Docker（四）Cgroups

什么是 cgroups

容器技术本身花了很多精力在解决两方面的问题：隔离与限制。 实现隔离的方式就是 linux 的名称空间， 这一点我之前曾经讲过关于网络 docker 是如何隔离的，以及 docker 是如何利用 linux 的各种其他技术来解决被隔离的容器之间如何通信的问题。
那么第二大主题就是限制， 限制一个容器中运行的进程能够使用的资源。 比如限制 CPU，内存，IO 等我们常见资源类型。
所以其实我们可以把一个容器就当成是一个进程组， 而 Cgroups 会去限制这个进程组所使用的资源上限。

Linux Cgroups 的全称是 Linux Control Group。它最主要的作用，就是限制一个进程组能够使用的资源上限，包括 CPU、内存、磁盘、网络带宽等等。而在 Linux 中，Cgroups 给用户暴露出来的操作接口是文件系统，即它以文件和目录的方式组织在操作系统的 /sys/fs/cgroup 路径下。 我们运行如下命令：

```
[root@m7-prod-ssd001] ~ # mount -t cgroup
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,release_agent=/usr/l
ib/systemd/systemd-cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_prio,net_cls)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpuacct,cpu)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
```

或者：

```
[root@m7-prod-ssd001] ~ # ll /sys/fs/cgroup/
total 0
drwxr-xr-x 7 root root  0 Jan  7 13:49 blkio
lrwxrwxrwx 1 root root 11 Jan  7 13:49 cpu -> cpu,cpuacct
lrwxrwxrwx 1 root root 11 Jan  7 13:49 cpuacct -> cpu,cpuacct
drwxr-xr-x 7 root root  0 Jan  7 13:49 cpu,cpuacct
drwxr-xr-x 6 root root  0 Jan  7 13:49 cpuset
drwxr-xr-x 7 root root  0 Jan  7 13:49 devices
drwxr-xr-x 5 root root  0 Jan  7 13:49 freezer
drwxr-xr-x 5 root root  0 Jan  7 13:49 hugetlb
drwxr-xr-x 7 root root  0 Jan  7 13:49 memory
lrwxrwxrwx 1 root root 16 Jan  7 13:49 net_cls -> net_cls,net_prio
drwxr-xr-x 5 root root  0 Jan  7 13:49 net_cls,net_prio
lrwxrwxrwx 1 root root 16 Jan  7 13:49 net_prio -> net_cls,net_prio
drwxr-xr-x 5 root root  0 Jan  7 13:49 perf_event
drwxr-xr-x 7 root root  0 Jan  7 13:49 pids
drwxr-xr-x 7 root root  0 Jan  7 13:49 systemd
```

我们看到，cgroups 以文件系统的方式提供给用户操作。
在这个目录下有很多的控制不同资源的目录。如果我们想要限制一个进程或者一组进程的 cpu 使用。 就可以在相应的目录下进行操作。
这里我借用张磊老师的一个例子来给大家演示一下。我们现在进入 /sys/fs/cgroup/cpu 目录下：

```
root@ubuntu:/sys/fs/cgroup/cpu$ mkdir container
root@ubuntu:/sys/fs/cgroup/cpu$ ls container/
cgroup.clone_children cpu.cfs_period_us cpu.rt_period_us  cpu.shares notify_on_release
cgroup.procs          cpu.cfs_quota_us  cpu.rt_runtime_us cpu.stat   tasks
```

这个目录就称为一个 “控制组” 。你会发现，操作系统会在你新创建的 container 目录下，自动生成该子系统对应的资源限制文件。现在，我们在后台执行这样一条脚本：

```
$ while : ; do : ; done &
[1] 226
```

很显然这是一个死循环，并且它可以把 cpu 资源吃满。 如果我们使用 top 命令查看 cpu 的使用率，会发现它处于 100% 的状态。接下来我们向 container 组里的 cfs_quota 文件写入 20 ms（20000 us）。 $ echo 20000 > /sys/fs/cgroup/cpu/container/cpu.cfs_quota_us。 接下来，我们把被限制的进程的 PID 写入 container 组里的 tasks 文件，上面的设置就会对该进程生效了：$ echo 226 > /sys/fs/cgroup/cpu/container/tasks

我们可以通过 top 命令看一下，cpu 的使用率降到了 20%。 所以我们可以通过向/sys/fs/cgroup/ 这个目录下写入影响的信息而达到控制进程的目的， 所以我们在一个启动了 k8s 的节点上看看它的 cgoup 目录下的内容。 如下：

```
ll /sys/fs/cgroup/memory/
total 0
-rw-r--r--   1 root root 0 Jan  7 13:49 cgroup.clone_children
--w--w--w-   1 root root 0 Jan  7 13:49 cgroup.event_control
-rw-r--r--   1 root root 0 Jan  7 13:49 cgroup.procs
-r--r--r--   1 root root 0 Jan  7 13:49 cgroup.sane_behavior
drwxr-xr-x   2 root root 0 Jan 10 19:56 docker
drwxr-xr-x   7 root root 0 Mar  9 16:14 kubepods
drwxr-xr-x   2 root root 0 Jan  7 13:50 kube-proxy
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.failcnt
--w-------   1 root root 0 Jan  7 13:49 memory.force_empty
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.kmem.failcnt
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.kmem.limit_in_bytes
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.kmem.max_usage_in_bytes
-r--r--r--   1 root root 0 Jan  7 13:49 memory.kmem.slabinfo
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.kmem.tcp.failcnt
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.kmem.tcp.limit_in_bytes
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.kmem.tcp.max_usage_in_bytes
-r--r--r--   1 root root 0 Jan  7 13:49 memory.kmem.tcp.usage_in_bytes
-r--r--r--   1 root root 0 Jan  7 13:49 memory.kmem.usage_in_bytes
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.limit_in_bytes
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.max_usage_in_bytes
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.memsw.failcnt
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.memsw.limit_in_bytes
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.memsw.max_usage_in_bytes
-r--r--r--   1 root root 0 Jan  7 13:49 memory.memsw.usage_in_bytes
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.move_charge_at_immigrate
-r--r--r--   1 root root 0 Jan  7 13:49 memory.numa_stat
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.oom_control
----------   1 root root 0 Jan  7 13:49 memory.pressure_level
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.soft_limit_in_bytes
-r--r--r--   1 root root 0 Jan  7 13:49 memory.stat
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.swappiness
-r--r--r--   1 root root 0 Jan  7 13:49 memory.usage_in_bytes
-rw-r--r--   1 root root 0 Jan  7 13:49 memory.use_hierarchy
-rw-r--r--   1 root root 0 Jan  7 13:49 notify_on_release
-rw-r--r--   1 root root 0 Jan  7 13:49 release_agent
drwxr-xr-x 380 root root 0 Mar  9 16:53 system.slice
-rw-r--r--   1 root root 0 Jan  7 13:49 tasks
drwxr-xr-x   2 root root 0 Jan  7 13:49 user.slice
```

我们在先知内存的 cgroup 下看到了控制组 (也就是目录) kubepods 和 docker。 这下大家就知道 docker 和 k8s 是如何限制各自的资源使用的了。 在前几篇文章中我们看到了在 POD 中针对资源限制的机制。其实就是通过 cgroups 来做的。

术语定义

上面都是在系统中实际给大家看 cgroup 是怎么做的，现在来看看一些枯燥的定义。我们把每种资源叫做子系统，比如 CPU 子系统，内存子系统。为什么叫做子系统呢，因为它是从整个操作系统的资源衍生出来的。然后我们创建一种虚拟的节点，叫做 cgroup，然后这个虚拟节点可以扩展，以树形的结构，有 root 节点，和子节点。这个父节点和各个子节点就形成了层级（hierarchiy）。每个层级都可以附带继承一个或者多个子系统，就意味着，我们把资源按照分割到多个层级系统中，层级系统中的每个节点对这个资源的占比各有不同。

下面我们想法子把进程分组，进程分组的逻辑叫做 css_set。这里的 css 是 cgroup_subsys_state 的缩写。所以 css_set 和进程的关系是一对多的关系。另外，在 cgroup 眼中，进程请不要叫做进程，叫做 task。这个可能是为了和内核中进程的名词区分开吧。

进程分组 css_set，不同层级中的节点 cgroup 也都有了。那么，就要把节点 cgroup 和层级进行关联，和数据库中关系表一样。这个事一个多对多的关系。为什么呢？首先，一个节点可以隶属于多个 css_set，这就代表这这批 css_set 中的进程都拥有这个 cgroup 所代表的资源。其次，一个 css_set 需要多个 cgroup。因为一个层级的 cgroup 只代表一种或者几种资源，而一般进程是需要多种资源的集合体。