# cgroups

## What are they and how do I use them??

Lucas Chaufournier

June 7, 2017

## Table of Contents

General Overview
- Definition
- General Interface
- Manual Cgroup Interaction
- cgroup Tools
- Other Frameworks

Available cgroups

## Definition

### From the Linux Kernel Documentation

"Control Groups provide a mechanism for aggregating/partitioning sets of tasks and all their future children, into hierarchical groups with specialized behaviors."

## Definition

### From the Linux Kernel Documentation

"Control Groups provide a mechanism for aggregating/partitioning sets of tasks and all their future children, into hierarchical groups with specialized behaviors."

### Layman's Terms

cgroups are a generic mechanism to group tasks together and apply difference resource controllers to them.

## General Interface

There are multiple ways to interact with the cgroup system.

1. Manually Interact with the filesystem
2. Use of the api.
3. Use of cgexec and cgcreate interface
4. Container Management Platforms such as Docker.

## General Interface

There are multiple ways to interact with the cgroup system.

1. Manually Interact with the filesystem
2. Use of the api.
3. Use of cgexec and cgcreate interface
4. Container Management Platforms such as Docker.

## Manual Cgroup Interaction

cgroups are typically mounted at the following location:

```
/sys/fs/cgroup
```

This directory contains the various resource types that we can control. These include:

- blkio
- cpu
- cpuacct
- cpuset
- devices
- freezer
- hugetlb
- memory
- net_cls
- net_prio

We'll go into a little more detail on these later.

## Manual cgroup Interaction: Creating a cgroup

To create a new cgroup, simply create a new directory in the resource that you want to limit. We will use memory as an example.

```
# mkdir /sys/fs/cgroup/memory/mycgroup1
```

If you run *ls* on that directory you will see that many new files have been added. Interacting with these files, allows one to set various limits on the resource.

## Manual cgroup Interaction: Adding a process to the cgroup.

To add a process to this cgroup we need to find the PID of the process. Once we have the PID, we can echo it to the cgroup.procs file in our cgroup directory.

### Add a process with PID 4950 to cgroup.

```
# echo 4950 > /sys/fs/cgroup/memory/mycgroup1/cgroup.procs
```

Our process is now in our cgroup and we can now begin to set limits.

## Manual cgroup Interaction: Setting Limits.

We can set the maximum amount of memory a process can use by writing the max number of bytes to the **memory.limit_in_bytes** file.

### Limit process to 1GB.

```
# echo 1000000000 > /sys/fs/cgroup/memory/mycgroup1/memory.limit_in_bytes
```

Our process now has only 1GB of memory available to it.

## Manual cgroup Interaction: Misc

- To remove a cgroup, you must first remove all processes from the cgroup.procs and tasks files.
- Processes can only belong to one cgroup at a time for a resource. Adding to a new cgroup removes membership of the other.
- You can view the cgroups a process is a member of by reading the **/proc/[pid]/cgroup** file.

## cgroup Tools

It turns out there is an even easier method for controlling cgroups in the form of *cgexec*, *cgset*, *cgdelete* and *cgcreate*.
These tools provide an easy to use interface for creating, managing, and removing cgroups.

To create a cgroup simply run the cgcreate command like so:

```
# cgcreate -g cpu,cpuacct,...:/mycgroup1
```

Replace *cpu,cpuacct,...* with your desired cgroup controllers.

## cgroup Tools

With *cgexec* you can start processes in a cgroup.

Run bash in a memory cgroup.

```
# cgexec -g cpu,memory:mycgroup1 bash
```

With *cgclassify* you can add existing processes to a cgroup.

Add process 4985 to mycgroup1

```
# cgclassify -g cpu,memory:mycgroup1 4985
```

With *cgset* you can set the limits on a cgroup.

Limit memory of cgroup to 5gb.

```
# cgset -r memory.limit_in_bytes=5000000000 mycgroup1
```

## cgroup Tools: Delete cgroups

With *cgdelete* you can delete a cgroup.

### Delete a cgroup.

```
# cgdelete cpu,memory:/mycgroup1
```

## Other Frameworks

Other frameworks create cgroups automatically when you start containers.

### Docker container with 8gb ram and cpus 1,2,3

```
# docker run -it -m=8g --cpuset-cpus="1-3" ubuntu /bin/bash
```

### LXD/LXC container with 8gb ram and cpus 1,2,3

```
# lxc launch ubuntu-daily:16.04 container1
# lxc config set container1 limits.memory 8GB
# lxc config set container1 limits.cpu  1,4
```

### rkt container with 8gb ram and cpus 1,2,3

```
# rkt run --interactive docker://ubuntu --insecure-options=image
# Manually Manage
```

## blkio

- Controls and monitors access to I/O on block devices by all tasks in a cgroup.
- Proportional Weight: Uses CFQ scheduling to make sure each cgroup has a set percentage of all I/O operations
- I/O Throttling: Used to set an upper limit for the number of I/O operations performed by specific device.
- blkio provides many fine tuning knobs as well as methods for reading I/O stats.
- blkio does not work for buffered write operations but does work for direct access and buffered read.

## blkio Tunables

blkio tunables and statistics:

- blkio.io_merged
- blkio.throttle.read_bps_device
- blkio.io_merged_recursive
- blkio.throttle.read_iops_device
- blkio.io_queued
- blkio.throttle.write_bps_device
- blkio.io_queued_recursive
- blkio.throttle.write_iops_device
- blkio.io_service_bytes
- blkio.time
- blkio.io_service_bytes_recursive
- blkio.time_recursive
- blkio.io_serviced
- blkio.weight
- blkio.io_serviced_recursive
- blkio.weight_device
- blkio.io_service_time
- blkio.io_service_time_recursive
- blkio.io_wait_time
- blkio.io_wait_time_recursive
- blkio.leaf_weight
- blkio.leaf_weight_device
- blkio.reset_stats
- blkio.sectors
- blkio.sectors_recursive
- blkio.throttle.io_service_bytes
- blkio.throttle.io_serviced

## cpuacct

- Generates automatic reports for cpu usage of a cgroup.
- `cpuacct.usage` reports the total cpu time used.
- `cpuacct.stat` reports the user and system time spent on cpu for all tasks in a cgroup.
- `cpuacct.usage_percpu` reports the time consumed on each cpu for all tasks.

## cpusets

- Specifies the cpus and memory nodes to be assigned to a cgroup.
- Allows you to specify a subset of cpus for tasks in the cgroup to run on.
- `cpuset.cpu_exclusive` allows you to prevent other cgroups from using the specified cpus. Effectively pins exclusively to the core.
- Available options
  - cpuset.cpus
  - cpuset.mems
  - cpuset.memory_migrate
  - cpuset.cpu_exclusive
  - cpuset.mem_exclusive
  - cpuset.mem_hardwall
  - cpuset.memory_pressure
  - cpuset.memory_pressure_enabled
  - cpuset.memory_spread_page
  - cpuset.memory_spread_slab
  - cpuset.sched_load_balance
  - cpuset.sched_relax_domain_level

## Devices

- Allows and disallows access to devices for a cgroup.
- Specify devices in a 4-tuple that represents type (block vs character), device type (i.e. SCSI),
  device node (i.e. partition1), access (r,rw,m).
- `b 8:1 rw` → a block device type SCSI with partition 1 and read/write access.
- `devices.list` can be used to list all devices a cgroup has access to.
- By default cgroups have access to all devices.

## freezer

- The freezer subsystem allows one to suspend and resume tasks.
- `freezer.state` only available to non-root cgroups.
- There are only three states:

- Frozen
- Freezing
- Thawed

- Simply move a process into a cgroup with freezer and then write Frozen to the cgroup.

## memory

- The memory subsystem allows the control of memory resources for the cgroup.
- Allows you to set hard and soft limits on the amount of memory for the cgroup as well as the swappiness.
- To limit the overall memory for a cgroup write to `memory.limit_in_bytes`. See example in earlier slides.
- Available options:

- memory.limit_in_bytes
- memory.numa_stat
- memory.oom_control
- memory.pressure_level
- memory.soft_limit_in_bytes
- memory.force_empty
- memory.swappiness
- memory.kmem.limit_in_bytes
- memory.use_hierarchy
- memory.kmem.slabinfo
- memory.kmem.tcp.limit_in_bytes
- memory.move_charge_at_immigrate

## net

- This subsystem allows you to tag packets and set the priority of network traffic for cgroups.
- `net_cls` tags network packets allowing `tc` to identify packets.
- `net_cls.classid` is a single value written in hex of the form 0xAAAABBBB that is the tag for packets.
- `net_prio.prioidx` is a read only file that contains the id for cgroup.
- `net_prio.ifpriomap` is a mapping of interfaces to priorities for the cgroup.