Bringing DevOps to Devices

Modern Update Approaches for Embedded Linux

Petros Angelatos
April 2016
About me

- Petros Angelatos
- CTO & Founder of resin.io

@petrosagg
Agenda

- The need for updates
- Update techniques
- Our solution
  - Device software architecture
  - Yocto architecture
  - Host OS updates
  - Application updates
- Drone demo
- Questions
The need for updates
More powerful devices, more complex software

Embedded software now demands the full lifecycle support we’ve been giving to web, cloud, and mobile.
Importance of updates

- Recalling products costs money
- Higher exposure to security exploits
- Longer release cycles
Importance of updates

Toyota to Recall 1.9 Million Priuses to Update Software

by Craig Trudell and Yuki Hagiwara

February 11, 2014 – 9:31 PM PST
Importance of updates

THE VW DIESEL CRISIS

VW will recall 2.46 million cars with illegal software in Germany, report says

November 30, 2015 11:03 CET -- UPDATED: Nov. 30 13:05 CET - adds German minister statement

FRANKFURT (Reuters) -- Volkswagen Group will recall 2.46 million diesel vehicles fitted with illegal emissions-control software in Germany, Die Welt newspaper reported today.

Some 1.54 million VW-brand cars and light commercial vehicles need to be fixed, as well as 531,813 from Audi, another 286,970 at Skoda and 104,197 at the Seat brand, the paper said, without naming its sources.
Importance of updates

Fiat Chrysler recalls 1.4 million vehicles to block hacking

*Fiat Chrysler Automobiles is recalling 1.4 million cars and trucks to update security to block possible hacking attempts.*

Fiat Chrysler Automobiles, under pressure from federal regulators, said Friday it will recall 1.4 million cars and trucks to protect them from cybersecurity attacks just days after Wired magazine revealed that a Jeep Cherokee could be hacked remotely.

The National Highway Traffic Safety Administration has launched an investigation so it can closely monitor the recall.
Importance of updates
We’ve been supporting a network with hundreds of screens in 5 countries, for two years. We’ve had to go out on weekends, in the snow, with drills and USB sticks, upgrading software.

We spent a lot of resources on infrastructure that had little to do with our specific application.
Updates are difficult

- Poor connectivity
- Intermittent power
- Devices can be anywhere
- Devices could be in the middle of a critical operation
- A failed updated is a bricked device
Various update techniques

- Image based
- Package based
- Containers
Our solution
On-device software architecture

The Vision: 100% updateable

- All containers update safely and reversibly. Our own agent (Supervisor) runs in its own container
- Layers shared between containers are stored only once
- Docker and Yocto userspace update using conventional methods
- All software projects we depend on are under open source licenses
Yocto layer architecture

- meta-resin
  - Jethro overlayer
  - Fido overlayer
  - Daisy overlayer

- meta-resin-common

- poky (yocto)

- meta-openembedded

- BSP x 10
Host OS updates

- Green/Blue method
- Has been discussed in Yocto mailing list
- Used by
  - CoreOS
  - ChromiumOS
  - Ubuntu Snappy
  - Probably others too
Typical partition layout

- Linux and Bootloader
- root
- Inactive
- Data
Atomic updates

- Immutable filesystem images
- Image as unit of deployment
Host OS updates

- Initial device state
- Bootloader points to first root partition
Host OS updates

- Version 2 of the OS is downloaded into the inactive partition
- This operation can be interrupted without issues
- At the end, we can verify integrity and sync to disk
Host OS updates

- Copy bootfiles from the OS image to boot partition
  - Kernel
  - DTBs
  - Initrd
  - etc.

- Do it in a atomic fashion
  - Write tmp file
  - Sync to disk
  - Rename to destination
  - Sync again
Host OS updates

- Flip flag in bootloader to point to the new OS image and to the new OS kernel
- Reboot
Host OS updates

- Final device state after reboot
Failsafe updates

- With the help of hardware watchdogs
- With the help of bootloader logic
- The new version marks itself as stable after running self-test
Container updates

- We use Docker
  - Originally ported Docker to ARM
- No reboot required
  - Move fast, brick nothing

- Efficient in bandwidth through layer sharing
- Efficient in disk IO through layer sharing
  - But we can do better with binary diffs
Container updates

- We can build update strategies depending on requirements
- Can achieve true downtime updates
Update strategies

Strategy 1: Download then Kill (default)

1. DOWNLOAD THE UPDATE
   DEVICE
   Supervisor
   Old Container → New Container

2. UPDATE DOWNLOADED
   DEVICE
   Supervisor
   Old Container → New Container

3. OLD CONTAINER KILLED, NEW ONE STARTED
   DEVICE
   Supervisor
   Old Container → New Container
Update strategies

Strategy 2: Hand Over

1. DOWNLOAD THE UPDATE
   - DEVICE
     - Supervisor
     - Old Container
     - New Container

2. UPDATE DOWNLOADED
   - DEVICE
     - Supervisor
     - Old Container
     - New Container

3. NEW CONTAINER STARTED
   - DEVICE
     - Supervisor
     - Old Container
     - New Container

4. NEW CONTAINER ASKS OLD CONTAINER TO GIVE UP
   - DEVICE
     - Supervisor
     - Old Container
     - New Container

5. OLD CONTAINER IS READY TO DIE
   - Device
     - Supervisor
     - Old Container
     - New Container
     - Notifies

6. OLD CONTAINER KILLED
   - Device
     - Supervisor
     - Old Container
     - New Container
Drone demo
Food for thought

- If you squint, containers look a lot like host OS images and vice versa

Can we unify?

We think yes. Come to our booth to talk about it :)
Open source

- Resin OS Github Organisation
  - https://github.com/resin-os
- Resin device supervisor
  - https://github.com/resin-io/resin-supervisor
- Gitter
  - https://gitter.im/resin-io/chat
Questions?

Get in touch: Petros Angelatos // petrosagg@resin.io // @petrosagg