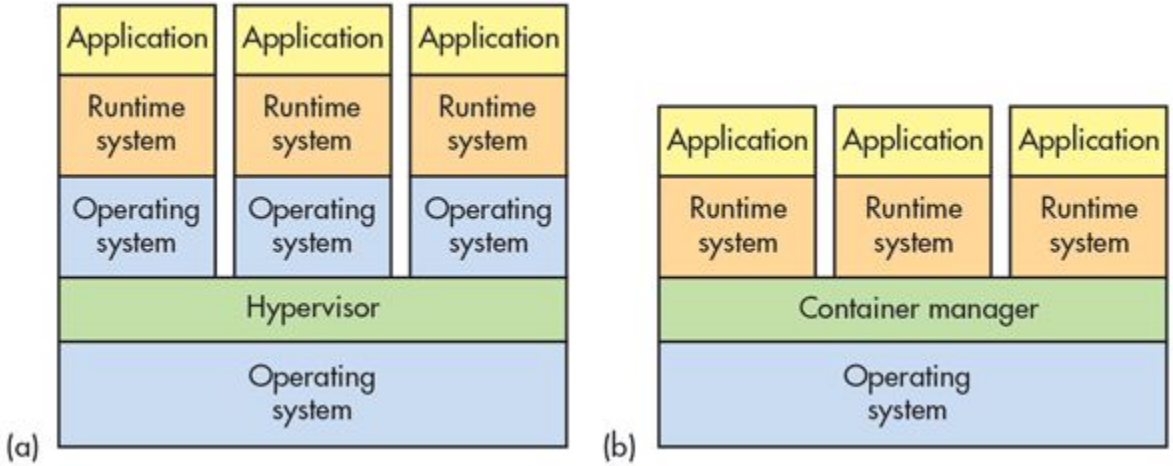
cgroups, namespaces, containers: the hype

Siddharth Yadav, Radhika Ghosal

Some jargon

- **VM** - provides an abstract machine (*virtual hardware*)
  - Can run OS of choice on emulated hardware (and accompanying device drivers)
- **Container** - provides an abstract OS (*virtual OS*)
  - Isolates application's view of the OS - process trees, network, user IDs, and mounted file systems.
  - More lightweight than VMs
- **Hypervisor / VMM** - manages VMs
  - Interface between emulated OS and actual hardware
  - **VirtualBox**, **QEMU**
- **Container engine** - manages containers
  - Interface between emulated isolated userspace and host OS
  - **Docker**

Some jargon


Application
Application
Application
Runtime system
Runtime system
Runtime system
Operating system
Operating system
Operating system
Hypervisor
Operating system
(a)
Application
Application
Application
Runtime system
Runtime system
Runtime system
Container manager
Operating system
(b)

Some jargon

A single physical machine can have multiple VMs, and each VM can run multiple containers!

Internals: Virtualization

- Dive into KVM, QEMU, Intel-VT!
  - In QEMU, check out: hw/i386/intel_iommu.c
  - Intel-VT reference: https://software.intel.com/sites/default/files/managed/c5/15/vt-directed-io-spec.pdf
  - VMware reference: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/VMware_paravirtualization.pdf

Internals: Virtualization

- How do hardware constructs work?
  - I/O?
  - Events?
  - Interrupts?
  - **Memory?** DMA accesses?
- Two routes: **emulating** via software (slow!), **expose** hardware to the VM (needs hardware support!)
  - See “A Comparison of Software and Hardware Techniques for x86 Virtualization”, Adams and Agesen, ASPLOS 2006

Virtualization: I/O

- Hardware: Direct pass-through
  - What happens when guest has more physical I/O than Host?
- **Software emulation**
  - Scope for more flexibility and features - virtual NICs, and more.
  - **QEMU** emulates hardware resources (hard disk, video, USB, etc.)
- Direct Memory Access has dedicated DMA controller.
  - **Intel VT-x**: IOMMU - address translation support for DMA

Virtualization: Events and Interrupts

- Syscalls
- Trap-and-emulate
  - Any privileged instruction in guest OS happens, hypervisor “traps” it, then emulates its operation on the host OS.
    - Unvirtualizable events
      - popf does not trap when it cannot modify system flags.
- **Intel VT-x**: Hypervisor runs at ring 1, VMs run at ring 3, hypervisor provides hardware virtualization, **KVM** takes advantage of this.
  - KVM is a kernel module, runs in privileged mode.
  - **QEMU talks to KVM** for emulating the CPU and its instructions.
    - If no KVM, then QEMU is forced to emulate everything in software (using **dynamic binary translation**).

Virtualization: **Memory**

- Software: shadow page table
- Hardware: **extended page table**

Virtualization: **Memory**

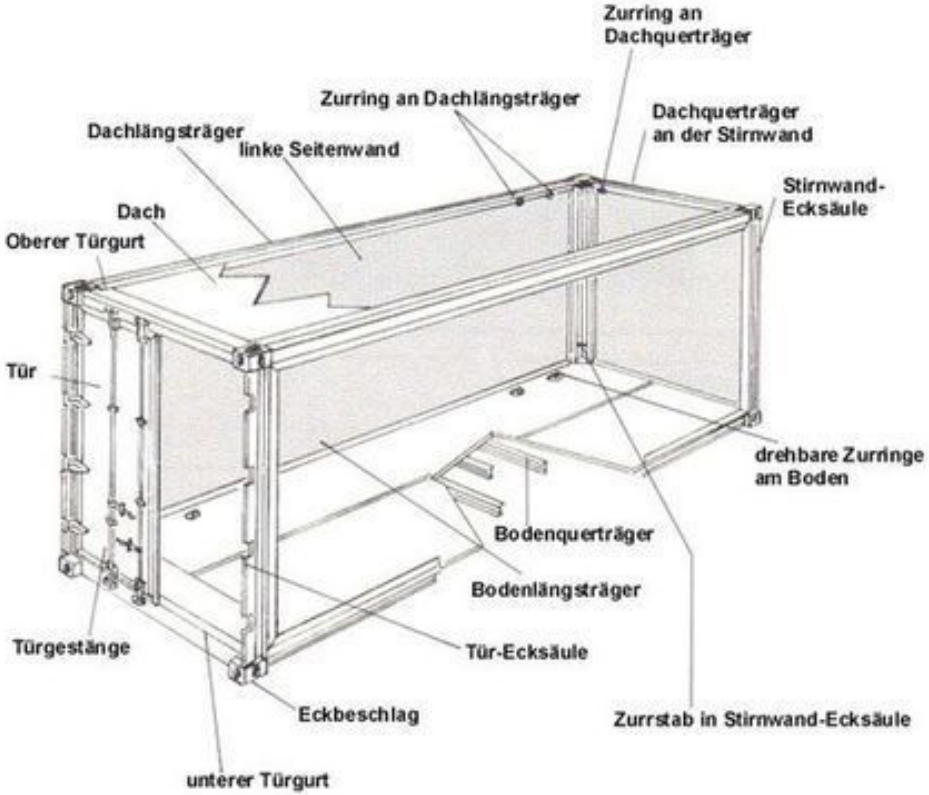
- Guest OS can’t have access to hardware page tables.
  - Hypervisor keeps the “real mappings” (guest virtual -> host physical).
  - **Shadow page table**.
    - Hypervisor can’t easily interpose on a TLB miss.
- Page faults?
  - If no VM: generates an exception, then redirected to handler.
  - If VM: needs to be forwarded to hypervisor.
- Expensive.

Virtualization: **Memory**

- Why not have the Guest physical memory -> Host physical memory mapping in hardware?
  - Intel VT-x supports **extended page tables** now.

Cool tricks

- **Time dilation!**
  - VMM can control rate of timer interrupts to Guest OS.
  - Can change how OS interprets passage of time.
  - If VMM slows timer by 10x, then other hardware (CPU, disk, network) appears 10x faster to OS and applications.
  - Applications run much slower.

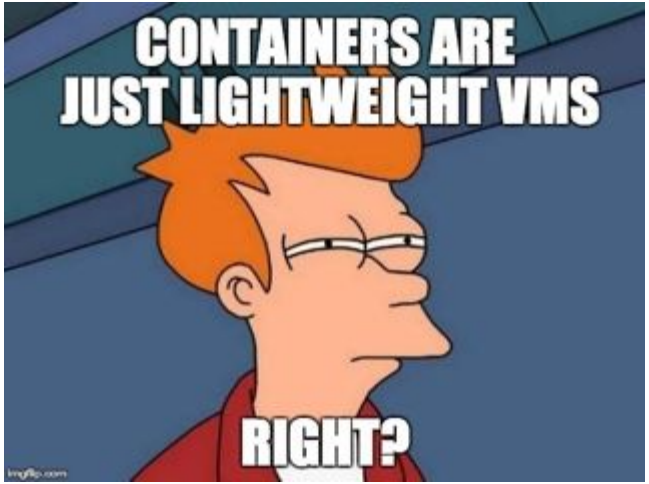

Zurring an Dachquerträger
Zurring an Dachlängsträger
Dachquerträger an der Stirnwand
Dachlängsträger
linke Seitenwand
Stirnwand-Ecksäule
Dach
Oberer Türgurt
Tür
drehbare Zurringe am Boden
Bodenquerträger
Bodenlängsträger
Türgestänge
Tür-Ecksäule
Eckbeschlag
Zurrstab in Stirnwand-Ecksäule
unterer Türgurt

Containers

They **feel** like a VM

- Own process space
- Can stuff as root
- Can install packages
- Can run services
- Own networking space
- Can mess up routing, iptables

You container seem to know nothing about your host.



Containers are not VM at all because...

- It uses the **hosts kernel**
- You can’t run a kernel other than the same host.
- You can’t have your own kernel modules.
- No hardware virtualisation.
- It’s too fast.
- Host can see what is running in side the containers.

You container will have some process running and those process are simple processes in your host.

So what are containers exactly?

Being like a VM, but not a VM? What?

Containers are NOT REAL..

- It’s just a bunch of Linux features/primitives tied together beautifully.
- You can see containers as **a duct tape**.
- It is made up of namespaces, cgroups, seccomp, etc!

namespaces

It handles what can a process ***see***.

namespaces

The different kinds of namespaces:

- pid
- net
- mnt
- uts
- ipc
- user

Each process is a part all the different kinds of namespaces mentioned above.

pid namespace

Processes in PID namespace **‘x’** can only see processes in PID namespace **‘x’**(there is catch to that)

pid namespace

- Each pid namespace has its own PID numbering starting at 1.
- If PID 1 goes away, whole namespace is killed
- Bonus: /proc/sys/kernel/ns_last_pi

pid namespace

- A process can end up having multiple PIDs in each namespace it belongs to.

pid namespace

- CLONE_NEWPID can be used while cloning to get a new namespace.
- Will give PID 1 to the children process and PID </proc/sys/kernel/ns_last_pi+1> of the children to the parent process.
- unshare can be used too

net namespace

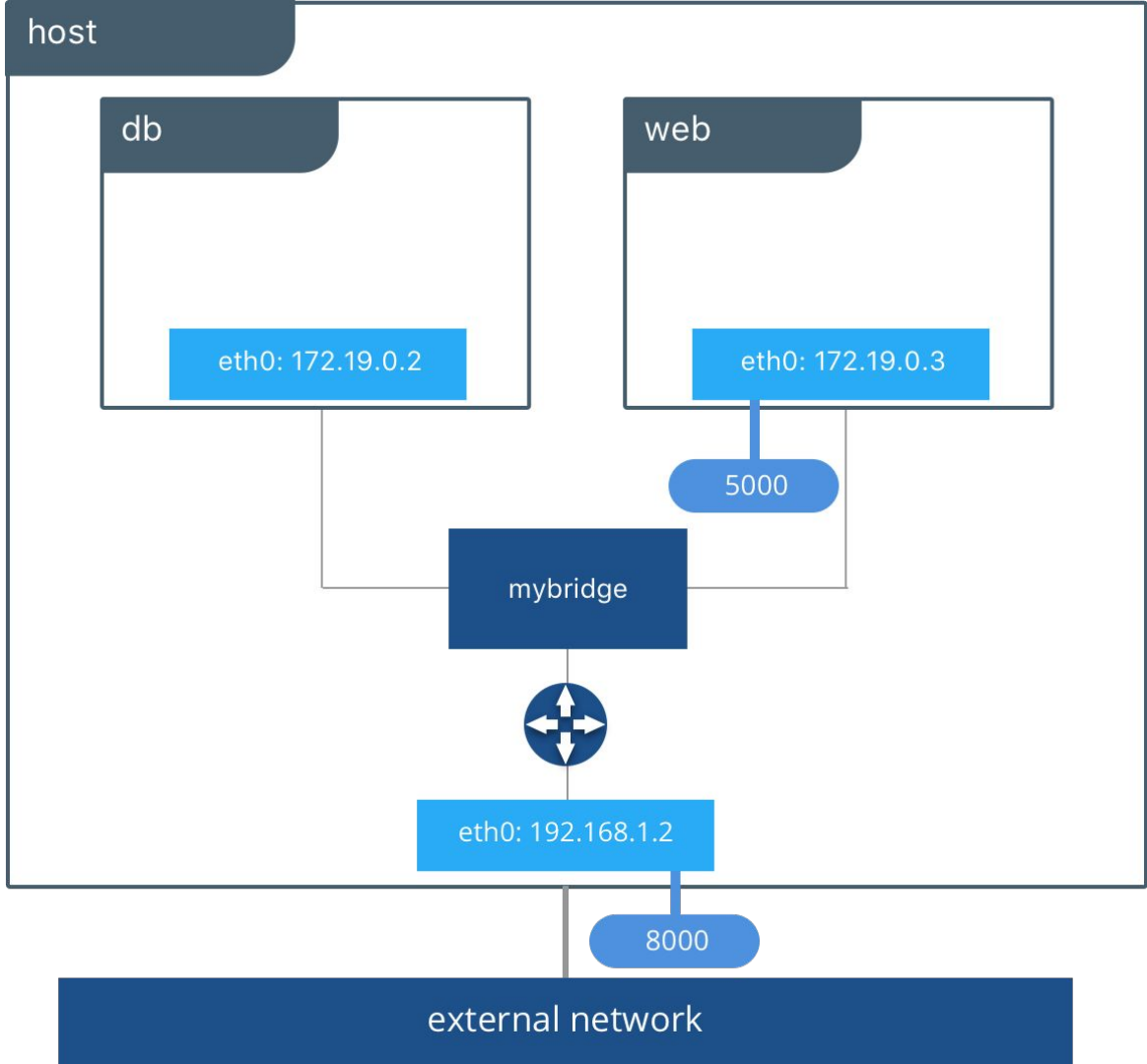
network namespace consists of

- network devices
- IP addresses
- IP routing tables
- port number
- whatever /proc/net directory can imply

net namespace

- Use CLONE_NEWNET during clone
- Or use ip netns add <ns_name>
- ip netns exec <ns_name> <the command>

Default Docker Networking



UTS Namespace

- isolates two system identifiers
  - nodename
  - domainname
- Returned by the uname() system call
- Set using the sethostname() and setdomainname() system calls
- Uses **CLONE_NEWUTS**

Other namespaces

- mnt
- ipc
- user

cgroups

It monitors and controls what “physical resources” can a process use.

And we all really need it.

Jargon: control group subsystems

Each subsystem represents some ***one*** kind of ***resources*** like a processor time, memory space or number of pids or in other words number of processes ***for a control group***. It applies per-cgroup limits on resource.

```
sid ) cat /proc/cgroups
#subsys_name    hierarchy       num_cgroups     enabled
cpuset  3       3       1
cpu     5       140     1
cpuacct 5       140     1
blkio   7       139     1
memory  8       364     1
devices 11      140     1
freezer 12      4       1
net_cls 10      3       1
perf_event      9       3       1
net_prio        10      3       1
hugetlb 6       3       1
pids    2       148     1
rdma    4       1       1
```

Jargon: control group subsystems

Each subsystem represents some ***one*** kind of ***resources*** like a processor time, memory space or number of pids or in other words number of processes ***for a control group***. It applies per-cgroup limits on resource.

```
sid ) ls -l /sys/fs/cgroup/
total 0
dr-xr-xr-x 5 root root  0 Feb  7 14:16 blkio/
lrwxrwxrwx 1 root root 11 Feb  6 18:51 cpu -> cpu,cpuacct/
lrwxrwxrwx 1 root root 11 Feb  6 18:51 cpuacct -> cpu,cpuacct/
dr-xr-xr-x 5 root root  0 Feb  7 14:16 cpu,cpuacct/
dr-xr-xr-x 3 root root  0 Feb  7 14:16 cpuset/
dr-xr-xr-x 6 root root  0 Feb  7 14:16 devices/
dr-xr-xr-x 4 root root  0 Feb  7 14:16 freezer/
dr-xr-xr-x 3 root root  0 Feb  7 14:16 hugetlb/
dr-xr-xr-x 5 root root  0 Feb  7 14:16 memory/
lrwxrwxrwx 1 root root 16 Feb  6 18:51 net_cls -> net_cls,net_prio/
dr-xr-xr-x 3 root root  0 Feb  7 14:16 net_cls,net_prio/
lrwxrwxrwx 1 root root 16 Feb  6 18:51 net_prio -> net_cls,net_prio/
dr-xr-xr-x 3 root root  0 Feb  7 14:16 perf_event/
dr-xr-xr-x 5 root root  0 Feb  7 14:16 pids/
dr-xr-xr-x 2 root root  0 Feb  7 14:16 rdma/
dr-xr-xr-x 6 root root  0 Feb  7 14:16 systemd/
dr-xr-xr-x 5 root root  0 Feb  7 14:16 unified/
```

cgroup: random points

- Each subsystem has its own hierarchy(tree)
- Hierarchies are independent, example memory and cpu are different and independent tree
- Each process is in only one node in each hierarchy
- Each hierarchy starts with 1 node (the root)
- Each node implies a group of processes sharing the same resources

demo

seccomp

Secure Computing
or
syscall filters