Methods of Cloud Computing

Chapter 2: Virtual Resources

Complex and Distributed Systems
Faculty IV
Technische Universität Berlin

Operating Systems and Middleware
Hasso-Plattner-Institut
Universität Potsdam

Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- **OS-Level Virtualization**
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

OS Virtualization

- Also known as Container Virtualization
- Motivation: HW virtualization comes with too much overhead, large images, and long boot times
- Idea: do not virtualize entire machine, but…
  - Reuse operating system kernel and isolate applications
  - Virtualize access to resources used by processes
    - File system
    - Devices
    - Network
    - Other processes
    - …

History of OS Virtualization

- Unix v7 **`chroot`** system call (1979)
  - Allows setting the file system root for processes
  - Unix philosophy: almost everything is accessed through the file system → almost everything can be virtualized though **`chroot`**
- Wave of container technologies in early 2000s
  - FreeBSD Jails, Linux VServer, Solaris Zones, …
  - Different degrees of isolation, different tool chains
  - Mostly specific to Linux distribution
  - Never popular with big masses, mainly used by system admins and large companies

History of OS Virtualization

- LXC (Linux Containers, 2008)
  - Easy-to-use user space tools for accessing Linux kernel process isolation features (cgroups, namespaces)
- Docker (2013)
  - Became the next big thing after the initial cloud hype
  - Most wide spread container technology and eco system
- Rocket (CoreOS, 2014)
  - Alternative to Docker
  - More focused on security and standardization
- LXD (Canonical)
  - Focused on entire Linux distributions (not applications)

Overview

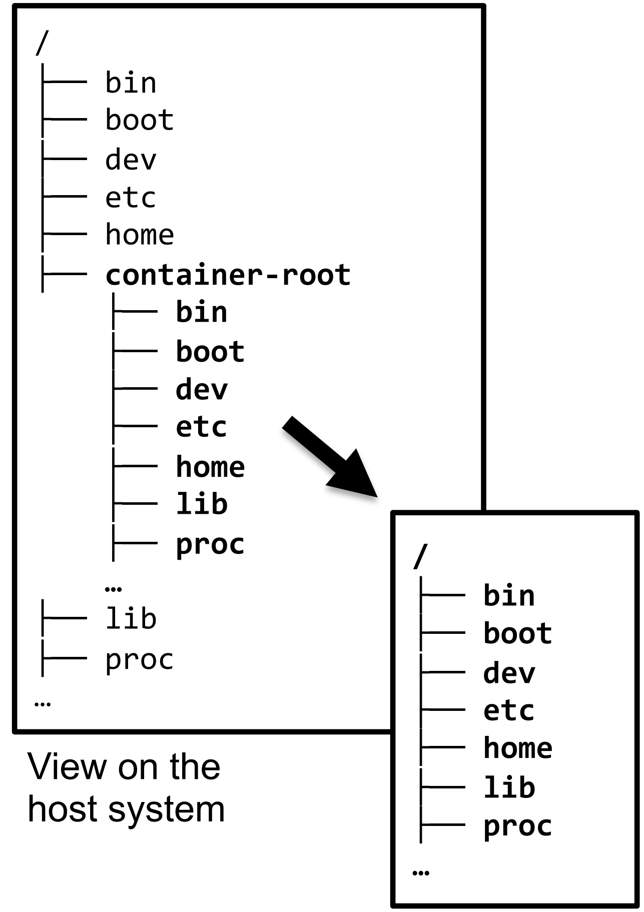
- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - **Linux Containerization**
  - LXC Containers and Docker
  - Comparison to Virtual Machines

Linux Kernel Features for OS Virtualization

- Linux kernel contains large number of mechanisms for process isolation:
  - `chroot` system call
  - Namespaces
  - Capabilities
  - cgroups
  - SELinux
  - `seccomp`
  - …
- Partially overlapping functionality, but different configuration approaches

chroot

- Chroot
  - Oldest mechanism for process isolation
  - System call, changes the root directory (“/”) of the calling process
  - Since “everything is a file” in Linux, many aspects of the underlying system can be virtualized
  - System calls, networking, etc. remain unchanged



View on the host system

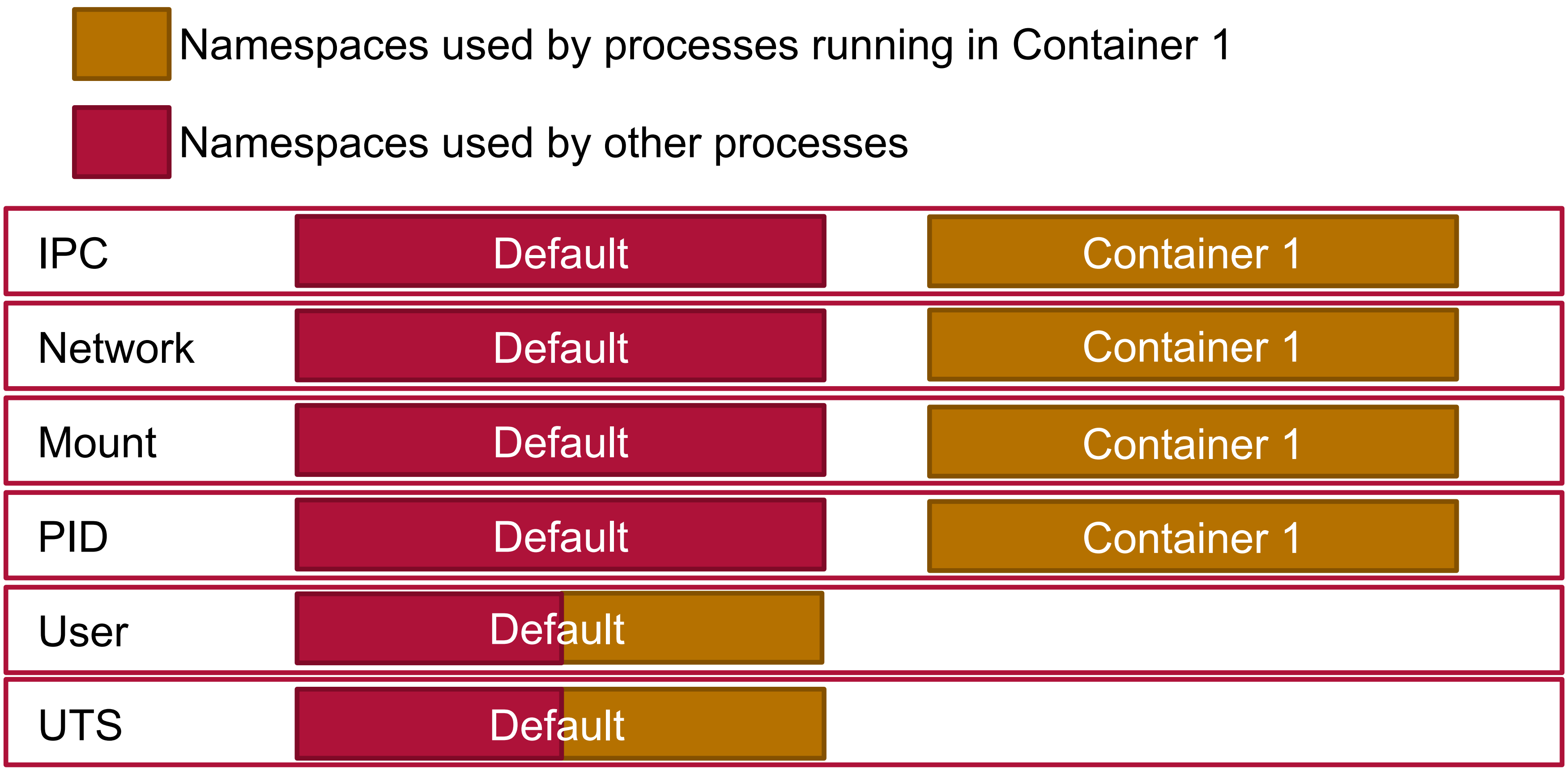
View in the chrooted process

Linux Namespaces

- Namespaces
  - Separate views on kernel resources for processes in different namespaces
  - Isolated resources:
    - IPC (Inter-process communication)
    - Network (devices, protocol stacks, firewalls, ports, …)
    - Mount (mount points, file system structure)
    - PID (processes)
    - User (users and groups)
    - UTS (hostname and domain name)

Linux Namespaces

- Container 1: Share users and hostname, but isolate processes, network and mount points

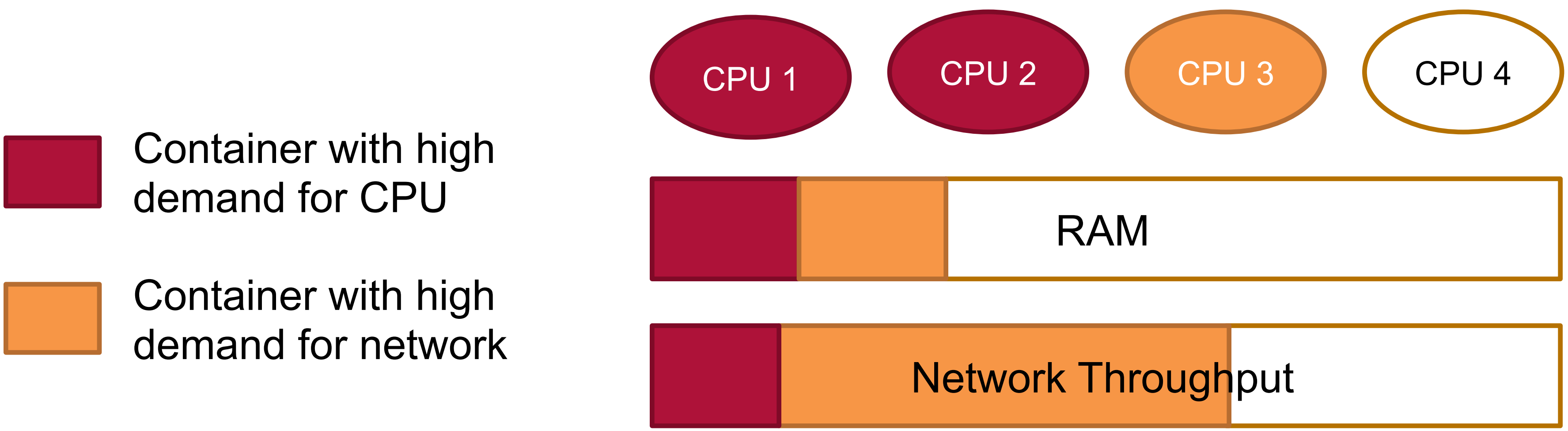


Linux Capabilities

- Capabilities
  - Traditionally, super user (root) is the only one to perform administrative tasks on a Linux system
  - New processes can be assigned selected capabilities
  - Long list of possible capabilities, including many system calls, device access, file system modifications, …

Linux cgroups

- cgroups (control groups)
  - Allows definition of resource usage constraints for parts of the process tree
  - Used for limiting CPU, RAM, network, and disk usage for containers



Security Policies

- SELinux, AppArmor
  - Optional security kernel modules
  - Allow rule-based confinement of processes to limit their access to certain files and devices only
  - SELinux: complex and powerful rule definitions
  - AppArmor: more straight-forward security profiles
- Seccomp policies
  - `seccomp` system call puts the process in a secure computation mode, where only selected system calls are allowed

Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - **LXC Containers and Docker**
  - Comparison to Virtual Machines

Examples for Container Technologies

- LXC: First Container technology widely adopted by end users

- Docker: Currently dominant container platform and eco system

Linux Containers

- LXC: Library and userspace tools (bash scripts) to access kernel process isolation features
- No daemon, containers and their file systems are stored in `/var/lib/lxc`

Language Bindings
(Python, Lua, Go, Python, Ruby, …)

Userspace tools
`lxc-create, lxc-start, lxc-stop, …`

`liblxc`

Kernel features:
Namespaces, capabilities, chroot, cgroups,
AppArmor, SELinux, seccomp

LXC Container Creation

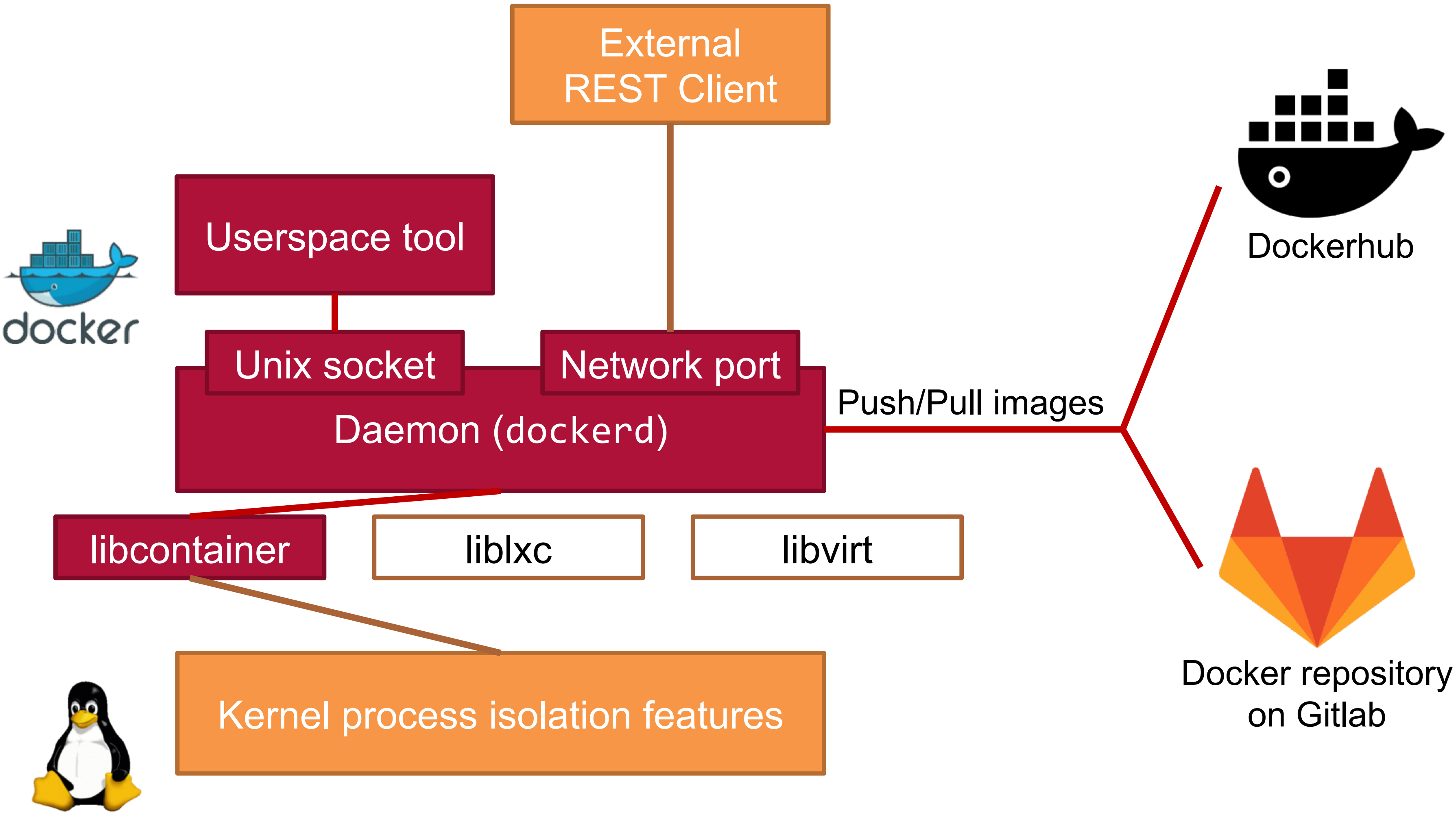
- Container Creation
  1. Container directory allocated under `/var/lib/lxc`
  2. “Template script” executed to populate the file system of the container
  3. Process is spawned and `chrooted` into its file system
  4. Isolation properties are configured
  5. Main application process is spawned, all child processes will inherit isolation context
- LXC does not rely on an image format
- No public repository for sharing images (introduced later in LXD)

Docker

- Most popular container technology at the moment

- Docker includes:
  - Daemon and user space tools for managing local containers and images
  - Hierarchical image format
  - Public and private repositories for sharing images
  - Included and external capabilities for automatic orchestration of container infrastructures

Docker Components

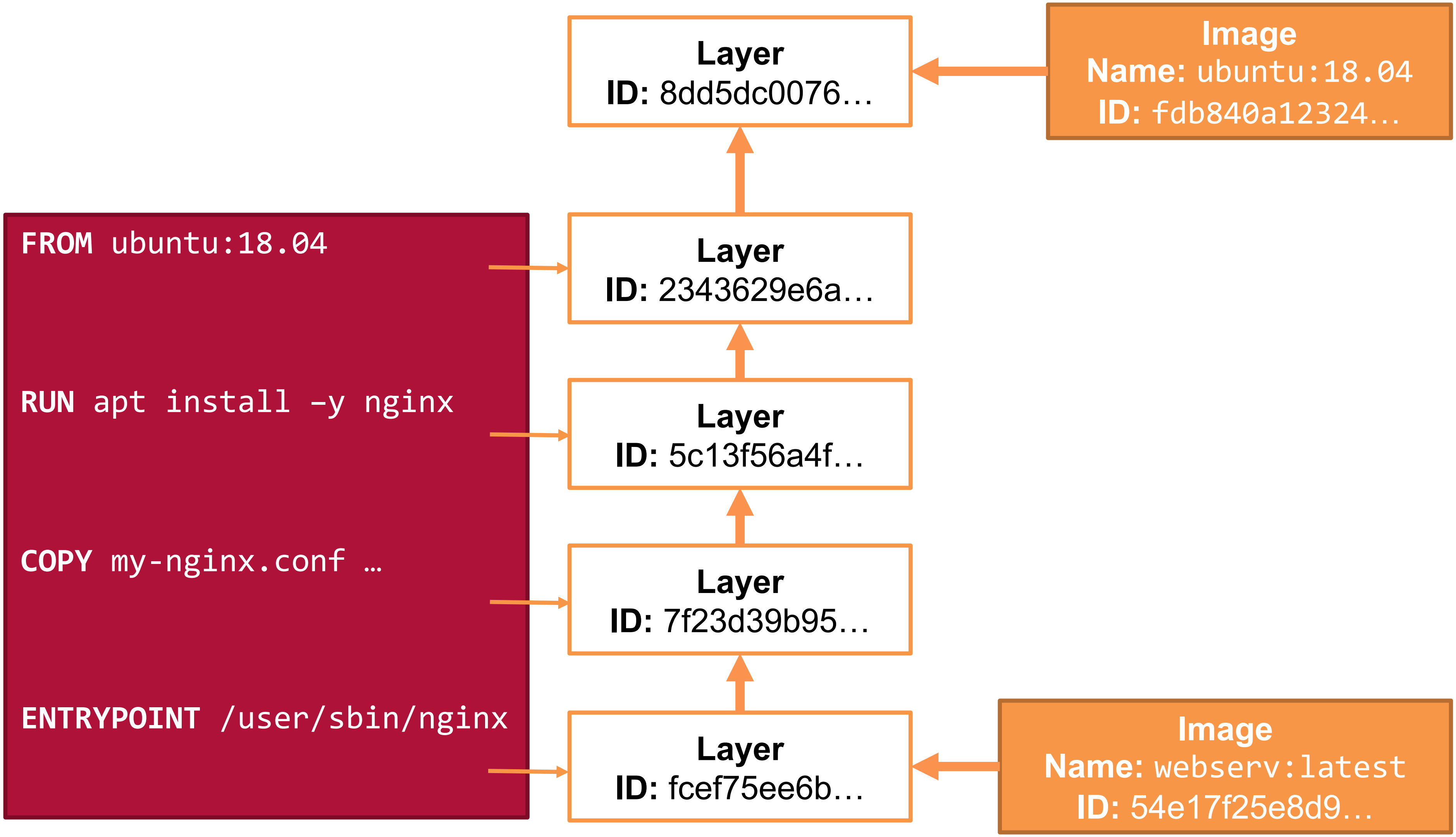


Dockerfiles

- Automatic way to create container images
- Text file with commands that modify files or change configuration values
- Intermediate steps can be cached using hashes, resulting in reduced image build times

```
FROM ubuntu:18.04
RUN apt install -y nginx
COPY my-nginx.conf /etc/nginx/nginx.conf
ENTRYPOINT /user/sbin/nginx
```

Docker Image Format



Docker Image Format

- Docker image contains:
  - List of layers that form the file system
  - ID (Hash of layer IDs and other config data)
  - Configuration data: ports, mounts, variables, …
  - Other meta info (creation time, author, history, …)
- Docker image **layer** contains:
  - Layer files (tarball): Files that were added/changed on this layer, relative to parent layer
  - ID of the layer (Hash of all files), ID of parent layer
  - Command that was used to create the layer

Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - **Comparison to Virtual Machines**

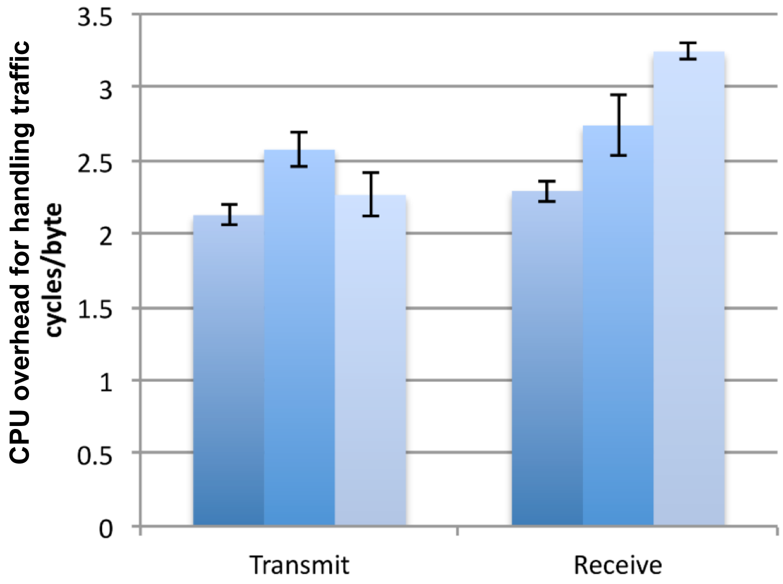
Containers vs VMs: Performance (CPU & RAM)

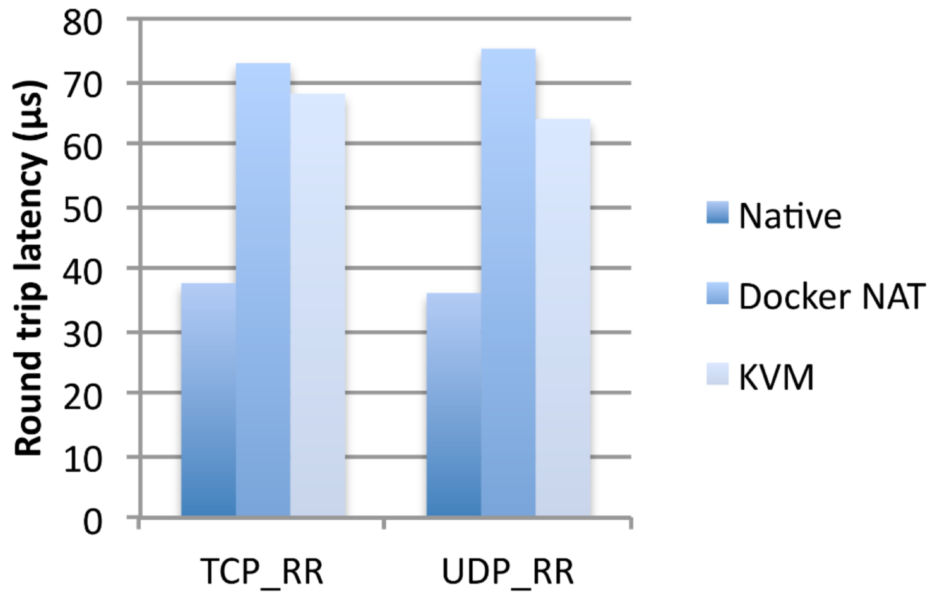
Workload	Docker	KVM
Linpack (GFLOPS)	290.9 [±0.98]	284.2 [±1.45]
Memory (Random Access, GIOps/s)	0.0124 [±0.00044]	0.0125 [±0.00032]
Memory (Sequential Access, GB/s)	45.6 [±0.55]	45.0 [±0.19]

- VMs introduce very low CPU and memory access overhead
  - Condition: exposing cache topology and CPU acceleration features (e.g. NUMA, FPUs, SSE)

Source: IBM Research Report (An Updated Performance Comparison of Virtual Machines and Linux Containers, 2014)
https://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195dd819c85257d2300681e7b/$file/rc25482.pdf

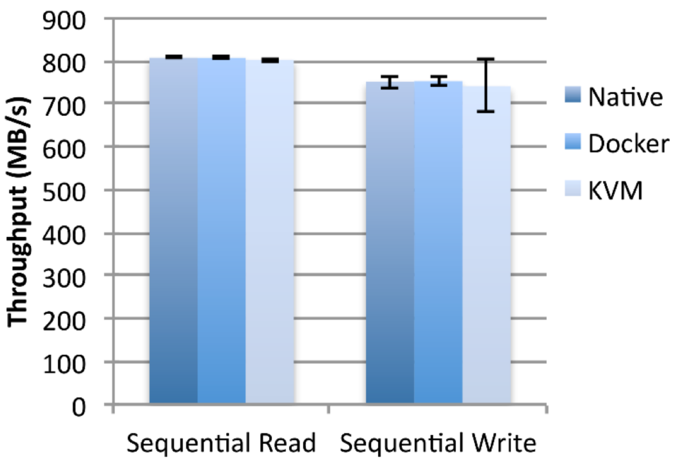
Containers vs VMs: Performance (Network)

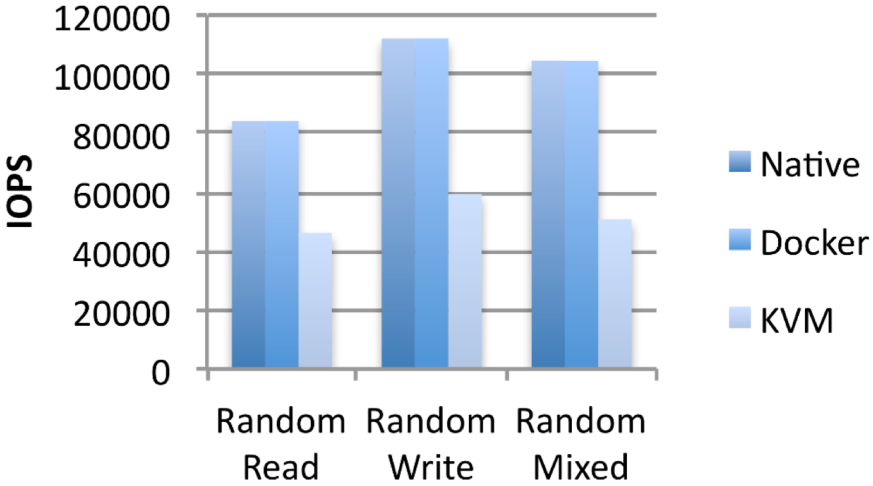




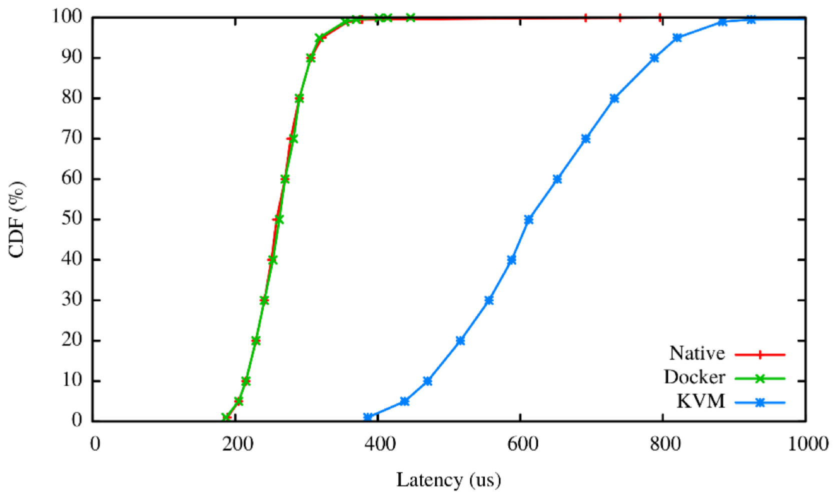
- Low CPU overhead per packet
- Reasons for latency increase:
  - NAT in Docker networking
  - Virtual network device in KVM (NAT might further increase latency)

Containers vs VMs: Performance (Disk IO)





- Similar throughput
- Penalty for latency and IOPs due to virtual IO device



Containers vs VMs: Image Size & Boot Time

- Virtual Machine images usually larger than container images (contain entire OS), but often image caching on execution host

- Boot time of VM can be orders of magnitude longer than container startup

Containers vs VMs: Isolation & Security

- Containers share the host OS kernel
  - Vulnerable to Linux kernel bugs
  - Denial-of-Service attacks: Resource usage, system calls, context switches of a container can starve others
  - Kernel parameters cannot be tuned to workload

- Virtualization technology older and VMM order of magnitude less code → more exploits discovered and fixed

Containers vs VMs: Summary

Virtual Machines

- Complete isolation
- Flexible OS
- Live migration

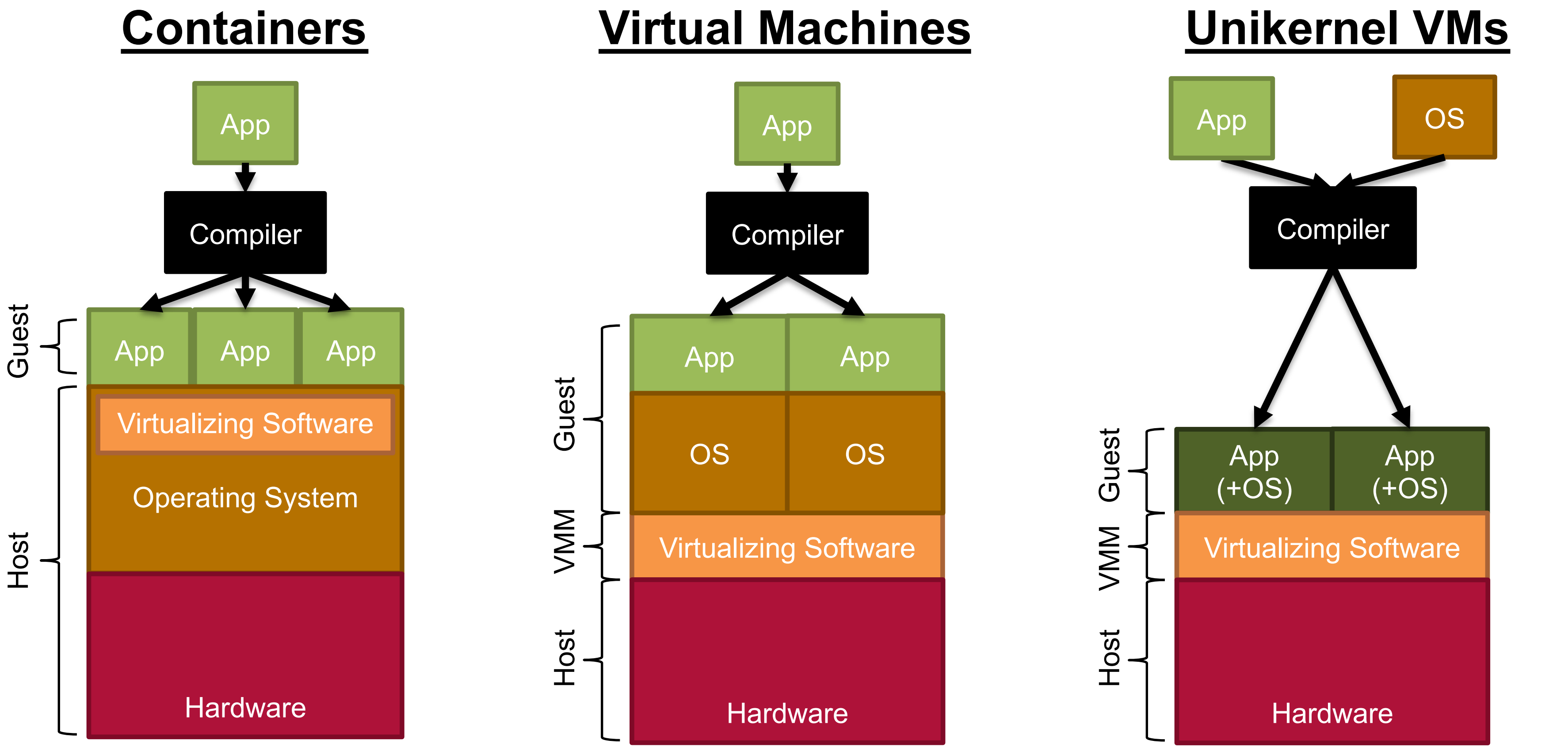
Containers

- Small images
- Quick startup
- Direct device access

→Tradeoff between performance and isolation & security

Sidenote: Unikernel VM Images

- Compile and link application code directly with all required OS functionality



Overview

- Virtual Resources and Infrastructure-as-a-Service
- Hardware Virtualization
  - Binary Translation, OS-Assisted Virtualization, Hardware-Assisted Virtualization
  - Virtual Machine Migration
  - Resource Isolation and Performance Implication
  - Case Study: Amazon EC2
- OS-Level Virtualization
  - Linux Containerization
  - LXC Containers and Docker
  - Comparison to Virtual Machines

Summary: Virtual Resources

- IaaS clouds let customers rent basic IT resources
  - Full control over OS and deployed applications in VMs
  - No long-term obligation or risk of over-/under-provisioning
- Virtualization as fundamental enabling technology
  - Several customers can share physical infrastructure
  - Different approaches to achieve virtualization
- Containers increasingly recognized as alternative or supplement to VMs

References

- Andrew S. Tanenbaum, Herbert Bos: Modern Operating Systems, Pearson, 2015
- William Stallings: Operating Systems – Internals and Design Principles, 2015

[3] G.J. Popek and R.P. Goldberg: "Formal Requirements for Virtualizable Third Generation Architectures", Communications of the ACM, 17 (7), 1974

[4] J.S. Robin, C.E. Irvine: "Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor", Proc. of the 9th Conference on USENIX Security Symposium, 2000

[6] K. Adams, O. Agesen: "A Comparison of Software and Hardware Techniques for x86 Virtualization", Proc. of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, 2006

[7] A. Whitaker , M. Shaw , S.D. Gribble: "Denali: Lightweight Virtual Machines for Distributed and Networked Applications", Proc. of the 2002 USENIX Annual Technical Conference, 2002

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield: "Xen and the Art of Virtualization", Proc. of the 19th ACM Symposium on Operating Systems principles, 2003

[11] O. Agesen: "Performance Aspects of x86 Virtualization", VMWORLD 2007

[13] G. Neiger, A. Santoni, F. Leung, D. Rodgers, R. Uhlig: "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization", Intel Technology Journal, 10 (3), 2006

[15] C. Clark, K. Fraser, S. Hand, J.G. Hanseny, E. July, C. Limpach, I. Pratt, A. Wareld: "Live Migration of Virtual Machines", Proc. of the 2nd conference on Symposium on Networked Systems Design & Implementation, 2005

[18] J. Schad, J. Dittrich, J.-A. Quiané-Ruiz: "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance", Proc. of the VLDB Endowment, 3 (1-2), 2010