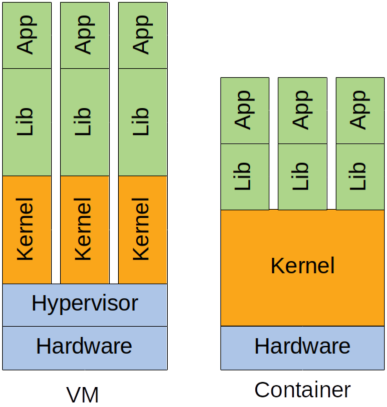
Containers: Namespaces and Dockers

Guillaume Urvoy-Keller[2]

January 7, 2022

[2] Special thanks to Benoit Benedetti for his slides

VMs versus Containers



Source: *Tom Goethals, Merlijn Sebrechts, Ankita Atrey, Bruno Volckaert, Filip De Turck: Unikernels vs Containers: An In-Depth Benchmarking Study in the Context of Microservice Applications. SC^2 2018: 1-8*

Containers (compared to VMs)

Pros

- Lightweight
- Little impact on performance

Cons

- Security as kernel is shared!
- Migration more complex

Containers key components (in Linux)

- Namespaces: logical separation of processes
- Cgroups: physical separation of processes, i.e. assignation of resources (CPU, Memory, Network) to processes
- Features of Linux kernel
- Containers solutions (Docker, LXC, OpenVZ, etc.) are built on top of these features

Namespaces

- Appeared 2.4.19 in 2002
- Limits what a process sees in terms of other processes, network interfaces, volumes mounted
- Types: pid, net, mnt, uts, ipc, user
- 6 namespaces corresponding to 6 different views/dimensions. Ex: a process can be separated from the others only along a single dimension, e.g. network
- A process in a Linux system belongs to a set of namespaces
    - ...even if it is not in a container. In this case it belongs to the default namepaces.

NET Namespace

A process sees only the networking stack of the NET namespace it belongs to

- its own interfaces (loopback at least)
- its own routing tables
- its own iptables (Linux firewall/NAT) rules
- its own sockets

UTS Namespace

- UNIX Time Sharing (misleading)
- Specific hostname

IPC Namespace

- Inter Process Communication
- Enable a set of processes to have their:
  - semaphores
  - message queues
  - shared memory

User Namespace

- Specific UID/GID (User ID / Group ID) for each User Namespace
- Map IDs of each User Namespace to the default IDs of Namesapces
    - uid 0 → 9999 of container C1 correspond to uid 10000 → 19999 on host
    - uid 0 → 9999 of container C2 correspond to uid 20000 → 29999 on host

Mount Namespace

- A namespace has its own rootfs
- can mask /proc, /sys
- can have private mount and /tmp

PID Namespace

A process in a specific namespace PID only sees the processes in the same namespace PID

- Processes ids start at 1 in each namespace PID
- If PID 1 disappears, the namespace disappears

Namespaces in practice

- unshare() and clone()
- For a given process, its namespaces are visible /proc/{pid}/ns

Namespaces in practice

```
root@stretch:/home/vagrant# echo $$
1125
root@stretch:/home/vagrant# ls —l /proc/1125/ns
lrwxrwxrwx 1 root root 0 Jan 3 20:57 cgroup —> cgroup:[4026531835]
lrwxrwxrwx 1 root root 0 Jan 3 20:57 ipc —> ipc:[4026531839]
lrwxrwxrwx 1 root root 0 Jan 3 20:57 mnt —> mnt:[4026531840]
lrwxrwxrwx 1 root root 0 Jan 3 20:57 net —> net:[4026531957] # Net namepace 1
lrwxrwxrwx 1 root root 0 Jan 3 20:57 pid —> pid:[4026531836]
lrwxrwxrwx 1 root root 0 Jan 3 20:57 user —> user:[4026531837]
lrwxrwxrwx 1 root root 0 Jan 3 20:57 uts —> uts:[4026531838]
root@stretch:/home/vagrant# unshare —n /bin/bash
root@stretch:/home/vagrant# echo $$
1138
root@stretch:/home/vagrant# ls —l /proc/1138/ns
lrwxrwxrwx 1 root root 0 Jan 3 20:57 cgroup —> cgroup:[4026531835]
lrwxrwxrwx 1 root root 0 Jan 3 20:57 ipc —> ipc:[4026531839]
lrwxrwxrwx 1 root root 0 Jan 3 20:57 mnt —> mnt:[4026531840]
lrwxrwxrwx 1 root root 0 Jan 3 20:57 net —> net:[4026532116] # Net namepace 2
lrwxrwxrwx 1 root root 0 Jan 3 20:57 pid —> pid:[4026531836]
lrwxrwxrwx 1 root root 0 Jan 3 20:57 user —> user:[4026531837]
lrwxrwxrwx 1 root root 0 Jan 3 20:57 uts —> uts:[4026531838]
```

Control groups (Cgroups)

- Appeared in 2008 (kernel 2.6.24)
- Enable to control and monitor the resources allocated to a process: cpu, memory, network, block io, device

Cgroups

Source: `http://www.tothenew.com/blog/cgroups-and-namespaces-on-ubuntu/`

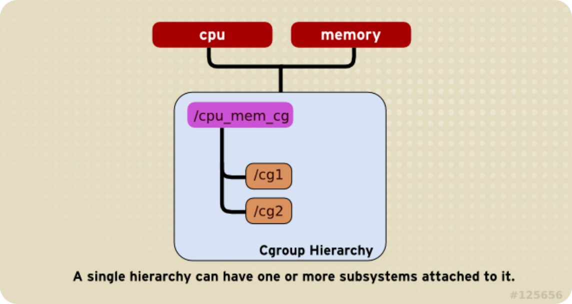
- Control groups (cgroups) are a kernel mechanism for grouping, tracking, and limiting the resource usage of processes.
- Terminology:
  - Subsystem: A subsystem represents a single resource, such as CPU time or memory
  - Common subsystems:
    - cpusets: tasks in a cpuset cgroup may only be scheduled on CPUS assigned to that cpuset.
    - blkio: limits per-cgroup block io
    - net_prio : allows setting network traffic priority on a per-cgroup basis.

Cgroups

Terminology (cont'd)

- Hierarchy: a set of subsystems mounted together forms a hierarchy.
- Tasks: processes are called tasks (in cgroups terminology).
- Cgroups : A cgroup associates a set of tasks with a set of parameters for one or more subsystems

Cgroups- Hierarchy



A single hierarchy can have one or more subsystems attached to it.

- One hierarchy with two subsystems, cpu and memory
- Hierarchy and cgroups materialized as filesystem and directory within this filesystem

Cgroups- Hierarchy

- cg1 contains a set of files where you write what is the resource consumption limit you impose and the processes managed under c1.

```
root@stretch:/sys/fs/cgroup/cpuset# cd cg1
root@stretch:/sys/fs/cgroup/cpuset/cg1# ls
cgroup.clone_children cpuset.effective_mems cpuset.memory_spread_page
        notify_on_release
cgroup.procs cpuset.mem_exclusive cpuset.memory_spread_slab tasks
cpuset.cpu_exclusive cpuset.mem_hardwall cpuset.mems
cpuset.cpus cpuset.memory_migrate cpuset.sched_load_balance
cpuset.effective_cpus cpuset.memory_pressure cpuset.sched_relax_domain_level
```

Cgroups- Hierarchy

- Assigning a process (with PID p) to cg1 is as simple as writing the p into the task file inside cg1 directory (see lab)
- During a fork, the child inherits the cgroup (and also namespace) of parent

Docker 101

- A service running inside a machine
- Relies on namespaces and cgroups to create containers
- A hub `https://hub.docker.com/` from which you can browse and download container images
- Official images are managed with names with single string, e.g. ubuntu or nginx
- Non official images are managed with names like urvoy/ubuntu

Docker 101: lifecylcle of a container

- A container is created with docker run command
- A container is running if at least one process is running inside
- If not, it is stopped
- but not killed

Docker 101: docker in practice

```
Guillaumes−MacBook−Pro−2:~ urvoy$ docker run −it ubuntu:bionic /bin/bash # start a bash
        inside a container
# with image downloaded from the hub
Unable to find image 'ubuntu:bionic' locally
bionic: Pulling from library/ubuntu
84ed7d2f608f: Pull complete
be2bf1c4a48d: Pull complete
a5bdc6303093: Pull complete
e9055237d68d: Pull complete
Digest: sha256:868fd30a0e47b8d8ac485df174795b5e2fe8a6c8f056cc707b232d65b8a1ab68
Status: Downloaded newer image for ubuntu:bionic
root@d8826d373f2f:/# we are in the container now!
```

Docker 101: listing containers and images

We can list the running containers on the host

```
Guillaumes—MacBook—Pro—2:~ urvoy$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
4f6217e9cb2b ubuntu "/bin/bash" 11 seconds ago Up 10 seconds recursing_spence
```

And also the images available to create containers:

```

Guillaumes—MacBook—Pro—2:~ urvoy$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
ubuntu bionic 1d9c17228a9e 6 days ago 86.7MB
```

Docker 101: lifecycle

```
Guillaumes−MacBook−Pro−2:~ urvoy$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
4f6217e9cb2b ubuntu "/bin/bash" 4 minutes ago Up 4 minutes recursing_spence
Guillaumes−MacBook−Pro−2:~ urvoy$ docker attach 4f6217e9cb2b
root@4f6217e9cb2b:/# read escape sequence # Simply type ^P^Q to exit without killing
Guillaumes−MacBook−Pro−2:~ urvoy$ docker attach 4f6217e9cb2b
root@4f6217e9cb2b:/# exit # now we (try to) kill
exit
Guillaumes−MacBook−Pro−2:~ urvoy$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
Guillaumes−MacBook−Pro−2:~ urvoy$ docker ps −a # but it is still there and only stopped (
        note the −a)
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
4f6217e9cb2b ubuntu "/bin/bash" 4 minutes ago Exited (0) 5 seconds ago recursing_spence
Guillaumes−MacBook−Pro−2:~ urvoy$ docker rm 4f6217e9cb2b # now we kill it
4f6217e9cb2b
Guillaumes−MacBook−Pro−2:~ urvoy$
```