资源限制介绍

默认情况下，容器没有资源限制，可以使用主机内核调度程序允许的尽可能多的给定资源，docker提供了控制容器可以限制容器使用多少内存或者cpu的方法，设置docker run命令的运行时配置标志。

其中一些功能要求宿主机的内核支持Linux功能，要检查支持，可以使用docker info命令，如果内核中禁用了某项功能，可能会在输出结尾处看到警告。

对于Linux主机，如果没有足够的内容来执行其他重要的系统任务，将会抛出OOM异常（内存溢出、内存泄漏、内存异常），随后系统会开始杀死进程以释放内存，凡是运行在宿主机的进程都有可能被kill，包括dockerd和其他的应用程序，如果重要的系统进程被kill，会导致和该进程相关的服务全部宕机。

产生OOM异常时，Dockerd尝试通过调整docker守护程序上的OOM优先级来减轻这些风险，以便它比系统上的其他进程更不可能被杀死，但是容器的OOM优先级未调整时单个容器被杀死的可能性更大（不推荐调整容器的优先级这种方式）。

Linux会每个进程算一个分数，最终他会将分数最高的进程kill掉

```
/proc/PID/oom_score_adj #范围为-1000到1000，值越高越容易被宿主机kill掉，如果将该值设置为-1000，则进程永远不会被宿主机kernel kill。
/proc/PID/oom_adj #范围为-17到+15，取值越高越容易被干掉，如果是-17,则表示不能被kill，该设置参数的存在是为了和旧版本的Linux内核兼容。
/proc/PID/oom_score #这个值是系统综合进程的内存消耗量、CPU时间(utime+ stime)、存活时间(uptime - start time)和oom_adj计算出的进程得分，消耗内存越多得分越高，越容易被宿主机kernel强制杀死。
```

容器的内存限制

Docker可以强制执行**硬性内存限制**，即只允许容器使用给定的内存大小

Docker也可以执行非**硬性内存限制**，即容器可以使用尽可能多的内存，除非内核检测到主机上的内存不够用了

内存限制参数

- -m or --memory：容器可以使用的最大内存量，如果设置此选项，则允许的最小值为4m
- --memory-swap：容器可以使用的交换分区和物理内存大小总和，必须要在设置了物理内存限制的前提才能设置交换分区的限制，经常将内存交换到磁盘的应用程序会降低性能。如果该参数设置未-1，则容器可以使用主机上swap的最大空间
- --memory-swappiness：设置容器使用交换分区的倾向性，值越高表示越倾向于使用swap分区，范围为0-100，0为能不用就不用，100为能用就用。
- --kernel-memory：容器可以使用的最大内核内存量，最小为4m，由于内核内存于用户空间内存隔离，因此无法于用户空间内存直接交换，因此内核内存不足的容器可能会阻塞宿主机主机资源，这会对主机和其他容器或者其他服务进程产生影响，因此不要设置内核内存大小
- --memory-reservation：允许指定小于--memory的软限制当Docker检测到主机上的争用或内存不足时会激活该限制，如果使用--memory-reservation，则必须将其设置为低于--memory才能使其优先。因为它是软限制，所以不能保证容器不超过限制。
- --oom-kill-disable：默认情况下，发生OOM时kernel会杀死容器内进程，但是可以使用该参数可以禁止oom发生在指定的容器上，仅在已设置-m选项的容器上禁用oom，如果-m参数未配置，产生oom时主机为了释放内存还会杀死进程

案例

如果一个容器未作内存使用限制，则该容器可以利用到系统内存最大空间，默认创建的容器没有做内存资源限制

- 拉取容器压测工具镜像

```
[root@docker-server1 ~]# docker pull lorel/docker-stress-ng
[root@docker-server1 ~]# docker run -it --rm lorel/docker-stress-ng -help
```

- 使用压测工具开启两个工作进程，每个工作进程最大允许使用内存256M，且宿主机不限制当前容器的最大内存

```
[root@docker-server1 ~]# docker run -it --rm --name test1 lorel/docker-stress-ng --vm 2 --vm-bytes 256m
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 2 vm

[root@docker-server1 ~]# docker stats
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O     BLOCK I/O   PIDS
3ca32774fc20   test1     185.16%   514.3MiB / 1.781GiB   28.21%    648B / 0B   0B / 0B     5

```

- 宿主机限制最大内存使用

```
[root@docker-server1 ~]# docker run -it --rm -m 256m --name test2 lorel/docker-stress-ng --vm 2 --vm-bytes 256m
[root@docker-server1 ~]# docker stats
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O     BLOCK I/O         PIDS
bfff488e6185   test1     169.76%   255.8MiB / 256MiB   99.91%    648B / 0B   3.53GB / 10.6GB   5
```

- 可以通过修改cgroup文件值来扩大内存限制，缩小会报错

```
[root@docker-server1 ~]# cat /sys/fs/cgroup/memory/docker/bfff488e618580b227b5411c91b35517850e95af2ac2225b45180937c14e70c2/memory.limit_in_bytes
```

内存软限制

软限制不会真正限制到内存的使用

```
[root@docker-server1 ~]# docker run -it --rm -m 256m --memory-reservation 128m --name test1 lorel/docker-stress-ng --vm 2 --vm-bytes 256m
[root@docker-server1 ~]# docker stats
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O     BLOCK I/O         PIDS
0ffb4b8fdbde   test1     174.52%   255.9MiB / 256MiB   99.95%    648B / 0B   5.33GB / 18.1GB   5
```

交换分区限制

```
[root@docker-server1 ~]# docker run -it --rm -m 256m --memory-swap 512m --name test1 lorel/docker-stress-ng --vm 2 --vm-bytes 256m
```

容器的CPU限制

一个宿主机，有几十个核心的cpu，但是宿主机上可以同时运行成百上千个不同的进程用以处理不同的任务，多进程共用一个cpu的核心依赖计数就是为可压缩资源，即一个核心cpu可以通过调度而运行多个进程，但是在同一个单位时间内只能由一个进程在cpu上运行，那么这么多的进程怎么在cpu上执行和调度的呢？（进程优先级）

默认情况下，每个容器对主机cpu周期的访问权限是不受限制的，但是我们可以人为干扰

参数

- --cpus： 指定容器可以使用多少可用cpu资源，例如，如果主机有两个cpu，并且设置了--cpu=1.5，那么该容器将保证最多可以访问1.5个的cpu（如果是4核cpu，那么还可以是4核心上的每核用一点，但是总计是1.5核心的cpu）
- --cpu-period：设置cpu的调度周期，必须于--cpu-quota一起使用
- --cpu-quota：在容器上添加cpu配额，计算方式为cpu-quota/cpu-period的结果值（现在通常使用--cpus）
- --cpuset-cpus：用于指定容器运行的cpu编号，也就是所谓的绑核
- --cpuset-mem：设置使用哪个cpu的内存，仅对非统一内存访问（NUMA）架构有效。
- --cpu-shares：值越高的容器将会得到更多的时间片（宿主机多喝cpu总数为100%，加入容器A为1024，容器B为2048，那么容器B将最大时容器A的可以CPU的两倍），默认的时间片时2014，最大为262144

案例

未限制容器cpu

- 启动1个进程，占用8核cpu，未限制容器会把cpu全部占完

```
[root@docker-server1 ~]# docker run -it --rm --name test1 lorel/docker-stress-ng --vm 1 --cpu 8

[root@docker-server1 ~]# docker stats
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O     BLOCK I/O         PIDS
7eb5882b9379   test1     812.96%   1.387GiB / 1.781GiB   77.88%    648B / 0B   4.02GB / 412MB    25

```

限制容器cpu

```
[root@docker-server1 ~]# docker run -it --rm  --cpus 4 --name test1 lorel/docker-stress-ng --vm 1 --cpu 8

[root@docker-server1 ~]# docker stats
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O     BLOCK I/O         PIDS
0a1c3805e5c9   test1     398.99%   1.414GiB / 1.781GiB   79.40%    648B / 0B   1.14GB / 127MB    25

[root@docker-server1 ~]# top
```

```
top - 21:19:53 up  2:33,  2 users,  load average: 12.50, 9.48, 4.62
Tasks: 175 total,  17 running, 158 sleeping,   0 stopped,   0 zombie
%Cpu0  : 52.1 us,  0.3 sy,  0.0 ni, 45.5 id,  1.7 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  : 49.8 us,  1.4 sy,  0.0 ni, 15.2 id, 32.5 wa,  0.0 hi,  1.0 si,  0.0 st
%Cpu2  : 48.6 us,  1.7 sy,  0.0 ni,  6.2 id, 41.4 wa,  0.0 hi,  2.1 si,  0.0 st
%Cpu3  : 49.1 us,  2.1 sy,  0.0 ni,  8.0 id, 39.4 wa,  0.0 hi,  1.4 si,  0.0 st
%Cpu4  : 51.6 us,  0.7 sy,  0.0 ni, 46.0 id,  1.4 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu5  : 48.4 us,  2.4 sy,  0.0 ni,  1.4 id, 46.3 wa,  0.0 hi,  1.4 si,  0.0 st
%Cpu6  : 50.2 us,  1.0 sy,  0.0 ni, 23.9 id, 24.2 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu7  : 45.3 us,  5.9 sy,  0.0 ni, 21.1 id, 26.6 wa,  0.0 hi,  1.0 si,  0.0 st
```

- 将容器运行到指定的cpu上

```
[root@docker-server1 ~]# docker run -it --rm  --cpus 2 --cpuset-cpus 1,3 --name test1 lorel/docker-stress-ng --vm 1 --cpu 8

[root@docker-server1 ~]# docker stats
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O     BLOCK I/O         PIDS
ee11d834dde5   test1     186.68%   1.488GiB / 1.781GiB   83.60%    648B / 0B   44.8GB / 95.7MB   25

[root@docker-server1 ~]# top
top - 21:27:31 up  2:41,  2 users,  load average: 14.97, 9.00, 5.77
Tasks: 176 total,  19 running, 157 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.3 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu1  : 87.5 us, 10.9 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  1.7 si,  0.0 st
%Cpu2  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 32.9 us, 46.1 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi, 21.1 si,  0.0 st
%Cpu4  :  0.0 us, 17.3 sy,  0.0 ni, 82.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us, 12.7 sy,  0.0 ni, 79.2 id,  0.0 wa,  0.0 hi,  8.2 si,  0.0 st
%Cpu6  :  0.0 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu7  :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

- 基于cpu-shares对cpu进行切分

```
[root@docker-server1 ~]# docker run -it --rm  -d --cpu-shares 1000 --name test1 lorel/docker-stress-ng --vm 1 --cpu 4
[root@docker-server1 ~]# docker run -it --rm  -d --cpu-shares 500 --name test2 lorel/docker-stress-ng --vm 1 --cpu 4

[root@docker-server1 ~]# docker stats
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O     BLOCK I/O       PIDS
d6dd34edb722   test1     543.41%   819.6MiB / 1.781GiB   44.95%    648B / 0B   102MB / 154MB   13
154b07a94e2f   test2     241.15%   711.1MiB / 1.781GiB   39.00%    648B / 0B   406MB / 145MB
```