Rootless Containers with runC

Aleksa Sarai
Software Engineer
SUSE Linux GmbH

asarai@suse.de

SUSE

OPEN CONTAINER INITIATIVE

whoami

- Aleksa Sarai → `cyphar`
- Software Engineer on the Containers team at SUSE.
- Undergraduate student at the University of Sydney.
- runC maintainer.
- Long-term contributor to the Open Container Initiative and Docker.
- Proponent of Free Software.

Open Container Initiative

- Standards body created in 2015 to standardise container formats and runtimes.
- Two main components:
  - Runtime configuration.
  - Image format.
- runC is the de-facto implementation of the runtime specification.
  - It just needs a root filesystem and configuration file.
- … and it's the runtime used by Docker.

A Hypothetical Researcher ...

- Researcher wants to run Python 3 code on a Python 2 cluster.

A Hypothetical Researcher …

- Researcher wants to run Python 3 code on a Python 2 cluster.
- So they use a container to package Python 3 — right?
  - But the administrator (quite rightly) doesn't want to install "new-fangled software".

A Hypothetical Researcher …

- Researcher wants to run Python 3 code on a Python 2 cluster.
- So they use a container to package Python 3 — right?
  - But the administrator (quite rightly) doesn't want to install "new-fangled software".
- Okay, so they compile all of the dependencies from scratch — right?
  - *Ha, ha.*

A Hypothetical Researcher …

- Researcher wants to run Python 3 code on a Python 2 cluster.
- So they use a container to package Python 3 — right?
  - But the administrator (quite rightly) doesn't want to install "new-fangled software".
- Okay, so they compile all of the dependencies from scratch — right?
  - *Ha, ha.*
- What should our researcher do?
  - What if we could create and run containers without privileges?

A Hypothetical Researcher …

- Researcher wants to run Python 3 code on a Python 2 cluster.
- So they use a container to package Python 3 — right?
  - But the administrator (quite rightly) doesn't want to install "new-fangled software".
- Okay, so they compile all of the dependencies from scratch — right?
  - *Ha, ha.*
- What should our researcher do?
  - What if we could create and run containers without privileges?
- Not actually a hypothetical — this was me in early 2016.
- … and many other researchers have this problem.

How do we do that?

- Containers are mostly made of Linux kernel namespaces.
  - `cgroup`s are not actually required.
  - Oh, and it's mostly duct tape and hacks.
- They isolate a process's view of parts of the system.
- One of the most interesting namespaces is the USER namespace.
  - You can "pretend" that an unprivileged user is root.

USER Namespaces

- Capabilities and uids are scoped to their `USER` namespace.
- Permission checks are done based on the namespaces a process is in.
- Based on mapping the host's `uid`s and `gid`s into the container.
- `EPERM` on all operations on `unmapped` `uid`s or `gid`s.

Unprivileged USER Namespaces

- Since Linux 3.8, unprivileged users can create new USER namespaces.
  - And it's been *mostly* safe since Linux 3.19.
- All other namespaces are "owned" by a given USER namespace.
  - You can create a fully namespaced environment without privileges!
  - Operations in these namespaces are more restricted than usual.
- Only your user and group are mapped!

Implementing it!

- Get a container runtime to implement this for you!

Implementing it!

- Get a container runtime to implement this for you!
  - … which I've already done for you!
- … or you could do it manually:

Implementing it!

- Get a container runtime to implement this for you!
  - … which I've already done for you!
- … or you could 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.
  - https://github.com/opencontainers/runc/pull/774
  - … and it's all tested!

Works	Broken
`run`	`checkpoint` [criu]
`exec`	`restore` [criu]
`kill`	`pause` [cgroups]
`delete`	`resume` [cgroups]
`list`	`events` [cgroups]
`state`	`ps` [cgroups]
`spec`	
`create`	
`start`	
`--detach`	

What about images?

- Runtime is only half of the story — what about images?
- Recently, two tools have been created to make this very easy.

  skopeo Download and convert images from various sources and registries.

  umoci Unpack, repack and otherwise modify local OCI images.
- Now you can use images as an unprivileged user too!
  - You don't get cool filesystem features though.

Live Demo!

May the demo gods have mercy.

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 will therefore fail to “drop privileges”.
- *Solution*: Write a tool to emulate GNU/Linux’s privilege model using `ptrace`(2).
  - It currently *sort of* works, but requires more shims and testing.
  - `https://github.com/cyphar/remainroot`

What about networking?

- Unprivileged `NET` namespaces aren't useful.
  - They only have a loopback interface.
- To create `veth` pairs you need `CAP_NET_ADMIN` in both `NET` namespaces.
- *Solution*: Don't set up the `NET` namespace and just use the **host** namespace like a normal process.
  - Which means you don't get to use `iptables`(8).
  - … but at least you get network access!
- But there is some movement in the kernel to fix this problem.

What about `cgroups`?

- `cgroup`s are a resource control mechanism.
- `cgroup`s interface is effectively a virtual filesystem.
  - Everything under `/sys/fs/cgroup` is owned by `root` and `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?
- *Solution*: Submit kernel patches that implement unprivileged subtree management.
  - Currently at version 10 of the patch set.
  - Maintainer doesn't seem to like the idea at all.

What other stuff is broken?

- I just found out that the fix for `CVE-2016-9962` broke rootless containers.
  - It was a kernel bug, sent a patch yesterday.
  - … the point being the interfaces are still janky.
- `runc ps` uses `cgroup`s but we could potentially implement it differently.
  - Unfortunately tricky because it requires `AF_UNIX` socket magic.
- *Please* test this code and tell me what you find!
  - `https://github.com/opencontainers/runc/pull/774`

Just for Researchers?

- Researchers are not the only target!
- What if you could use container features in desktop applications?
  - … or start Kubernetes as part of an application?
  - … or build images in a build system without privileges?
  - … or *<insert your idea here>*
  - … or **PROFIT!**

Future of Free Software?

- Back to the theme of the conference: “The Future of Free Software”.
- In my mind the future is refining and extending existing ideas.
  - In containers, this is allowing everyone to use them.
- But more importantly, the future is **what we make it**.

Acknowledgements

- Jessie Frazelle — started working on this first and inspired me with her PoC, binctr.
- Eric Biederman — working tirelessly to get USER namespaces working.
- James Bottomley — helped me with kernel patches trying to fix the `cgroup` issue.

Questions?