1=ro
7 | /root/aufs/image-layer2=ro
8 | /root/aufs/image-layer3=ro
9 | 64
10 | 65
11 | 66
12 | 67
13 | /root/aufs/container-layer/.aufs.xino
```

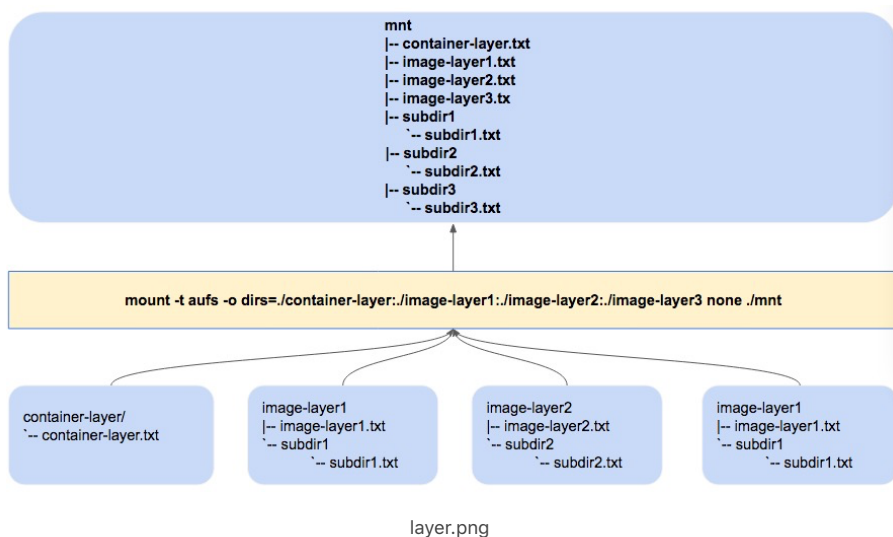
可以看到 **aufs** 文件系统会默认把 **dirs** 后的第一个文件设置为可读写, 其余文件设置为只读. 接下来看一下整个目录结构.

```

1 root@nicktming:~/aufs# tree
2 .
3 |-- container-layer
4 |   |-- container-layer.txt
5 |-- image-layer1
6 |   |-- image-layer1.txt
7 |   |-- subdir1
8 |       |-- subdir1.txt
9 |-- image-layer2
10 |   |-- image-layer2.txt
11 |   |-- subdir2
12 |       |-- subdir2.txt
13 |-- image-layer3
14 |   |-- image-layer3.txt
15 |   |-- subdir3
16 |       |-- subdir3.txt
17 |-- mnt
18 |   |-- container-layer.txt
19 |   |-- image-layer1.txt
20 |   |-- image-layer2.txt
21 |   |-- image-layer3.txt
22 |   |-- subdir1
23 |       |-- subdir1.txt
24 |   |-- subdir2
25 |       |-- subdir2.txt
26 |   |-- subdir3
27 |       |-- subdir3.txt
28 |-- run.sh
29
30 11 directories, 15 files

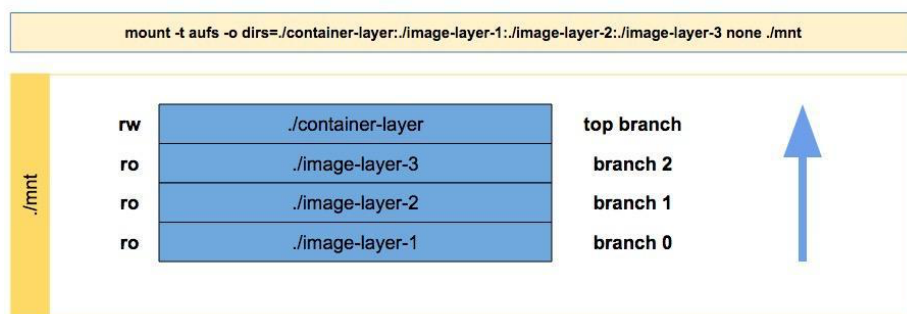
```

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`.



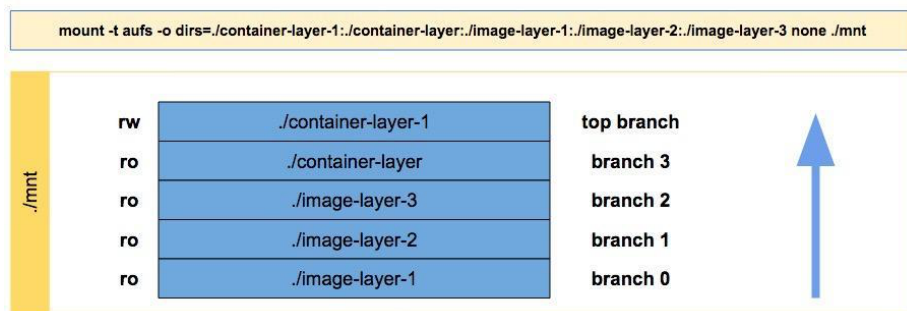
将多个目录合并成一个虚拟文件系统, 成员目录称为虚拟文件系统的分支 (branch). 也就是说 `container-layer`, `image-layer1`, `image-layer2`, `image-layer3` 都被称为 `branch`. 每个 `branch` 有3个权限, 只读 (ro), 读写 (rw), 写隐藏 (wo). 一般情况下, `aufs` 只有最上层的

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. 接下来可以做一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.

```
1 root@nicktming:~/aufs# cat mnt/image-layer1.txt
2 I am image layer 1
3 root@nicktming:~/aufs# echo "\n write something to image/layer1/image-layer1.txt\n" >> mnt/image-layer1.txt
4 root@nicktming:~/aufs# cat mnt/image-layer1.txt
5 I am image layer 1
6 \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.

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

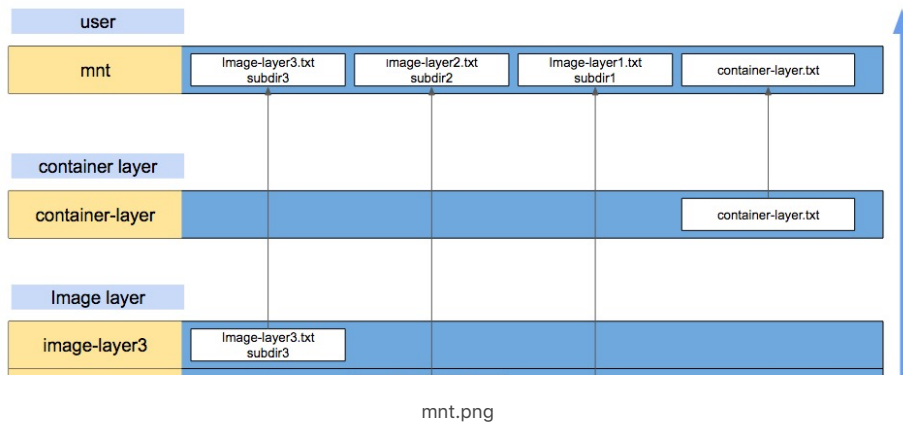
可以看到 `image-layer1/image-layer1.txt` 中的内容没有丝毫改变. 因为 `mnt` 只是个挂载点, 当取消挂载的时候 `mnt` 里面的内容都会没有了的, 那真正写上去的内容在哪里呢? 此时查看一下整个目录的结构.

```
1 | root@nicktming:~/aufs# tree
2 | .
3 | |-- container-layer
4 | |   |-- container-layer.txt
5 | |   |-- image-layer1.txt
6 | |-- image-layer1
7 | |   |-- image-layer1.txt
8 | |   |-- subdir1
9 | |     |-- subdir1.txt
10 |-- image-layer2
11 | |   |-- image-layer2.txt
12 | |   |-- subdir2
13 | |     |-- subdir2.txt
14 |-- image-layer3
15 | |   |-- image-layer3.txt
16 | |   |-- subdir3
17 | |     |-- subdir3.txt
18 |-- mnt
19 | |   |-- container-layer.txt
20 | |   |-- image-layer1.txt
21 | |   |-- image-layer2.txt
22 | |   |-- image-layer3.txt
23 | |   |-- subdir1
24 | |     |-- subdir1.txt
25 | |   |-- subdir2
26 | |     |-- subdir2.txt
27 | |   |-- subdir3
28 | |     |-- subdir3.txt
29 |-- run.sh
30 |
31 | 11 directories, 16 files
```

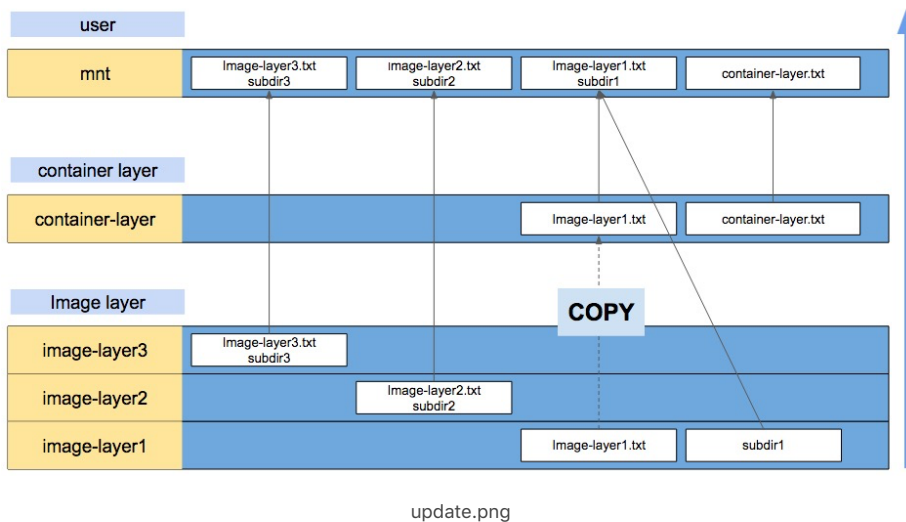
可以看到 `container-layer` 文件夹下面多了一个 `image-layer1.txt`, 查看里面的内容.

```
1 | root@nicktming:~/aufs# cat container-layer/image-layer1.txt
2 | I am image layer 1
3 | \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.



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:



删除

删除 `mnt/image-layer2.txt` 看看会有什么改变.

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

```

6 | root@nicktming:~/aufs# tree
7 | .
8 | |-- container-layer
9 | |   |-- container-layer.txt
10 | |   |-- image-layer1.txt
11 | |-- image-layer1
12 | |   |-- image-layer1.txt
13 | |   |-- subdir1
14 | |       |-- subdir1.txt
15 | |-- image-layer2
16 | |   |-- image-layer2.txt
17 | |   |-- subdir2
18 | |       |-- subdir2.txt
19 | |-- image-layer3
20 | |   |-- image-layer3.txt
21 | |   |-- subdir3
22 | |       |-- subdir3.txt
23 | |-- mnt
24 | |   |-- container-layer.txt
25 | |   |-- image-layer1.txt
26 | |   |-- image-layer3.txt
27 | |   |-- subdir1
28 | |       |-- subdir1.txt
29 | |       |-- subdir2
30 | |       |-- subdir2.txt
31 | |       |-- subdir3
32 | |       |-- subdir3.txt
33 | |-- run.sh
34 |
35 | 11 directories, 15 files

```

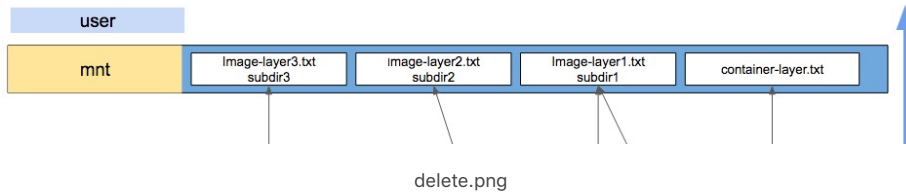
发现 `mnt` 目录下看不到 `image-layer2.txt`, 但是 `image-layer2` 层的 `image-layer2.txt` 仍然存在, 所以可以说只是逻辑删除了该 `image-layer2.txt` 文件, 或者是此 `aufs` 做了什么操作让用户看不到该 `image-layer2.txt` 文件.

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.

```

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

```



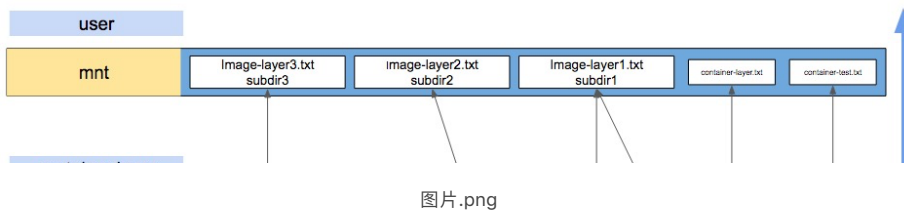
增加

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.

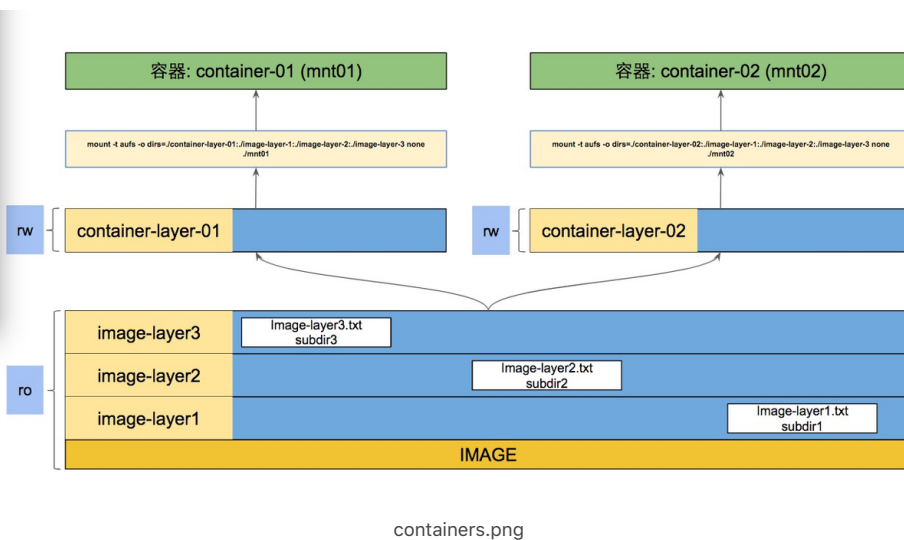
```

1 | root@nicktming:~/aufs# echo "container-01" > mnt/container-test.txt
2 | root@nicktming:~/aufs# cat mnt/container-test.txt
3 | container-01
4 | root@nicktming:~/aufs# cat container-layer/container-test.txt
5 | container-01
6 | root@nicktming:~/aufs# tree
7 | .
8 | |-- container-layer
9 | |   |-- container-layer.txt
10 | |   |-- container-test.txt
11 | |   `-- image-layer1.txt
12 | |-- image-layer1
13 | |   |-- image-layer1.txt
14 | |   `-- subdir1
15 | |       |-- subdir1.txt
16 | |-- image-layer2
17 | |   |-- image-layer2.txt
18 | |   `-- subdir2
19 | |       |-- subdir2.txt
20 | |-- image-layer3
21 | |   |-- image-layer3.txt
22 | |   `-- subdir3
23 | |       |-- subdir3.txt
24 | |-- mnt
25 | |   |-- container-layer.txt
26 | |   |-- container-test.txt
27 | |   |-- image-layer1.txt
28 | |   |-- image-layer3.txt
29 | |   |-- subdir1
30 | |       |-- subdir1.txt
31 | |   |-- subdir2
32 | |       |-- subdir2.txt
33 | |   `-- subdir3
34 | |       |-- subdir3.txt
35 | `-- run.sh
36 |
37 | 11 directories, 17 files

```

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.



umount

```

1 root@nicktming:~/aufs# df -h
2 df: '/tmp/tmp5RAi0E': No such file or directory
3 Filesystem      Size  Used Avail Use% Mounted on
4 /dev/vda1       50G   2.7G   45G   6% /
5 none            4.0K   0    4.0K   0% /sys/fs/cgroup
6 udev            487M   12K  487M   1% /dev
7 tmpfs           100M   356K  100M   1% /run
8 none            5.0M   0    5.0M   0% /run/lock
9 none            497M   24K  497M   1% /run/shm
10 none            100M   0    100M   0% /run/user
11 none            50G   2.7G   45G   6% /root/aufs/mnt
12 root@nicktming:~/aufs# umount /root/aufs/mnt
13 root@nicktming:~/aufs# tree
14 .
15 |-- container-layer
16 |   |-- container-layer.txt
17 |   |-- container-test.txt
18 |   `-- image-layer1.txt
19 |-- image-layer1
20 |   |-- image-layer1.txt
21 |   `-- subdir1
22 |       |-- subdir1.txt
23 |-- image-layer2
24 |   |-- image-layer2.txt
25 |   `-- subdir2
26 |       |-- subdir2.txt

```

```
27 | |-- image-layer3
28 | |   |-- image-layer3.txt
29 | |   |-- subdir3
30 | |       |-- subdir3.txt
31 | |-- mnt
32 | |-- run.sh
33 |
34 | 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



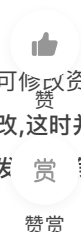
nicktming

关注

赞赏支持

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 赏 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

推荐阅读

看了28岁华为员工的工资表才知道: 牛逼的人注定会牛逼
阅读 13,155

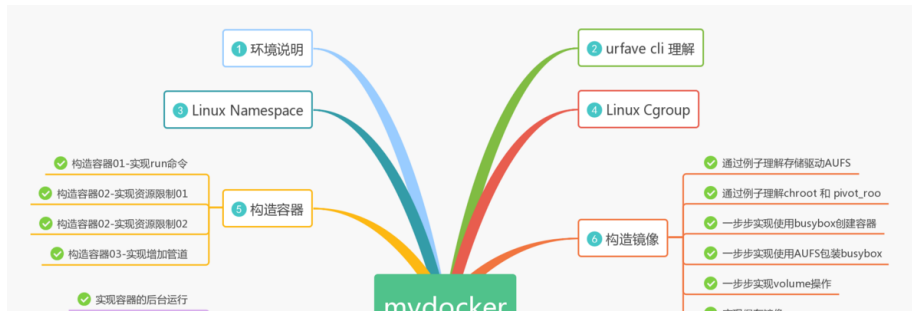
自律的人 生活处处有惊喜
阅读 22,028

邓超宣布息影退出娱乐圈, 孙俪一条微博让1000万人泪目
阅读 26,124

生活中有哪些残忍的真相?
阅读 41,758

今天我被辞退了
阅读 10,070

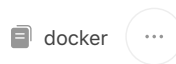




写下你的评论...

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



"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



nicktming 逢山开路 遇水架桥
总资产98 (约9.01元) 共写了13.3W字 获得224个赞 共353个粉丝

关注



推荐阅读

更多精彩内容 >

(3) UFS

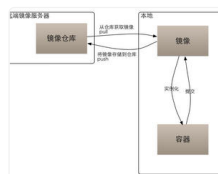
Union File System Union File System是一种为linux FreeBSD和Net...

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...

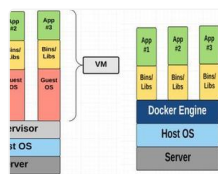
阿里云云栖号 阅读 1,945 评论 0 赞 13



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...

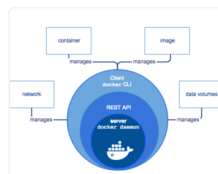
Java大蜗牛 阅读 2,665 评论 2 赞 10



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

