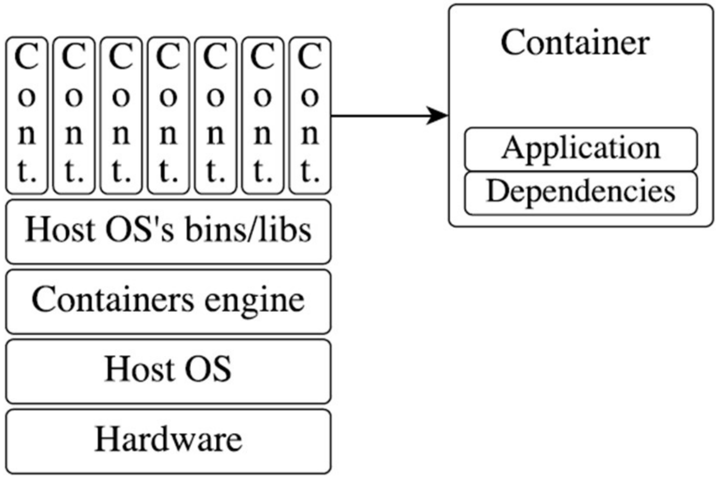
DSG-SOA-M 2024:
- Linux Containers -

Dr. Andreas Schönberger

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

What are Linux Containers?

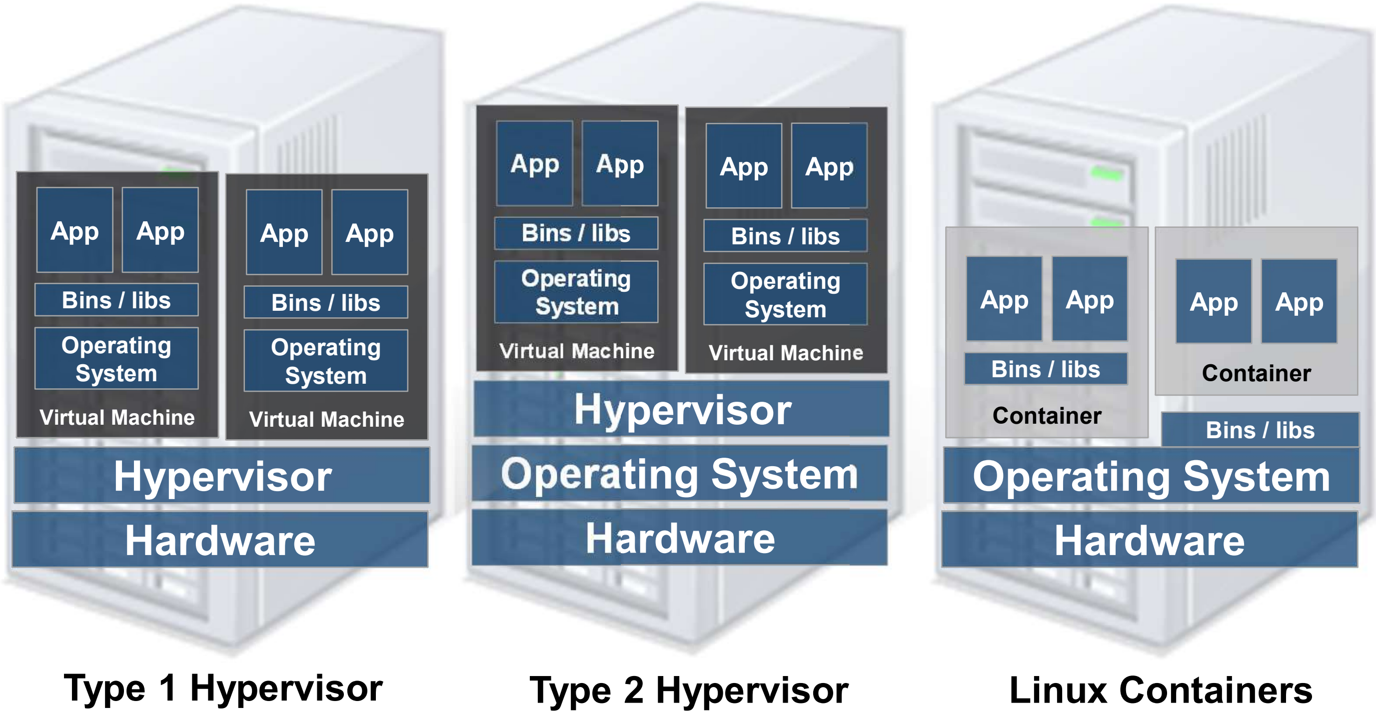
- Lightweight alternative to hypervisor-based virtualization
  - Higher density of virtualized instances
  - Efficient use of scarce embedded resources
- Containers implement isolation of processes at the OS level
- Run on top of the same shared OS kernel of the underlying host machine



Src.: Roberto Morabito
Ericsson Research

Hypervisors vs. Linux Containers

Containers share the OS kernel of the host and thus are lightweight. However, each container must have the same OS kernel.



Containers are isolated, but share OS and, where appropriate, libs / bins.

Src.:Boden Russell, IBM

Where Do We Come from?

- Unix chroot (1979)
- FreeBSD Jails (2000)
- Linux VServer (2001)
- Solaris zones (2004)
- OpenVZ (2005)
- Linux Containers – LXC (2009)
- Docker (2013)
- Podman (2018)

Note: LXC as a technology != LXC tools

Why Linux Containers?

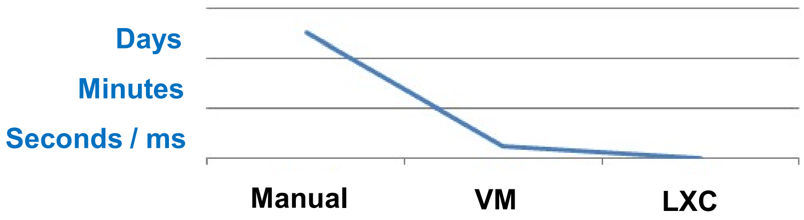
Run many applications in a fast, portable and isolated way in many different environments!

Containers are like light-weight VMs.
Small footprint, fast boot, shared kernel.

Why Linux Containers – in more detail

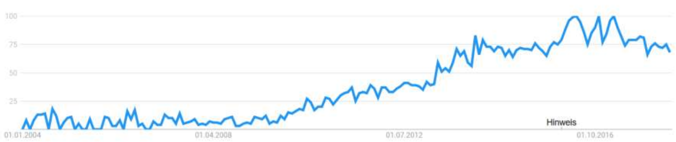
- Provision in seconds / milliseconds
- Near bare metal runtime performance
- VM-like agility – it's still "virtualization"
- Flexibility
  - Containerize a "system"
  - Containerize "application(s)"
- Lightweight
  - Just enough Operating System
  - Minimal per container penalty
- Open source – free – lower TCO
- Supported with OOTB modern Linux kernel

Provision Time

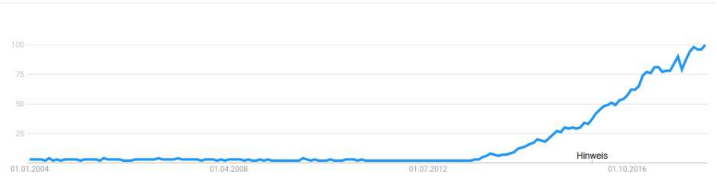


Src.:Boden Russell, IBM

Google trends - *lxc*

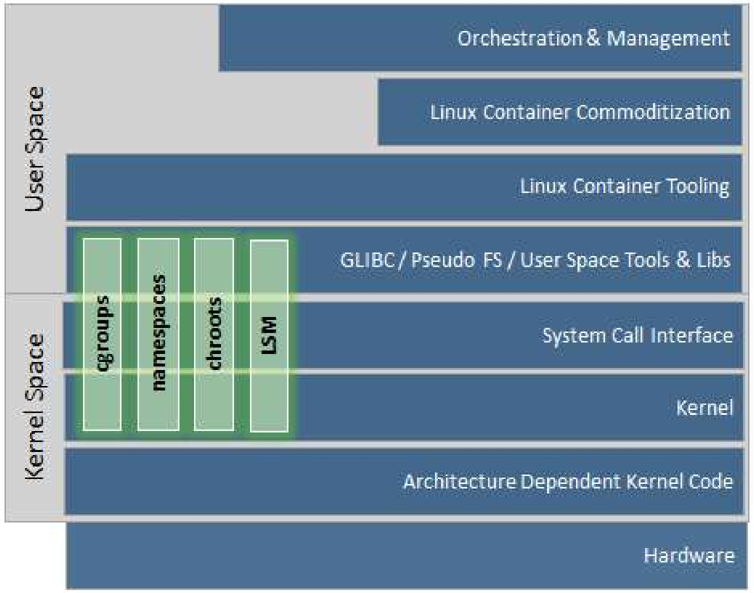


Google trends - *docker*



Ok, What is the Stack Beneath?

- LXCs are built on modern kernel features
  - cgroups; limits, prioritization, accounting & control
  - namespaces; process based resource isolation
  - chroot; apparent root FS directory
  - Linux Security Modules (LSM); Mandatory Access Control (MAC)
- User space interfaces for kernel functions
- LXC tools
  - Tools to isolate process(es) virtualizing kernel resources
- LXC commoditization
  - Dead easy LXC
  - LXC virtualization
- Orchestration & management
  - Scheduling across multiple hosts
  - Monitoring
  - Uptime



Src.:Boden Russell, IBM

Linux cgroups – Core Tool for Resource Isolation

- Functionality
  - Access; which devices can be used per cgroup
  - Resource limiting; memory, CPU, device accessibility, block I/O, etc.
  - Prioritization; who gets more of the CPU, memory, etc.
  - Accounting; resource usage per cgroup
  - Control; freezing & check pointing
  - Injection; packet tagging
- Usage
  - cgroup functionality exposed as “resource controllers” (aka “subsystems”)
  - Subsystems mounted on FS
  - Top-level subsystem mount is the root cgroup; all procs on host
  - Directories under top-level mounts created per cgroup
  - Procs put in `tasks` file for group assignment
  - Interface via read / write pseudo files in group

Src.:Boden Russell, IBM

Linux cgroups Subsystems

- cgroups provided via kernel modules (should be 3.10 or higher)
  - Not always loaded / provided by default
  - Locate and load with `modprobe`
- See: https://www.kernel.org/doc/Documentation/cgroup-v1/

Subsystem	Tunable Parameters
blkio	- Weighted proportional block I/O access. Group wide or per device. - Per device hard limits on block I/O read/write specified as bytes per second or IOPS per second.
cpu	- Time period (microseconds per second) a group should have CPU access. - Group wide upper limit on CPU time per second. - Weighted proportional value of relative CPU time for a group.
cpuset	- CPUs (cores) the group can access. - Memory nodes the group can access and migrate ability. - Memory hardwall, pressure, spread, etc.
devices	- Define which devices and access type a group can use.
freezer	- Suspend/resume group tasks.
memory	- Max memory limits for the group (in bytes). - Memory swappiness, OOM control, hierarchy, etc..
hugetlb	- Limit HugeTLB size usage. - Per cgroup HugeTLB metrics.
net_cls	- Tag network packets with a class ID. - Use tc to prioritize tagged packets.
net_prio	- Weighted proportional priority on egress traffic (per interface).

How to Use cgroups - Linux cgroups Pseudo FS Interface

- Linux pseudo FS is the interface to cgroups
  - Read / write to pseudo file(s) in your cgroup directory
- Some libs exist to interface with pseudo FS programmatically

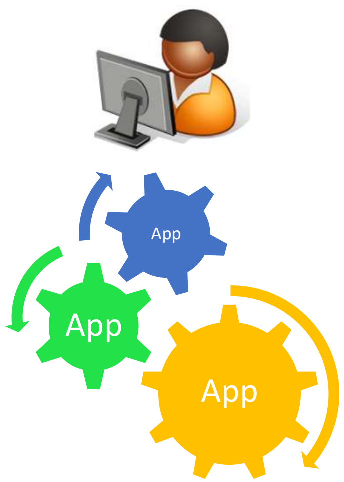
Find details on kernel.org

```
/sys/fs/cgroup/my-lxc

|-- blkio
|    |-- blkio.io_merged
|    |-- blkio.io_queued
|    |-- blkio.io_service_bytes
|    |-- blkio.io_serviced
|    |-- blkio.io_service_time
|    |-- blkio.io_wait_time
|    |-- blkio.reset_stats
|    |-- blkio.sectors
|    |-- blkio.throttle.io_service_bytes
|    |-- blkio.throttle.io_serviced
|    |-- blkio.throttle.read_bps_device
|    |-- blkio.throttle.read_iops_device
|    |-- blkio.throttle.write_bps_device
|    |-- blkio.throttle.write_iops_device
|    |-- blkio.time
|    |-- blkio.weight
|    |-- blkio.weight_device
|    |-- cgroup.clone_children
|    |-- cgroup.event_control
|    |-- cgroup.procs
|    |-- notify_on_release
|    |-- release_agent
|    `-- tasks
|-- cpu
|    |-- ...
|-- ...
`-- perf_event
```

```
echo "8:16 1048576“ >
blkio.throttle.read_bps_de
vice
```

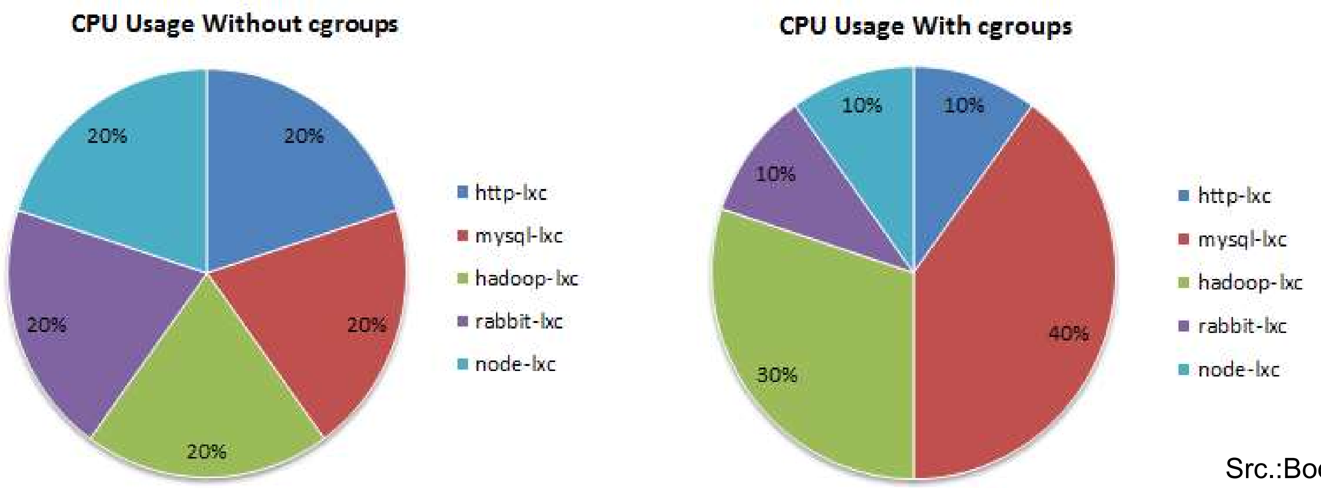
```
cat blkio.weight_device
dev     weight
8:1     200
8:16    500
```



Src.:Boden Russell, IBM

Linux cgroups: CPU Usage

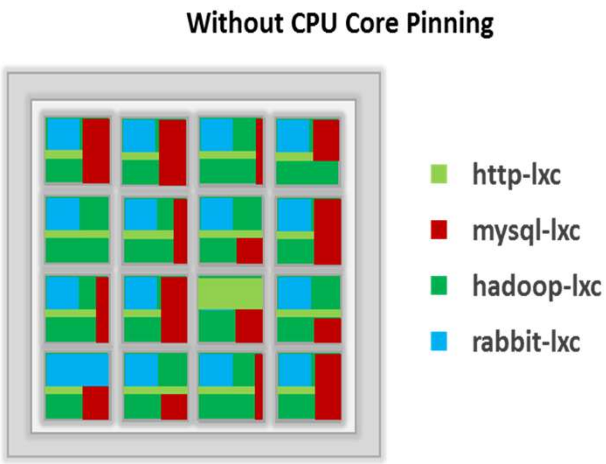
- Use CPU shares (and other controls) to prioritize jobs / containers
- Carry out complex scheduling schemes
- Segment host resources
- Adhere to SLAs

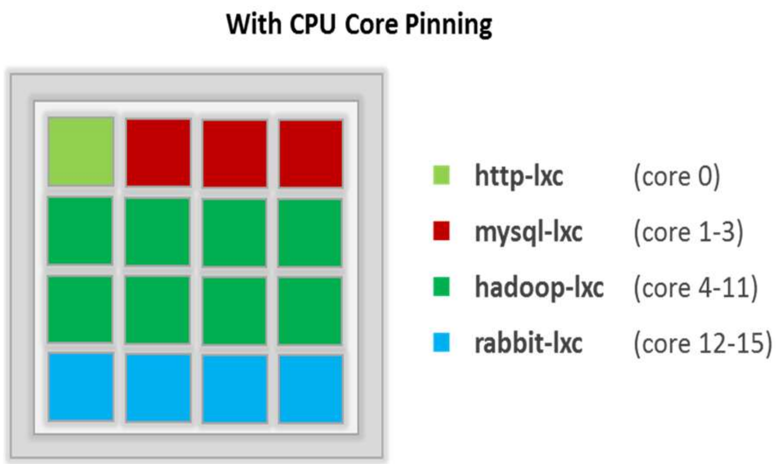


Src.:Boden Russell, IBM

Linux cgroups: CPU Pinning

- Pin containers / jobs to CPU cores
- Carry out complex scheduling schemes
- Reduce core switching costs
- Adhere to SLAs

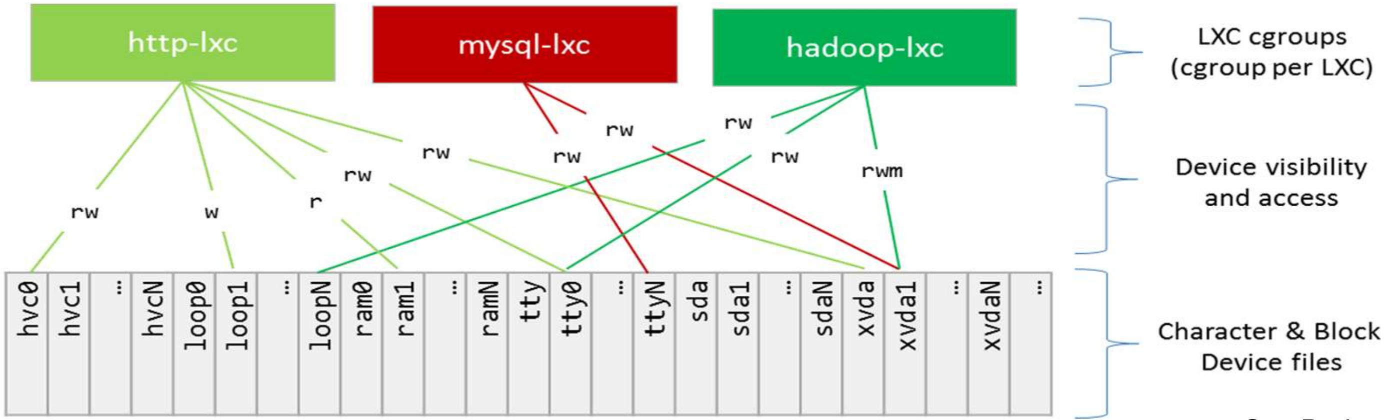




Src.:Boden Russell, IBM

Linux cgroups: Device Access

- Limit device visibility; isolation
- Implement device access controls
  - Secure sharing
- Segment device access
- Device whitelist / blacklist



Src.:Boden Russell, IBM

Linux Namespaces

- Why namespaces?
  Namespaces provide isolation of Linux resources for a group of processes.

Src.:Boden Russell, IBM

- Functionality
  - Provide process level isolation of global resources
    - **MNT** (mount points, file systems, etc.)
    - **PID** (process)
    - **NET** (NICs, routing, etc.)
    - **IPC** (System V IPC resources)
    - **UTS** (host & domain name)
    - **USER** (UID + GID)
  - Process(es) in namespace have illusion they are the only processes on the system
  - Generally constructs exist to permit “connectivity” with parent namespace
- Usage
  - Construct namespace(s) of desired type
  - Create process(es) in namespace (typically done when creating namespace)
  - If necessary, initialize “connectivity” to parent namespace
  - Process(es) in name space internally function as if they are only proc(s) on system

Typical Container Engine / Tooling Functionality

- Standard container operations to realize / manage LXCs
  - Run, start, stop, delete, attach, snapshot, etc.
- APIs
  - CLI, Web API (RESTful or other), Automation files (e.g. dockerfile)
- Images
  - Format (typically RAW), required dev files, etc..
  - Image repository or catalog
  - Container FS layers (union FS style)
  - Import / export images and containers
- Metrics
  - CPU, memory, block IO, etc..
- Eventing
  - Push events on container changes
- Logs
  - Inspect / manage container logs
- Checkpointing
  - Freeze a container's state

Src.:Boden Russell, IBM

What are / were the Liabilities?

- Live migration still an excitement
- Full orchestration across resources (compute / storage / networking)
- Fears of security
- Not a well known technology… but maturing very fast
- Integration with existing virtualization and Cloud tooling
- Not much / any industry standards
- Missing skillset
- Slower upstream support due to kernel dev process

Src.:Boden Russell, IBM