Soft Reboot

Keep your containers running while your image-based Linux host gets updated

Luca Boccassi
Linux Systems Group
Microsoft

What? Why?

- Performance, of course
- On a headless system providing services, a reboot (even a kexec one) causes a service interruption
  - That's not nice, nobody likes interruptions
- Also on some systems a double reboot is necessary (e.g.: dnf offline upgrade), can be used to cut down on downtime

- Reboot → kexec: cut firmware/hardware reset, save time
- Kexec → soft reboot: cut kernel reset, save time

- Overarching goal: minimize service disruption as much as possible, in any way possible
  - …that doesn't cost money, enjoy that eMMC disk, hope you like single-channel slow I/O

Coming to an Image-Based Linux near you

- Fits well with image-based model
  - rootfs and UKI are distinct, discrete components of the OS
  - updated independently
  - Kernel is not updated that often (once or twice a month on Debian stable)
- Pairs nicely with kernel ‘live patching’
- Whole userspace replaced ‘atomically’ and rebooted into
- On package-based OSes might be useful when non-restartable components are updated (e.g.: D-Bus broker/daemon)

How?

- As far as the kernel is concerned, nothing at all is happening
    - …which means boot-id doesn’t change, which means ‘journalctl –list-boots’ also doesn’t notice, whoops - we should probably fix that, it’s on the TODO list
- As far as any userspace process is concerned, it is a normal shutdown
- systemd goes through the usual shutdown phases and stops every service
- …but instead of giving control back to the kernel, it re-execs itself and starts up again
    - Either in-place, or if it exists pivot roots to /run/nextroot
    - All disks can be pre-prepared under /run/nextroot, no need to redo decryption
    - Kernel configuration is maintained as-is (sysctl)
- New ‘soft-reboot’ systemctl verb and equivalent D-Bus APIs
    - A new PrepareForShutdownWithMetadata signal is sent that lists the type of reboot (v255)

Is that it?

- Given systemd doesn’t exit, arbitrary state can be passed through the re-exec
- File Descriptor Store is the prime example, services can hand FDs to systemd and get them back after soft-reboot
  - Units need to set the new FileDescriptorStorePreserve= directive
  - FD Store can be used for open sockets (e.g.: connections stay alive) but also for memory buffers via memfd
- Network stack can be configured to preserve configuration
  - e.g.: KeepConfiguration=dhcp-on-stop in systemd-networkd
- /run/ is also transitioned through wholesale, so services will find any saved state still available after soft-reboot
  - Note that this is not recursive, any sub-mounts of /run/ will have to be set up again
  - /tmp/ is a new tmpfs

Cool, but what does this have to do with containers?

- Here’s a neat idea: some payloads are independent of the rootfs…
  - E.g.: Portable Services run in their own image-based filesystem
- …so what if we configured them to keep running during the soft-reboot?
  - Network still accessible, disks still accessible
  - From a little to zero service interruption on updates!
- …but there’s a catch: they really, really need to be disconnected from the rootfs
  - Any reference kept around from the old rootfs will make it so the mounts are not garbage collected, occupying memory
  - Any IPC to the rest of the OS might at the very least block until soft-reboot has finished, if not outright fail. E.g.: sd-bus need to be configured to reconnect to D-Bus
  - Future improvement: for each BindPaths= in a systemd service figure out if it’s from the old rootfs, and if so have systemd swap it (using mount-beneath)

Demo time: Podman container surviving soft-reboot

Demo time: Azure Boost production node

Thanks!

Questions?

Appendix - podman quadlet diff

https://paste.debian.net/1306203/