云计算技术
大作业

中国科学院大学计算机与控制学院
中国科学院计算技术研究所

开发一个容器引擎

- 实验目的：
  - 利用Linux操作系统的Namespace和Cgroup机制，实现一个简单的容器引擎，开发语言不限（shell脚本、C、Python等都可以）
- 实验要求：
  - 容器引擎应具有以下功能：
    - 实现进程、用户、文件系统、网络等方面的隔离
    - 能够在Ubuntu系统上运行CentOS环境
    - 能够实现同一操作系统下两个容器之间的网络通信
    - 能够为容器分配定量的CPU和内存资源
- 实验报告要求
  - 1. 容器引擎的源代码
  - 2. 容器引擎的实现说明
  - 3. 容器引擎支持容器隔离性、两个容器之间通信连通性、容器资源定量分配等三个核心功能的证明材料，如实验截图和说明文字

Linux资源隔离技术——cgroup

- CPU subsystem
  - 控制cgroup中进程可以使用的CPU时间片
  - 相关控制接口
    - /sys/fs/cgroup/cpu/[cgroup]/tasks
      - 该文件中可以写入1个或多个进程的PID，记录该cgroup中的进程
    - /sys/fs/cgroup/cpu/[cgroup]/cpu.cpu_period_us
      - CPU时间片的分配周期，以微秒为单位
    - /sys/fs/cgroup/cpu/[cgroup]/cpu.cpu_quota_us
      - 在一个CPU时间片分配周期中，该cgroup中的所有进程能够使用的总的CPU时间，以微秒为单位
    - /sys/fs/cgroup/cpu/[cgroup]/cpu.shares
      - 也是用来限制cgroup中进程对CPU的使用的，它限制的不是进程能够占用的绝对时间，而是控制CPU不同cgroup进程间的分配比例
      - CPU就绪队列中各个进程所属的cgroup的cpu.shares比值就是各个进程在该CPU上能够获得的CPU时间的比例
      - 在同一个CPU就绪队列中如果其它cgroup中的进程没有用完/用满自己的时间片，即提前放弃了自己的CPU时间片，那么同队列中的其它cgroup中进程可以使用这部分空闲的时间片，从而能够更加充分地利用资源

Linux资源隔离技术——cgroup

- Cgroup操作演示——将进程加入到cgroup实现资源限制
  - **方法一：命令行直接在cgroup文件系统下创建目录，配置资源限额，最后将进程PID写入到tasks列表中。该方法的配置在系统重启后失效。**
    - 1. 启动一个进程，该进程默认属于user.slice（cgroup）
      - /home/course/while1& 查看并获得该进程的PID
      - cat /sys/fs/cgroup/cpu/user.slice/tasks 能够查到刚启动进程的PID
    - 2. 在cpu子系统的Hierarchy下创建一个子目录（即生成一个对应的cgroup），系统将自动为该子目录创建相关文件
      - mkdir /sys/fs/cgroup/cpu/demo
      - ls /sys/fs/cgroup/cpu/demo
    - 3. 设置CPU时间片分配调度周期和周期内该cgroup下进程能够获得的运行时间配额
      - echo 100000 > cpu.cfs_period_us
      - echo 50000 > cpu.cfs_quota_us
    - 4. 将进程从默认的user.slice迁入到新建的cgroup
      - echo PID >> /sys/fs/cgroup/cpu/demo/tasks
      - ps -eo pid,cgroup,cmd | grep -i while1

Linux资源隔离技术——cgroup

- Cgroup操作演示——将进程加入到cgroup实现资源限制
  - **方法一：命令行直接在cgroup文件系统下创建目录，配置资源限额，最后将进程PID写入到tasks列表中。该方法的配置在系统重启后失效。**
    - 5. 删除创建的cgroup目录
      - cgdelete cpu:demo
      - 不能直接用rm -rf demo/，会报“Operation not permitted”错误
    - 6. 利用cpu.shares限制进程能够使用的CPU时间比例
      - 在/sys/fs/cgroup/cpu目录下新建一个子目录，如democpu；设置cpu.shares为4096（默认为1024）
      - 启动两个能够跑满CPU的进程
      - 查看两个进程默认所在的cgroup
      - 将其中一个进程迁移到democpu cgroup中
      - 查看两个进程所占用的时间比例
      - 将两个进程绑定到相同的CPU核上去，使其处于同一个CPU就绪队列，再次查看二者所占用的CPU时间比例
      - 调整另一个进程所在的cgroup，观察进程所占用的CPU时间比例是否有变化

Linux资源隔离技术——cgroup

- Cgroup操作演示——将进程加入到cgroup实现资源限制
  - **方法二：**通过修改cgconfig.conf和cgrules.conf两个配置文件，然后按序重启服务，自动创建cgroup并关联subsystem，该方法创建的cgroup在系统重启后能够自动创建。
    - 1. 在/etc/cgconfig.conf中定义cgroup及其与subsystem的关联关系

```
group demogrp{
   cpu{
                cpu.cfs_quota_us = 50000;
                cpu.cfs_period_us = 100000;
   }

   cpuset{
                cpuset.cpus = "3";
                cpuset.mems = "0";
   }

   memory{
                memory.limit_in_bytes=104857600;
                memory.swappiness=0;
   }

   blkio{
                blkio.throttle.read_bps_device="8:0 524288";
                blkio.throttle.write_bps_device="8:0 524288";
   }
}
```

Linux资源隔离技术——cgroup

- Cgroup操作演示——将进程加入到cgroup实现资源限制
  - **方法二：**通过修改cgconfig.conf和cgrules.conf两个配置文件，然后按序重启服务，自动创建cgroup并关联subsystem，该方法创建的cgroup在系统重启后能够自动创建。
    - 2. 在/etc/cgrules.conf中定义用户、进程与subsystem和cgroup的隶属关系

```
*:while1  *  demogrp/
```

    - 3. 重启cgconfig配置服务
      - systemctl restart cgconfig.service
    - 4. 重启cgred规则引擎服务
      - systemctl restart cgred.service

Linux资源隔离技术——cgroup

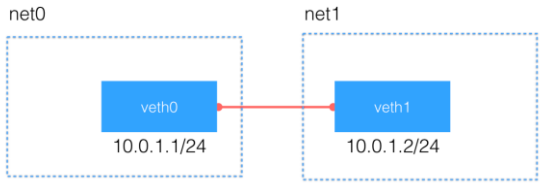
- Cgroup操作演示——将进程加入到cgroup实现资源限制
  - **方法二：通过修改cgconfig.conf和cgrules.conf两个配置文件，然后按序重启服务，自动创建cgroup并关联subsystem，该方法创建的cgroup在系统重启后能够自动创建。**
    - 5. 启动进程，查看该进程的cgroup隶属关系以及资源限额情况
    - 6. 删除配置文件中定义的cgroup和隶属关系
      - 执行删除命令：cgclear -l /etc/cgconfig.conf -e
      - 删除cgconfig.conf和cgrules.conf中定义的cgroup和进程隶属关系
      - 重启cgconfig和cgred服务

Linux资源隔离技术——Namespace

- Network Namespace
  - 是实现Linux下网络虚拟化的基础，每个network namespace有独立的网络栈信息，包括独立的网卡、路由表、ARP表、iptables等网络资源，因此可以给主机上的虚拟机或者容器提供独立的、相互隔离的网络资源
  - Linux主机上network namespace操作相关的命令
    - 查看：ip netns ls
    - 增加：ip netns add <nsname>
    - 在命名空间中执行命令：ip netns exec <nsname> ip link
  - 默认情况下，network namespace之间不能通信，也不能与主机网络通信

Linux资源隔离技术——Namespace

- Network Namespace
  - **演示1**：通过veth pair连接两个network namespace
    - 1. 创建两个network namespace
      - ip netns add net1; ip netns add net2
    - 2. 创建一对veth pair
      - ip link add type veth



    - 3. 将创建出来的veth分别放入两个network namespace中
      - ip link set veth0 netns net1; ip link set veth2 netns net2
    - 4. 分别在两个namespace中为新建的虚拟网卡分配ip地址，并配置上线
      - ip netns exec net1 ip addr add 10.0.1.1/24 dev veth0
      - ip netns exec net1 ip link set veth0 up
      - ip netns exec net2 ip addr add 10.0.1.2/24 dev veth1
      - ip netns exec net2 ip link set veth1 up
    - 5. 从其中一个namespace向另一个namespace发包
      - ip netns exec net1 ping 10.0.1.2

Linux资源隔离技术——Namespace

- Network Namespace
  - **演示2**：通过Linux bridge连接多个network namespace
    - 1. 创建一个bridge
      - ip link add br0 type bridge; ip link set dev br0 up
    - 1. 为每个namespace创建一对veth pair
      - ip link add type veth
    - 2. 把veth pair其中的一个放入network namespace中，为其配置ip地址
      - ip link set dev veth0 netns net1
      - ip netns exec net1 ip link set dev veth0 name eth0
      - ip netns exec net1 ip addr add 10.0.2.1/24 dev eth0
      - ip netns exec net1 ip link set dev eth0 up
    - 3. 将veth pair中的另一个连接到创建的bridge上
      - ip link set dev veth1 master br0
      - ip link set dev veth1 up
    - 4. 测试网络连通性
      - ip netns exec net1 ping 10.0.2.2