Linux Containers

Basic Concepts

Lucian Carata
FRESCO Talklet, 3 Oct 2014
<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Function</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

/subsystems (controllers)

TL: top level cgroup (mount)

△ built as kernel module
cgroups - user space view

cgroup hierarchies

/sys/fs/cgroup

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

TL /mem  memory
  /opus
  /normal
  /experiment_1

common
  tasks
cgroup.procs
release_agent  TL
notify_on_release
cgroup.clone_children
cgroup.sane_behavior

cpuacct
  cpuacct.stat
  cpuacct.usage
  cpuacct.usage_percpu

cpu
  cpu.stat
  cpu.shares
  cpu.cfs_period_us
  cpu.cfs_quota_us
  cpu.rt_period_us
  cpu.rt_runtime_us

cpuset
memory
hugetbl
devices
blkio
net_cls
net_prio
freezer
perf
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]`

- `include / linux / cgroup.h`

kernel code for attach/detaching task from css_set

```
init/main.c
fork(), exit()
```
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`

- `include / linux / cgroup.h`
- `include / linux / cgroup_subsys.h`

- `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
```
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

hugetlb

devices

blkio

net_cls

net_prio

freezer

perf

* each subsystem can be used at most once*

TL top level cgroup (mount)

△ built as kernel module
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**

Logical copy of the network stack:

- loopback device
- all network tables (routing, etc)
- all sockets
- /procfs and /sysfs entries
namespaces - summary

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
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: cpu.lat, cpuacct.histogram

LXC: user-space containerisation tools

Docker

systemd-nspawn

Forks
Vserver, OpenVZ
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`
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)