Docker核心实现技术

兰迪

目录

概述

Docker作为一项虚拟化技术，底层技术主要围绕两点：

- 隔离
- 可用

Namespace

Namespace 是 Linux 提供的一种内核级别的资源隔离技术，使处于不同 namespace 的进程拥有独立的全局系统资源。

命名空间	系统调用参数	Linux发行版本
Mount namespaces	CLONE_NEWNS	Linux 2.4.19
UTS namespaces	CLONE_NEWUTS	Linux 2.6.19
IPC namespaces	CLONE_NEWIPC	Linux 2.6.19
PID namespaces	CLONE_NEWPID	Linux 2.6.24
Network namespaces	CLONE_NEWNET	Linux 2.6.29
User namespaces	CLONE_NEWUSER	Linux 3.8
Cgroup namespaces	CLONE_NEWCGROUP	Linux 4.6

- clone()
- setns()
- unshare()

```
int clone(int (*child_func)(void *), void *child_stack
          , int flags, void *arg);
```

```
root@cb7e4ba53cad:/#
root@cb7e4ba53cad:/#
root@cb7e4ba53cad:/# ls -l /proc/$$/ns
total 0
lrwxrwxrwx 1 root root 0 Sep  4 03:27 cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 root root 0 Sep  4 03:27 ipc -> ipc:[4026532419]
lrwxrwxrwx 1 root root 0 Sep  4 03:27 mnt -> mnt:[4026532417]
lrwxrwxrwx 1 root root 0 Sep  4 03:27 net -> net:[4026532422]
lrwxrwxrwx 1 root root 0 Sep  4 03:27 pid -> pid:[4026532420]
lrwxrwxrwx 1 root root 0 Sep  4 03:27 user -> user:[4026531837]
lrwxrwxrwx 1 root root 0 Sep  4 03:27 uts -> uts:[4026532418]
root@cb7e4ba53cad:/#
```

Mount namespaces

- Mount
- CLONE_NEWNS
- 继承关系
- peer group
- propagation type
- shared subtrees

```
root@5d0d7283a15b:/proc/1# ls /proc/$$
attr       clear_refs       cpuset   fd       limits     mem         net        oom_score_adj  root        setgroups  statm    timers
autogroup  cmdline          cwd      fdinfo   loginuid   mountinfo   ns         pagemap        sched       smaps      status   timerslack_ns
auxv       comm             environ  gid_map  map_files  mounts      oom_adj    personality    schedstat   stack      syscall  uid_map
cgroup     coredump_filter  exe      io       maps       mountstats  oom_score  projid_map     sessionid   stat       task     wchan
```

UTS namespaces

- hostname & NIS domain name
- 老版本内核：全局变量
- 新版本内核：task_struct

```
 */
struct nsproxy {
        atomic_t count;
        struct uts_namespace *uts_ns;
        struct ipc_namespace *ipc_ns;
        struct mnt_namespace *mnt_ns;
        struct pid_namespace *pid_ns_for_children;
        struct net           *net_ns;
        struct cgroup_namespace *cgroup_ns;
};
extern struct nsproxy init_nsproxy;
```

IPC namespaces

- 进程间通信 Inter-Process Communication：
  - 管道：pipe 匿名管道、s_pipe 流管道和FIFO 命名管道
  - 信号 Signal
  - 消息队列
  - 共享内存
  - 信号量
  - 套接字 Socket
- System V IPC objects 和 POSIX message queues

PID namespaces

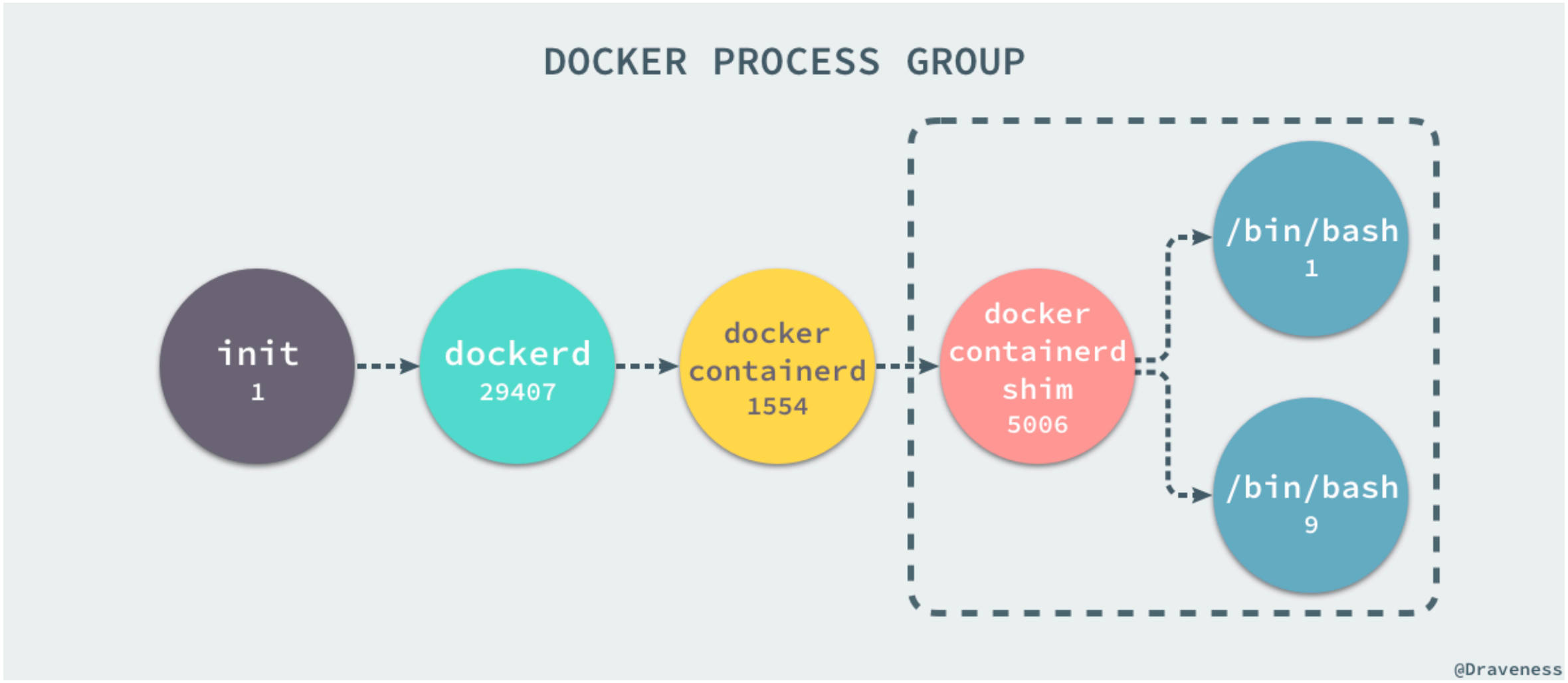
- /proc/[PID]
- init 进程 & systemd
- 嵌套关系：最多32层&MAX_PID_NS_LEVEL
- PID namespaces 在进程创建时决定

```
root@5d0d7283a15b:/proc/1# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 06:11 pts/0    00:00:00 bash
root        15     1  0 06:12 pts/0    00:00:00 ps -ef
root@5d0d7283a15b:/proc/1# cd /proc/1
root@5d0d7283a15b:/proc/1# ls
attr       clear_refs       cpuset   fd       limits     mem         net        oom_score_adj  root       setgroups  statm    timers
autogroup  cmdline          cwd      fdinfo   loginuid   mountinfo   ns         pagemap        sched      smaps      status   timerslack_ns
auxv       comm             environ  gid_map  map_files  mounts      oom_adj    personality    schedstat  stack      syscall  uid_map
cgroup     coredump_filter  exe      io       maps       mountstats  oom_score  projid_map     sessionid  stat       task     wchan
root@5d0d7283a15b:/proc/1#
```

Docker 进程模型

- rpc
- dockerd、ctr、containerd、shim、runc

```
docker     ctr
  |         |
  V         V
dockerd -> containerd ---> shim -> runc -> runc init -> process
                      |--- > shim -> runc -> runc init -> process
                      +--- > shim -> runc -> runc init -> process
```



User namespaces

- 隔离user权限相关的Linux资源：**user id**、**group id**、**capabilities**等
- 嵌套
- /proc/PID/uid_map & /proc/PID/gid_map
- linux 下的每个 namespace 都有一个 user namespace 与之相连

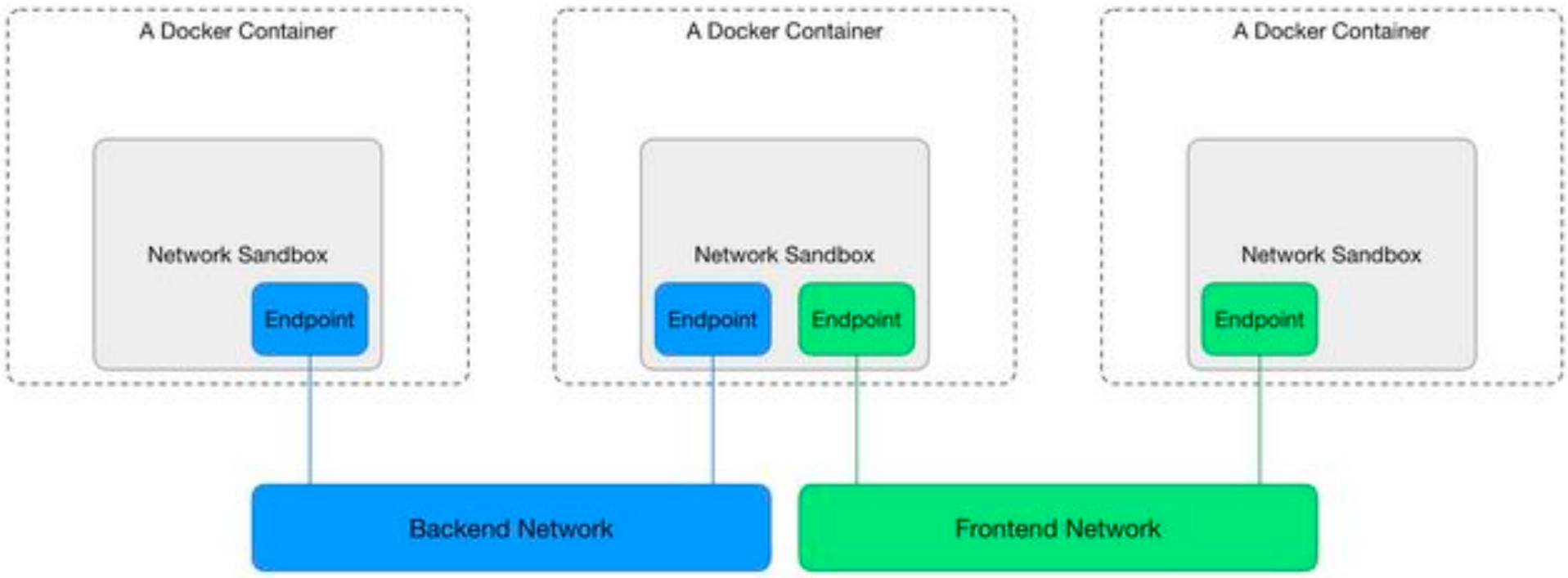
改变文件的所属者(chown())	CAP_CHOWN
向进程发送信号(kill(), signal())	CAP_KILL
改变进程的uid(setuid(), setreuid(), setresuid()等)	CAP_SETUID
trace进程(ptrace())	CAP_SYS_PTRACE
设置系统时间(settimeofday(), stime()等)	CAP_SYS_TIME

Network namespace

- 隔离网络设备、IP地址、端口等。
- 拥有独立的网络栈、路由表、防火墙规则等

网络

- Container Network Model，CNM网络模型
  - Sandbox
  - Endpoint
  - Network
- libnetwork



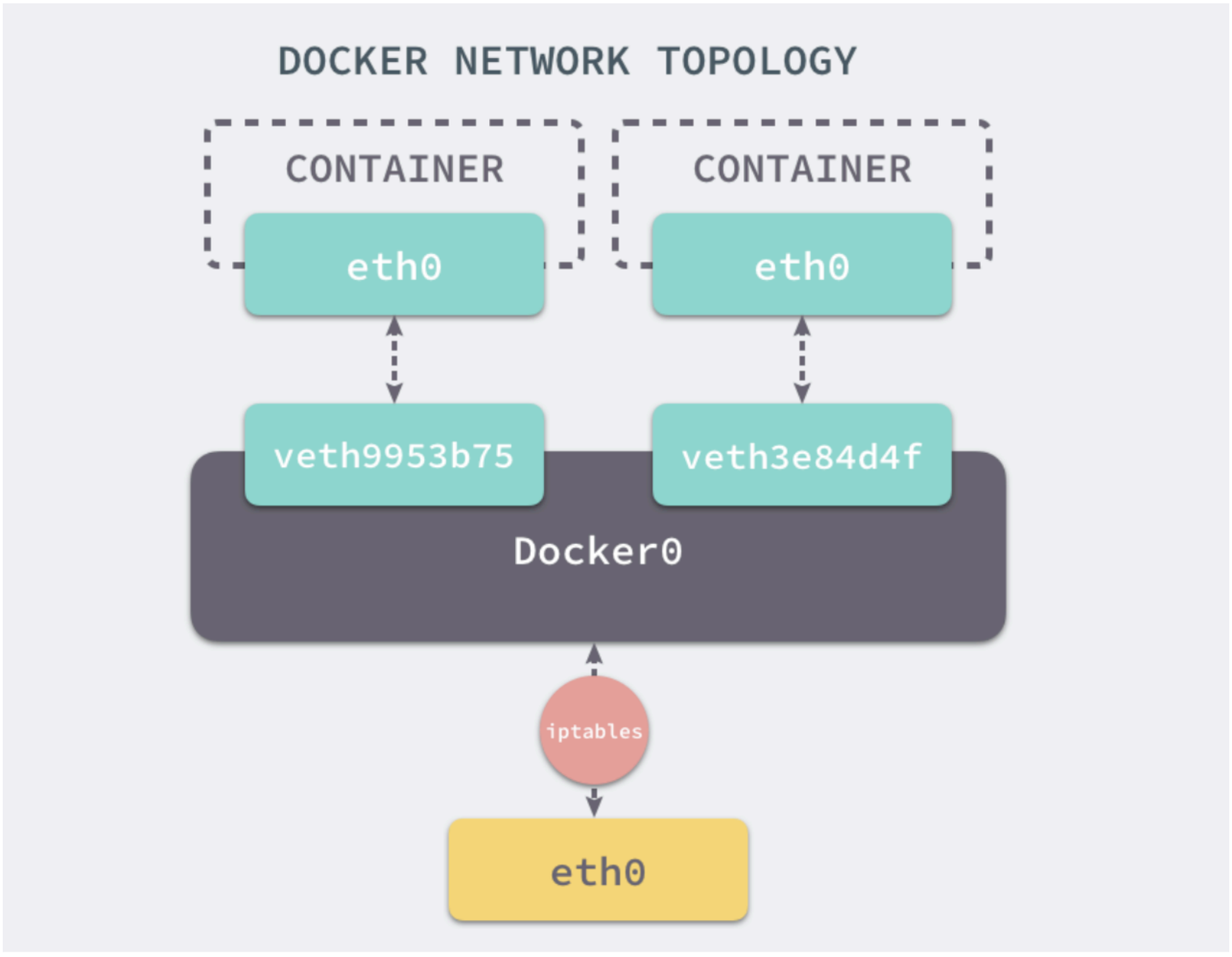
Docker 提供了五种网络模式：

- Bridge
- Host
- Container
- None
- user-defined：bridge、host、overlay、macvlan、third-party plugins

Network settings

```
--dns=[]           : Set custom dns servers for the container
--network="bridge" : Connect a container to a network
                      'bridge': create a network stack on the default Docker bridge
                      'none': no networking
                      'container:<name|id>': reuse another container's network stack
                      'host': use the Docker host network stack
                      '<network-name>|<network-id>': connect to a user-defined network
--network-alias=[] : Add network-scoped alias for the container
--add-host=""      : Add a line to /etc/hosts (host:IP)
--mac-address=""   : Sets the container's Ethernet device's MAC address
--ip=""            : Sets the container's Ethernet device's IPv4 address
--ip6=""           : Sets the container's Ethernet device's IPv6 address
--link-local-ip=[] : Sets one or more container's Ethernet device's link local IPv4/IPv6 addresses
```

- Bridge模式下
  - sandbox = network namespace
  - endpoint = 虚拟的以外网网卡接口
  - network = Docker0 Bridge

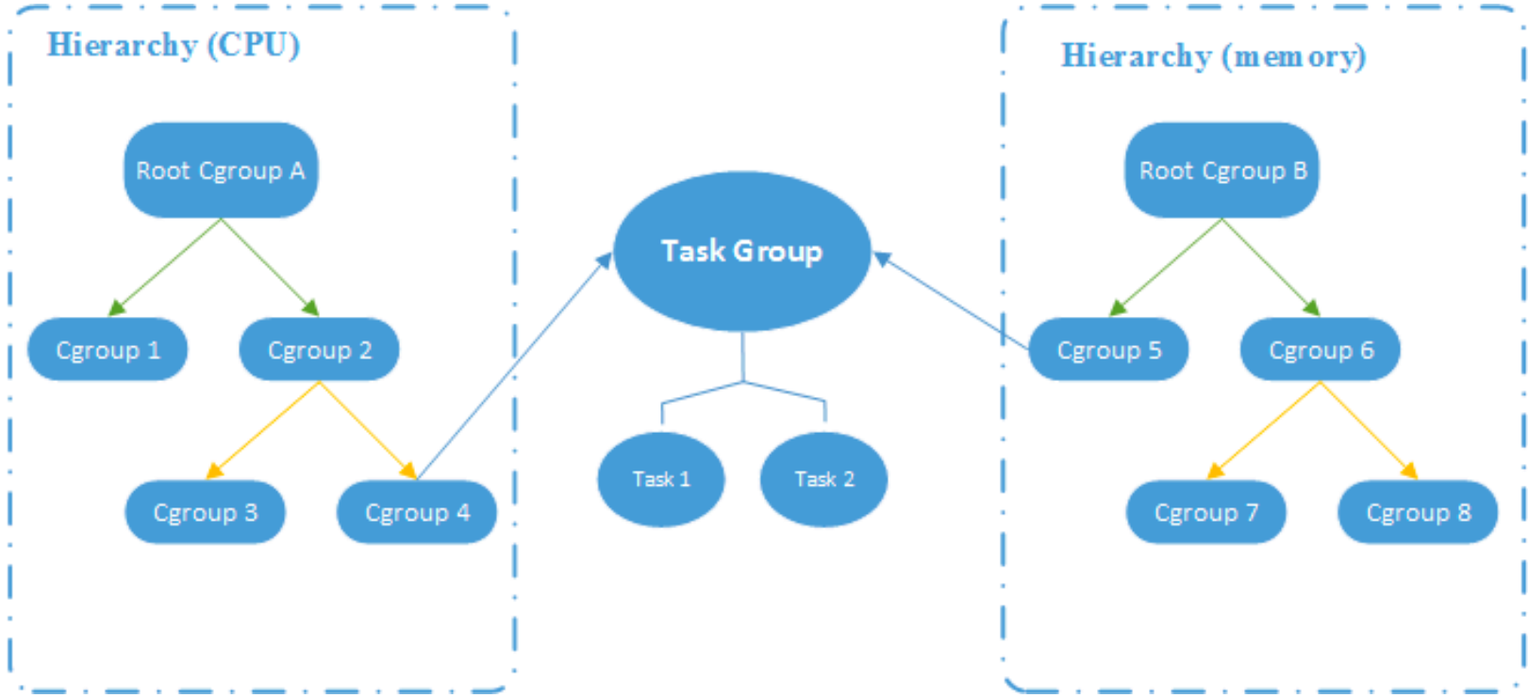


```
[read-only@bj1b-144 ~]$ brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.1a8e5f146478       no              bond1
                                                        veth08a34c3
                                                        veth15045ad
                                                        veth2cc19e5
                                                        veth4101a46
                                                        veth4342764
                                                        veth4b3797a
                                                        veth51c7d7e
                                                        veth5251292
                                                        veth532994d
                                                        veth5e2bf35
                                                        veth619a656
                                                        veth66e6b28
                                                        veth6960efd
                                                        veth7b2df98
                                                        veth844488c
                                                        vethba6b9ab
                                                        vethbae8539
                                                        vethd131d8d
                                                        vethd80163e
                                                        vethda9b417
                                                        vethdf75022
docker0         8000.000000000000       no
foobridge0              8000.000000000000       no
```

```
[read-only@bj1b-144 ~]$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond1 state UP mode DEFAULT qlen 1000
    link/ether 48:df:37:0d:7e:7c brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500 qdisc mq master bond1 state UP mode DEFAULT qlen 1000
    link/ether 48:df:37:0d:7e:7c brd ff:ff:ff:ff:ff:ff
4: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN mode DEFAULT qlen 1
    link/ipip 0.0.0.0 brd 0.0.0.0
5: bond0: <BROADCAST,MULTICAST,MASTER> mtu 1500 qdisc noop state DOWN mode DEFAULT qlen 1000
    link/ether 6e:e3:49:34:da:bd brd ff:ff:ff:ff:ff:ff
6: bond1: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UP mode DEFAULT qlen 1000
    link/ether 48:df:37:0d:7e:7c brd ff:ff:ff:ff:ff:ff
7: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT qlen 1000
    link/ether 1a:8e:5f:14:64:78 brd ff:ff:ff:ff:ff:ff
8: foobridge0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT qlen 1000
    link/ether de:a4:d5:04:04:3f brd ff:ff:ff:ff:ff:ff
9: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN mode DEFAULT qlen 1000
    link/ether 12:f8:8d:0e:1b:98 brd ff:ff:ff:ff:ff:ff
2315: veth51c7d7e@if2314: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UP mode DEFAULT
    link/ether 1a:8e:5f:14:64:78 brd ff:ff:ff:ff:ff:ff link-netnsid 26
13: veth677b90f@if12: <BROADCAST,MULTICAST> mtu 1450 qdisc noop state DOWN mode DEFAULT
    link/ether 82:e3:be:49:53:e6 brd ff:ff:ff:ff:ff:ff link-netnsid 1
2331: veth844488c@if2330: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UP mode DEFAULT
    link/ether 72:68:b9:62:7d:33 brd ff:ff:ff:ff:ff:ff link-netnsid 42
2081: veth5e2bf35@if2080: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UP mode DEFAULT
    link/ether de:32:d3:99:ad:6c brd ff:ff:ff:ff:ff:ff link-netnsid 30
1573: veth2cc19e5@if1572: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UP mode DEFAULT
    link/ether aa:93:a5:17:b5:f4 brd ff:ff:ff:ff:ff:ff link-netnsid 24
811: veth6960efd@if810: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br0 state UP mode DEFAULT
    link/ether aa:c8:c9:19:6c:0b brd ff:ff:ff:ff:ff:ff link-netnsid 25
```

Control Groups

- 对物理资源如CPU、网络带宽等分组管理。
- task、group、hierarchy、subsystem



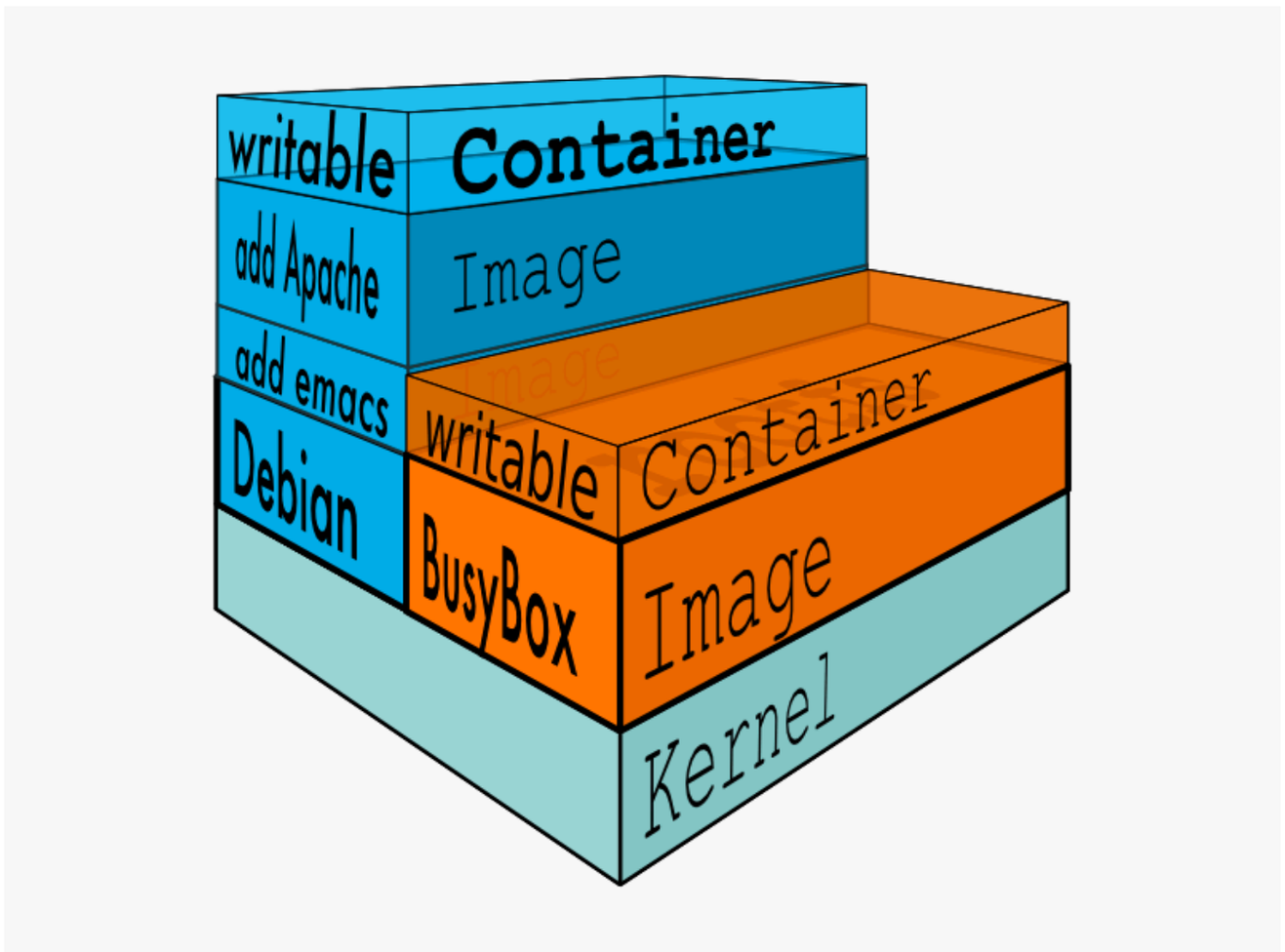
- Resource limiting 限制资源使用
- Prioritization 优先级控制
- Accounting 记录使用资源数
- Control 挂起、恢复执行进程。

Union File System

将不同物理位置下的文件合并到一个目录下，用来存储**镜像**和**容器**的层级数据。& linux mount

优点&特点：

- copy-on-write 写时复制
- allocate-on-damand 用时分配

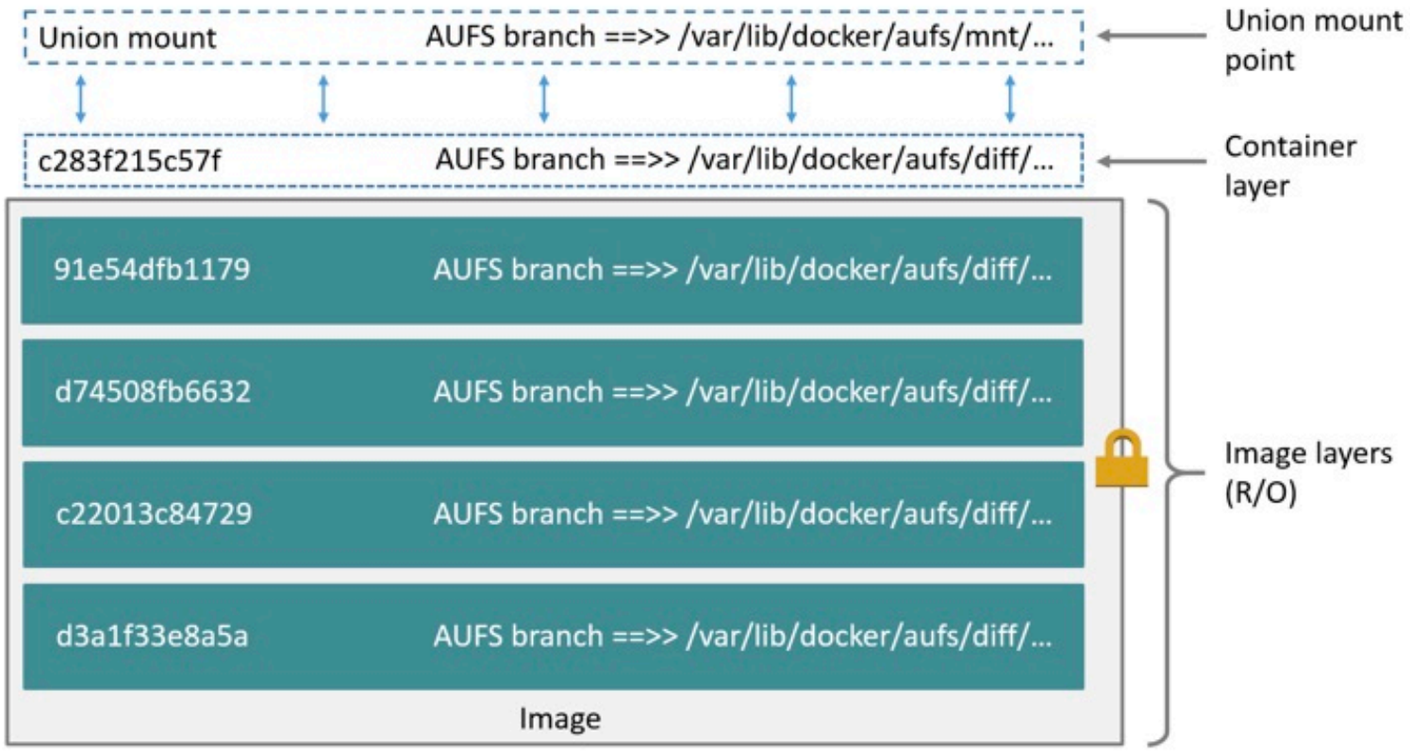


- AUFS
- devicemapper
- Overlay
- Overlay2
- vfs
- zfs

```
# landi @ landideMacBook in ~ [18:26:52]
$ docker info | grep Storage
Storage Driver: aufs
```

AUFS

支持将不同目录挂载到同一个虚拟文件系统下。



- diff、layers、mnt
- udfa参数：none、reval、notify
- whiteout隐藏文件

```
/var/lib/docker/aufs

➜ tree -L 1 /var/lib/docker/aufs
/var/lib/docker/aufs
├── diff
├── layers
└── mnt
```

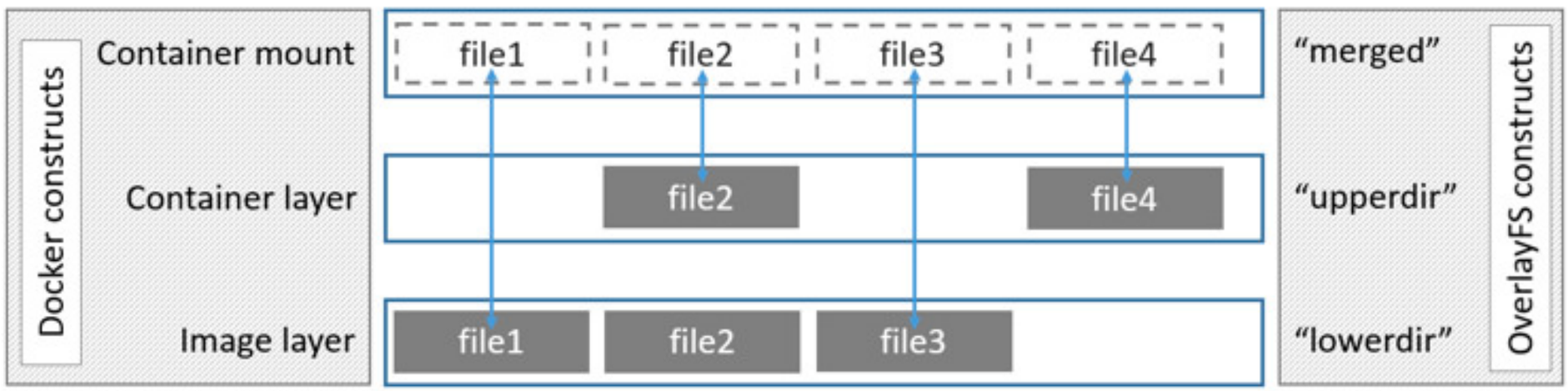
OverlayFS

- overlay 3.18 版本加入 linux 内核，overlay2 在 4.0版本
- /var/lib/docker/overlay/
- 硬链接、物理索引（inode）

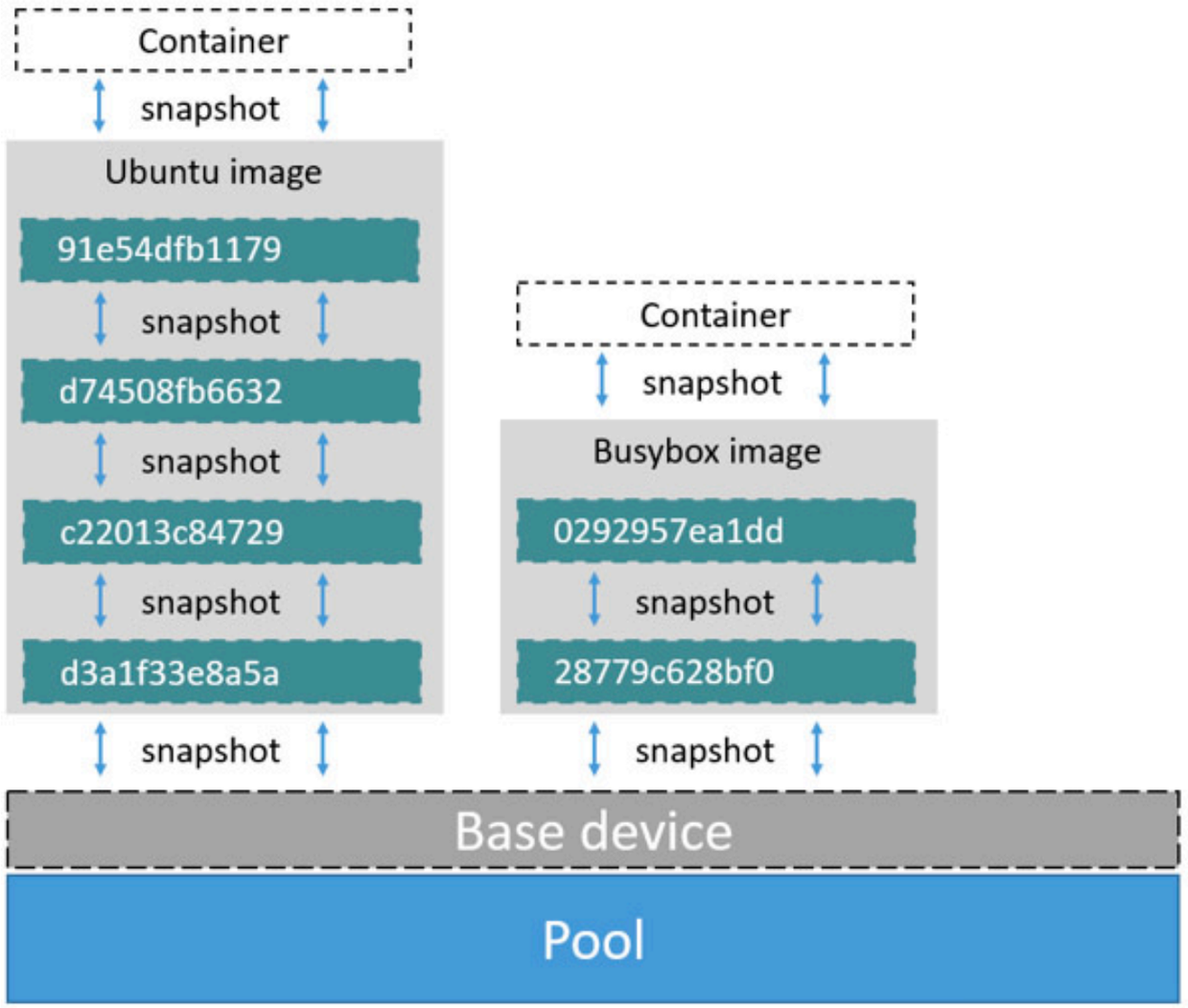
```
$ ls -l /var/lib/docker/overlay/

total 20
drwx------ 3 root root 4096 Jun 20 16:11 38f3ed2eac129654acef11c32670b534670c3a06e483fce313d72e3e0a15baa8
drwx------ 3 root root 4096 Jun 20 16:11 55f1e14c361b90570df46371b20ce6d480c434981cbda5fd68c6ff61aa0a5358
drwx------ 3 root root 4096 Jun 20 16:11 824c8a961a4f5e8fe4f4243dab57c5be798e7fd195f6d88ab06aea92ba931654
drwx------ 3 root root 4096 Jun 20 16:11 ad0fe55125ebf599da124da175174a4b8c1878afe6907bf7c78570341f308461
drwx------ 3 root root 4096 Jun 20 16:11 edab9b5e5bf73f2997524eebeac1de4cf9c8b904fa8ad3ec43b3504196aa3801
```

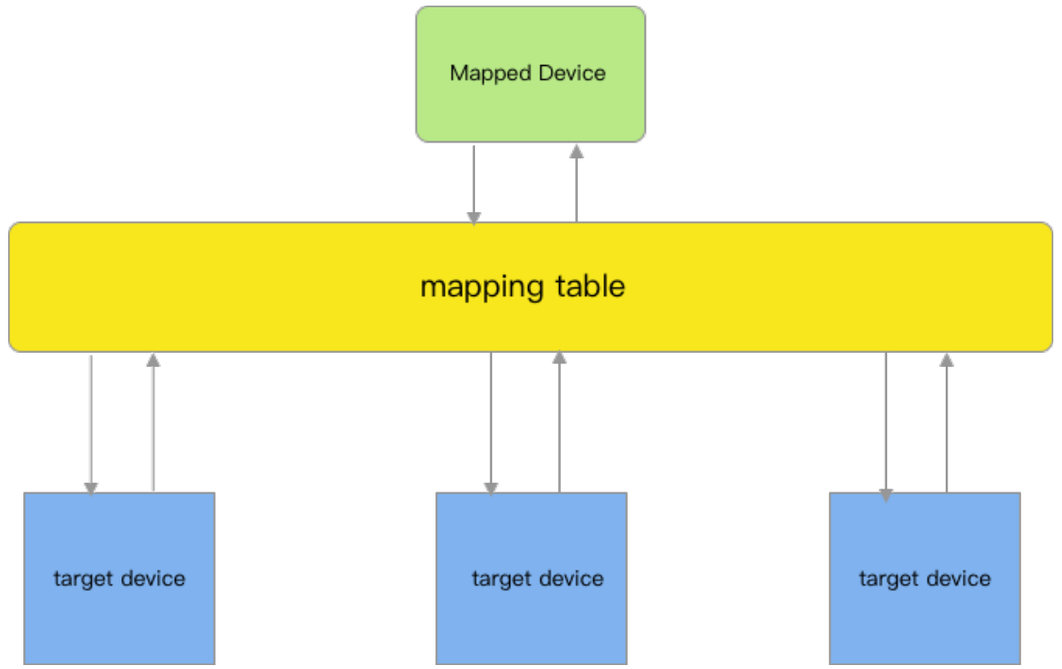
```
$ ls -i /var/lib/docker/overlay/38f3ed2eac129654acef11c32670b534670c3a06e483fce313d72e3e0a15baa8/root/bin/ls
19793696 /var/lib/docker/overlay/38f3ed2eac129654acef11c32670b534670c3a06e483fce313d72e3e0a15baa8/root/bin/ls
$ ls -i /var/lib/docker/overlay/55f1e14c361b90570df46371b20ce6d480c434981cbda5fd68c6ff61aa0a5358/root/bin/ls
19793696 /var/lib/docker/overlay/55f1e14c361b90570df46371b20ce6d480c434981cbda5fd68c6ff61aa0a5358/root/bin/ls
```



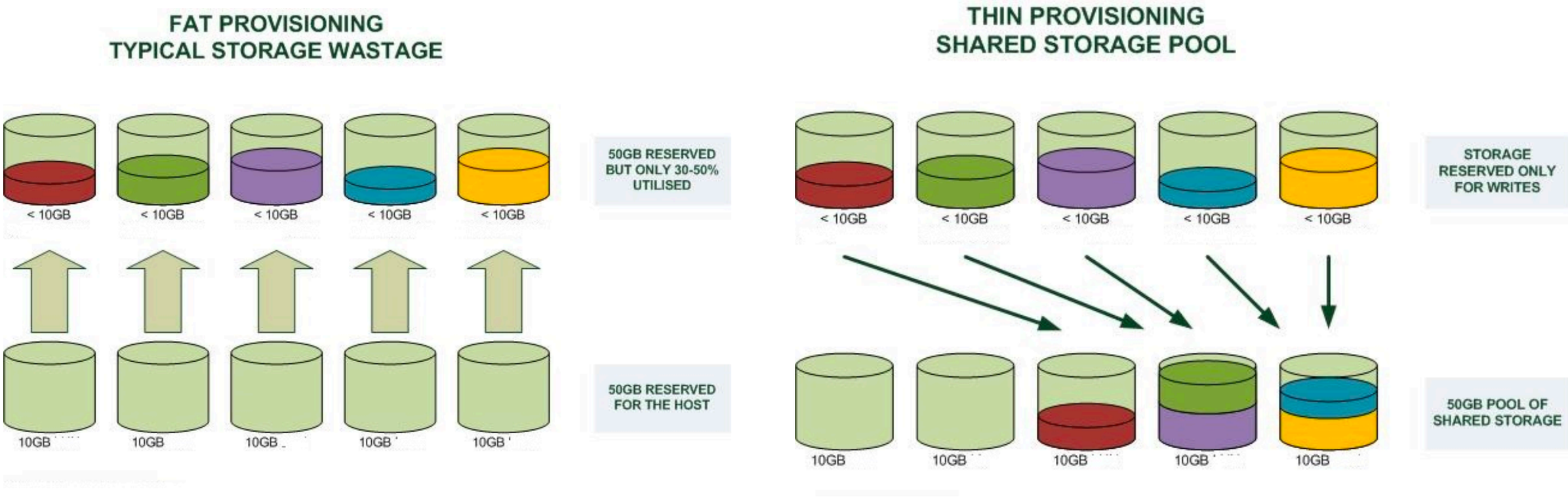
devicemapper



- 块设备存储
- Device Mapper
- Fat Provisioning & Thin Provisioning
- loop-lvm 和 direct-lvm
- 不支持共享存储 & 磁盘溢出



Fat Provisioning & Thin Provisioning



Q & A