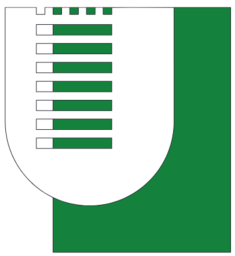
Containers

Linux Kernel and Docker

Roberto Masocco
roberto.masocco@uniroma2.it

University of Rome "Tor Vergata"
Department of Civil Engineering and Computer Science Engineering
Intelligent Systems Lab

May 5, 2022



TOR VERGATA
UNIVERSITY OF ROME

School of Engineering

Disclaimer

What follows is heavily based on specific features of the Linux kernel.
Compatibility with different platforms cannot be guaranteed.

Roadmap

1. Containers
2. Docker
3. Docker Compose

Roadmap

Why Containers?

Example: Packaging Applications

Suppose you are ready to distribute your new application:

- you need to be sure that it is compatible with all the **platforms** you chose to support;
- you need to figure out a way to deal with **dependencies**;
- you want to publish some kind of self-contained, easily-identifiable **package**.

Why Containers?

Example: Isolating Applications

Suppose you are deploying applications on a server:

- you want to define **resource quotas** and **permissions** for each;
- you want to be sure that each module has what it needs to operate, but **nothing more**;
- you want to **isolate** each module for security reasons, in case something goes wrong.

Why Containers?

Example: Replicating Environments

Suppose you are developing applications for a specific system (maybe with a different architecture):

- you want to have a **local copy** of such system without carrying one with you;
- you want to have all **libraries** and **dependencies** installed without tainting your own system;
- you would like to **deploy** the entire installation with just a few commands, without running any script but simply copying data.

Why Containers?

A possible solution to many of the previous situations could be a set of **virtual machines**.
However, virtual machines are **slow**, hypervisors take up **system resources** and guest kernels must always be **tweaked**.
In each of the above scenarios something simpler would be enough, especially since **the OS is not involved**, only applications are.
This is what a **container** is.

Figure 1: FreeBSD jail logo

Containers in the Linux kernel

Support for containers was added to the Linux kernel with a set of **features** starting from kernel 2.6 (2003), mainly:

- **control groups** (`cgroups`): defining different resource usage policies for groups of processes;
- **namespaces**: isolating processes and users in different "realms", both hardware (e.g. network stack) and software (e.g. PIDs);
- **capabilities**: defining what a process can do, with both hardware and software resources.

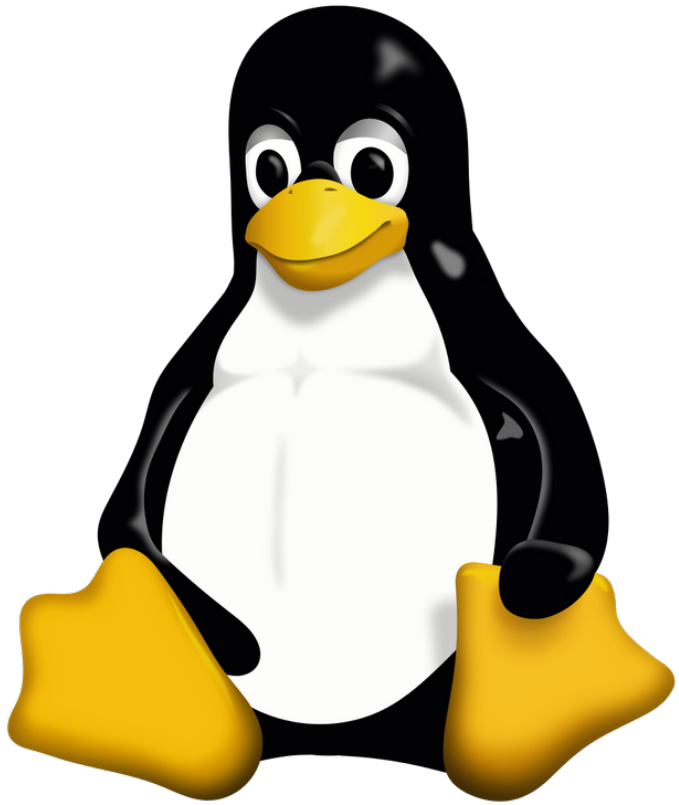


Figure 2: Tux

Containers in the Linux kernel

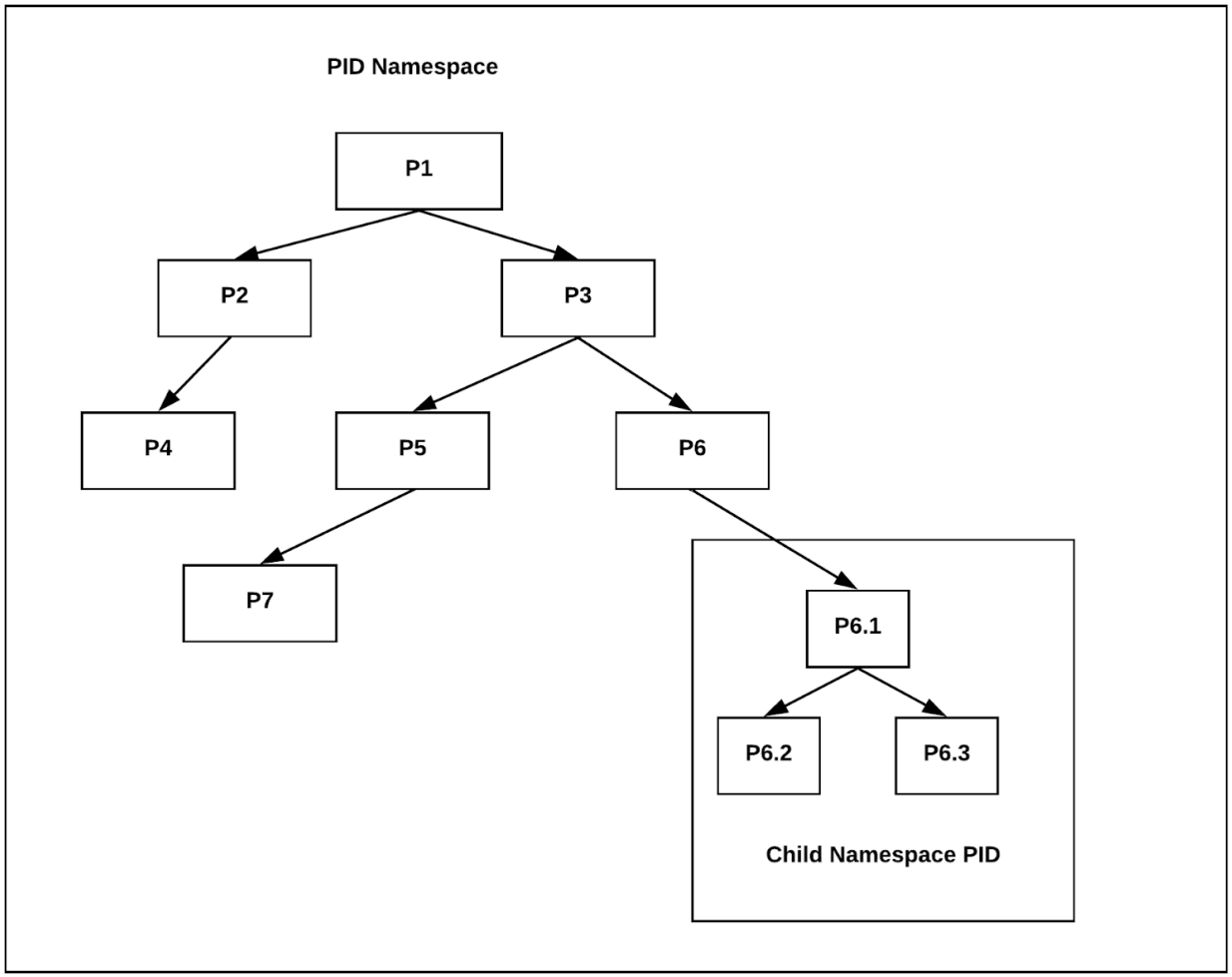


Figure 3: Nested PID namespaces

Roadmap

Roadmap