A Comparison of Linux Software Update Technologies

Matt Porter, Konsulko Group
Embedded Linux Conference Europe 2016
Overview

- Background of Linux software update
- Linux software update strategies
- Detailed look at each FOSS project
  - Strategy employed
  - Other features
  - Maturity
  - Community
  - Downstream projects
Linux software update history

● H.J. Lu’s Boot-root distributions
  ○ Boot and build the rest, no update mechanism
● MCC, TAMU, and SLS
  ○ Packages in tarballs, no dependencies
● Slackware
  ○ Packages in tarballs, no dependencies
  ○ Per release scripted updates, flaky if too old
Linux software update history

- Debian, Red Hat Linux / Fedora, SUSE
  - Modern deb or rpm package management with complete set of dependencies
  - Non-atomic incremental updates
  - Release updates by designating a set of package versions
  - Driven by complex set of pre/post install scripts which can leave updated systems in a non-working state.
Linux software update requirements

- There are many requirements
  - Tradeoffs are unique for each product.
  - No exact steps only guidelines
- Power fail safe?
- Frequent/infrequent updates?
- Bandwidth of update delivery channel vs. size of updates?
- Speed of update?
- Verification/Authentication?
Linux software update strategies

Traditional method

- Traditional non-atomic package-based releases
  - Package based granularity with dependency hierarchy
  - apt and yum based updates may require luck or other methods for reliability.
- Unacceptable for the embedded zoo.
Linux software update strategies

Full image updates

- Has been the standard approach since Linux in embedded systems was popularized.
- Single image approach assumes the new update will boot
  - Recoverable if update can be performed from an immutable mechanism (fallback factory bootloader)
- Dual image approach is inherently atomic
  - Bootloader will fallback to previously working image on failure of update
- Update speed relative to size of full image
Linux software update strategies
Full image updates

Completely unrealistic and simplified dual image example

Bootloader

1) Boot active image

Linux image A (active -> inactive)

2) Receive and install update to partition B.

3) Toggle B active and A inactive. Reboot.

Linux image B (inactive -> active)

4) Boot new active image B. Fallback to A if boot fails (typically using watchdog and/or heartbeat).
Linux software update strategies
Incremental atomic updates

- The new kid on the block that does crazy stuff, likely to be called *balderdash* by the old guard.
- Driven initially by server needs
  - Incremental atomic upgrades that can be quickly deployed or rolled back on demand.
  - Complete history of deployments
- Releases are composed of binary deltas
  - Not a package granularity
  - Deltas are per file modified
  - Size of updates are minimized
Linux software update strategies

Containers

- Not usually a complete upgrade solution
- Built on top of a core immutable distribution
- Applications only exist in a container instance
- Updates rolled out in container deltas
SWUpdate

- Single or dual image update framework
  - [https://github.com/sbabic/swupdate](https://github.com/sbabic/swupdate)
  - Written in C. GPL2 license.
  - Attempt to be modular with plugins
  - Supports signed images, local/remote updates, and U-Boot.
  - meta-swupdate layer for Yocto Project
- Has several contributors besides original author
- Used at least by Siemens ([http://sched.co/7rrA](http://sched.co/7rrA)) and Stefano’s own projects
SWUpdate

- Updates delivered in simple CPIO archive
- Each individual image is described in sw-description and integrity is validated with a SHA256 hash.
- Handler plugins implement the details of how each described image is handled.
  - U-Boot env update
  - NOR, NAND, UBI partition and write
  - MMC/SD/eMMC partition and write
  - Custom installers can enable FPGA bitstream or uC firmware updates
SWUpdate

- **sw-descriptions** can be extended using custom LUA parsers to support new features
  - multiple hardware platforms in one image
- Configuration file support via libconfig or XML format by default
- Uses Kbuild for configuration
- Supports Mongoose web server and REST interface to Hawkbit server for remote update
- Strange stuff exists like an implementation of a userspace GPIO library that duplicates other projects.
mender.io

- Dual image update framework
- https://github.com/mendersoftware/mender
- Designed as a client/server system for OTA updates. Written in Go. Apache 2 license.
- meta-mender layer supports building the client into a device image using YP/OE
- https://github.com/mendersoftware/meta-mender
- Project contributors are overwhelmingly represented by Mender employees.
Two client modes
  ○ Standalone - updates are triggered locally (suitable for physical media or any network update in pull mode)
  ○ Managed - client is a daemon and will poll the server for updates.

mender’s dual image or “A/B” scheme uses a notion of “commit” when and update has booted properly. On failure it will toggle the inactive/active partitions as with a standard dual image approach.
● QEMU and BeagleBone Black reference platforms
  ○ That’s the bee’s knees for getting started easily
● As a complete demonstrable solution, mender relies on some assumptions:
  ○ U-Boot Boot Count Limit, ext2/3/4fs, and Linux env tools, and a specific U-Boot configuration
  ○ systemd (and required kernel config options) for managed mode
  ○ a fixed layout of U-Boot in one partition, a persistent data partition, and two A/B partitions with rootfs/kernel.
mender.io

- Does not support raw NOR, NAND, UBI partitions and volumes.
- Excellent documentation on use and customization.
- Ready to use platforms to test operation.
- Established project CI loop.
- Test/QA tools all available freely.
OSTree

- Incremental atomic upgrade mechanism
- [https://github.com/ostreedevel/ostree](https://github.com/ostreedevel/ostree)
- Self-described as “git for operating system binaries”.
- Uses a git-like object store to record and deploy complete file system trees using binary deltas.
- Depends on an immutable filesystem hierarchy for the updated root filesystem ([https://www.freedesktop.org/wiki/Software/systemd/TheCaseForTheUsrMerge/](https://www.freedesktop.org/wiki/Software/systemd/TheCaseForTheUsrMerge/))
- Persistent data kept in /etc
OSTree

- How does it work?
  - Target has a local copy of a repository in /ostree/repo
  - Target has any number of “deployments” stored in /ostree/deploy
- A deployment is stored physically in /ostree/deploy/$OSNAME/$CHECKSUM and uniquely identified with a SHA256 checksum
- Each deployment has its own copy of /etc
- Activation requires a reboot
OSTree

- Deploy and rollback
  - ostree-admin-upgrade
  - ostree-admin-deploy {REFSPEC}
  - ostree-admin-status
  - ostree-admin-undeploy {INDEX}
- Atomic updates are guaranteed by atomically swapping a /boot symlink to a new deployment /ostree/boot.foo directory
- A bind mount is established at boot time pointing to the currently deployed filesystem.
There are many projects that have adopted OSTree

- Gnome Continuous
  https://wiki.gnome.org/Projects/GnomeContinuous

- Project Atomic
  http://www.projectatomic.io/

- Flatpak
  https://github.com/flatpak/flatpak

- Pulp Platform
  https://github.com/pulp/pulp_ostree

- Automotive Grade Linux
  https://jira.automotivelinux.org/browse/SPEC-194

- https://git.automotivelinux.org/gerrit/gitweb?p=AGL/meta-agl-extra.git;a=summary
swupd

- Incremental atomic upgrade mechanism
- Originally part of ClearLinux project
  - https://github.com/clearlinux/swupd-client
  - https://github.com/clearlinux/swupd-server
- Functionality is very similar to OSTree.
- Updates are delivered as a stream of bundles containing binary filesystem deltas.
- meta-swupd supports YP/OE target image builds
  - http://git.yoctoproject.org/cgit/cgit.cgi/meta-swupd
Key difference is that the swupd-client does not require a reboot to activate a newly released bundle.

swupd-server tool handles creation of bundles and feed update streams to a client.

Project shows no contributors outside of Intel.

Only projects adopting swupd are ClearLinux and Ostro OS, both Intel projects.
Container-based solutions

- Resin.io
  - Base OS is flexible, Docker-based deltas
- Ubuntu Snappy
  - Base OS is minimal Ubuntu with deltaed containers
- Project Atomic
  - Base OS managed with OSTree, Docker-based deltas
- Focus on application and middleware update
Related sessions

- Generic System for Safe Upgrades
  - Tuesday 10:00 [http://sched.co/7rrp](http://sched.co/7rrp)
- ResinOS
  - Tuesday 15:00 [http://sched.co/8PTZ](http://sched.co/8PTZ)
- OSS Remote update for IoT Devices
  - Tuesday 15:00 [http://sched.co/7rrA](http://sched.co/7rrA)
- Mender.io BoF
  - Tuesday 18:10 [http://sched.co/8PeA](http://sched.co/8PeA)
- Continuous Delivery with Yocto (Ostro)
  - Wednesday 10:45 [http://sched.co/7rrB](http://sched.co/7rrB)
- Software update for IoT
  - Wednesday 14:00 [http://sched.co/7rrJ](http://sched.co/7rrJ)
- Software updates for connected devices
  - Wednesday 15:00 [http://sched.co/7rrK](http://sched.co/7rrK)
Questions?