Preview Deck: Containers on Embedded

How to Make an App Enabled Embedded Linux Product that Fits in 16MB of Flash

Hands On Workshop @ ELC Europe 2019

Alexander Sack <asac@pantacor.com> et al.

Pantacor

The “Goal”

QEMU Malta 16M (e.g. MIPS)

OpenWRT

w/ App via OTA

Pantacor

16MB of flash for products is not a lot …

Must fit

- Bootloader
- Config
- Kernel
- Modules
- Firmware Blobs
- UserSpace
- Persistent Storage

Must support

- A full networking stack
- A management Client
- Failsafe Upgrades
- Telemetry data client
- Ability to add Third Party features without integrating into the main firmware → aka Applications

Pantacor

The “Host”

Modern Linux Host (x64) w/ PVR

```
wget -qO - http://bit.ly/elc-pvr-tarball | tar xz
```

The “Target Host”

QEMU Malta 16M (e.g. MIPS)

```
docker pull pantacor/qemu-malta-16m
```

Running the Target Host

```
# Download the factory image
wget -qO - http://bit.ly/elc2019-workshop-img-xz2 \
    | unxz > flash.img

# we use qemu mips emulator with 16M flash
docker run --rm -it \
    -v $PWD/flash.img:/tmp/pflash.img \
    --privileged --net host \
    pantacor/qemu-malta-16m
```

Claim your Device

```
# Login at https://www.pantahub.com
Username: elc19-workshop / Password: elc19ws

# Claim your Device
DeviceID: ```cat /pantavisor/device-id```
Challenge: ```cat /pantavisor/challenge```

# Find your device: Search for device id on pantahub
```

The File System

A single UbiFS/JFFS2 partition
Bootloader loads kernel/initrd/fdt from here!

0K-“Recovery” Disk

Transactional Update & Rollback
Presence Factory Reset with USB

Pantacor

Small Kernel ~2MB

All built-in peripherals compiled in
Basic Modules support
Only minimal modules need installing

Pantacor

Initrd Root (<1M)

Ship container Engine outside
Allow sharing of files across containers
Bundle All Orchestration Logic

Full Network Stack by OpenWrt ~1.5MB

All built-in peripherals compiled in
Basic Modules support
Only minimal modules need installing

Pantacor

A Basic WebApp

Exemplary httpd server with basic REST cgis.
Can be replaced with more useful things like bluetooth telemetry daemon etc.

Building the Web App

```
# Use buildroot to build a minimal tinyhttpd rootfs
git clone https://github.com/buildroot/buildroot

# OR …
wget -qO - http://bit.ly/elc2019-rootfs > rootfs.tar.xz

# Use docker as packaging and distribution format
docker import rootfs.tar.xz mips-darkhttpd-raw
```

⇒ Total Size XZ compressed: 650K

Building the Web App (2)

```
# Create runnable docker with volume and entrypoint
cat > Dockerfile <<EOF
FROM mips-darkhttpd-raw

VOLUME /var/www
CMD /usr/sbin/darkhttpd /var/www
EOF

docker build -t mips-darkhttpd .
```

Deploying the App

We use Pantavisor (PV) Container Engine
We use PVR to manage PV state
we add pantacor/mips32-tinyhttpd container

Deploying the App

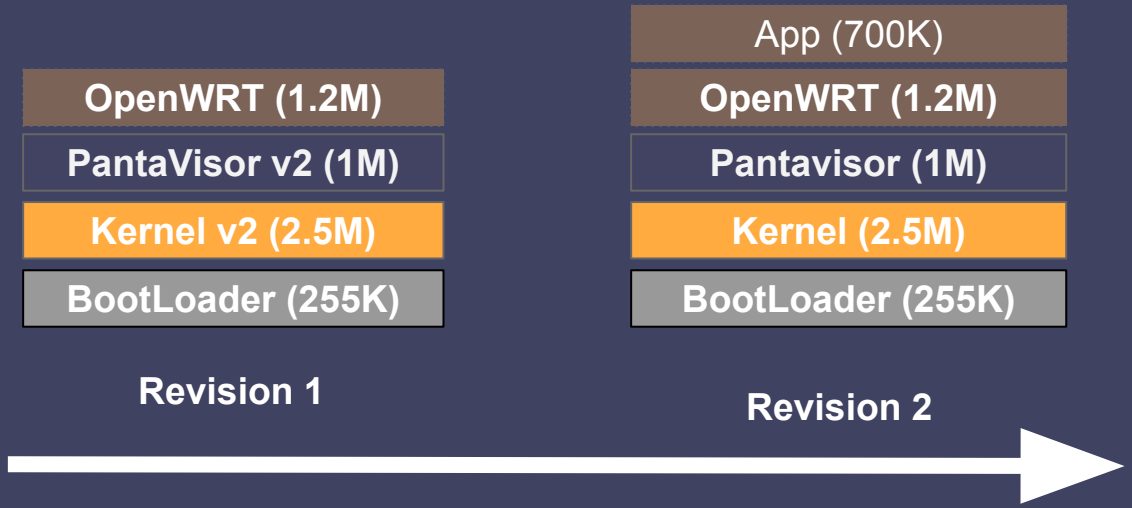
```
# Clone the device to patch
pvr clone elc19-workshop/YOURDEVICE

# add and commit application
fakeroot pvr app add --from mips-darkhttpd darkhttpd

# commit and post the update
pvr add . && pvr commit && pvr post

# System will restart and tinyhttpd will serve on port 80
curl http://192.168.1.1/
```

Result



16M System with APP by Third Party
Automatic Telemetry: Logs go to pantahub
Fail Safe Software Upgrade adds App through PH
Small Disk Footprint feasible for low cost devices

Pantacor