



容器底层实现技术



前言

- 本章主要介绍容器实现的核心技术Namespace和Cgroup，学习容器技术的本质，并对容器使用的资源进行限制。



目标

- 学完本课程后，您将能够：
 - 描述namespace实现
 - 描述cgroup实现
 - 掌握容器资源限制方法



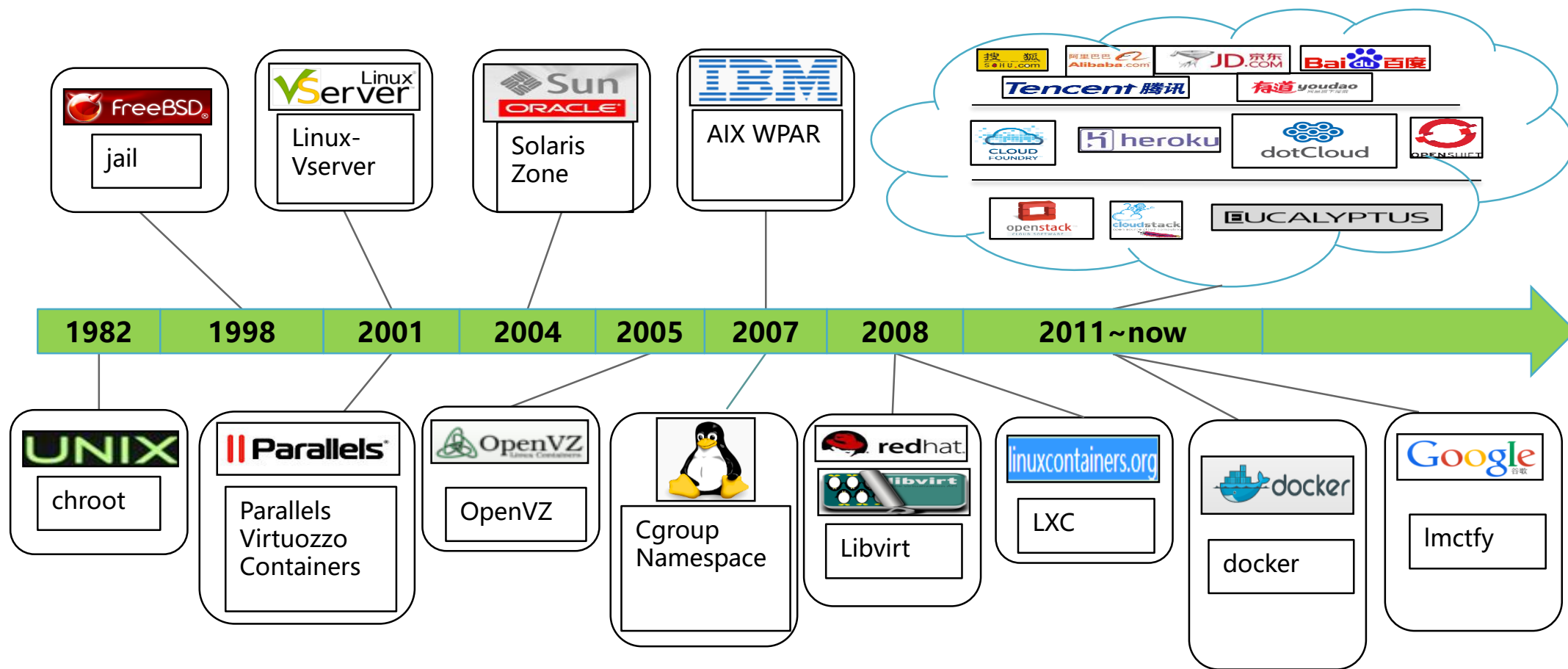
目录

1. Namespace和Cgroup

2. 容器资源限制



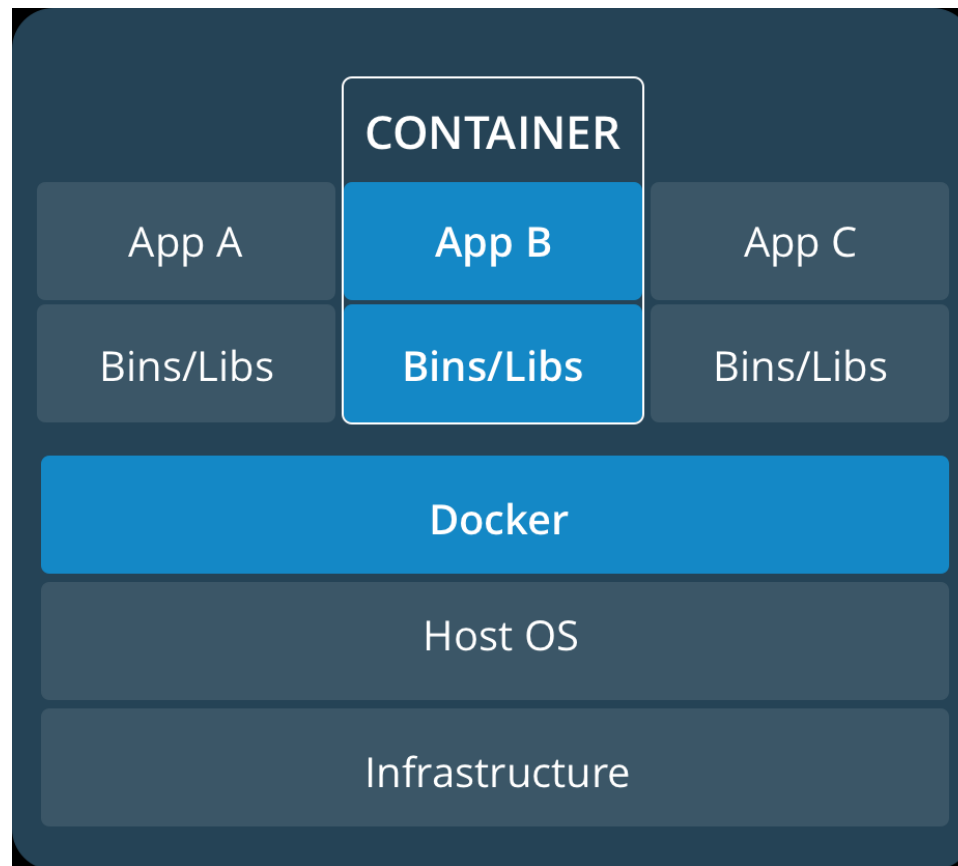
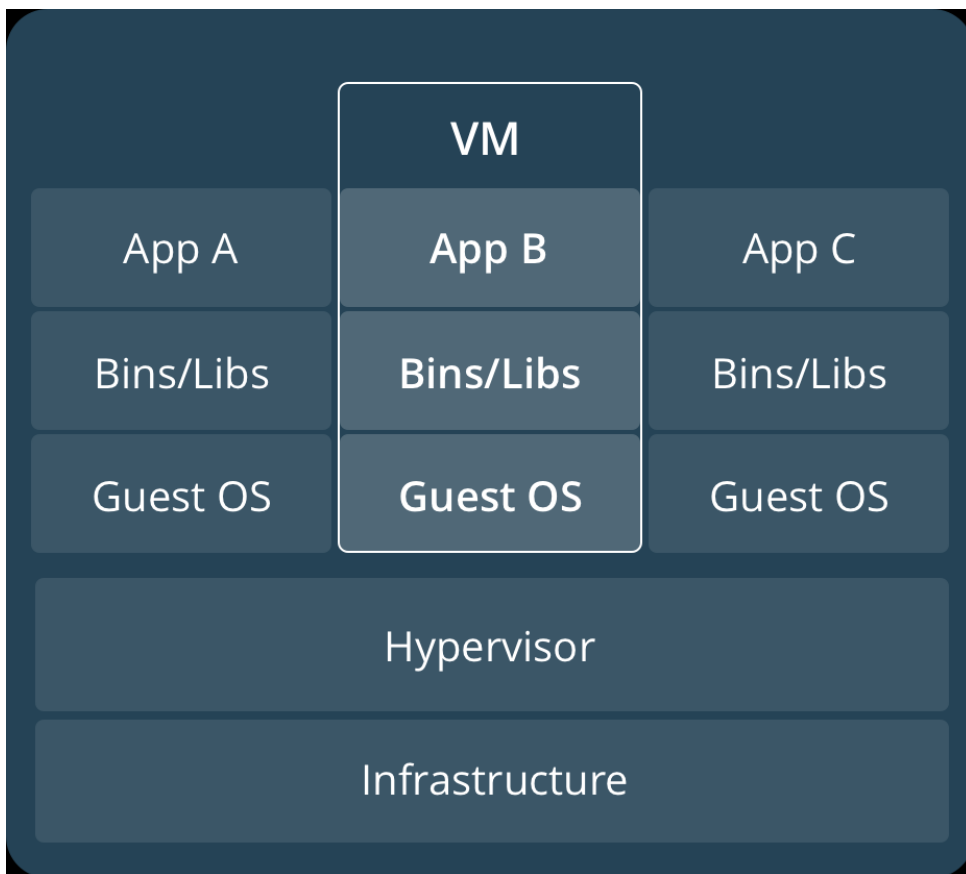
容器技术发展历史





Docker容器实现原理

- Docker容器在实现上通过namespace技术实现进程隔离，通过cgroup技术实现容器进程可用资源的限制。docker启动一个容器时，实际是创建了带多个namespace参数的进程。





Namespace

- Namespace：命名空间
 - 作用：资源隔离
 - 原理：namespace将内核的全局资源进行封装，使得每个namespace都有一份独立的资源。因此不同进程在各自namespace内对同一种资源的使用不会相互干扰。

Namespace类型	系统调用参数	隔离内容	引入的内核版本
PID namespace	CLONE_NEWPID	进程空间（进程ID）	Linux 2.6.24
Mount namespace	CLONE_NEWNS	文件系统挂载点	Linux 2.6.19
Network namespace	CLONE_NEWNET	网络资源：网络设备、端口等	始于Linux 2.6.24完成于Linux 2.6.29
User namespace	CLONE_NEWUSER	用户ID和用户组ID	始于Linux 2.6.23完成于Linux 3.8
UTS namespace	CLONE_NEWUTS	主机名和域名	Linux 2.6.19
IPC namespace	CLONE_NEWIPC	信号量、消息队列和共享内存	Linux 2.6.19



PID namespace隔离示例 (1)

- 以交互模式启动一个centos容器，并在其中运行/bin/bash程序。执行ps命令查看到“/bin/bash”是PID=1的进程，即Docker将其隔离于宿主机中的其他进程。

```
[root@localhost ~]# docker run -it centos /bin/bash
[root@24b87937f13d /]# ps axf
  PID TTY          STAT       TIME COMMAND
    1 pts/0        Ss          0:00   /bin/bash
   14 pts/0        R+          0:00   ps axf
```

- 打开另一个终端，使用docker inspect查看容器进程在宿主机上的真实PID。实际上，该容器上运行的“/bin/bash”在宿主机上是PID=96745的进程。

```
[root@localhost ~]# docker inspect 24b87937f13d | grep Pid
  "Pid": 96745,
  "PidMode": "",
  "PidsLimit": 0,
```

**还有什么方式可以查看到
容器进程真实的PID?**



PID namespace隔离示例 (2)

- 分别在宿主机和容器中查看该容器进程相关的namespace信息，发现两者是一致的。
 - 注：每个进程在/proc下都有一个目录，存放namespace相关信息。

```
[root@localhost ns]# ll /proc/96745/ns
total 0
lrwxrwxrwx. 1 root root 0 Aug  9 13:56 ipc -> ipc:[4026532192]
lrwxrwxrwx. 1 root root 0 Aug  9 13:56 mnt -> mnt:[4026532190]
lrwxrwxrwx. 1 root root 0 Aug  9 13:17 net -> net:[4026532195]
lrwxrwxrwx. 1 root root 0 Aug  9 13:56 pid -> pid:[4026532193]
lrwxrwxrwx. 1 root root 0 Aug  9 13:56 user -> user:[4026531837]
lrwxrwxrwx. 1 root root 0 Aug  9 13:56 uts -> uts:[4026532191]
```

```
[root@24b87937f13d 1]# ll /proc/1/ns
total 0
lrwxrwxrwx. 1 root root 0 Aug  9 19:16 ipc -> ipc:[4026532192]
lrwxrwxrwx. 1 root root 0 Aug  9 19:16 mnt -> mnt:[4026532190]
lrwxrwxrwx. 1 root root 0 Aug  9 19:16 net -> net:[4026532195]
lrwxrwxrwx. 1 root root 0 Aug  9 19:16 pid -> pid:[4026532193]
lrwxrwxrwx. 1 root root 0 Aug  9 19:16 user -> user:[4026531837]
lrwxrwxrwx. 1 root root 0 Aug  9 19:16 uts -> uts:[4026532191]
```



Cgroups

- Cgroups: Linux Control Group
 - 作用：限制一个进程组对系统资源的使用上限，包括CPU、内存、Block I/O等。
 - Cgroups还能设置进程优先级，对进程进行挂起和恢复等操作。
 - 原理：将一组进程放在一个Cgroup中，通过给这个Cgroup分配指定的可用资源，达到控制这一组进程可用资源的目的。
 - 实现：在Linux中，Cgroups以文件和目录的方式组织在操作系统的/sys/fs/cgroup路径下。该路径中所有的资源种类均可被cgroup限制。

```
[root@localhost /]# mount -t cgroup
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,xattr,release_agent=/usr/l
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,memory)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,hugetlb)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,cpuset)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,freezer)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,cpuacct,cpu)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,net_prio,net_cls)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,perf_event)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,blkio)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,pids)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,seclabel,devices)
```



目录

1. Namespace和Cgroup
- 2. 容器资源限制**



CPU资源限制

- 可通过如下参数，对容器的可用CPU资源进行限制：
 - `--cpu-shares`：权重值，表示该进程能使用的CPU资源的权重值。
 - `cpu.cfs_period_us`和`cpu.cfs_quota_us`：这两个配置参数一般配合使用，表示限制进程在长度为`cpu.cfs_period_us`的一段时间内，只能被分配到总量为`cpu.cfs_quota_us`的CPU时间。
 - 例如：某容器的`cpu.cfs_period_us=100000`，`cpu.cfs_quota_us=10000`。则表示该容器只能使用10%的CPU资源。



CPU资源限制示例 (1)

- 启动一个容器，进行压力测试，设置cpu权重值为1024。

```
[root@localhost /]# docker run --name huawei1 -it -c 1024 progrium/stress --cpu 1
stress: info: [1] dispatching hogs: 1 cpu, 0 io, 0 vm, 0 hdd
stress: debug: [1] using backoff sleep of 3000us
stress: debug: [1] --> hogcpu worker 1 [6] forked
```

- 使用top命令查看系统宿主机cpu使用率。

```
top - 11:02:46 up 30 days, 5:48, 4 users, load average: 2.09, 2.07, 2.10
Tasks: 105 total, 2 running, 102 sleeping, 1 stopped, 0 zombie
%Cpu(s):100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 1777256 total, 120904 free, 341496 used, 1314856 buff/cache
KiB Swap: 1048572 total, 1037820 free, 10752 used. 1053336 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
70900	root	20	0	7304	100	0	R	99.3	0.0	200:14.67	stress
1	root	20	0	54404	6212	3624	S	0.3	0.3	0:51.94	systemd
89355	root	20	0	161880	2216	1572	R	0.3	0.1	0:00.11	top
2	root	20	0	0	0	0	S	0.0	0.0	0:00.08	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:02.37	ksoftirqd/0

Cpu权重是否生效?



CPU资源限制示例 (2)

- 启动第二个容器，cpu权重值依旧设置为1024。

```
[root@localhost /]# docker run --name huawei2 -it -c 1024 polinux/stress-ng --cpu 1
stress-ng: info:  [1] defaulting to a 86400 second run per stressor
stress-ng: info:  [1] dispatching hogs: 1 cpu
stress-ng: info:  [1] cache allocate: default cache size: 16384K
```

- 再次使用top命令查看宿主机系统cpu使用率。

```
top - 11:01:25 up 30 days, 5:47, 4 users, load average: 2.03, 2.06, 2.10
Tasks: 107 total, 3 running, 103 sleeping, 1 stopped, 0 zombie
%Cpu(s):100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 1777256 total, 117180 free, 342692 used, 1317384 buff/cache
KiB Swap: 1048572 total, 1037820 free, 10752 used. 1049804 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
72058	root	20	0	7304	96	0	R	49.7	0.0	177:53.70	stress
70900	root	20	0	7304	100	0	R	49.3	0.0	199:28.70	stress
70822	root	20	0	308436	16880	10652	S	0.3	0.9	0:00.87	docker
89355	root	20	0	161880	2216	1572	R	0.3	0.1	0:00.01	top
175586	root	20	0	336432	29168	12460	S	0.3	1.6	2:37.74	docker-containe
1	root	20	0	54404	6212	3624	S	0.0	0.3	0:51.91	systemd

你能找到对该容器进行CPU限制的配置文件吗?



CPU资源限制示例 (3)

- 查看刚才创建的两个名为huawei1和huawei2容器的CONTAINER ID。

```
[root@localhost ~]# docker ps -f name="huawei*"
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
784e2e058291	progrium/stress	"/usr/bin/stress --v..."	24 seconds ago	Up 22 seconds		huawei2
bca245a8e3ba	progrium/stress	"/usr/bin/stress --v..."	39 seconds ago	Up 28 seconds		huawei1

- 已经自动为这两个容器在/sys/fs/cgroup/cpu/docker目录下创建了相应的文件夹。

```
[root@localhost ~]# cd /sys/fs/cgroup/cpu/docker/
```

```
[root@localhost docker]# ll
```

```
total 0
```

```
drwxr-xr-x. 2 root root 0 Aug  9 12:33 784e2e05829128bdc4c38cd249f0c2691a10c19fc4794f86602184ba990042a1
drwxr-xr-x. 2 root root 0 Aug  7 00:38 bca245a8e3ba26cc967c35d10be51b3c3e95faf22aae988e45b76bc1441e56ee
-rw-r--r--. 1 root root 0 Jul 10 09:14 cgroup.clone_children
--w--w--w-. 1 root root 0 Jul 10 09:14 cgroup.event_control
-rw-r--r--. 1 root root 0 Jul 10 09:14 cgroup.procs
```

- 查看容器的cpu.shares参数值和tasks值。其中tasks值即为该容器进程在宿主机上的PID。

```
[root@localhost docker]# cat 784e2e05829128bdc4c38cd249f0c2691a10c19fc4794f86602184ba990042a1/cpu.shares
1024
```

```
[root@localhost docker]# cat 784e2e05829128bdc4c38cd249f0c2691a10c19fc4794f86602184ba990042a1/tasks
94164
```

- 可在宿主机中查看cat /proc/94164/cgroup验证该容器进程当前生效的cgroup配置。



内存资源限制

- 默认情况下，宿主机不限制容器对内存资源的使用。可使用如下参数来控制容器对内存资源的使用：
 - `--memory`: 设置内存资源的使用限额
 - `--memory-swap`: 设置内存和SWAP资源的使用限额
- 对进程内存使用限制的详细配置参数在`/sys/fs/cgroup/memory`目录:

```
[root@localhost ~]# ls /sys/fs/cgroup/memory/
cgroup.clone_children  memory.kmem.limit_in_bytes      memory.limit_in_bytes      memory.oom_control      release_agent
cgroup.event_control   memory.kmem.max_usage_in_bytes  memory.max_usage_in_bytes  memory.pressure_level   system.slice
cgroup.procs           memory.kmem.slabinfo           memory.memsw.failcnt       memory.soft_limit_in_bytes tasks
cgroup.sane_behavior   memory.kmem.tcp.failcnt        memory.memsw.limit_in_bytes memory.stat               user.slice
docker                 memory.kmem.tcp.limit_in_bytes  memory.memsw.max_usage_in_bytes memory.swappiness
memory.failcnt          memory.kmem.tcp.max_usage_in_bytes memory.memsw.usage_in_bytes memory.usage_in_bytes
memory.force_empty      memory.kmem.tcp.usage_in_bytes  memory.move_charge_at_immigrate memory.use_hierarchy
memory.kmem.failcnt     memory.kmem.usage_in_bytes      memory.numa_stat           notify_on_release
```




内存资源限制示例

- 启动一个容器，使其最多使用400M内存和100M swap。并进行压力测试。

```
[root@localhost /]# docker run -it -m 400M --memory-swap=500M progrium/stress --vm 1 --vm-bytes 450M
stress: info: [1] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hdd
stress: debug: [1] using backoff sleep of 3000us
stress: debug: [1] --> hogvm worker 1 [6] forked
stress: debug: [6] allocating 471859200 bytes ...
stress: debug: [6] touching bytes in strides of 4096 bytes ...
stress: debug: [6] freed 471859200 bytes
stress: debug: [6] allocating 471859200 bytes ...
stress: debug: [6] touching bytes in strides of 4096 bytes ...
```

- 以上过程循环在“分配450M内存，释放450M内存”。



Block IO限制

- Block IO指的是磁盘的读写，可通过如下3种方式限制容器读写磁盘的带宽。
 - 设置相对权重
 - `--blkio-weight` Block IO (relative weight)
 - 设置bps：每秒读写的数据量
 - `--device-read-bps` Limit read rate (bytes per second) from a device
 - `--device-write-bps` Limit write rate (bytes per second) to a device
 - 设置iops：每秒IO次数
 - `--device-read-iops` Limit read rate (IO per second) from a device
 - `--device-write-iops` Limit write rate (IO per second) to a device



知识小考

- Docker到底为Container虚拟了什么资源?
- 容器的安全性、隔离性为什么达不到虚拟机的级别?
- Cgroups能否对容器的资源使用下限进行锁定?
- Namespace是否可以隔离系统上的所有资源?



实验&实训任务

- 实验任务
 - 请按照实验手册1.6部分完成Namespace和cgroups部分实验。
- 实训任务
 - 请灵活使用本章节课程及实验手册中学到的知识，按照实验手册1.6.4章节完成Namespace和Cgroups实训任务。



思考题

1. 限制Block IO带宽就可以限制所有IO的读写速度。 T or F
2. 默认情况下，系统不对容器的CPU资源进行限制。 T or F
3. 宿主机是否有权限杀死一个docker启动的容器？ 命令是什么？
4. 下列哪一项为Container分配所需的资源？
 - A. Linux Kernel
 - B. Docker Engine
 - C. Docker Daemon
 - D. namespace



本章总结

- Namespace实现原理
- Cgroups实现原理
- 容器对CPU、内存、Block IO资源的使用限制

The background of the slide features a blue-tinted image of several business professionals in a modern office. They are standing on a highly reflective floor, and their silhouettes are clearly visible. The people are engaged in various interactions, some holding documents or tablets. The overall aesthetic is professional and corporate.

谢谢

www.huawei.com