Introduction to Charliecloud

Megan Phinney
mphinney@lanl.gov
Los Alamos National Laboratory

TAMU Research Computing Symposium
05/21/24

LA-UR 24-24388

Charliecloud Team (Current)

Reid Priedhorsky

Jordan Ogas

Shane Goff

Megan Phinney

Lucas Caudill

Layton McCafferty

Krishna Chilleri

Agenda

01 What are containers?

02 What is Charliecloud?

03 Fully unprivileged build

01

What are containers?

Users need different software

Standard HPC software stacks have a specific purpose:

- Specifically: MPI-based physics simulations

What if your thing is different?

- non-MPI simulations
- Artificial intelligence
- *Spicy* software dependencies

Admins will install software for you

- **IF** there is enough demand
- Unusual software needs go unmet

User-defined software stacks

BYOS (bring your own software)

- Lets users install software of their own choice
- … up to and including a complete Linux distribution
- … and run it on compute resources they don't own

But, possible problems include …

- Missing functionality
  - *high speed network, accelerators, filesystems*
- Performance
  - *many opportunities for overhead*
- Security problems
  - *multiple root exploits*
- Excessive complexity
  - *See Spack*

A container is ***not***

- a lightweight virtual machine
  - or something you boot
- a container image
  - filesystem tree
- something that requires a specific tool
- the container runtime itself
  - ex. Docker

A container is

- a process
  - with its own view of kernel resources
  - or perhaps a group of processes sharing that view

An **image** is: said filesystem

In whatever form it tak

Containers are just processes!

Containers are mostly for abstraction/encapsulation.

- Moving between containers is explicitly supported.
- `setns(2)`, `/proc`, etc.

Privileged/setuid containers need more to be safe.

- SELinux/AppArmor, `seccomp-bpf`, etc.
- (this is hard! Lots of CVEs)

Unprivileged containers get kernel safety measures

- Lots of smart people's time has gone into this
- You already trust the Linux kernel to keep unprivileged processes secure. Keep doing that.

Container Ingredients

01

Linux namespaces

- **Mount**: filesystem tree and mounts
- **PID**: process IDs
- **UTS**: host name
- **Network**: all other network stuff
- **IPC**: System V and POSIX

privileged
need root to create

- **User**: UID/GID/capabilities

unprivileged

02

cgroups: limit resource consumption per process

03

`prctl(PR_SET_NO_NEW_PRIVS)`

04

`seccomp(2)`

05

SELinux, AppArmor, etc.

Charliecloud privilege taxonomy

type	namespace	setup	IDs in container	examples
	mount	privileged	shares UID and GID with host	Docker, Singularity, Podman
	mount + privileged user	privileged	arbitrary UIDs and GIDs separate from host	Singularity, Podman (rootless)
	mount + unprivileged user	unprivileged	only 1 UID and 1 GID in container	Charliecloud

Priedhorsky, Canon, Randles, Younge. SC21. https://dx.doi.org/10.1145/3458817.3476187

Reproducibility

Distros have been working on bit-identical software builds for years and *(plot twist)* it's still not done

- e.g., timestamps get embedded everywhere

Prescriptive builds do help.

- e.g., Dockerfile ⇒ standard

But unsolved challenges remain

- `FROM centos:7` ⇒ maybe different tomorrow
- `FROM centos:9f38484` ⇒ maybe gone tomorrow

02

What is Charliecloud?

Charliecloud Philosophy

1) transparent; not opaque

- **Treat containers as regular files**
- **Examine/debug containers with standard UNIX tools**
- **Things should be explicit**

Charliecloud Philosophy

2) simple;
not complex

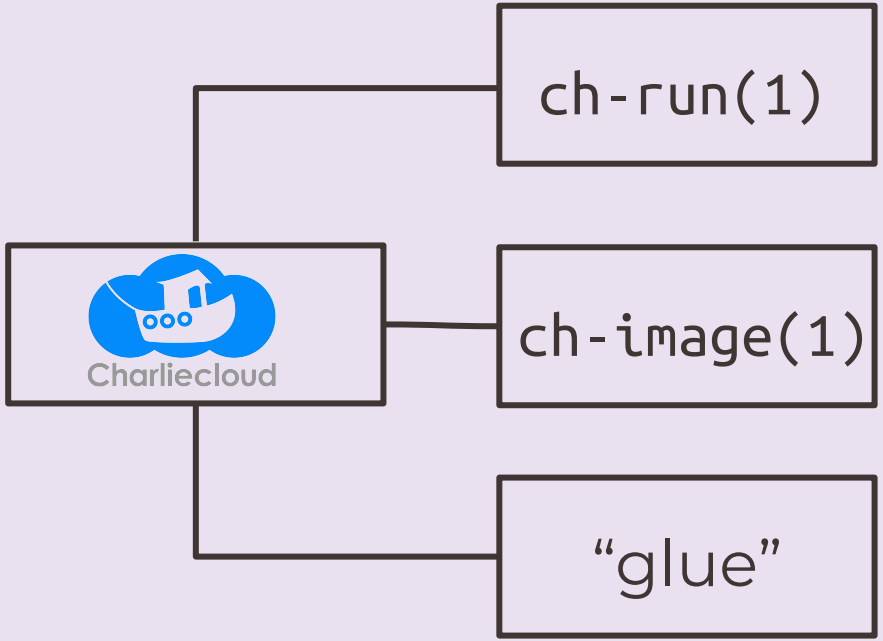
- **Everything is a user process**
- **Implement the right features; Minimize dependencies**
- **Use mount and user namespaces only**
- **Embrace UNIX: *make each program do one thing well***

Charliecloud Philosophy

3) trust the kernel

- **Don’t maintain a security boundary**
- **Stay unprivileged**
- **Avoid responsibility**

Charliecloud Components



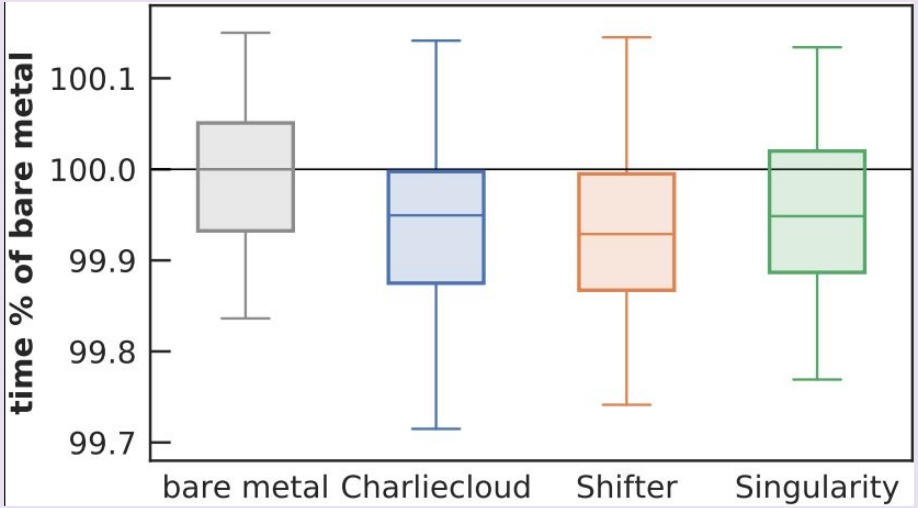
C: container runtime

Python: Docker interpreter
a.k.a “builder”; push/pull/etc.

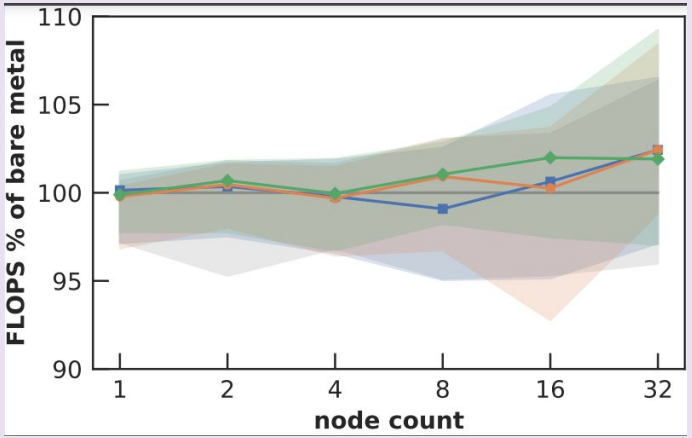
POSIX sh: helper scripts for image conversions & foreign builder wrapping

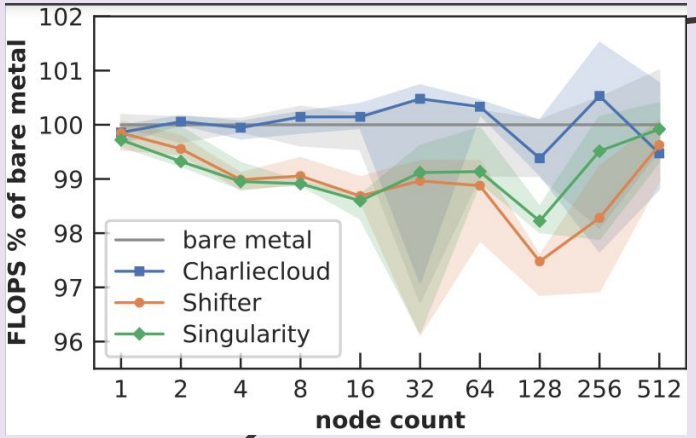
Performance impact: probably zero

SysBench



HPCG





03

Fully Unprivileged Builds

Basic Pitch

- Users want more flexibility ⇒ containers
- Containers need root to build ⇒ HPC policy mismatch
- Build on generic x86 VMs ⇒ HPC hardware mismatch
- Low-privilege containers? ⇒ Build directly on HPC

- ***The Key:*** Linux user namespaces
- New taxonomy of container privilege
- Fully-unprivileged Charliecloud

- Better workflow now & future is bright

Container image workflow

root ⇒ easy

Old 1

generic x86-64

uninformative

Old 2

laptop/workstation

build → test

CI/CD virtual machine

build → test

supercomputer

run

run

specific arch; unprivileged

low privilege?

Solution?

supercomputer

build → test → run

Charliecloud privilege taxonomy

type	namespace	setup	IDs in container	examples
I	mount	privileged	shares UID and GID with host	Docker, Singularity, Podman
II	mount + privileged user	privileged	arbitrary UIDs and GIDs separate from host	Singularity, Podman (rootless)
	mount + unprivileged user	unprivileged	only 1 UID and 1 GID in container	Charliecloud

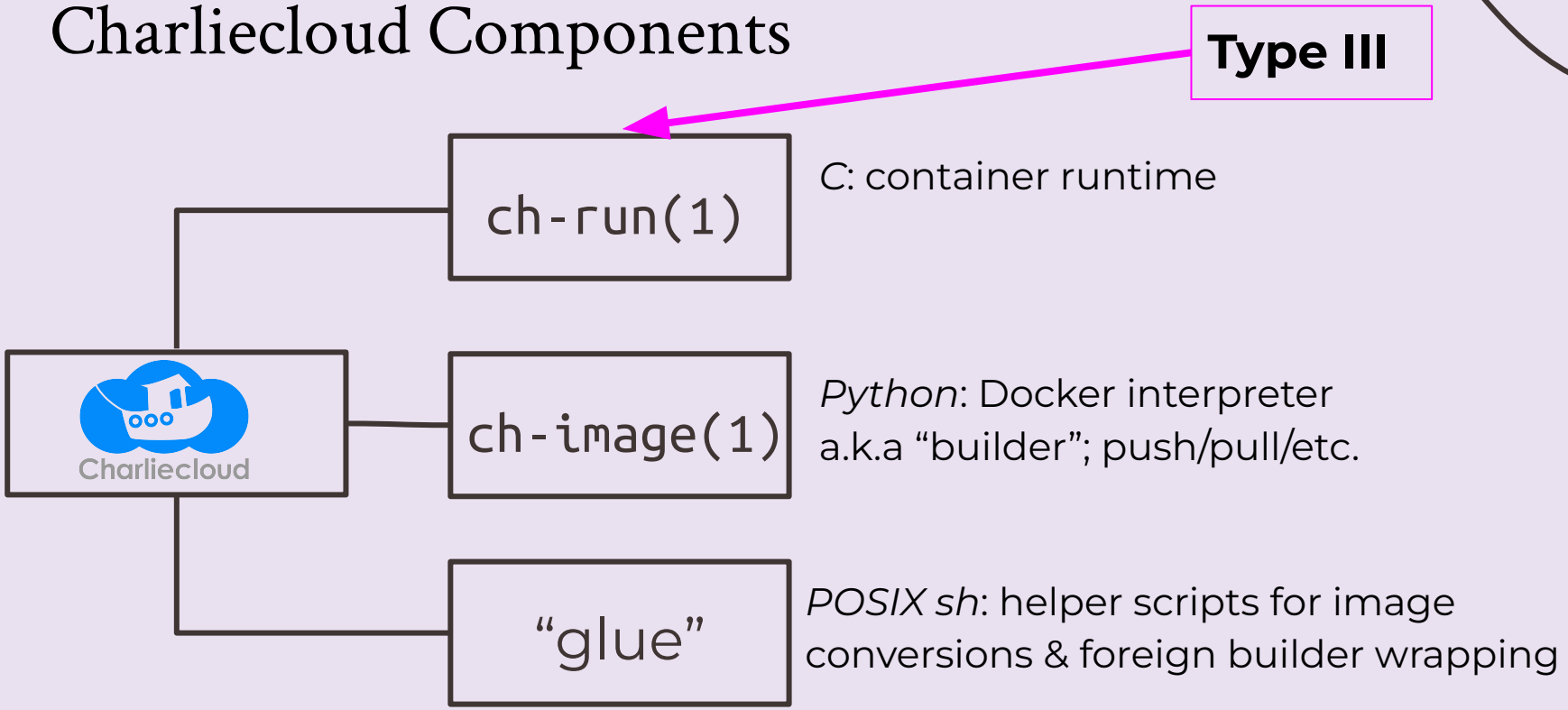
Priedhorsky, Canon, Randles, Younge. SC21. https://dx.doi.org/10.1145/3458817.3476187

Only **Type III containers** are fully unprivileged throughout the container lifetime

Build options

type	namespace	setup	IDs in container	approach
I	mount	privileged	shares UID and GID with host	sandboxed build system
II	mount + privileged user	privileged	arbitrary UIDs and GIDs separate from host	privileged helper tools; careful configuration
III	mount + unprivileged user	unprivileged	only 1 UID and 1 GID in container	`fakeroot(1)` wrapper

Charliecloud Components



Type II vs. Type III build

type	Unprivileged?	File Ownership	ID Management on Host	Works with Network FS	No Root Emulation
II	mostly	preserved	security boundary	no	yes
III	fully	flattened	only 1 UID and 1 GID in container	yes	no

New Root Emulation Mode: `seccomp`

- Why do we need this?
  - We need to tell programs that we have real root privileges even though we are running as a normal user

- Uses the kernel's `seccomp(2)` system call filtering to intercept certain privileged system calls, do absolutely nothing, and return success to the program

New Root Emulation Mode: `seccomp`

- Advantages:
  - Simpler
  - Faster
  - Completely agnostic to libc
  - Mostly agnostic to distribution

- Disadvantages:
  - Lacks consistency

- Our previous root emulation mode, `fakeroot`, has already been adopted by SingularityCE and Apptainer.

Charliecloud User Group

- If you want to join our low traffic mailing list for more information: https://groups.io/g/charliecloud
- Charliecloud User Group Meetings: First Tuesday of the month, 10 am - 11 am MDT; virtual
- Full tutorial: https://hpc.github.io/charliecloud/tutorial.html

04

Interactive Demo

Pre-Demo Notes

- User namespaces are enabled on the compute nodes and not the front-end nodes at this time.
- Access to internet resources is allowed on the front-end nodes and not the compute nodes

- **Suggestion**: have 2 tabs open, one with access to a front-end node and one with access to a compute node.

Introduction to Charliecloud

Megan Phinney
mphinney@lanl.gov
Los Alamos National Laboratory

TAMU Research Computing Symposium
05/21/24

LA-UR 24-24388

Thanks

Do you have any questions?
your email@freepik.com
+91 620 421 838
yourcompany.com

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, infographics & images by **Freepik**

Please keep this slide for attribution