[**理解Docker（4）：Docker 容器使用 cgroups 限制资源使用**](http://www.cnblogs.com/sammyliu/p/5886833.html)

本系列文章将介绍Docker的有关知识：

（1）[Docker 安装及基本用法](http://www.cnblogs.com/sammyliu/p/5875470.html)

（2）[Docker 镜像](http://www.cnblogs.com/sammyliu/p/5877964.html%20)

（3）[Docker 容器的隔离性 - 使用 Linux namespace 隔离容器的运行环境](http://www.cnblogs.com/sammyliu/p/5878973.html%20)

（4）[Docker 容器的隔离性 - 使用 cgroups 限制容器使用的资源](http://www.cnblogs.com/sammyliu/p/5886833.html%20)

（5）[Docker 网络](http://www.cnblogs.com/sammyliu/p/5894191.html%20)

    上一篇文章将到 Docker 容器使用 linux namespace 来隔离其运行环境，使得容器中的进程看起来就像爱一个独立环境中运行一样。但是，光有运行环境隔离还不够，因为这些进程还是可以不受限制地使用系统资源，比如网络、磁盘、CPU以及内存 等。关于其目的，一方面，是为了防止它占用了太多的资源而影响到其它进程；另一方面，在系统资源耗尽的时候，linux 内核会触发 OOM，这会让一些被杀掉的进程成了无辜的替死鬼。因此，为了让容器中的进程更加可控，Docker 使用 Linux cgroups 来限制容器中的进程允许使用的系统资源。

**1. 基础知识：Linux control groups**

**1.1 概念**

  Linux Cgroup 可​​​让​​​您​​​为​​​系​​​统​​​中​​​所​​​运​​​行​​​任​​​务​​​（进​​​程​​​）的​​​用​​​户​​​定​​​义​​​组​​​群​​​分​​​配​​​资​​​源​​​ — 比​​​如​​​ CPU 时​​​间​​​、​​​系​​​统​​​内​​​存​​​、​​​网​​​络​​​带​​​宽​​​或​​​者​​​这​​​些​​​资​​​源​​​的​​​组​​​合​​​。​​​您​​​可​​​以​​​监​​​控​​​您​​​配​​​置​​​的​​​ cgroup，拒​​​绝​​​ cgroup 访​​​问​​​某​​​些​​​资​​​源​​​，甚​​​至​​​在​​​运​​​行​​​的​​​系​​​统​​​中​​​动​​​态​​​配​​​置​​​您​​​的​​​ cgroup。所以，可以将 controll groups 理解为 controller （system resource） （for） （process）groups，也就是是说它以一组进程为目标进行系统资源分配和控制。

  它主要提供了如下功能：

* **Resource limitation**: 限制资源使用，比如内存使用上限以及文件系统的缓存限制。
* **Prioritization**: 优先级控制，比如：CPU利用和磁盘IO吞吐。
* **Accounting**: 一些审计或一些统计，主要目的是为了计费。
* **Control**: 挂起进程，恢复执行进程。

使​​​用​​​ cgroup，系​​​统​​​管​​​理​​​员​​​可​​​更​​​具​​​体​​​地​​​控​​​制​​​对​​​系​​​统​​​资​​​源​​​的​​​分​​​配​​​、​​​优​​​先​​​顺​​​序​​​、​​​拒​​​绝​​​、​​​管​​​理​​​和​​​监​​​控​​​。​​​可​​​更​​​好​​​地​​​根​​​据​​​任​​​务​​​和​​​用​​​户​​​分​​​配​​​硬​​​件​​​资​​​源​​​，提​​​高​​​总​​​体​​​效​​​率​​​。

在实践中，系统管理员一般会利用CGroup做下面这些事（有点像为某个虚拟机分配资源似的）：

* 隔离一个进程集合（比如：nginx的所有进程），并限制他们所消费的资源，比如绑定CPU的核。
* 为这组进程分配其足够使用的内存
* 为这组进程分配相应的网络带宽和磁盘存储限制
* 限制访问某些设备（通过设置设备的白名单）

Linux 系统中，一切皆文件。Linux 也将 cgroups 实现成了文件系统，方便用户使用。在我的 Ubuntu 14.04 测试环境中：

[复制代码](javascript:void(0);)

root@devstack:/home/sammy# mount -t cgroup

cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,relatime,cpuset)

cgroup on /sys/fs/cgroup/cpu type cgroup (rw,relatime,cpu)

systemd on /sys/fs/cgroup/systemd type cgroup (rw,noexec,nosuid,nodev,none,name=systemd)

root@devstack:/home/sammy# lssubsys -m

cpuset /sys/fs/cgroup/cpuset

cpu /sys/fs/cgroup/cpu

cpuacct /sys/fs/cgroup/cpuacct

memory /sys/fs/cgroup/memory

devices /sys/fs/cgroup/devices

freezer /sys/fs/cgroup/freezer

blkio /sys/fs/cgroup/blkio

perf\_event /sys/fs/cgroup/perf\_event

hugetlb /sys/fs/cgroup/hugetlb

root@devstack:/home/sammy# ls /sys/fs/cgroup/ -l

total 0

drwxr-xr-x 3 root root 0 Sep 18 21:46 blkio

drwxr-xr-x 3 root root 0 Sep 18 21:46 cpu

drwxr-xr-x 3 root root 0 Sep 18 21:46 cpuacct

drwxr-xr-x 3 root root 0 Sep 18 21:46 cpuset

drwxr-xr-x 3 root root 0 Sep 18 21:46 devices

drwxr-xr-x 3 root root 0 Sep 18 21:46 freezer

drwxr-xr-x 3 root root 0 Sep 18 21:46 hugetlb

drwxr-xr-x 3 root root 0 Sep 18 21:46 memory

drwxr-xr-x 3 root root 0 Sep 18 21:46 perf\_event

drwxr-xr-x 3 root root 0 Sep 18 21:46 systemd

[复制代码](javascript:void(0);)

我们看到 /sys/fs/cgroup 目录中有若干个子目录，我们可以认为这些都是受 cgroups 控制的资源以及这些资源的信息。

* blkio — 这​​​个​​​子​​​系​​​统​​​为​​​块​​​设​​​备​​​设​​​定​​​输​​​入​​​/输​​​出​​​限​​​制​​​，比​​​如​​​物​​​理​​​设​​​备​​​（磁​​​盘​​​，固​​​态​​​硬​​​盘​​​，USB 等​​​等​​​）。
* cpu — 这​​​个​​​子​​​系​​​统​​​使​​​用​​​调​​​度​​​程​​​序​​​提​​​供​​​对​​​ CPU 的​​​ cgroup 任​​​务​​​访​​​问​​​。​​​
* cpuacct — 这​​​个​​​子​​​系​​​统​​​自​​​动​​​生​​​成​​​ cgroup 中​​​任​​​务​​​所​​​使​​​用​​​的​​​ CPU 报​​​告​​​。​​​
* cpuset — 这​​​个​​​子​​​系​​​统​​​为​​​ cgroup 中​​​的​​​任​​​务​​​分​​​配​​​独​​​立​​​ CPU（在​​​多​​​核​​​系​​​统​​​）和​​​内​​​存​​​节​​​点​​​。​​​
* devices — 这​​​个​​​子​​​系​​​统​​​可​​​允​​​许​​​或​​​者​​​拒​​​绝​​​ cgroup 中​​​的​​​任​​​务​​​访​​​问​​​设​​​备​​​。​​​
* freezer — 这​​​个​​​子​​​系​​​统​​​挂​​​起​​​或​​​者​​​恢​​​复​​​ cgroup 中​​​的​​​任​​​务​​​。​​​
* memory — 这​​​个​​​子​​​系​​​统​​​设​​​定​​​ cgroup 中​​​任​​​务​​​使​​​用​​​的​​​内​​​存​​​限​​​制​​​，并​​​自​​​动​​​生​​​成​​​​​内​​​存​​​资​​​源使用​​​报​​​告​​​。​​​
* net\_cls — 这​​​个​​​子​​​系​​​统​​​使​​​用​​​等​​​级​​​识​​​别​​​符​​​（classid）标​​​记​​​网​​​络​​​数​​​据​​​包​​​，可​​​允​​​许​​​ Linux 流​​​量​​​控​​​制​​​程​​​序​​​（tc）识​​​别​​​从​​​具​​​体​​​ cgroup 中​​​生​​​成​​​的​​​数​​​据​​​包​​​。​​​
* net\_prio — 这个子系统用来设计网络流量的优先级
* hugetlb — 这个子系统主要针对于HugeTLB系统进行限制，这是一个大页文件系统。

默认的话，在 Ubuntu 系统中，你可能看不到 net\_cls 和 net\_prio 目录，它们需要你手工做 mount：

[复制代码](javascript:void(0);)

root@devstack:/sys/fs/cgroup# modprobe cls\_cgroup

root@devstack:/sys/fs/cgroup# mkdir net\_cls

root@devstack:/sys/fs/cgroup# mount -t cgroup -o net\_cls none net\_cls

root@devstack:/sys/fs/cgroup# modprobe netprio\_cgroup

root@devstack:/sys/fs/cgroup# mkdir net\_prio

root@devstack:/sys/fs/cgroup# mount -t cgroup -o net\_prio none net\_prio

root@devstack:/sys/fs/cgroup# ls net\_prio/cgroup.clone\_children cgroup.procs net\_prio.ifpriomap notify\_on\_release tasks

cgroup.event\_control cgroup.sane\_behavior net\_prio.prioidx release\_agent

root@devstack:/sys/fs/cgroup# ls net\_cls/

cgroup.clone\_children cgroup.event\_control cgroup.procs cgroup.sane\_behavior net\_cls.classid notify\_on\_release release\_agent tasks

[复制代码](javascript:void(0);)

**1.2 实验**

**1.2.1 通过 cgroups 限制进程的 CPU**

写一段最简单的 C 程序：

int main(void)

{

int i = 0;

for(;;) i++;

return 0;

}

编译，运行，发现它占用的 CPU 几乎到了 100%：

top - 22:43:02 up 1:14, 3 users, load average: 0.24, 0.06, 0.06 PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

2304 root 20 0 4188 356 276 R 99.6 0.0 0:11.77 hello

接下来我们做如下操作：

[复制代码](javascript:void(0);)

root@devstack:/home/sammy/c# mkdir /sys/fs/cgroup/cpu/hello

root@devstack:/home/sammy/c# cd /sys/fs/cgroup/cpu/hello

root@devstack:/sys/fs/cgroup/cpu/hello# ls

cgroup.clone\_children cgroup.procs cpu.cfs\_quota\_us cpu.stat tasks

cgroup.event\_control cpu.cfs\_period\_us cpu.shares notify\_on\_release

root@devstack:/sys/fs/cgroup/cpu/hello# cat cpu.cfs\_quota\_us

-1

root@devstack:/sys/fs/cgroup/cpu/hello# echo 20000 > cpu.cfs\_quota\_us

root@devstack:/sys/fs/cgroup/cpu/hello# cat cpu.cfs\_quota\_us

20000

root@devstack:/sys/fs/cgroup/cpu/hello# echo 2428 > tasks

[复制代码](javascript:void(0);)

然后再来看看这个进程的 CPU 占用情况：

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

2428 root 20 0 4188 356 276 R 19.9 0.0 0:46.03 hello

它占用的 CPU 几乎就是 20%，也就是我们预设的阈值。这说明我们通过上面的步骤，成功地将这个进程运行所占用的 CPU 资源限制在某个阈值之内了。

如果此时再启动另一个 hello 进程并将其 id 加入 tasks 文件，则两个进程会共享设定的 CPU 限制：

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

2428 root 20 0 4188 356 276 R 10.0 0.0 285:39.54 hello

12526 root 20 0 4188 356 276 R 10.0 0.0 0:25.09 hello

**1.2.2 通过 cgroups 限制进程的 Memory**

同样地，我们针对它占用的内存做如下操作：

[复制代码](javascript:void(0);)

root@devstack:/sys/fs/cgroup/memory# mkdir hello

root@devstack:/sys/fs/cgroup/memory# cd hello/

root@devstack:/sys/fs/cgroup/memory/hello# cat memory.limit\_in\_bytes

18446744073709551615

root@devstack:/sys/fs/cgroup/memory/hello# echo 64k > memory.limit\_in\_bytes

root@devstack:/sys/fs/cgroup/memory/hello# echo 2428 > tasks

root@devstack:/sys/fs/cgroup/memory/hello#

[复制代码](javascript:void(0);)

上面的步骤会把进程 2428 说占用的内存阈值设置为 64K。超过的话，它会被杀掉。

**1.2.3 限制进程的 I/O**

运行命令：

sudo dd if=/dev/sda1 of=/dev/null

通过 iotop 命令看 IO （此时磁盘在快速转动），此时其写速度为 242M/s：

TID PRIO USER DISK READ DISK WRITE SWAPIN IO> COMMAND

2555 be/4 root 242.60 M/s 0.00 B/s 0.00 % 61.66 % dd if=/dev/sda1 of=/dev/null

接着做下面的操作：

root@devstack:/home/sammy# mkdir /sys/fs/cgroup/blkio/io

root@devstack:/home/sammy# cd /sys/fs/cgroup/blkio/io

root@devstack:/sys/fs/cgroup/blkio/io# ls -l /dev/sda1

brw-rw---- 1 root disk 8, 1 Sep 18 21:46 /dev/sda1

root@devstack:/sys/fs/cgroup/blkio/io# echo '8:0 1048576' > /sys/fs/cgroup/blkio/io/blkio.throttle.read\_bps\_device

root@devstack:/sys/fs/cgroup/blkio/io# echo 2725 > /sys/fs/cgroup/blkio/io/tasks

结果，这个进程的IO 速度就被限制在 1Mb/s 之内了：

TID PRIO USER DISK READ DISK WRITE SWAPIN IO> COMMAND

2555 be/4 root 990.44 K/s 0.00 B/s 0.00 % 96.29 % dd if=/dev/sda1 of=/dev/null

**1.3 术语**

cgroups 的术语包括：

* **任务（Tasks）**：就是系统的一个进程。
* **控制组（Control Group）**：一组按照某种标准划分的进程，比如官方文档中的Professor和Student，或是WWW和System之类的，其表示了某进程组。Cgroups中的资源控制都是以控制组为单位实现。一个进程可以加入到某个控制组。而资源的限制是定义在这个组上，就像上面示例中我用的 hello 一样。简单点说，cgroup的呈现就是一个目录带一系列的可配置文件。
* **层级（Hierarchy）**：控制组可以组织成hierarchical的形式，既一颗控制组的树（目录结构）。控制组树上的子节点继承父结点的属性。简单点说，hierarchy就是在一个或多个子系统上的cgroups目录树。
* **子系统（Subsystem）**：一个子系统就是一个资源控制器，比如CPU子系统就是控制CPU时间分配的一个控制器。子系统必须附加到一个层级上才能起作用，一个子系统附加到某个层级以后，这个层级上的所有控制族群都受到这个子系统的控制。Cgroup的子系统可以有很多，也在不断增加中。

**2. Docker 对 cgroups 的使用**

**2.1 默认情况**

默认情况下，Docker 启动一个容器后，会在 /sys/fs/cgroup 目录下的各个资源目录下生成以容器 ID 为名字的目录（group），比如：

/sys/fs/cgroup/cpu/docker/03dd196f415276375f754d51ce29b418b170bd92d88c5e420d6901c32f93dc14

此时 cpu.cfs\_quota\_us 的内容为 -1，表示默认情况下并没有限制容器的 CPU 使用。在容器被 stopped 后，该目录被删除。

运行命令 docker run -d --name web41 --cpu-quota 25000 --cpu-period 100 --cpu-shares 30 training/webapp python app.py 启动一个新的容器，结果：

root@devstack:/sys/fs/cgroup/cpu/docker/06bd180cd340f8288c18e8f0e01ade66d066058dd053ef46161eb682ab69ec24# cat cpu.cfs\_quota\_us

25000

root@devstack:/sys/fs/cgroup/cpu/docker/06bd180cd340f8288c18e8f0e01ade66d066058dd053ef46161eb682ab69ec24# cat tasks

3704

root@devstack:/sys/fs/cgroup/cpu/docker/06bd180cd340f8288c18e8f0e01ade66d066058dd053ef46161eb682ab69ec24# cat cpu.cfs\_period\_us

2000

Docker 会将容器中的进程的 ID 加入到各个资源对应的 tasks 文件中。表示 Docker 也是以上面的机制来使用 cgroups 对容器的 CPU 使用进行限制。

相似地，可以通过 docker run 中 mem 相关的参数对容器的内存使用进行限制：

--cpuset-mems string MEMs in which to allow execution (0-3, 0,1)

--kernel-memory string Kernel memory limit

-m, --memory string Memory limit

--memory-reservation string Memory soft limit

--memory-swap string Swap limit equal to memory plus swap: '-1' to enable unlimited swap

--memory-swappiness int Tune container memory swappiness (0 to 100) (default -1)

比如  docker run -d --name web42 --blkio-weight 100 --memory 10M --cpu-quota 25000 --cpu-period 2000 --cpu-shares 30 training/webapp python app.py：

root@devstack:/sys/fs/cgroup/memory/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410# cat memory.limit\_in\_bytes

10485760

  root@devstack:/sys/fs/cgroup/blkio/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410# cat blkio.weight  
  100

目前 docker 已经几乎支持了所有的 cgroups 资源，可以限制容器对包括 network，device，cpu 和 memory 在内的资源的使用，比如：

[复制代码](javascript:void(0);)

root@devstack:/sys/fs/cgroup# find -iname ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./net\_prio/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./net\_cls/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./systemd/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./hugetlb/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./perf\_event/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./blkio/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./freezer/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./devices/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./memory/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./cpuacct/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./cpu/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

./cpuset/docker/ec8d850ebbabaf24df572cb5acd89a6e7a953fe5aa5d3c6a69c4532f92b57410

[复制代码](javascript:void(0);)

**2.2 net\_cls**

   net\_cls 和 tc 一起使用可用于限制进程发出的网络包所使用的网络带宽。当使用 cgroups network controll net\_cls 后，指定进程发出的所有网络包都会被加一个 tag，然后就可以使用其他工具比如 iptables 或者 traffic controller （TC）来根据网络包上的 tag 进行流量控制。关于 TC 的文档，网上很多，这里不再赘述，只是用一个简单的例子来加以说明。

  关于 classid，它的格式是 0xAAAABBBB，其中，AAAA 是十六进制的主ID（major number），BBBB 是十六进制的次ID（minor number）。因此，0X10001 表示 10：1，而 0x00010001 表示 1：！。

  （1）首先在host 的网卡 eth0 上做如下设置：

tc qdisc del dev eth0 root                                        #删除已有的规则  
tc qdisc add dev eth0 root handle 10: htb default 12                
tc class add dev eth0 parent 10: classid 10:1 htb rate 1500kbit ceil 1500kbit burst 10k         #限速  
tc filter add dev eth0 protocol ip parent 10:0 prio 1 u32 match ip protocol 1 0xff flowid 10:1  #只处理 ping 参数的网络包

其结果是：

* 在网卡 eth0 上创建了一个 HTB root 队列，hangle 10： 表示队列句柄也就是major number 为 10
* 创建一个分类 10：1，限制它的出发网络带宽为 80 kbit （千比特每秒）
* 创建一个分类器，将 eth0 上 IP IMCP 协议 的 major ID 为 10 的 prio 为 1 的网络流量都分类到 10：1 类别

（2）启动容器

容器启动后，其 init 进程在host 上的 PID 就被加入到 tasks 文件中了：

root@devstack:/sys/fs/cgroup/net\_cls/docker/ff8d9715b7e11a5a69446ff1e3fde3770078e32a7d8f7c1cb35d51c75768fe33# ps -ef | grep 10047

231072 10047 10013 1 07:08 ? 00:00:00 python app.py

设置 net\_cls classid：

echo 0x100001 > net\_cls.classid

再在容器启动一个 ping 进程，其 ID 也被加入到 tasks 文件中了。

（3）查看tc 情况： tc -s -d class show dev eth0

[复制代码](javascript:void(0);)

Every 2.0s: tc -s class ls dev eth0 Wed Sep 21 04:07:56 2016

class htb 10:1 root prio 0 rate 1500Kbit ceil 1500Kbit burst 10Kb cburst 1599b  
Sent 17836 bytes 182 pkt (dropped 0, overlimits 0 requeues 0)  
rate 0bit 0pps backlog 0b 0p requeues 0  
lended: 182 borrowed: 0 giants: 0  
tokens: 845161 ctokens: 125161

[复制代码](javascript:void(0);)

我们可以看到 tc 已经在处理 ping 进程产生的数据包了。再来看一下 net\_cls 和 ts 合作的限速效果：

10488 bytes from 192.168.1.1: icmp\_seq=35 ttl=63 time=12.7 ms

10488 bytes from 192.168.1.1: icmp\_seq=36 ttl=63 time=15.2 ms

10488 bytes from 192.168.1.1: icmp\_seq=37 ttl=63 time=4805 ms

10488 bytes from 192.168.1.1: icmp\_seq=38 ttl=63 time=9543 ms

其中：

* 后两条说使用的 tc class 规则是 tc class add dev eth0 parent 10: classid 10:1 htb rate 1500kbit ceil 15kbit burst 10k
* 前两条所使用的 tc class 规则是 tc class add dev eth0 parent 10: classid 10:1 htb rate 1500kbit ceil 10Mbit burst 10k

**3. Docker run 命令中 cgroups 相关命令**

[复制代码](javascript:void(0);)

**block IO:**

--blkio-weight value Block IO (relative weight), between 10 and 1000

--blkio-weight-device value Block IO weight (relative device weight) (default [])

--cgroup-parent string Optional parent cgroup for the container

**CPU:**

--cpu-percent int CPU percent (Windows only)

--cpu-period int Limit CPU CFS (Completely Fair Scheduler) period

--cpu-quota int Limit CPU CFS (Completely Fair Scheduler) quota

-c, --cpu-shares int CPU shares (relative weight)

--cpuset-cpus string CPUs in which to allow execution (0-3, 0,1)

--cpuset-mems string MEMs in which to allow execution (0-3, 0,1)

**Device:**

--device value Add a host device to the container (default [])

--device-read-bps value Limit read rate (bytes per second) from a device (default [])

--device-read-iops value Limit read rate (IO per second) from a device (default [])

--device-write-bps value Limit write rate (bytes per second) to a device (default [])

--device-write-iops value Limit write rate (IO per second) to a device (default [])

**Memory:**

--kernel-memory string Kernel memory limit

-m, --memory string Memory limit

--memory-reservation string Memory soft limit

--memory-swap string Swap limit equal to memory plus swap: '-1' to enable unlimited swap

--memory-swappiness int Tune container memory swappiness (0 to 100) (default -1)

[复制代码](javascript:void(0);)

一些说明：

1. cgroup 只能限制 CPU 的使用，而不能保证CPU的使用。也就是说， 使用 cpuset-cpus，可以让容器在指定的CPU或者核上运行，但是不能确保它独占这些CPU；cpu-shares 是个相对值，只有在CPU不够用的时候才其作用。也就是说，当CPU够用的时候，每个容器会分到足够的CPU；不够用的时候，会按照指定的比重在多个容器之间分配CPU。

2. 对内存来说，cgroups 可以限制容器最多使用的内存。使用 -m 参数可以设置最多可以使用的内存。