Linux Containers

Basic Concepts

Lucian Carata
FRESCO Talklet, 3 Oct 2014
### Underlying kernel mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cgroups</td>
<td>manage resources for groups of processes</td>
</tr>
<tr>
<td>namespaces</td>
<td>per process resource isolation</td>
</tr>
<tr>
<td>seccomp</td>
<td>limit available system calls</td>
</tr>
<tr>
<td>capabilities</td>
<td>limit available privileges</td>
</tr>
<tr>
<td>CRIU</td>
<td>checkpoint/restore (with kernel support)</td>
</tr>
</tbody>
</table>
cgroups - user space view

low-level filesystem interface similar to sysfs (/sys) and procfs (/proc)
new filesystem type “cgroup”, default location in /sys/fs/cgroup

cgroup hierarchies

/subsystem

TL /cpu
  /cpu
  /high-priority
  /normal
  /experiment_1

TL /mem
  /mem
  /opus
  /normal
  /experiment_1

subsystems (controllers)

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

* each subsystem can be used at most once

▲ built as kernel module

TL top level cgroup (mount)
cgroups - kernel space view

- `task_struct`
  - `css_set *cgroups`
  - `list_head cg_list`

- `css_set`
  - `list_head tasks`
  - `cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT]`

- kernel code for attach/detaching task from `css_set`
  - `init/main.c`
  - `fork()`, `exit()`

List of all tasks using the same `css_set`
cgroups - kernel space view

- task_struct
  - css_set *cgroups
  - list_head cg_list

- css_set
  - list_head tasks
  - cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT]

- cgroup_subsys
  - int (*attach)(...)
  - void (*fork)(...)
  - void (*exit)(...)
  - void (*bind)(...)
  - ...
  - const char* name;
  - cgroupfs_root *root;
  - cftype *base_cftypes

- cgroup_subsys cpuset_subsys
- cgroup_subsys freezer_subsys
- cgroup_subsys mem_cgroup_subsys
cgroups - kernel space view

```c
#include <linux/cgroup_subsys.h>

cgroup_subsys

int (*attach)(...)
void (*fork)(...)
void (*exit)(...)
void (*bind)(...)

...

const char* name;
cgroupfs_root *root;
cftype *base_cftypes

cgroup_subsys cpuset_subsys
    .base_cftypes = files
```

```c
static struct cftype files[] = {
    {   .name = "cpus",
        .seq_show = cpuset_common_seq_show,
        .write_string = cpuset_write_remask,
        .max_write_len = \(100U + 6 * NR_CPUS\),
        .private = FILE_CPULIST,
        ...
    },
    {   .name = "mems",
        .seq_show = cpuset_common_seq_show,
        .write_string = cpuset_write_remask,
        .max_write_len = \(100U + 6 * MAX_NUMNODES\),
        .private = FILE_MEMLIST,
        ...
    },
    {   .name = "cpu_exclusive",
        .read_u64 = cpuset_read_u64,
        .write_u64 = cpuset_write_u64,
        .private = FILE_CPU_EXCLUSIVE,
        ...
    },
    {   .name = "mem_exclusive",
        .read_u64 = cpuset_read_u64,
        .write_u64 = cpuset_write_u64,
        .private = FILE_MEM_EXCLUSIVE,
        ...
    },
};
```
cgroups - summary

cgroup hierarchies

/sys/fs/cgroup

TL /cpu  
   /high-priority
   /normal
   /experiment_1

TL /mem  
   /opus
   /normal
   /experiment_1

subsystems (controllers)

cpuset  cpu  cpuacct
memory  hugetbl
devices  blkio  net_cls ▲  net_prio▲
freezer  perf

* each subsystem can be used at most once*

▲ built as kernel module
TL top level cgroup (mount)
Namespaces limit the scope of kernel-side *names* and *data structures* at process granularity.

- **mnt** (mount points, filesystems) - `CLONE_NEWNS`
- **pid** (processes) - `CLONE_NEWPID`
- **net** (network stack) - `CLONE_NEWNET`
- **ipc** (System V IPC) - `CLONE_NEWIPC`
- **uts** (unix timesharing - domain name, etc) - `CLONE_NEWUTS`
- **user** (UIDs) - `CLONE_NEWUSER`
Namespaces limit the scope of kernel-side **names** and **data structures** at process granularity

Three system calls for management

- **clone()** new process, new namespace, attach process to ns
- **unshare()** new namespace, attach current process to it
- **setns(int fd, int nstype)** join an existing namespace
namespaces - user space view

- each namespace is identified by an *inode* (unique)
- six (?) entries (inodes) added to /proc/<pid>/ns/

- two processes are in the same namespace if they see the same inode for equivalent namespace types (mnt, net, user, ...)

**User space utilities**

* IPROUTE (ip netns add, etc)
* unshare, nsenter (part of util-linux)
* shadow, shadow-utils (for user ns)
For each namespace type, a default namespace exists (the global namespace).

- **struct nsproxy** is shared by all tasks with the same set of namespaces.
namespaces - kernel space view

Example for **uts** namespace

- global access to hostname: `system_utsname.nodename`
- namespace-aware access to hostname: `&current->nsproxy->uts_ns->name->nodename`
Example for **net** namespace

- a **network device** belongs to exactly one network namespace
- a **socket** belongs to exactly one network namespace
- a new network namespace only includes the loopback device
- communication between namespaces using **veth** or **unix sockets**
Namespaces limit the scope of kernel-side names and data structures at process granularity.

- **mnt** (mount points, filesystems)
- **pid** (processes)
- **net** (network stack)
- **ipc** (System V IPC)
- **uts** (unix timesharing - domain name, etc)
- **user** (UIDs)
Containers

- A light form of resource virtualization based on kernel mechanisms
- A container is a *user-space* construct

- Multiple containers run on top of the *same kernel*
  - illusion that they are the only one using resources
    (cpu, memory, disk, network)

- some implementations offer support for
  - container templates
  - deployment / migration
  - union filesystems

[Diagram of Container layers](taken from the Docker documentation)
Container solutions

Mainline

Google containers (lmctfy)
- uses cgroups only, offers CPU & memory isolation
- no isolation for: disk I/O, network, filesystem, checkpoint/restore
- adds some cgroup files: \texttt{cpu.lat, cpuacct.histogram}

LXC: user-space containerisation tools

Docker

systemd-nspawn

Forks

Vserver, OpenVZ
Container solutions - LXC

An LXC container is a userspace process created with the `clone()` system call

- with its own pid namespace
- with its own mnt namespace
- net namespace (configurable) - lxc.network.type

Offers container templates `/usr/share/lxc/templates`

- shell scripts
  - `lxc-create -t ubuntu -n containerName`
    - also creates cgroup `/sys/fs/cgroup/<controller>/lxc/containerName`
Container solutions - Docker

A Linux container engine

- multiple backend drivers
- application rather than machine-centric
- app build tools
- diff-based deployment of updates (AUFS)
- versioning (git-like) and reuse

- links (tunnels) between containers

taken from the Docker documentation
Questions?

Thank you!

Lucian Carata
lc525@cam.ac.uk

More details

cgroups:  http://media.wix.com/ugd/295986_d73d8d6087ed430c34c21f90b0b607fd.pdf

namespaces:  http://lwn.net/Articles/531114/  (and series)