第九章　docker 容器命名和资源配额控制

本节所讲内容：

9.1　docker 容器命名和重命名

9.2　创建 docker 容器实例时指定主机名

9.3　docker 容器资源配额控制之 cpu

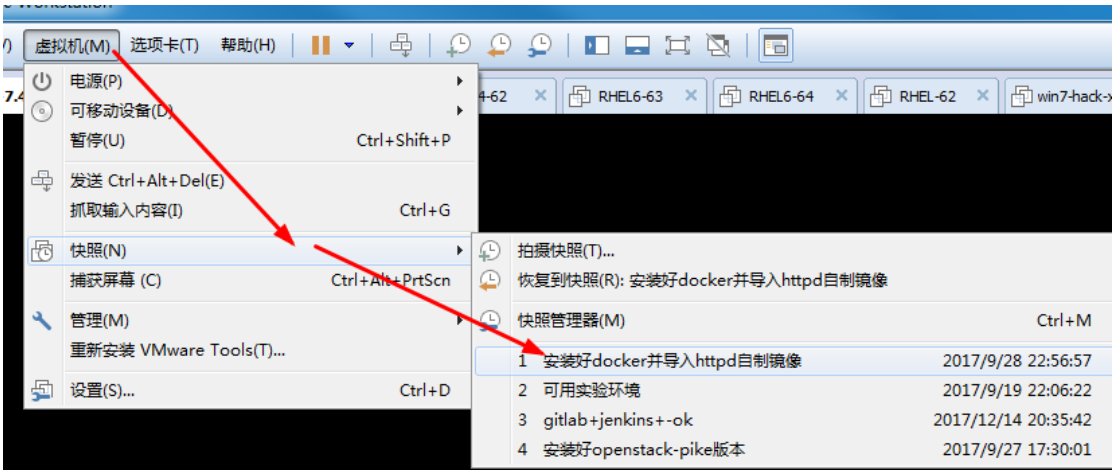
9.4　docker 容器资源配额控制之内存

9.5　docker 容器资源配额控制之 IO

9.6　docker 数据映射

前期准备：

还原快照到已经安装好 docker 的状态：



[root@xuegod63 ~]# systemctl　start docker

[root@xuegod63 ~]# docker　images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
docker.io/centos	httpd	0fe32512008f	23 hours ago	304.5 MB
docker.io/centos	apache	38fa5452c20d	23 hours ago	304.3 MB
docker.io/centos	latest	196e0ce0c9fb	2 weeks ago	196.6 MB

9.1　docker 容器命名和重命名

9.1.1　docker 容器命名和重命名

容器命名语法：docker run -d --name 容器实例名 容器镜像名 要执行的命令

容器重命名语法：　docker rename　旧容器名　新容器名

例 1：运行一个名字为 docker1 的容器

[root@xuegod63 ~]# docker run -itd　--name docker1 docker.io/centos:latest /bin/bash

a651acdb6b4af511ce568a3a24762c56ba868b5adafaae0aa4ab9cd47d578062

[root@xuegod63 ~]# docker ps

```
[root@xuegod63 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
a651acdb6b4a        centos              "/bin/sh"           3 minutes ago       Up 3 minutes                            docker1
```

例 2：将 docke1 容器重命名

[root@xuegod63 ~]# docker rename docker1 docker2

[root@xuegod63 ~]# docker ps

9.2 创建 docker 容器实例时指定主机名

9.2.1 创建 docker 容器实例时指定主机名

语法：docker run -it --name 容器名 -h 指定主机名 镜像 /bin/bash

例 1：

[root@xuegod63 ~]# docker run -it --name docker3 -h docker63.cn centos /bin/sh

sh-4.2# hostname #查看

docker63.cn

9.3 docker 容器资源配额控制之 cpu

9.3.1 docker 容器资源配额控制

启动 docker 容器时，指定 cpu，内存，硬盘性能等的硬件资源使用份额

Docker 通过 cgroup 来控制容器使用的资源配额，包括 CPU、内存、磁盘三大方面，基本覆盖了常见的资源配额和使用量控制。

cgroup 概述：

cgroup 是 Control Groups 的缩写，是 Linux 内核提供的一种可以限制、记录、隔离进程组所使用的物理资源(如 cpu、memory、磁盘 IO 等等) 的机制，被 LXC、docker 等很多项目用于实现进程资源控制。cgroup 将任意进程进行分组化管理的 Linux 内核功能。cgroup 本身是提供将进程进行分组化管理的功能和接口的基础结构，I/O 或内存的分配控制等具体的资源管理功能是通过这个功能来实现的。

为什么要进行硬件配额？ 当多个容器运行时，防止某容器把所有的硬件都占用了。（比如一台被黑的容器）

例 1：给容器实例分配 512 权重的 cpu 使用份额

[root@xuegod63 ~]# docker run --help | grep cpu-shares

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

cpu 配额参数：

-c, --cpu-shares int CPU shares (relative weight) 在创建容器时指定容器所使用的 CPU 份额值。

cpu-shares 的值不能保证可以获得 1 个 vcpu 或者多少 GHz 的 CPU 资源，仅仅只是一个弹性的加权值。

默认情况下，每个 docker 容器的 cpu 份额都是 1024。单独一个容器的份额是没有意义的，只有在同时运行多个容器时，容器的 cpu 加权的效果才能体现出来。

例： 两个容器 A、B 的 cpu 份额分别为 1000 和 500，结果会怎么样？

情况 1：A 和 B 正常运行，在 cpu 进行时间片分配的时候，容器 A 比容器 B 多一倍的机会获得 CPU 的时间片。

情况 2：分配的结果取决于当时主机和其他容器的运行状态，实际上也无法保证容器 A 一定能获得 CPU 时间片。比如容器 A 的进程一直是空闲的，那么容器 B 是可以获取比容器 A 更多的 CPU 时间片的。极端情况下，比如说主机上只运行了一个容器，即使它的 cpu 份额只有 50，它也可以独占整个主机的 cpu 资源。

问：两个容器 A、B 的 cpu 份额分别为 1000 和 500， 1000+500> 1024 是超出了吗？

答：没有。A 使用 1024 的 2/3 ,B 使用 1024 的 1/3 。

cgroups 只在容器分配的资源紧缺时，也就是说在需要对容器使用的资源进行限制时，才会生效。因此，无法单纯根据某个容器的 cpu 份额来确定有多少 cpu 资源分配给它，资源分配结果取决于同时运行的其他容器的 cpu 分配和容器中进程运行情况。

例 1：给容器实例分配 512 权重的 cpu 使用份额

参数： --cpu-shares 512

[root@xuegod63 ~]# docker run -it --cpu-shares 512 centos /bin/bash

[root@df176dd75bd4 /]# cat /sys/fs/cgroup/cpu/cpu.shares #查看结果：

512

注：稍后，我们启动多个容器，测试一下是不是只能使用 512 份额的 cpu 资源。单独一个容器，看不出来

9.3.2 了解 CPU 周期控制

希望开拓一下眼界！

docker 提供了--cpu-period(周期)、--cpu-quota 两个参数控制容器可以分配到的 CPU 时钟周期。

--cpu-period 是用来指定容器对 CPU 的使用要在多长时间内做一次重新分配。 指定周期

--cpu-quota 是用来指定在这个周期内，最多可以有多少时间片断用来跑这个容器。 指定在这个周期中使用多少时间片

跟--cpu-shares 不同的，--cpu-period 和--cpu-quota 是指定一个绝对值，而且没有弹性在里面，容器对 CPU 资源的使用绝对不会超过配置的值。

cpu-period 和 cpu-quota 的单位为微秒（μs）。cpu-period 的最小值为 1000 微秒，最大值为 1 秒（10^6 μs），默认值为 0.1 秒（100000 μs）。cpu-quota 的值默认为-1，表示不做控制。

1 秒=1000 毫秒 1 毫秒=1000 微秒

例 1：如果容器进程需要每 1 秒使用单个 CPU 的 0.2 秒时间，可以将 cpu-period 设置为 1000000（即 1 秒），cpu-quota 设置为 200000（0.2 秒）。

[root@xuegod63 ~]# docker run -it --cpu-period 1000000 --cpu-quota 200000 centos /bin/bash

[root@0363ce23f262 /]# cat /sys/fs/cgroup/cpu/cpu.cfs_period_us #查看

1000000

[root@0363ce23f262 /]# cat /sys/fs/cgroup/cpu/cpu.cfs_quota_us

200000

9.3.3 CPU core 核心控制

参数：--cpuset 可以绑定 CPU

对多核 CPU 的服务器，docker 还可以控制容器运行限定使用哪些 cpu 内核和内存节点，即使用–cpuset-cpus 和–cpuset-mems 参数。对具有 NUMA 拓扑（具有多 CPU、多内存节点）的服务器尤其有用，可以对需要高性能计算的容器进行性能最优的配置。如果服务器只有一个内存节点，则–cpuset-mems 的配置基本上不会有明显效果。

扩展：

服务器架构一般分： SMP、NUMA、MPP 体系结构介绍

从系统架构来看，目前的商用服务器大体可以分为三类：

1、即对称多处理器结构(SMP ： Symmetric Multi-Processor) 例： x86 服务器，双路服务器。主板上有两个物理 cpu

2、非一致存储访问结构 (NUMA ：Non-Uniform Memory Access) 例： IBM 小型机 pSeries 690

3、海量并行处理结构 (MPP ： Massive ParallelProcessing) 。 例： 大型机

做运维有好处： 软，硬 ，网络 运维

学完 linux 后--》学开发（python，php,java）-> 技术主管

9.3.4 taskset 命令

taskset 设定 cpu 亲和力，taskset 能够将一个或多个进程绑定到一个或多个处理器上运行。

参数：

-c, --cpu-list 以列表格式显示和指定 CPU

-p, --pid 在已经存在的 pid 上操作

例 1：设置只在 1 和 2 号 cpu 运行 sshd 进程程序

```
[root@xuegod63 ~]# ps -axu | grep sshd
root       1450  0.0  0.0  82544  3604 ?        Ss   06:52   0:00 /usr/sbin/sshd -D
```

```
[root@xuegod63 ~]#  taskset -cp 1,2 1450
pid 1450's current affinity list: 0-3
pid 1450's new affinity list: 1,2
```

affinity [əˈfɪnəti] 密切关系

例 2：查看 ID 为 1 的进程在哪个 cpu 上运行

```
[root@xuegod63 ~]#  taskset -cp  1
pid 1's current affinity list: 0-3
```

RHEL7 中进程 ID 为 1 的进程是： systemd ； RHEL6 中进程 ID 为 1 的进程是：init

例 3：设置 nginx cpu 亲和力

在 conf/nginx.conf 中，有如下一行：

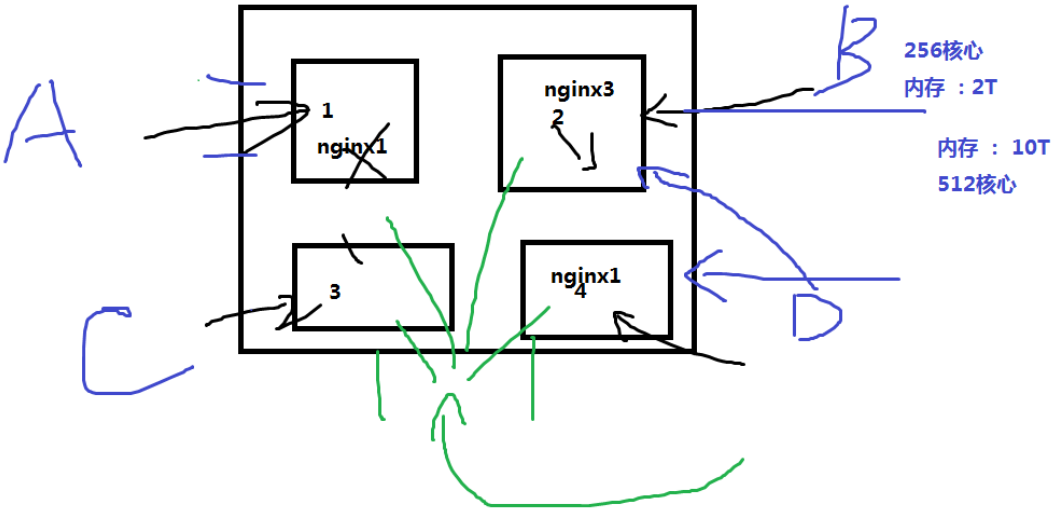
```
worker_processes  1;
```

这是用来配置 nginx 启动几个工作进程的 默认为 1。而 nginx 还支持一个名为 worker_cpu_affinity 的配置项，也就是说，nginx 可以为每个工作进程绑定 CPU。如下配置：

```
worker_processes  4;
worker_cpu_affinity    0001 0010  0100 1000;
```

这里 0001 0010 0100 1000 是掩码，分别代表第 1、2、3、4 颗 cpu 核心。

重启 nginx 后，4 个工作进程就可以各自用各自的 CPU 了。



例 4：物理机一共有 16 个核心，创建的容器只能用 0、1、2 这三个核心。

[root@xuegod63 ~]# docker run -it --name cpu1 --cpuset-cpus 0-2 centos

[root@ad554956e989 /]# cat /sys/fs/cgroup/cpuset/cpuset.cpus

0-2

[root@b3f567c41a1a /]# taskset -cp 1

pid 1's current affinity list: 0-2

9.3.5 CPU 配额控制参数的混合使用

当上面这些参数中时，cpu-shares 控制只发生在容器竞争同一个 cpu 的时间片时，如果通过 cpuset-cpus 指定容器 A 使用 cpu 0，容器 B 只是用 cpu1，在主机上只有这两个容器使用对应内核的情况，它们各自占用全部的内核资源，cpu-shares 没有明显效果。

cpu-period、cpu-quota 这两个参数一般联合使用，在单核情况或者通过 cpuset-cpus 强制容器使用一个 cpu 内核的情况下，即使 cpu-quota 超过 cpu-period，也不会使容器使用更多的 CPU 资源。

cpuset-cpus、cpuset-mems 只在多核、多内存节点上的服务器上有效，并且必须与实际的物理配置匹配，否则也无法达到资源控制的目的。

例 1：测试 cpuset-cpus 和 cpu-shares 混合使用运行效果，就需要一个压缩力测试工具 stress 来让容器实例把 cpu 跑满。

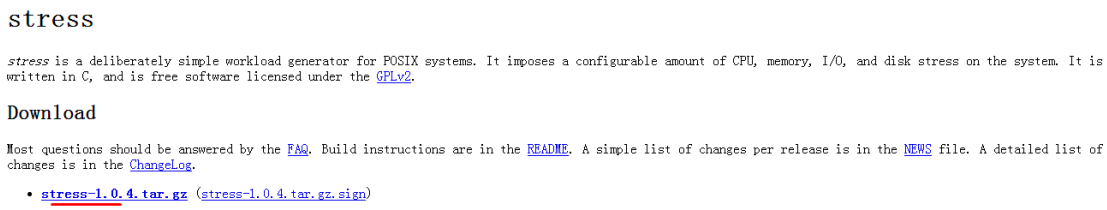
如何把 cpu 跑满？ 如何把 4 核心的 cpu 中第一和第三核心跑满？

先扩展：stress 命令

概述：linux 系统压力测试软件 Stress 。 stress 可以测试 Linux 系统 cpu/menory/IO/disk 的负载。

下载页：

http://people.seas.harvard.edu/~apw/stress/


stress

stress is a deliberately simple workload generator for POSIX systems. It imposes a configurable amount of CPU, memory, I/O, and disk stress on the system. It is written in C, and is free software licensed under the GPLv2.

Download

Most questions should be answered by the FAQ. Build instructions are in the README. A simple list of changes per release is in the NEWS file. A detailed list of changes is in the ChangeLog.

- stress-1.0.4.tar.gz (stress-1.0.4.tar.gz.sign)

注：也可以使用 epel 源中的 stress-xxx.rpm 。 MK 教你一个源码编译通用版本。
今天的这个内容不难。 因为你今天第一次知道这个知识 !!! 回去练习。

方法 1：
[root@xuegod63 ~]# yum install -y epel-release
[root@xuegod63 ~]# yum install stress -y

方法 2：
安装 stress，进行压力测试
上传 stress 到 linux 系统中。
[root@xuegod63 ~]# tar zxvf stress-1.0.4.tar.gz
[root@xuegod63 ~]# cd stress-1.0.4/
[root@xuegod63 stress-1.0.4]# ./configure
[root@xuegod63 stress-1.0.4]# make -j 4 # 学神一统 linux 江湖
[root@xuegod63 stress-1.0.4]# make install

tress 参数解释
-? 显示帮助信息
-v 显示版本号
-q 不显示运行信息
-n 显示已完成的指令情况
-t --timeout N 指定运行 N 秒后停止
--backoff N 等待 N 微妙后开始运行
-c 产生 n 个进程 每个进程都反复不停的计算随机数的平方根，测试 cpu
-i 产生 n 个进程 每个进程反复调用 sync()，sync()用于将内存上的内容写到硬盘上，测试 io
-m --vm n 产生 n 个进程,每个进程不断调用内存分配 malloc 和内存释放 free 函数 ，测试内存
--vm-bytes B 指定 malloc 时内存的字节数 （默认 256MB）
--vm-hang N 指定在 free 栈的秒数
-d --hadd n 产生 n 个执行 write 和 unlink 函数的进程
-hadd-bytes B 指定写的字节数
--hadd-noclean 不 unlink
注：时间单位可以为秒 s，分 m，小时 h，天 d，年 y，文件大小单位可以为 K，M，G

例 1：产生 2 个 cpu 进程，2 个 io 进程，20 秒后停止运行
[root@xuegod63 stress-1.0.4]# stress -c 2 -i 2 --verbose --timeout 20s
#如果执行时间为分钟，改 20s 为 1m

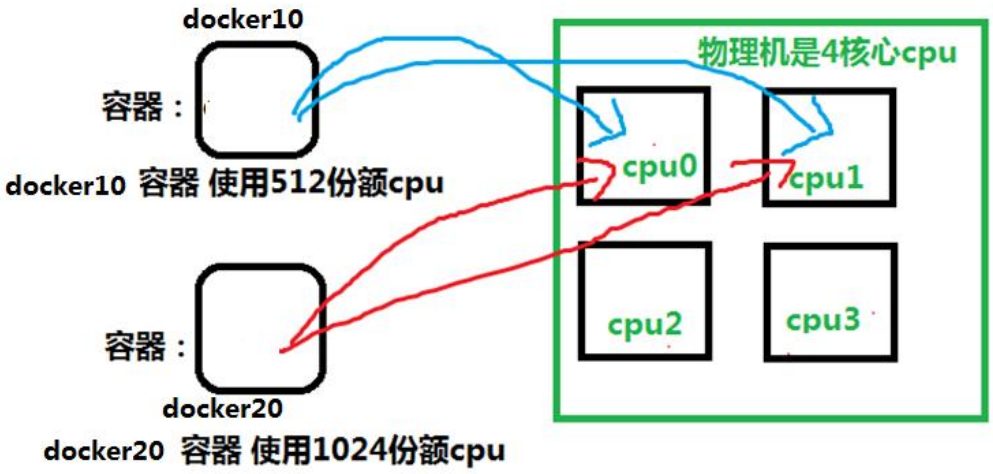
查看：

```
top - 00:46:19 up  1:27,  4 users,  load average: 1.03, 0.25, 0.12
Tasks: 414 total,   6 running, 408 sleeping,   0 stopped,   0 zombie
%Cpu(s): 52.1 us, 38.1 sy,  0.0 ni,  9.6 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3941508 total,  2504548 free,   568944 used,   868016 buff/cache
KiB Swap:  1023996 total,  1023996 free,        0 used.  3102552 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 9705 root      20   0    7260     92      0 R  99.7  0.0   0:21.99 stress
 9703 root      20   0    7260     92      0 R  99.3  0.0   0:21.97 stress
 9704 root      20   0    7260     92      0 R  78.8  0.0   0:17.72 stress
 9706 root      20   0    7260     92      0 R  78.5  0.0   0:17.72 stress
 7456 root      20   0       0      0      0 S   3.6  0.0   0:01.03 kworker/u128
```

例 1：测试 cpuset-cpus 和 cpu-shares 混合使用运行效果，就需要一个压缩力测试工具 stress 来让容器实例把 cpu 跑满。当跑满后 ,会不会去其他 cpu 上运行。如果没有在其他 cpu 上运行 ,说明 cgroup 资源限制成功。

实例 1：创建两个容器实例:docker10 和 docker20。 让 docker10 和 docker20 只运行在 cpu0 和 cpu1 上，最终测试一下 docker10 和 docker20 使用 cpu 的百分比。



创建两个容器实例：

[root@xuegod63 ~]# docker run -itd --name docker10 --cpuset-cpus 0,1 --cpu-shares 512 centos /bin/bash

[root@xuegod63 ~]# docker run -tid --name docker20 --cpuset-cpus 0,1 --cpu-shares 1024 centos /bin/bash

测试 1： 进入 docker10，使用 stress 测试进程是不是只在 cpu0,1 上运行：

[root@xuegod63 ~]# docker exec -it docker10 /bin/bash

[root@d1a431815090 /]# yum install -y epel-release #安装 epel 扩展源

[root@d1a431815090 /]# yum install stress -y

在物理机上使用 top 按 1 快捷链查看，每个 cpu 使用情况：

```
top - 01:14:57 up  1:55,  5 users,  load average: 0.33, 0.26, 0.19
Tasks: 425 total,   3 running, 422 sleeping,   0 stopped,   0 zombie
%Cpu0  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  1.2 us,  1.8 sy,  0.0 ni, 97.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3941508 total,  2209168 free,   587824 used,  1144516 buff/cache
KiB Swap:  1023996 total,  1023996 free,        0 used.  3078988 avail Mem
```

可看到正常。只在 cpu0,1 上运行

测试 2： 然后进入 docker20，使用 stress 测试进程是不是只在 cpu0,1 上运行，且 docker20 上运行的 stress 使用 cpu 百分比是 docker10 的 2 倍

[root@xuegod63 ~]# docker exec -it docker20 /bin/bash
[root@d1a431815090 /]# yum install -y epel-release #安装 epel 扩展源
[root@d1a431815090 /]# yum install stress -y
[root@f24e75bca5c0 /]# stree -c 2 -v -t 10m

在物理机上使用 top 按 1 快捷链查看，每个 cpu 使用情况：

```
top - 08:38:56 up  1:47,  4 users,  load average: 2.16, 0.91, 0.39
Tasks: 468 total,   5 running, 463 sleeping,   0 stopped,   0 zombie
%Cpu0  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3941508 total,  2262164 free,   477936 used,  1201408 buff/cache
KiB Swap:  1023996 total,  1023996 free,        0 used.  3131348 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 7322 root      20   0    7260     96      0 R  66.8  0.0   0:27.50 stress
 7323 root      20   0    7260     96      0 R  66.4  0.0   0:28.55 stress
 7325 root      20   0    7260     92      0 R  33.2  0.0   0:12.37 stress
 7326 root      20   0    7260     92      0 R  33.2  0.0   0:12.37 stress
```

注 :两个容器只在 cpu0,1 上运行。且 docker20 是 docker10 使用 cpu 的 2 倍。说明--cpu-shares 限制资源成功。

9.3.6 当容器命令运行结束后，自动删除容器

--rm 当容器命令运行结束后，自动删除容器，自动释放资源
--rm 这个参数有什么用？
[root@xuegod63 ~]# docker run --help | grep rm
--rm Automatically remove the container when it exits
作用： 当容器命令运行结束后，自动删除容器。
应用场景：在某些环境下，可能需要大量的新建 docker 实例，然后仅仅运行几秒钟，然后就彻底删除。 如运行单元测试或测试弹性云计算。

例：阿里云，要模拟双 11 的压力，需要快速创建 1 万实例，运行 ab 命令，拼命访问 tmall.com 首页，运行 1 个小时，1 小时后自动删除。

例：
[root@xuegod63 ~]# docker run -it --rm --name mk centos sleep 5
物理上查看：
[root@xuegod63 ~]# docker ps | grep mk
6c75a9317a6b centos "sleep 5" 6 seconds ago Up 4 seconds mk
等 5s 后，再查看：

[root@xuegod63 ~]# docker ps | grep mk #自动删除了

9.4 docker 容器资源配额控制之内存

9.4.1 内存

Docker 提供参数-m, --memory=""限制容器的内存使用量。

例 1：允许容器使用的内存上限为 128M：

[root@xuegod63 ~]# docker run -it -m 128m centos

查看：

[root@40bf29765691 /]# cat /sys/fs/cgroup/memory/memory.limit_in_bytes

134217728

注：也可以使用 tress 进行测试

9.5 docker 容器资源配额控制之 IO

9.5.1 IO

[root@xuegod63 ~]# docker run --help | grep write-b

--device-write-bps value Limit write rate (bytes per second) to a device (default []) #限制此设备上的写速度（bytes per second），单位可以是 kb、mb 或者 gb。

--device-read-bps value #限制此设备上的读速度（bytes per second），单位可以是 kb、mb 或者 gb。

为什么阿云平台上 普通云盘的 IO 为： 1000 IOPS ，为什么这么小？

原因是 一存储 给 2000 台云主机使用，需要控制一下 。防止某台云主机吃光你的磁盘 I/O 资源

情景：防止某个 Docker 容器吃光你的磁盘 I/O 资源

例 1：容器对硬盘的最高写入速度设定为 1MB/s。

--device 参数：将主机设备添加到容器

[root@xuegod63 ~]# docker run -it -v /var/www/html/:/var/www/html --device /dev/sda:/dev/sda --device-write-bps /dev/sda:1mb centos /bin/bash

root@81242cb3f485:~# time dd if=/dev/sda of=/var/www/html/test.out bs=1M count=50 oflag=direct,nonblock

注：dd 参数：

direct：读写数据采用直接 IO 方式；

nonblock：读写数据采用非阻塞 IO 方式

查看结果：

[root@xuegod63 ~]# ll -h /var/www/html/

总用量 21M

-rw-r--r-- 1 root root 0 12 月 28 22:37 index.html

-rw-r--r-- 1 root root 20M 12 月 28 22:40 test.out

[root@xuegod63 ~]# ll -h /var/www/html/

总用量 24M

-rw-r--r-- 1 root root 0 12 月 28 22:37 index.html

-rw-r--r-- 1 root root 23M 12 月 28 22:40 test.out

注： 发现 1 秒写 1M。 限制成功。

[root@xuegod63 ~]# docker pull ubuntu:16.04 下载 ubuntu 镜像

[root@xuegod63 ~]# docker run -it --device /dev/sda:/dev/sda --device-read-bps /dev/sda:1mb ubuntu:16.04 bash

root@331c3bac0c13:/# time dd if=/dev/sda of=/test.out bs=1M count=30 oflag=direct,nonblock

30+0 records in

30+0 records out

31457280 bytes (31 MB, 30 MiB) copied, 30.0186 s, 1.0 MB/s

#使用 ubuntu 做时，限制效果是明显的。

9.6 docker 数据映射

9.6.1 docker 数据映射。 docker 用来做计算，存储外挂

语法：docker run -itd -v /src:/dst centos bash #

-v 用来指定挂载目录， 冒号: 前面的/src 为物理机本地目录,:后面的/dst 为容器里的目录:

例 1：把物理机上的/var/www/html 映射到 docker 实例的/var/www/html 。

好处： 这样当 docker 坏了，数据还在物理机上，再使用 apache 镜像启动一个 docker 就可以了。数据不丢失。

[root@xuegod63 ~]# mkdir -p /var/www/html

[root@xuegod63 ~]# docker run -it --name web1 -v /var/www/html/:/var/www/html centos bash

[root@8fb49e536af4 /]# touch /var/www/html/index.html

物理机查看：

[root@xuegod63 ~]# ls /var/www/html/

index.html

总结：

9.1 docker 容器命名和重命名

9.2 创建 docker 容器实例时指定主机名

9.3 docker 容器资源配额控制之 cpu

9.4 docker 容器资源配额控制之内存

9.5 docker 容器资源配额控制之 IO

9.6 docker 数据映射