



ICT academy

Linux Containers

April 2024

Leon Štefanič Južnič, Matej Rabzelj

www.ict-academy.eu



IKT Katedra za informacijske
in komunikacijske
tehnologije

LTFE Laboratorij za
telekomunikacije

LMMFE Laboratorij za
multimedijo
LTFE ICT academy

www.ltfe.org

Linux containers, 8.4.2024

08:30 – 09:15

- Introduction to Linux Containers

09:15 – 11:30

- LXC

11:30 – 12:30

- Lunch

12:30 – 15:30

- LXD



AVTORSKE PRAVICE GRADIVA

Vsa gradiva usposabljanja z naslovom **Linux containers**, december 2024 (predavanja, demonstracije, vaje), so last Univerze v Ljubljani, Fakultete za elektrotehniko (UL FE).

- Uporaba je dovoljena izključno v okviru usposabljanja.
- Noben del gradiva ne sme biti reproduciran, shranjen ali prepisan v katerikoli obliki oziroma na katerikoli način (elektronsko, mehansko, s fotokopiranjem, snemanjem) brez predhodnega pisnega dovoljenja.

Vse pravice pridržane. © December 2024, UL FE.

UL Fakulteta za elektrotehniko



news



01 October

WELCOME DAY FOR INTERNATIONAL STUDENTS

Erasmus+ welcome day will take place at 1st October 2018 at 10:00 at Faculty of Electrical Engineering in "KuFE" (follow the routing board in lobby). All necessary informations you can find here Timetable for first semester

30 August

LECTURE OF PROF HYOUNGSEOP KIM

University of Ljubljana Faculty of Electrical Engineering ICT department IEEE Slovenia Section cordially invite you to attend the lecture of prof. dr. Hyoungseop Kim Kyutech, Japan Non-rigid Image registration technique for detection of lung nodules on MDCT images The lecture will be held on Monday, 3. 9. 2018 at 13:00 in Multimedia Hall' at the Faculty...

links

[studis](#)
[timetable](#)
[study calendar](#)
[staff contacts](#)
[exchange student guide](#)
[students office](#)
[enrolment](#)
[study in slovenia](#)



2.200+ študentov

**300+ raziskovalcev
in akademskega
osebja**

**50 % sredstev
pridobljenih na trgu**

Laboratorij za telekomunikacije LTFE

Laboratorij za multimedijo LMMFE

50+

Sodelavcev

100+

Projektov

600+

Akademskih objav

150+

Zadovoljnih partnerjev



Bločne verige v ekosistemih pametnih mest



Najdete nas tudi na:



@ltfe.org



@LTFE



@ltfe.lmmfe

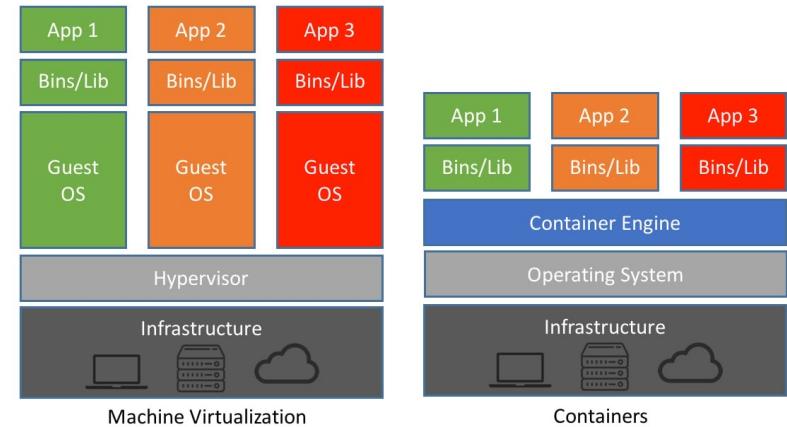


**Laboratory for Telecommunications (LTFE) & Laboratory for
Multimedia (LMMFE)**

Introduction to Linux Containers

Containerization vs traditional virtualization

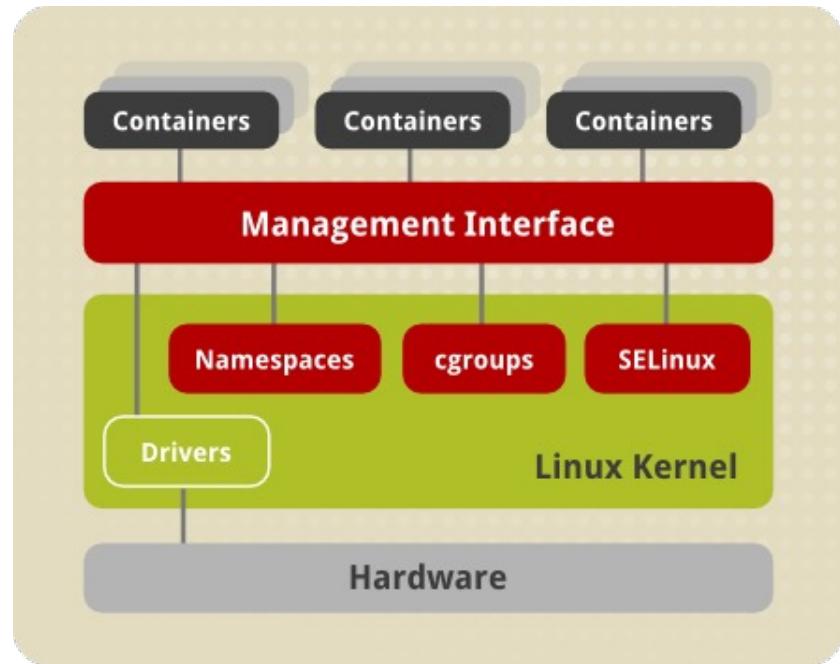
- **Virtualization** uses hypervisors to run multiple virtual machines on a single host, each with its own isolated operating system.
 - Virtual machines are heavy, slow, resource-intensive, and less flexible.
- **Containerization** uses user space and process isolations to run multiple containers on a single host, each with its own application environment.
 - Containers are **lightweight**, fast, resource-efficient, and portable



Source: <https://www.netapp.com/blog/containers-vs-vms/>

Introduction to Containers

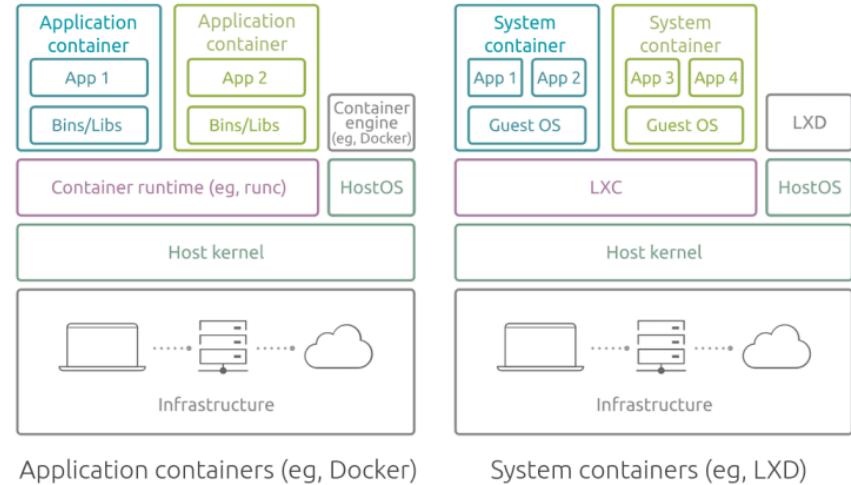
- Containers can **virtualize** both the operating system and the application level
- Containers can **isolate** and package applications with their dependencies
- Containers are **portable** and lightweight, and can run on any host
- Containers can **optimize** resource utilization and scale easily
- Containers can **facilitate** development, test, and production workflows



Source: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/introduction_to_linux_containers

What types of containers are there?

- A **container** is a single operating system image that runs isolated applications and their dependent resources on a host machine.
- There are two types of containers: **operating system level** and **application level**.
- **Operating system level containers** run a full operating system and can host multiple applications. They are long-lasting and similar to virtual machines. An example is LXD.
- **Application level containers** run a single process or service and are stateless and ephemeral. They are lightweight and easy to create and delete. An example is Docker.
- Both types of containers **share the kernel with the host** operating system and create isolated processes.



Source: <https://ubuntu.com/blog/lxd-vs-docker>

Container History

Container technology has a **long history** in different operating systems, but became popular in Linux with kernel support. **System containers are the oldest type of containers**, and they run a second operating system on the same kernel as the host.

Some of the milestones in container history are:

- 1982: Chroot (Unix-like operating systems)
- 1999: BSD introduced jails, a way of running a second BSD system on the same kernel as the main system.
- 2001: Linux implementation of the concept through Linux vServer.
- 2004: Solaris (Sun Solaris, Open Solaris) grew Zones which was the same concept but a part of Solaris OS.
- 2005: OpenVZ project started to implement multiple VPSs (virtual private servers) on Linux.
- 2008: LXC (Linux)
- 2013: Docker (Linux, FreeBSD, Windows)

LXC and linuxcontainers.org

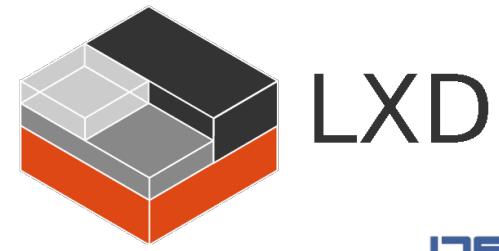
- LXC is the first system container technology based on mainline Linux features.
- LXC can create both system containers and application containers with a low-level interface.
- LXC containers are similar to chroot, but with more isolation and flexibility.
- LXC containers run a standard Linux installation on the same kernel as the host, with no virtualization overhead.
- linuxcontainers.org is the umbrella project for various Linux container technologies.
- linuxcontainers.org aims to provide a distro and vendor neutral environment for container development.
- linuxcontainers.org offers both containers and virtual machines that run full Linux systems.

When should you use Linux containers?

- Anytime when you're running Linux on Linux, you should be considering using containers instead of virtual machines.
- For almost any use case, you could run the exact same workload in a system container and not get any of the overhead that you usually get when using virtual machines.
- The only exception would be if you needed a specific version of the kernel different from the kernel of the host, for a specific feature of that virtual machine.
- System containers are much easier to manage than virtual machines.

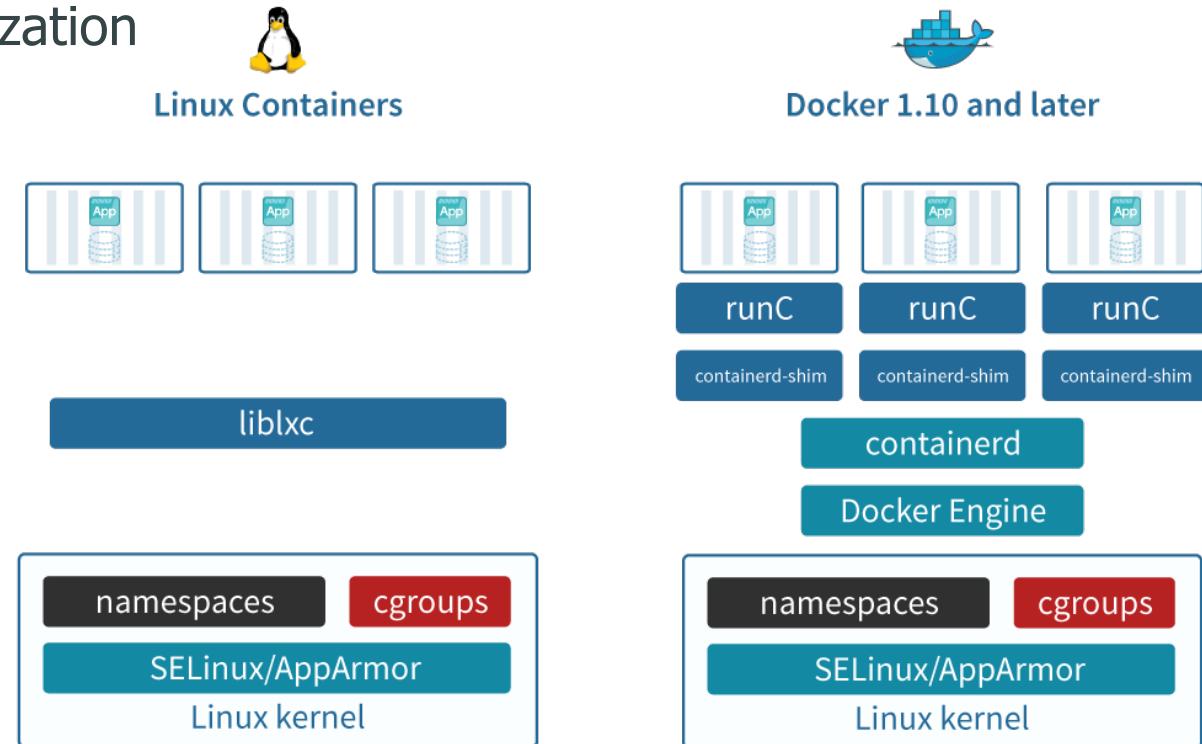
— What is LXD?

- **LXD is a system container and a virtual machine manager** that runs on top of LXC, enhancing the experience and enabling easier control and maintenance.
- Benefits:
 - System container and virtual machine manager built on top of LXC, enabling easier management, control and integration
 - Supports container and VMs
 - Better user experience through a simple REST API



Docker vs Linux Containers

- Host-Machine Utilization
- Simplicity
- Tooling
- Use Cases

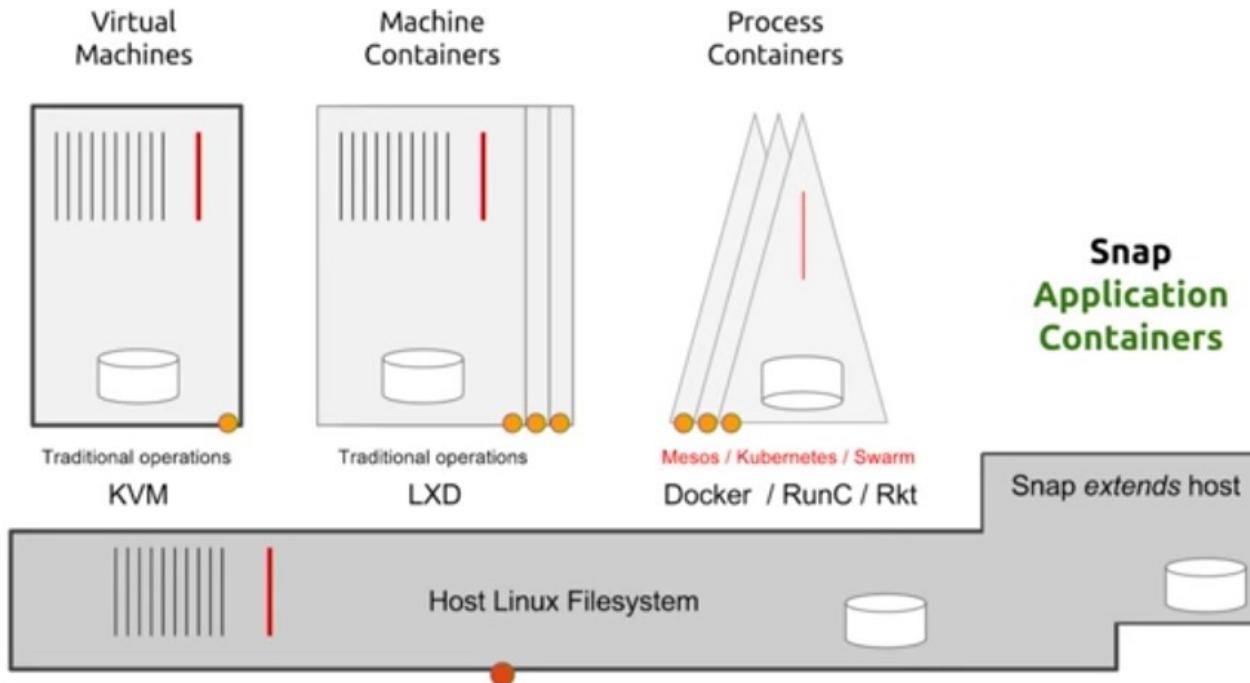


Source: <https://www.techdivision.com/aktuelles/blog/lxc-vs-docker-wir-setzen-bei-techdivision-inzwischen-verstaerkt-auf-lxc>

Features to Enable Containers

Recap

VM vs. VE



A look under the hood



Containerization (LXC)

is based on **limiting** and **isolating** processes.

Linux uses two key technologies to achieve this:

- **Control groups (cgroups)**
 - “resource management / limiting”
- **Namespaces (ns)**
 - “process isolation / virtual resources”

Both of these technologies are part of the modern Linux kernel, while their userland tooling is typically installed separately.

Containerization (LXC)

LXC and LXD packages only provide high-level abstraction / tools to efficiently configure and manage containers, ran by OS and enabled by the kernel's features.

Under the hood, cgroups and namespaces do most of the work in combination with other features enabling modern containers:

- **efficient filesystems with volume managers** (snapshots, cow, ...)
- security mechanisms, such as:
 - **seccomp** (limiting system calls)
 - **linux capabilities** (limiting root permissions)

Control groups (cgroups)

Introduced by Google in 2007, enter mainline in 2008 => LXC

Cgroups enable **resource management**:

- allocate, prioritize, deny, manage, and monitor
- CPU time, system memory, network bandwidth, etc.

Core idea: group processes (tasks) and apply group resource limits

Use in LXC / Docker: container resource limits

- prevent monopolizing host system resources
- performance optimization in shared environments

Control groups (cgroups)

Under the hood, cgroups make use of **kernel resource controllers**:

- **io** (set input/output limits/shares on resources)
- **memory** (set memory use limits and reporting)
- **pids** (limit number of processes and their children)
- rdma (set Remote DMA / InfiniBand resource limits)
- **cpu** (adjust scheduler parameters e.g., Completely Fair Scheduler)
- **cpuset** (restrict available cpu and memory nodes)
- perf_event (performance monitoring and reporting)

Under the hood => cpu controller integrates with linux scheduler (cpu time)
memory controller integrates with kernel memory manager

Control groups (cgroups)

We don't need LXC and LXD to create and manage cgroups:

- interaction via filesystem interface **/sys/fs/cgroup/**
 - in UNIX "everything is a file", pseudo-filesystem
- using userland tools e.g. **cgroup-tools**, **cgutils**, **libcgroup**

```
ls -lh /sys/fs/cgroup

-r--r--r-- 1 root root 0 Dec 12 16:38 cgroup.controllers
-rw-r--r-- 1 root root 0 Dec 12 16:38 cgroup.max.depth
-rw-r--r-- 1 root root 0 Dec 12 16:38 cgroup.max.descendants
-rw-r--r-- 1 root root 0 Dec 12 16:38 cgroup.pressure
-rw-r--r-- 1 root root 0 Dec 12 16:38 cgroup.procs
-r--r--r-- 1 root root 0 Dec 12 16:38 cgroup.stat
-rw-r--r-- 1 root root 0 Dec 12 16:38 cgroup.subtree_control
-rw-r--r-- 1 root root 0 Dec 12 16:38 cgroup.threads
```

Control groups (cgroups)

We don't need LXC and LXD to create and manage cgroups:

We can now view or modify maximum allowed cgroup memory using standard file operations:

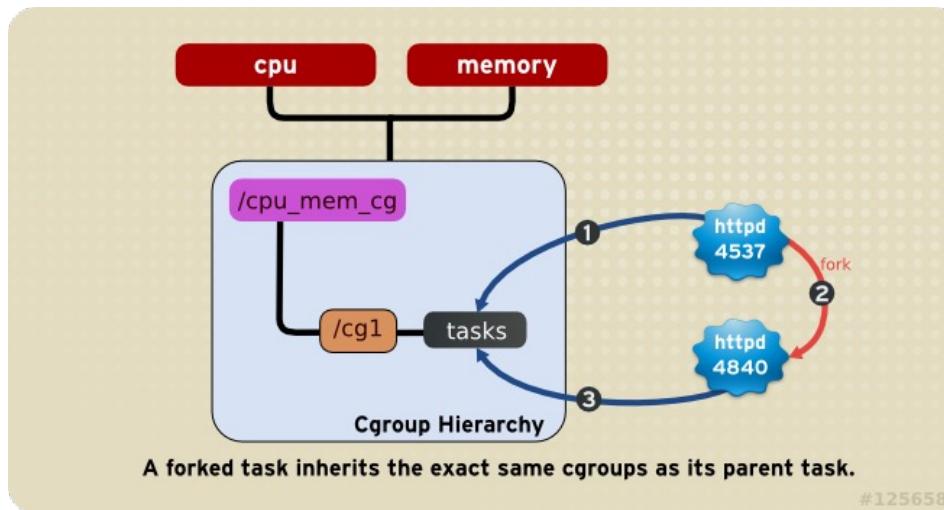
```
# get max memory  
cat /sys/fs/cgroup/mygroup/memory.max # result in bytes  
# add PID 1234 to group  
echo 1234 >> /sys/fs/cgroup/mygroup/cgroup.procs
```

Linux uses inotify to monitor for fs changes and apply resource controller policies in real-time.

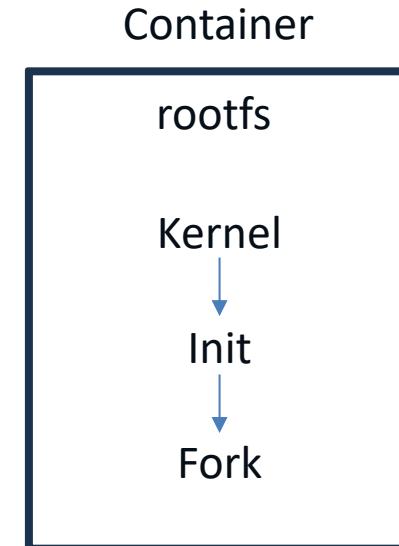
Control groups (cgroups)

Towards containers

cgroups hierarchies follow a set of rules – child task automatically inherits cgroup membership of its parent, but can be moved



#125658



Namespaces

enable **resource and process isolation** and allow for the partitioning of various aspects of the operating system, so that each set of processes sees its own isolated instance of the global resource.

Namespaces provide **virtual environments for processes**, isolating them from other processes and resources. Environments / namespace types:

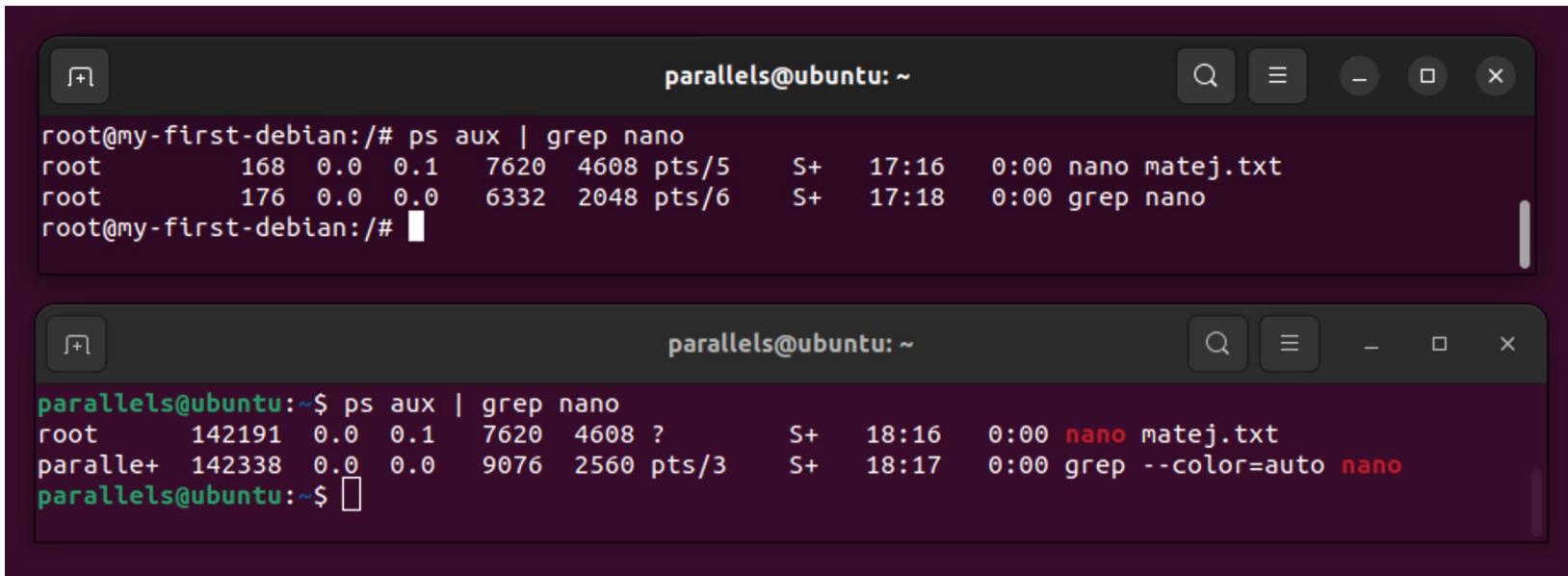
- **Mount**: CLONE_NEWNS - Mount points
- **Network**: CLONE_NEWNET - Network devices, stacks, firewall, routing tables, etc.
- **User**: CLONE_NEWUSER - User and group IDs
- **PID**: CLONE_NEWPID - Process IDs
- **UTS**: CLONE_NEWUTS - Hostname and NIS domain name
- **IPC**: CLONE_NEWIPC - System V IPC, POSIX message queues
- **Cgroup**: CLONE_NEWCGROUP - Cgroup root directory

Namespaces

Towards containers

Mount namespace (chroot / pivot_root)

PID Namespace (remapping process IDs)



The image shows two terminal windows side-by-side. Both windows have a dark theme with a light-colored title bar and a dark background.

Top Terminal:

```
parallels@ubuntu: ~
root@my-first-debian:/# ps aux | grep nano
root      168  0.0  0.1    7620  4608 pts/5      S+   17:16   0:00 nano matej.txt
root      176  0.0  0.0    6332  2048 pts/6      S+   17:18   0:00 grep nano
root@my-first-debian:/#
```

Bottom Terminal:

```
parallels@ubuntu: ~
parallels@ubuntu:~$ ps aux | grep nano
root      142191  0.0  0.1    7620  4608 ?          S+   18:16   0:00 nano matej.txt
parallels  142338  0.0  0.0    9076  2560 pts/3      S+   18:17   0:00 grep --color=auto nano
parallels@ubuntu:~$
```

Namespaces

Towards containers

Network namespace

```
sudo ip netns add mynetns # add a network namespace
```

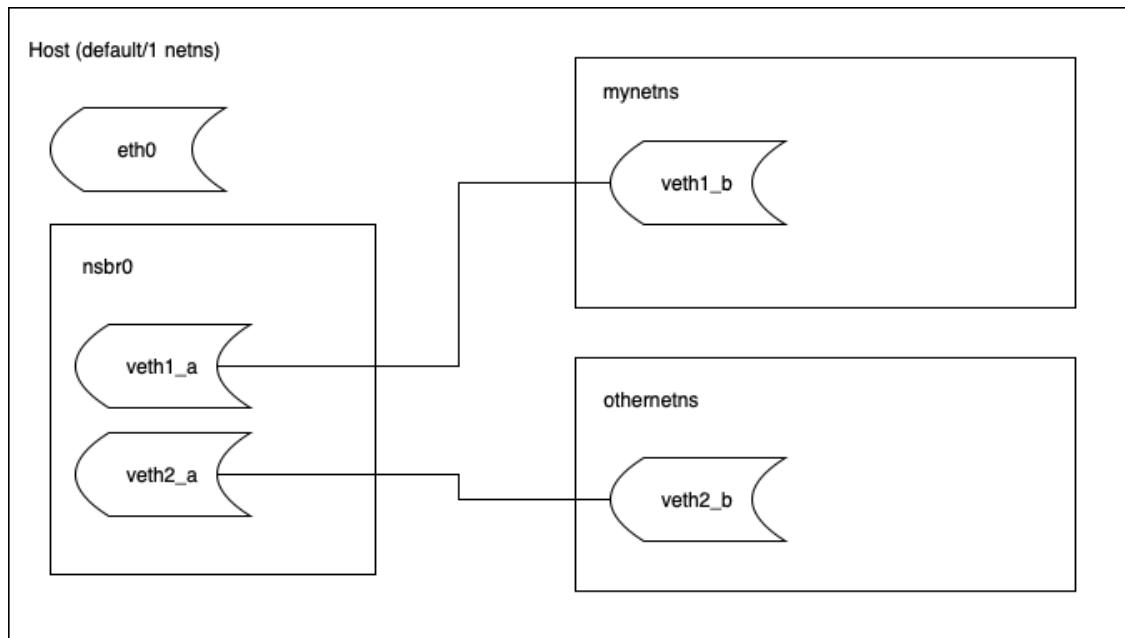
.... configure new IP, routes, etc. inside mynetns

```
sudo ip netns exec mynetns bash  
ping 1.1.1.1
```

Namespaces

Towards containers

Network namespace



Creating containers

We can now add a **rootfs (root filesystem)** of a desired guest operating system and combine the described mechanisms to start a new container:

1. Creating isolated namespaces.
 - e.g., isolating the filesystem similar to chroot, creating new netns with independent networking stack, etc.
2. Applying resource limits with cgroups.
 - e.g., setting cpu and memory limits, throttling network interfaces, etc.
3. Implementing security measures with capabilities.
 - e.g., allow creating sockets, define container "privilege" level (more on that later)
4. Starting an init process within the container.
 - init assumes PID 1 inside the container PID ns and starts userspace environment, the init system may differ from the one used by the host (e.g., sysvinit vs systemd)

Securing containers

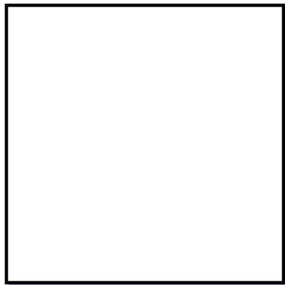
Additionally, we can implement security mechanisms to further limit the contained processes (DAC vs MAC).

Traditional linux security controls are based on filesystem permissions and **users** and **groups** (e.g., root can do everything).

Since we may need to run some containers as root (e.g., to access a host device), we need to limit what other actions this container can do.

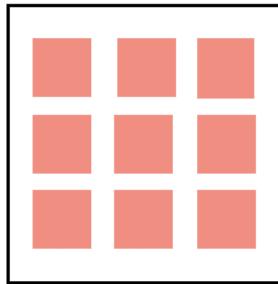
We can use **Linux capabilities** mechanism
to further subdivide “unlimited” permission transfer
from root user to root-owned process (e.g., userland libcap2-bin).

Securing containers



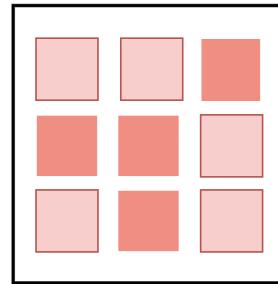
without capabilities
Kernel < 2.2

(A)



full capabilities

(B)



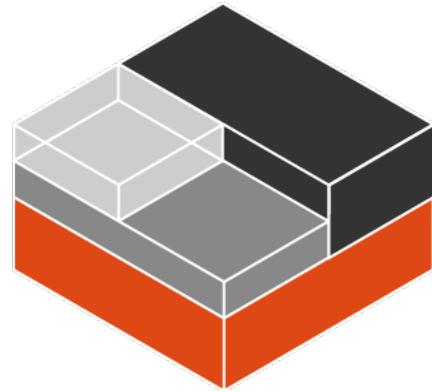
specific capabilities

(C)

Example:

- + add CAP_SYS_RAWIO to access IO devices
- remove CAP_NET_BIND to prevent network port binding

— That's it!



Thank you!

Leon Štefanič Južnič

leon.stefanic@ltfe.org

Matej Rabzelj

matej.rabzelj@ltfe.org



Laboratorij za telekomunikacije



Laboratorij za multimedijo



Katedra za informacijske
in komunikacijske
tehnologije



Univerza v Ljubljani
Fakulteta za elektrotehniko

