Introduction to Charliecloud

Megan Phinney
mphinney@lanl.gov
Los Alamos National Laboratory

ACES Workshop 07/14/23

LA-UR 23-27202
Charliecloud Team (Current)

Reid Priedhorsky
Jordan Ogas
Shane Goff
Megan Phinney
Lucas Caudill
Layton McCafferty
Hank Wikle
Nick Volpe
Agenda

01 What are containers?

02 What is Charliecloud?

03 Fully unprivileged build
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 takes*
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
- **User**: UID/GID/capabilities

02

cgroups: limit resource consumption per process

03

prctl(PR_SET_NO_NEW_PRIVS)

04

seccomp(2)

05

SELinux, AppArmor, etc.
### Charliecloud privilege taxonomy

<table>
<thead>
<tr>
<th>type</th>
<th>namespace</th>
<th>setup</th>
<th>IDs in container</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>mount</td>
<td></td>
<td>privileged</td>
<td>shares UID and GID with host</td>
<td>Docker, Singularity, Podman</td>
</tr>
<tr>
<td>mount + privileged user</td>
<td></td>
<td>privileged</td>
<td>arbitrary UIDs and GIDs separate from host</td>
<td>Singularity, Podman (rootless)</td>
</tr>
<tr>
<td>mount + unprivileged user</td>
<td></td>
<td>unprivileged</td>
<td>only 1 UID and 1 GID in container</td>
<td>Charliecloud</td>
</tr>
</tbody>
</table>

Priedhorsky, Canon, Randies, Younge. SC21. [https://dx.doi.org/10.1145/3458817.3476187](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 $\Rightarrow$ standard

But unsolved challenges remain

- FROM centos:7 $\Rightarrow$ maybe different tomorrow
- FROM centos:9f38484 $\Rightarrow$ 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

- **ch-run(1)**: C: container runtime

- **ch-image(1)**: Python: Docker interpreter a.k.a “builder”; push/pull/etc.

- **“glue”**: POSIX sh: helper scripts for image conversions & foreign builder wrapping
Performance impact: probably zero

SysBench

HPCG

Torrez, Randles, Priedhorsky / CANOPIE Workshop @ SC, 2019
03

Fully Unprivileged Builds
Basic Pitch

- Users want more flexibility ⇒ containers
- Container build needs root ⇒ HPC management mismatch
- Build on generic x86 VMs ⇒ HPC hardware mismatch
- Low-privilege containers ⇒ build directory on HPC

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

- Better workflow now & future is bright
Container image workflow

- **root ⇒ easy**
- **Old 1**
  - generic x86-64
  - uninformative
- **Old 2**
- **CI/CD virtual machine**
  - build ➔ test ➔ run
- **laptop/workstation**
  - build ➔ test ➔ run
- **supercomputer**
  - run

- **specific arch; unprivileged**
- **Solution?**
  - build ➔ test ➔ run
  - low privilege?
  - specific arch; unprivileged
## Charliecloud privilege taxonomy

<table>
<thead>
<tr>
<th>type</th>
<th>namespace</th>
<th>setup</th>
<th>IDs in container</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>mount</td>
<td>privileged</td>
<td>shares UID and GID with host</td>
<td>Docker, Singularity, Podman</td>
</tr>
<tr>
<td>II</td>
<td>mount + privileged user</td>
<td>privileged</td>
<td>arbitrary UIDs and GIDs separate from host</td>
<td>Singularity, Podman (rootless)</td>
</tr>
<tr>
<td></td>
<td>mount + unprivileged user</td>
<td>unprivileged</td>
<td>only 1 UID and 1 GID in container</td>
<td>Charliecloud</td>
</tr>
</tbody>
</table>

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

Priedhorsky, Canon, Randies, Younge. SC21. [https://dx.doi.org/10.1145/3458817.3476187](https://dx.doi.org/10.1145/3458817.3476187)
# Build options

<table>
<thead>
<tr>
<th>type</th>
<th>namespace</th>
<th>setup</th>
<th>IDs in container</th>
<th>approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>mount</td>
<td>privileged</td>
<td>shares UID and GID with host</td>
<td>sandboxed build system</td>
</tr>
<tr>
<td>II</td>
<td>mount + privileged user</td>
<td>privileged</td>
<td>arbitrary UIDs and GIDs separate from host</td>
<td>privileged helper tools; careful configuration</td>
</tr>
<tr>
<td>III</td>
<td>mount + unprivileged user</td>
<td>unprivileged</td>
<td>only 1 UID and 1 GID in container</td>
<td>fakerooot(1) wrapper</td>
</tr>
</tbody>
</table>
Charliecloud Components

- **ch-run(1)**: C: container runtime
- **ch-image(1)**: Python: Docker interpreter a.k.a “builder”; push/pull/etc.
- **“glue”**: POSIX sh: helper scripts for image conversions & foreign builder wrapping

**Type III**
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.
<table>
<thead>
<tr>
<th>type</th>
<th>Unprivileged?</th>
<th>File Ownership</th>
<th>ID Management on Host</th>
<th>Works with Network FS</th>
<th>No fakeroot(1) Wrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>II</td>
<td>mostly</td>
<td>preserved</td>
<td>security boundary</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>III</td>
<td>fully</td>
<td>flattened</td>
<td>only 1 UID and 1 GID in container</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
Recommendations

Type II implementations:

- add Type III
- fix shared FS (xattr on NFS, Lustre, GPFS?)

Type III implementations:

- robustify fakeroot(1)
- use its ownership data

Distributions:

- add unprivileged mode to package managers

Linux kernel:

- move ID maps into kernel
- make supplemental groups mappable
Introduction to Charliecloud

Megan Phinney
mphinney@lanl.gov
Los Alamos National Laboratory

ACES Workshop 07/14/23

LA-UR 23-27202
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