LA-UR-19-23481
Approved for public release; distribution is unlimited.

Title: Charliecloud: Unprivileged Containers for High-Performance Computing

Author(s): Jennings, Michael E.

Intended for: Campus Research Computing Consortium (CaRCC) online meeting

Issued: 2019-04-18

Disclaimer:
Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Triad National Security, LLC for the National Nuclear Security Administration of U.S. Department of Energy under contract 89233218CNA000001. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

75 Los Alamos

NATIONAL LABORATORY

EST. 1943

Charliecloud

Unprivileged Containers for High-Performance Computing

Michael Jennings (@mej0) – *mej@lanl.gov*
Platforms Team Lead, HPC Systems Group
Los Alamos National Laboratory

Campus Research Computing Consortium
Systems-Facing Track
Zoom Online Meeting
Thursday, 18 April 2019

Operated by Triad National Security, LLC for the U.S. Department of Energy's NNSA

Los Alamos National Laboratory

Introduction

- Established in 1943 as “Site Y” of the Manhattan Project
- Mission: To solve National Security challenges through Scientific Excellence
- One of the largest science and technology institutes in the world, conducting multidisciplinary research in fields such as national security, space exploration, renewable energy, medicine, nanotechnology, and supercomputing.

- Funded primarily by the Department of Energy, we also do extensive work for/with the Departments of Defense and Homeland Security, the Intelligence Community, et al.
- Our strategy reflects US government priorities including nuclear security, intelligence, defense, emergency response, nonproliferation, counterterrorism, and more.
- We help to ensure the safety, security, and effectiveness of the US nuclear stockpile.
- Since 1992, the United States no longer performs full-scale testing of nuclear weapons. This has necessitated continuous, ongoing leadership in large-scale simulation capabilities realized through investment in high-performance computing.

LANL High-Performance Computing Division

Introduction

- **LANL's history in HPC dates back to the early '50s.**
- **Accomplishments include:**
  - Helped IBM develop Stretch, the 1st transistor-based supercomputer
  - The 1st vector computer, Cray-1, deployed here
  - Our CM-5 was #1 on the inaugural Top500 List
  - 1st hybrid supercomputer (using IBM POWER and PlayStation Cell processors), Roadrunner, was also 1st to break the PetaFLOP/s barrier
- **Led by Gary Grider, creator of Burst Buffer technology**

- We support over 2000 unique users across more than 100 different classified/open science projects on 20+ clusters

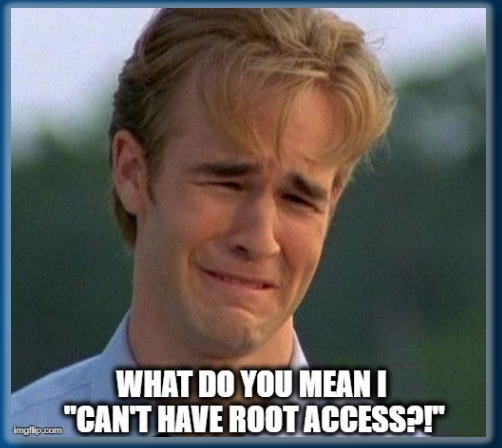
LANL has been a leader in HPC since before HPC was HPC!

First World Problems in HPC

Mo' Users, Mo' Problems!

Problem #1: HPC clusters have narrowly focused software stacks.

- They do serial and parallel-MPI tasks well...but that's it.
- Compute node images are often in RAM/NFS & kept small.
- Managing multiple OSs, and required expertise, is rare & labor intensive.



Problem #2: We're old. And not getting any younger.

- Schools are teaching clusters, "parallel"/scalable/distributed software development...but usually "embarrassingly parallel" (e.g., map/reduce).
- All the "cool kids" are using Ubuntu (or Arch, or Alpine...), not RHEL.
- Modern Machine Learning and Data Analytics toolsuites are non-trivial.

Problem #3: We are finite, as is our time.

- We generally won't install extra software with low user demand.
- Unique or unusual use cases tend to be lower priority.
- The line between "innovator" and "crackpot" is often rather blurry….
- Particularly in academic/research computing, teams are small with broad focus and must prioritize among many diverse problems/tasks.

Software build combinatorics is among the hardest problems in HPC management!

The Solution: Pawn It Off on the Users via UDSS/UDI/BYOE!

Make It Someone Else's!

User-Defined Software Stacks (UDSS), also known as **User-Defined Images (UDI)** or **Bring-Your-Own-Environment (BYOE)**, allow users to supply not only their own applications/source to run on HPC systems but also the environment – up to and including entire OS images – in which they should run!

Advantages include portability, usability, consistency, time savings…

Potential disadvantages include missing functionality (HSN, accelerators, filesystems), performance degradation; thus, addressing these should be part of the design of any HPC-focused solution!

In rare, specific cases, certain packages may address this independently by building static binaries or coupling dependencies with executables:

- Works everywhere with no privileges required
- Requires build-time (and sometimes run-time) support
- May not be feasible in all situations
- For everyone else, we have a few options…

This sounds counter to making our customers' lives easier, but it's not!

Option #1: Compile It Yourself

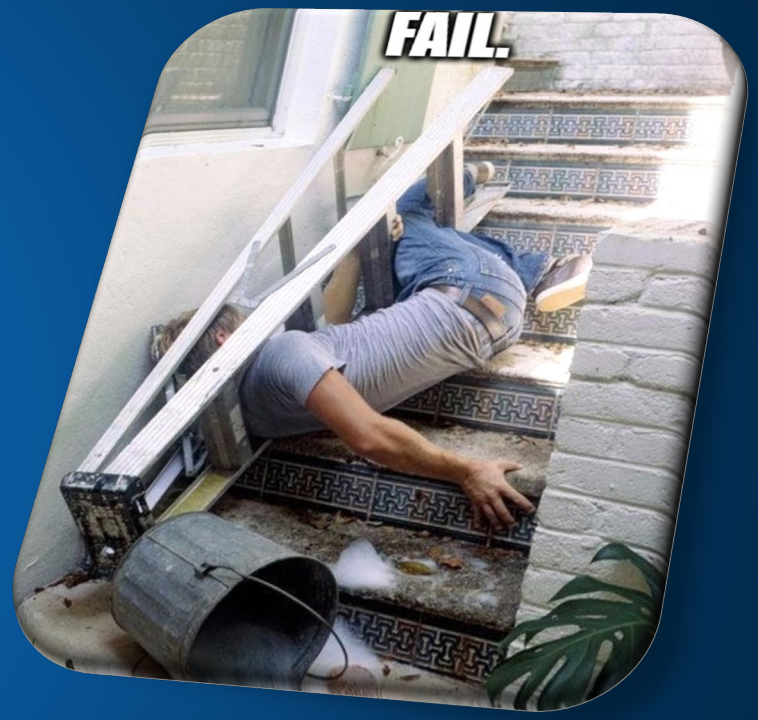
The Old-Fashioned Way

Advantages

- Available everywhere immediately
- Requires no privileges
- No additional privacy/security risk
- Direct access to all hardware
- No performance penalty
- Theoretically applies to any open source software library/app/stack

Disadvantages

- It's the 21st Century; we no longer *want* to party like it's 1999!
- Tedious and time-consuming
- Error-prone with exponential propagation of errors/time-to-fix
- Hard to update; start over each time
- No standard workflow/reproducibility
- Provides neither portability nor consistency



Nope.

Computers are awesome at repetitive, consistent tasks. People…not so much.

Option #2: Environment Manager

The Slightly Less Old-Fashioned Way

Advantages

- Available for most IA32/x64 systems
- Requires no privileges
- No additional privacy/security risk
- Direct access to all hardware
- No performance penalty

Disadvantages

- Frequently still requires building from source (time and space constraints)
- Varying degrees of HPC support
- Varying degrees of reproducibility, portability, consistency, workflow
- Users/consultants bear entire burden

If You Must…

- Good options include EasyBuild, Lmod, Spack
- Also Anaconda, nixOS

"There's GOT to be a BETTER WAY!!"

This is the current solution at most HPC & RC centers.

Option #3: Virtualization (VMs)

The Cloudy Way

Advantages

- Ultra-flexible (any kernel/OS/arch)
- Strong-to-complete isolation
- Common use cases perform well

Disadvantages

- Performance suffers for most HPC use cases, often significantly.
- Performance/security tradeoffs (i.e., paravirtualization sacrifices isolation/security to increase performance)
- Infrastructure can be complex.
- Direct/performant access to hardware may require privileges that are not easily granted at runtime or with minimal risk.
- Not all “exotic” HPC hardware supported.
- Entire OS must be provisioned and booted.
- Requires separate hostname/IP, separate network configuration, separate firewall, etc.
- User-supplied VMs often suffer from lack of CM/prescriptiveness.

Should HPC become the Cloud?

Someday HPC and Cloud may very well merge…but not today.

Option #4: Containers

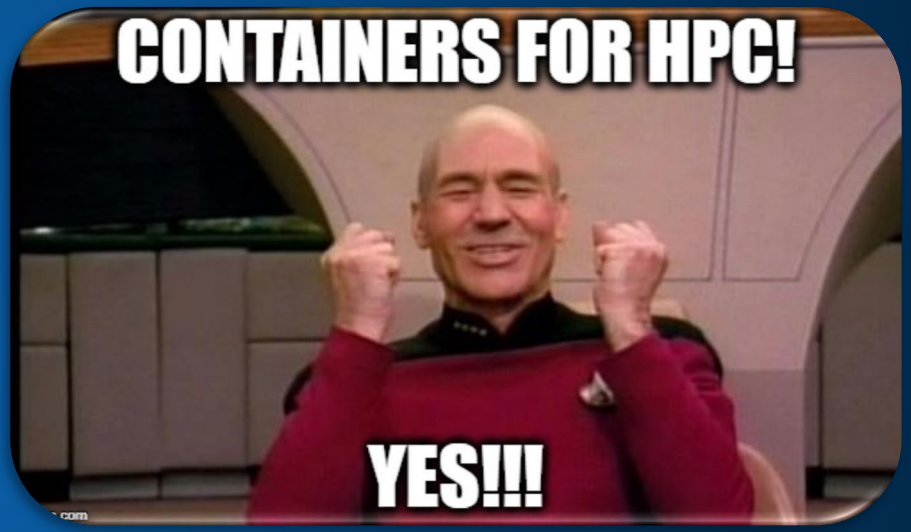
Containers for the Win!

Advantages

- Enough flexibility (only share kernel)
- Enough isolation (namespaces, etc.)
- Standard, reproducible workflow
- Bare-metal performance (or close)
- Minimal-to-no burden on users or consultants in most cases (presumes correct solution choice)
- Lightweight, simple, and easy to use (again, presumes correct solution choice)

Disadvantages

- Require recent Linux kernel/distro (SLES 12SP2, RHEL 7.4, Ubuntu 16.04, Linux LTS 4.4/4.9) *or* privilege
- Occasional growing pains due to newness (feature-complete since 2013, secure since 2015)
- Container expertise in HPC still rare; thus, bad/misleading info & myths are ubiquitous (even in publications)!



Should HPC use containers?

While (arguably) new, containers offer the ideal performance/isolation balance.

So...Umm...These Container Things...Like, What Are They?

Terminology

Good question! Not everyone agrees.
Here's our take:

Linux Containers:

- Use one or more kernel *namespaces*, along with zero or more additional optional kernel features (e.g., cgroups) to provide isolation for (i.e., "contain") a process, along with its child processes, if any;
- Provide a false/distorted view of the underlying system and its available resources;
- Envelope/restrict unprivileged process(es) such that escape/escalation is "impossible;" and
- Facilitate application security by providing capability constraints, integrity assurance, and content validation as required via industry-standard formats and workflows (for application containers).

Containers are <u>processes</u> that we want to group together and collectively lie to.

Namespaces, You Say?

The Linux Kernel supports 6 namespaces as of version 3.8, 7 as of 4.6.

- **6 Privileged Namespaces (require `CAP_SYS_ADMIN` to create)**
  - `mount` – Private filesystem mount points, recursion/propagation controls
  - `pid` – Private view of process IDs and processes, `init` semantics
  - `uts` – Private hostname and domainname values
  - `net` – Private network resources (devices, IPs, routes, ports, etc.)
  - `ipc` – Private IPC resources (SysV IPC objects, POSIX msg queues)
  - `cgroup` – Private control group hierarchy (Linux 4.6+ only)
- **1 Unprivileged Namespace (requires no capabilities to create)**
  - `user` – Private UID and GID mappings
    - Can be combined with other namespaces, even if unprivileged
- **System Call API: `unshare(2)`, `clone(2)`, `setns(2)`**

Further reading: "Namespaces in Operation" (https://lwn.net/Articles/531114/)

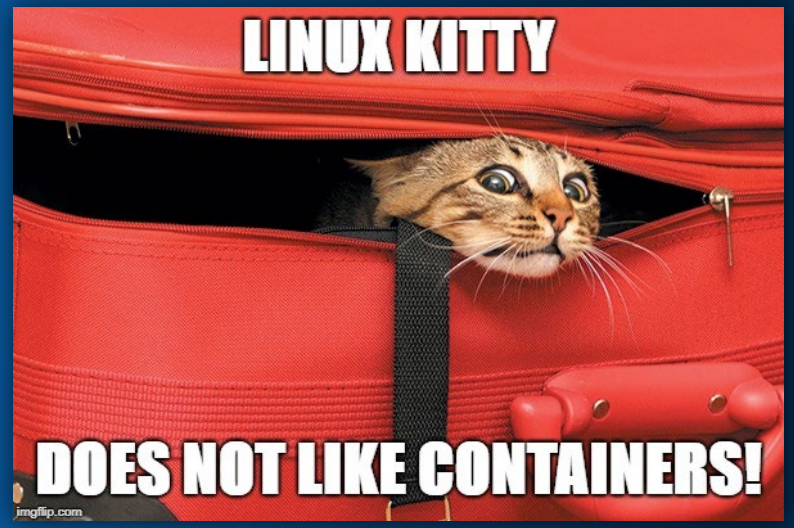
Namespaces are the ways in which the kernel empowers us to lie about our host.

Additional (Optional) Container Elements

Protecting the Lies

The Linux kernel has several additional subsystems that containers sometimes use:

- **`cgroups` – Control hierarchical resource management and usage constraints**
  - Latest kernels (4.6+) even have namespaces for this!
  - Schedulers/RMs use to track/control job resource utilization
- **`seccomp-bpf` – Berkeley Packet Filter-based syscall filtering**
  - Frequently used to prevent containers from exceeding their scope
- **`prctl(*_NO_NEW_PRIVS)` – Prevent privilege escalation**
  - Kernel-level flag that prevents `execve()` granting privileges.
  - Persists across all calls to `fork()`, `clone()`, and `execve()`
  - Privileged containerization is unsafe without this.
- **SELinux – MLS/MAC Labeling system for files/processes**
  - Allows admins precise control over actions, roles of applications
- **AppArmor – Profile-based MAC system for limiting apps' abilities**
  - Similar to SELinux but without filesystem labeling features



These features help us protect the lies, enforce the lies, and insulate the liars!

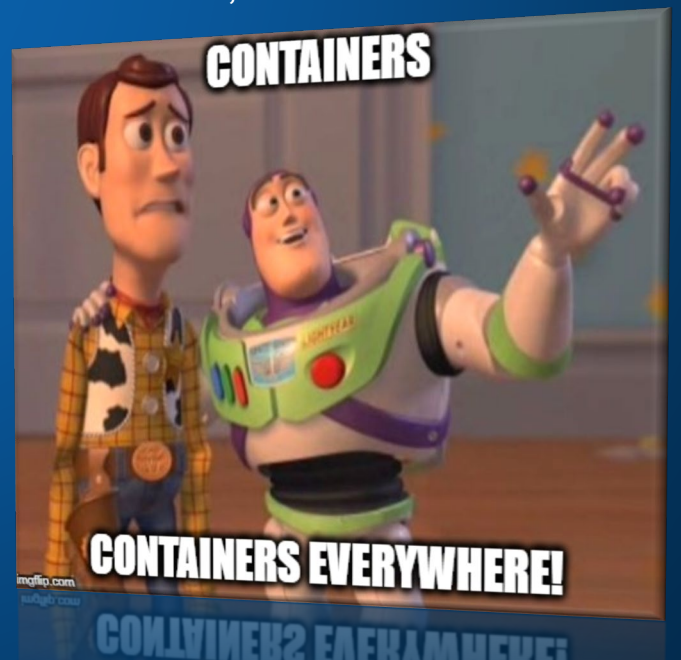
The Container Landscape

Full-featured Container Systems

- Support building, distribution, validation, and execution
- Provide for complete handling of containers throughout lifecycle
- Examples: Docker, CRI-O, LXC/LXD
- Most modern container engines now implement Open Container Initiative (OCI) Image and/or Runtime Spec(s)
- Building containers is still a per-system function. OCI does NOT "do" building!
  - `Dockerfiles` are the de facto standard; robust, capable DSL
  - CoreOS Rocket supplied `acbuild` based around traditional shell-fu
  - ProjectAtomic's `buildah` supports both!

Lightweight Container Systems

- Generally only provide runtime; most leverage Docker/OCI/K8s ecosystem
- Tend to require existing directory tree (i.e., flattened image) to run in
- Examples: RunC, CCon, NsJail, `unshare(1)`, `systemd-nspawn(1)`, SARUS
- …and of course, Charliecloud!

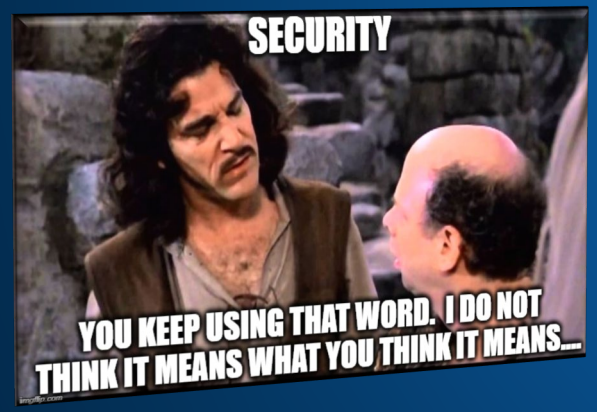


Let's Talk Security

Container Security

Modern Cybersecurity requires many layers; true security isn't merely the absence of access but rather the presence of protection!

- Prior to invocation, container security relies on a wide variety of content provenance and assurance methodologies (e.g., non-repudiation, CAS) to guarantee end-to-end data integrity and transport security.
- At runtime, container security relies entirely on the separation of authorities/roles within the Linux kernel (e.g., `seccomp`, MLS). If the kernel fails, the container fails.
- Lightweight container solutions have the advantage of leaving much of that to others.
- Most mindshare is in Linux kernel hardening and native unprivileged containers.



All container solutions expose (formerly) **privileged operations** to **unprivileged users**, requiring us to accept a new security boundary. So which one would you choose?

- one that's new and known to be flawed?
- a well-understood one that still relies on setuid-root binaries?
- the one that provides all relevant information to the kernel regarding identity and separation of privilege?

There's "Secure," and there's "Not Exactly." Make sure you choose the right one!

Charliecloud

Our Recommended Solution

- **LANL's Container Runtime**
  - Available on GitHub: https://github.com/hpc/charliecloud
- **2018 R&D 100 Winner!**

- **Recent developments in version 0.9.x (currently 0.9.9):**
  - New `Vagrantfile` for generating Charliecloud-enabled (and Docker-enabled) VM images based on CentOS 7 Virtualbox image.
  - New example containers and tutorials based on MPICH, Spack, `spokeo`, `umoci`, OpenMPI 3.1.3, and more.
  - New `ch-fromhost` utility to seamlessly integrate host-based resources into Charliecloud containers (HSN, GPU, libraries, etc.)
  - Improved spec file for potential future inclusion in upstream distros.
  - Significantly improved documentation (also fixed building of documentation on RHEL-based platforms)

Charliecloud is LANL's fully unprivileged, user-installable container suite

Why Charliecloud?

Choosy Users Choose `userns`!

- **Docker/Moby aren't (yet?) a good fit for HPC:**
  - Performance - OverlayFS is slooooooow due to layering!
  - Integrity - File removal is done by "whiting out" which causes subtle issues.
  - Associativity - Docker containers are children of daemon, not CLI.
- **NERSC's Shifter offers appealing user experience but has overhead**
  - User downloads container(s) from Docker Hub, submits jobs to run in them.
  - Admin must set up & maintain Image Gateway.
  - Relies primarily on `chroot()`; requires privilege.
- **Small/simple APIs should yield small/simple code!**
  - Charliecloud weighs in at ***just over 1000*** LoC.
  - Compare to NSJail (4,000), Shifter (19,000), Singularity (15,000), and Docker (160,000)!
- **Charliecloud's security boundary <u>is</u> the kernel**
  - Also worth watching: Moby, Rootless RunC, CRI-O/podman, Rootless Docker

Charliecloud

SMALL SOFTWARE DELIVERING BIG FLEXIBILITY TO HPC

Charliecloud Demo/Walkthrough

```
$ cd ~/charliecloud/examples/serial/hello
$ ls
Dockerfile  hello.sh  README  test.bats
$ ch-build -t hello ~/charliecloud
Sending build context to Docker daemon 15.19 MB
[...]
Successfully built 30662b3f94f3
$ ch-docker2tar hello /var/tmp
57M /var/tmp/hello.tar.gz
```

```
$ ch-tar2dir /var/tmp/hello.tar.gz /var/tmp/hello
creating new image /var/tmp/hello
/var/tmp/hello unpacked ok
$ ch-run /var/tmp/hello -- cat /etc/debian_version
8.9
```

Charliecloud Resources

For More Information…

- **Supercomputing 2017 Paper** by Reid Priedhorsky and Tim Randles
  - "Charliecloud: Unprivileged Containers for UDSS in HPC"
  - Los Alamos Tech Report LA-UR-17-30438
  - http://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-17-30438
- **;login: Article "Linux Containers for Fun & Profit in HPC"** by Reid Priedhorsky
  - https://www.usenix.org/publications/login/fall2017/priedhorsky
- **Documentation:** https://hpc.github.io/charliecloud (includes tutorials!)
- **Source Code:** https://github.com/hpc/charliecloud
- **Mailing List:** charliecloud@groups.io || https://groups.io/charliecloud
- Contact **Reid** (reidpr@lanl.gov), **Tim** (trandles@lanl.gov), or **Michael** (mej@lanl.gov, @mej0 on Twitter)

Try it out, and let us know what you think! Admins, devs, and users welcome!

Any Questions?

Michael Jennings
Los Alamos National Lab
mej@lanl.gov || ***mej@eterm.org***
f/kainx || t/@mej0 || i/kainx || m/@mej0