Rootless Containers with runC

Aleksa Sarai
Software Engineer
asarai@suse.de

Who am I?

- Software Engineer at SUSE.
- Student at University of Sydney.
  - Physics and Computer Science.
- Maintainer of runC.
- Long-time Docker contributor and user.
- Free Software advocate.

The Problem

- Researcher wants to run some Python 3 code on a computing cluster.
  - The cluster only supports Python 2.
- So, researcher uses a container to package Python 3 – right?
  - Drat! The administrator doesn't want to install any new-fangled software.
- The researcher tries to compile dependencies from scratch.
  - *Ha, ha.* Don't even get me started.
- So, what should the researcher do?
  - What if we could create and run containers without any privileges?

What are Linux containers made of?

- Short answer: Namespaces.
  - cgroups are not *really* required.
- Long answer: A lot of duct tape, and some Linux Namespaces.
- They isolate a process's view of parts of the system.
  - Except the things that don't have namespaces. Like the kernel keyring.
- The most interesting of which is the user namespace.
  - You can "pretend" that an unprivileged user is root.

Unprivileged User Namespaces

- Since Linux 3.8, unprivileged users can create user namespaces.
  - It's been *mostly* safe* since Linux 3.19.
- All other namespaces are pinned to a user namespace.
  - You can create a fully namespaced environment without privileges!
  - Operations in the namespaces are more restricted than usual.
- Only your user and group are mapped.

The Solution

- Get a container runtime to implement rootless containers.
  - Disable features in the runtime until the container runs!
- ... or you can just do it manually:
  - `unshare -UrmunipCf bash`
  - `mount --make-rprivate / && mount --rbind rootfs/ rootfs/`
  - `mount -t proc proc rootfs/proc`
  - `mount -t tmpfs tmpfs rootfs/dev`
  - `mount -t devpts -o newinstance devpts rootfs/dev/pts`
  - `# ... skipping over a lot more mounting ...`
  - `pivot_root rootfs/ rootfs/.pivot_root && cd /`
  - `mount --make-rprivate /.pivot_root && umount -l /.pivot_root`
  - `exec bash # finally`

What works?

- All basic functionality works with rootless containers.

Working	Broken
`run`	**`checkpoint`** *[criu]*
`exec`	**`restore`** *[criu]*
`kill`	**`pause`** *[cgroups]*
`delete`	**`resume`** *[cgroups]*
`list`	**`events`** *[cgroups]*
`state`	**`ps`** *[cgroups]*
`spec`	Detached containers *[console]*
`create`	
`start`	

Demo time!

May the demo gods have mercy.

Consoles and runC

- Pseudo-TTY allocation is done using the **host's** `/dev/ptmx`.
  - This can break in user namespaces.
- This is a long-standing bug in libcontainer.
  - Responsible for breaking **sudo** in Docker for years.
- We need this to run our integration tests, and for create / start.
- *Solution:* Do the allocation in the container and send a file descriptor over an **`AF_UNIX`** socket.

remainroot(1)

- Certain syscalls will always fail inside a rootless container.
  - **setuid**(2), **setgid**(2), **chown**(2), **setgroups**(2), **mknod**(2), etc.
- Others will give confusing results.
  - **getgroups**(2), **waitid**(2), etc.
- Package managers and other tools can't "drop privileges".
  - But we don't have any privileges!
- *Solution:* Write a tool to emulate GNU/Linux's privilege model using **ptrace**(2).
  - Currently works for most things, needs some more shims.
  - https://github.com/cyphar/remainroot

What about cgroups?

- cgroup access control is essentially a virtual filesystem.
  - Everything under `/sys/fs/cgroup` is owned by root and has `chmod go-w`.
- But *most* cgroupv1 controllers are hierarchical!
  - And cgroupv2 is **entirely** hierarchical, by design.
  - So why don't we have unprivileged subtree management?
- We need cgroups for a lot of different runC operations.
- *Solution:* Submit kernel patches that implement unprivileged subtree management.
  - Submitted and rejected.
- This would be useful for regular processes too (think Chromium).

Networking

- Unprivileged network namespaces aren't useful.
  - They only have a loopback interface.
- To create a link to the host's interface, you need **`CAP_NET_ADMIN`** in the host user namespace.
- *Solution:* Don't **`unshare`** the network namespace – use the **host's**.
  - This means you don't get to use **`iptables`**`(8)`.:
  - … but at least you get network access!
- There's some movement in the kernel to fix this problem.

Other things left to do

- **`ps`** uses cgroups to get the list of processes in a container.
  - *Solution:* More **`AF_UNIX`** socket magic.
- **`checkpoint`** and **`restore`** are currently disabled.
  - CRIU 2.0 has support for unprivileged checkpointing.
    - Not sure if it correctly checkpoints a rootless container.
  - Unprivileged restore is on the roadmap.
- Whilst cgroups are not generally solved, we can use them opportunistically.
  - If we have write access to a controller, we should use it.

Show me the code!

- Everything is in this pull request: opencontainers/runc#774.
  - Please help us test this!
  - Still needs some review and cleaning up.
- “When will this be finished?”
  - How many additional features do you need working?

Questions?


SUSE
We adapt. You succeed.