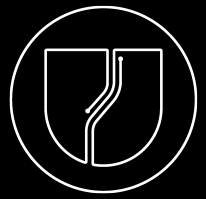
kynetics
enjoy the art of coding

Update Remotely IoT Devices using Eclipse hawkBit and SWUpdate

Nicola La Gloria



Update Factory

Thanks:
Diego Rondini, Andrea Zoleo, Will Martindale, Daniele Sergio, Eric Nelson, Gary Bisson (Boundary Devices) and Amit Pundir (Linaro).

and...

Thanks to the little Marianna for the drawing!

Agenda

- Motivations for our work with OTA updates on Embedded Linux
- The Android way for managing updates
- SWUpdate - a Linux Update agent
- Remote management and rollout: Eclipse hawkBit
- Our implementation to manage and deploy software updates Android-like: Update Factory

Motivations

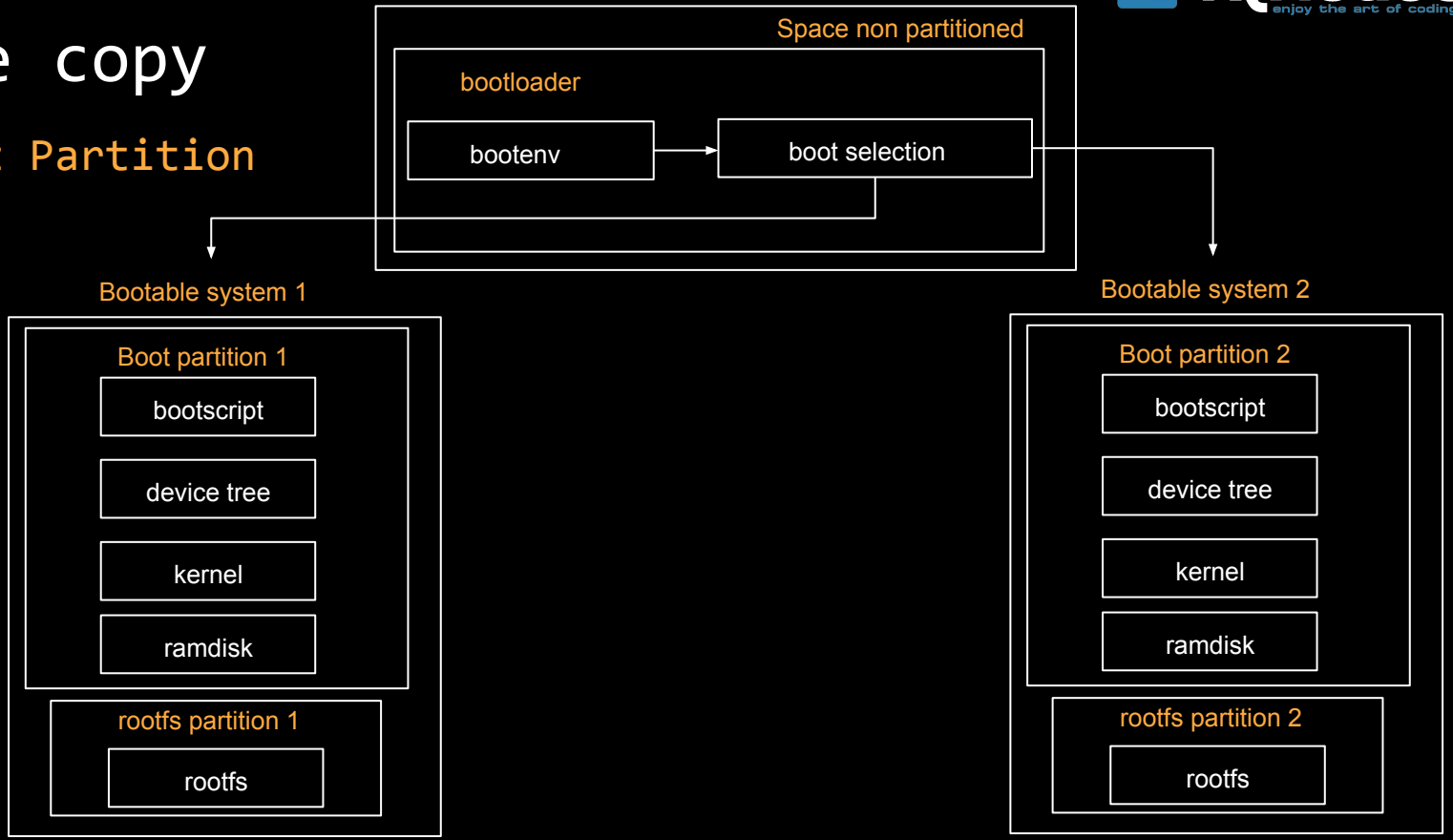
- Support medium scale general purpose CPU-SOC modules
- Ability to implement different update approaches on Linux
- Create a neutral platform to support both Linux and Android devices
- Track updates and divide them per device types and use cases
- Support custom device metadata sent to the Remote Update Management Platform

Part One: Device Update Approaches

- Double copy:
  - The devices features two copy of the Application/OS/RootFS
  - Each copy must contain the kernel, the root file system, and each further component that can be updated.
  - Cooperation with the boot loader is necessary, to decide which copy should be booted
- Single copy:
  - An upgrading software is required
  - Used usually to upgrade the partition containing the rootfs
  - You may update Kernel and Device Tree if the update environment is *segregated*
  - Cooperation with the boot loader is necessary to boot in *update mode*.

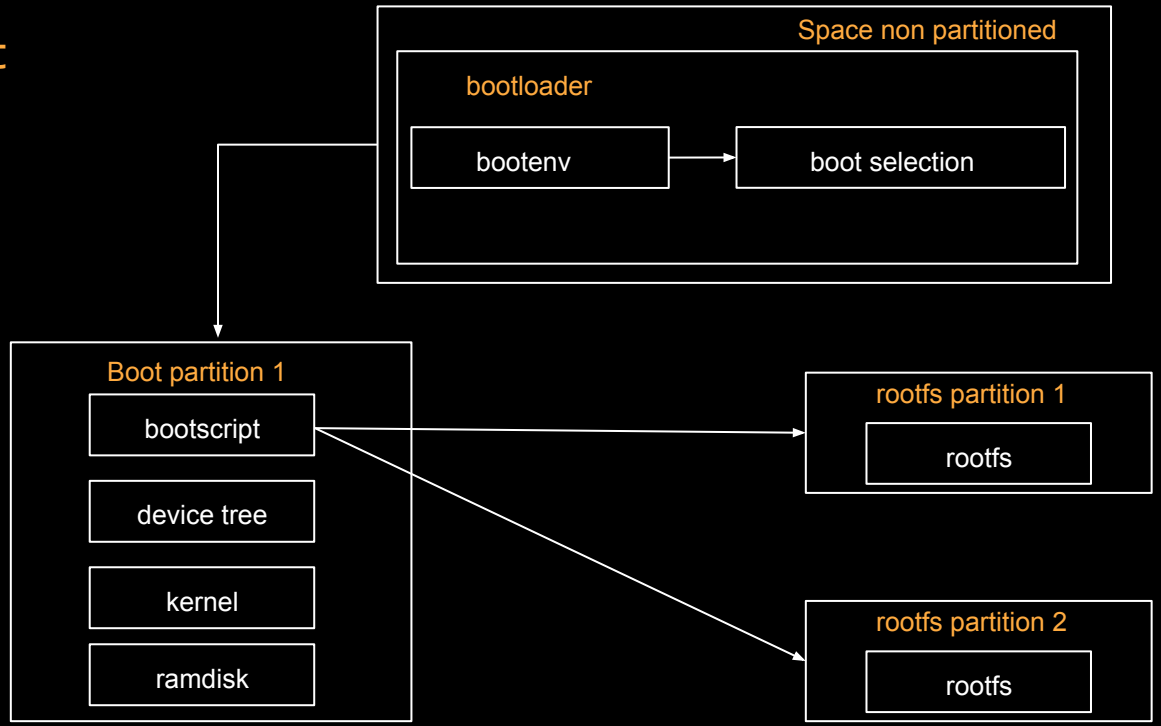
Double copy

Dual Boot Partition



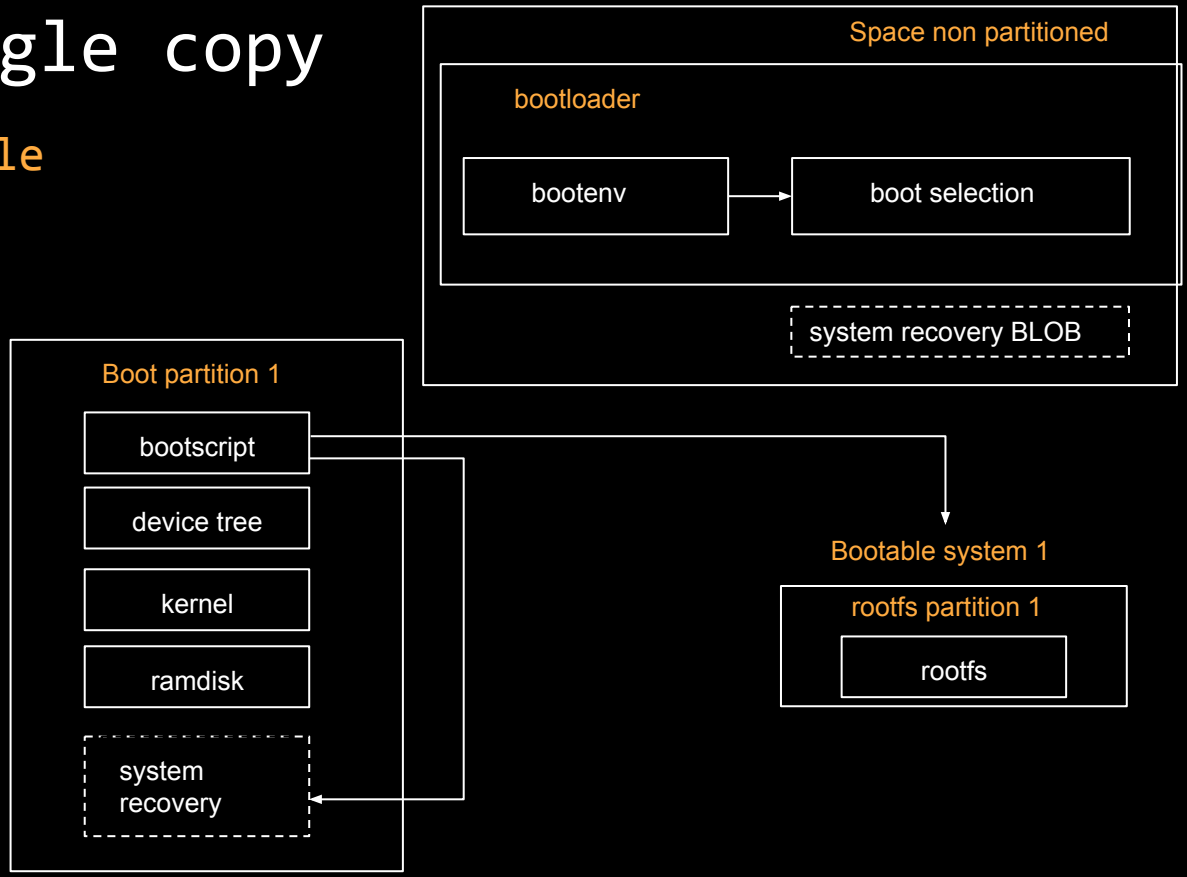
Double copy

Single Boot Partition



Single copy

Simple



Double Copy: Pros and Cons

- Pros:
  - Fallback in case of failure
  - Pretty easy to implement
- Cons:
  - Expensive in terms of storage resources, double the space
  - Requires a mechanism to switch between running and other copy if multiple partitions are doubled (e.g. boot, root)
  - Identify which copy is running

Single Copy: Pros and Cons

- Pros:
  - Requires smaller amount of space
  - “Update mode” lives in RAM
  - Can freely access whole storage (rewrite from scratch, including partition table)
  - Can be morphed to a factory reset artifact (tftpboot / USB boot)
- Cons:
  - No fallback if write fails (e.g. power interruption). Restart recovery mode to try it again
  - Simple scenario has one boot partition, kernel is shared between regular OS and Updater

Android update: approach to OTA updates

- Android approach splits the upgrade process in two phases:
  - preparation for the upgrade → performed in the full fledged Regular OS
  - execution of the upgrade → performed in a purpose built Recovery OS
- Preparation on the Device
  - Device flow:
    - registers to the cloud
    - polls for available updates
    - notifies update is available (Download? Y/n)
    - notifies update is ready to install (Proceed? Y/n)
    - reboot to Recovery OS
- Execution performed by the *recovery* binary

Android update: execution

- Bootloader/bootscript gets “reset cause” (i.MX6 Family) and boots in ramdisk-based Recovery Mode
- *recovery* starts
- *recovery* unpacks the update file provided (signed zip)
- *update-binary* executes actions in the *updater-script* (edifi)
- log and result files are written in the partition
- reboot to Regular OS

- *https://source.android.com/devices/tech/ota/device_code*
- *https://github.com/boundarydevices/android_device_boundary/commit/f069efd28d7d55e1cc298662881b9ceabb4650e3#diff-a55e09ca16b027ed99c01ca6765d9cca*

Snippet: bootscript (i.MX6)

```
+setenv bootpart 1
+
+setexpr rval *0x020CC068 \& 0x180        # get reset cause
+if itest.s "x$rval" -eq "x100"; then
+    echo "----------- run fastboot here";
+else
+    if itest.s "x$rval" -eq "x80"; then
+        setenv bootpart 2;
+    fi
+fi
+
+mw.l 0x020cc068 0 1
```

Android Update: advantages

- Single copy update featuring a recovery OS
- OTA agent runs in regular OS
  - No need to interrupt normal operation (yet)
  - Network access (e.g. pre-configured Wifi)
  - Interaction with the user (notifications / acknowledgment)
  - Full API access (Wifi or 3G/4G? Low battery?)
- Recovery has no need of network access, all artifacts are pre-fetched
- Update script support binary writing (no mount is required)
- Recovery environment is RO, minimal, isolated

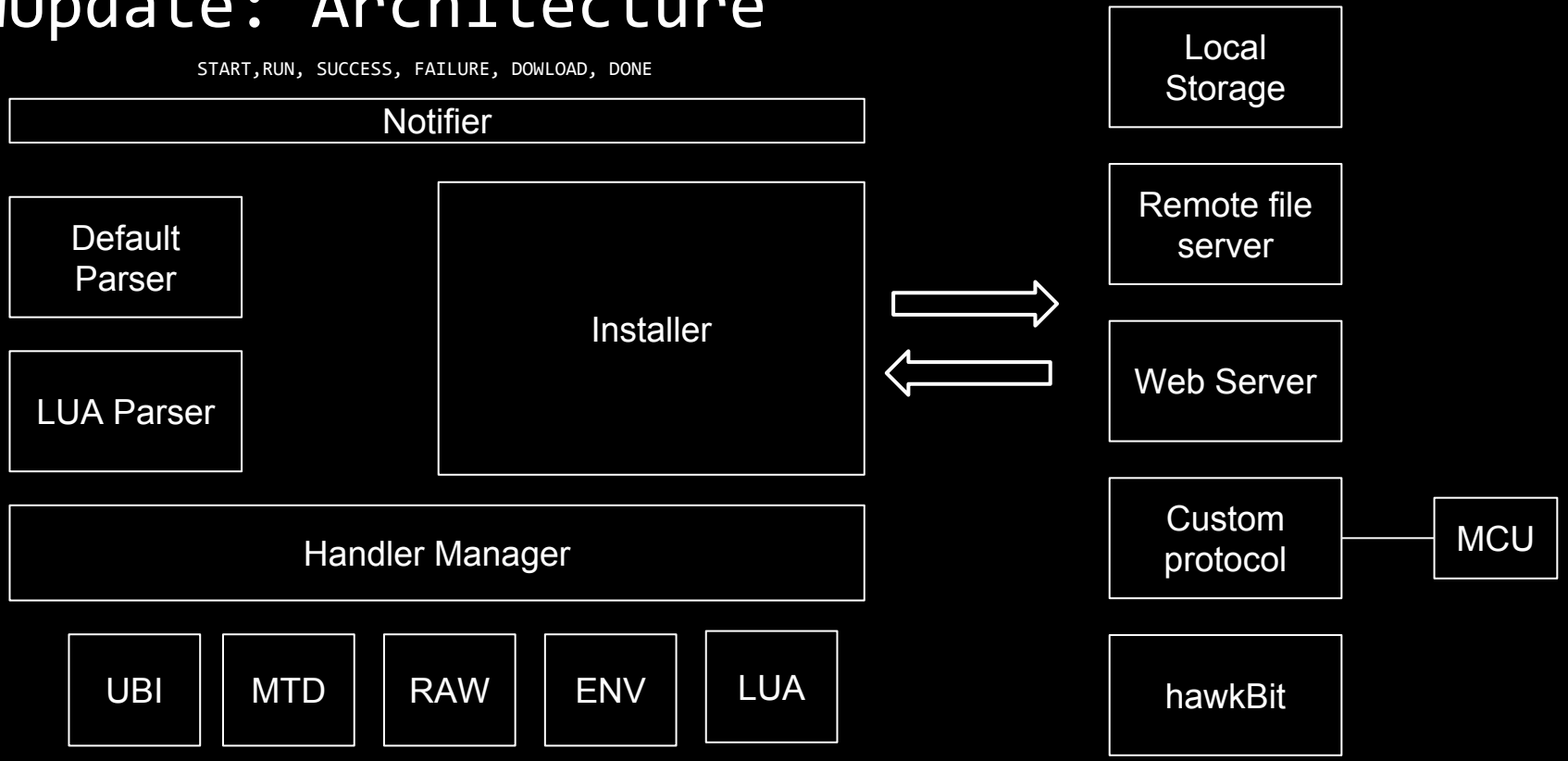
Embedded Linux like Android

- A good option for building a recovery system “Android Like” Linux is SWUpdate:
  - Written in C by Stefano Babic (Denx)
  - Run as Daemon
  - Update files (.swu) based on CPIO format
  - Several handlers (e.g. write raw data, write single file)
  - Update files scripting features (LUA)

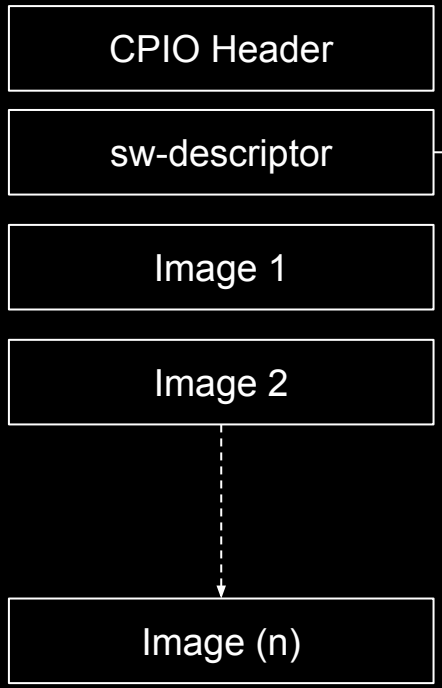
SWUpdate: features

- Local interfaces:
  - Local storage (USB, SD) as artifacts source
  - Support local peripheral devices, through USB/UART for streaming update (i.e MCU)
  - Embedded Web Server as local UI
- Remote interfaces:
  - HTTP, FTP
  - hawkBit (Suricatta embedded client)
- Signature and encryption of update files
- Handlers
  - U-boot for reading environment variables
  - Shell pre/post install scripts (also LUA)
  - Default config parser using libconfig (to parse update description file)

SWUpdate: Architecture



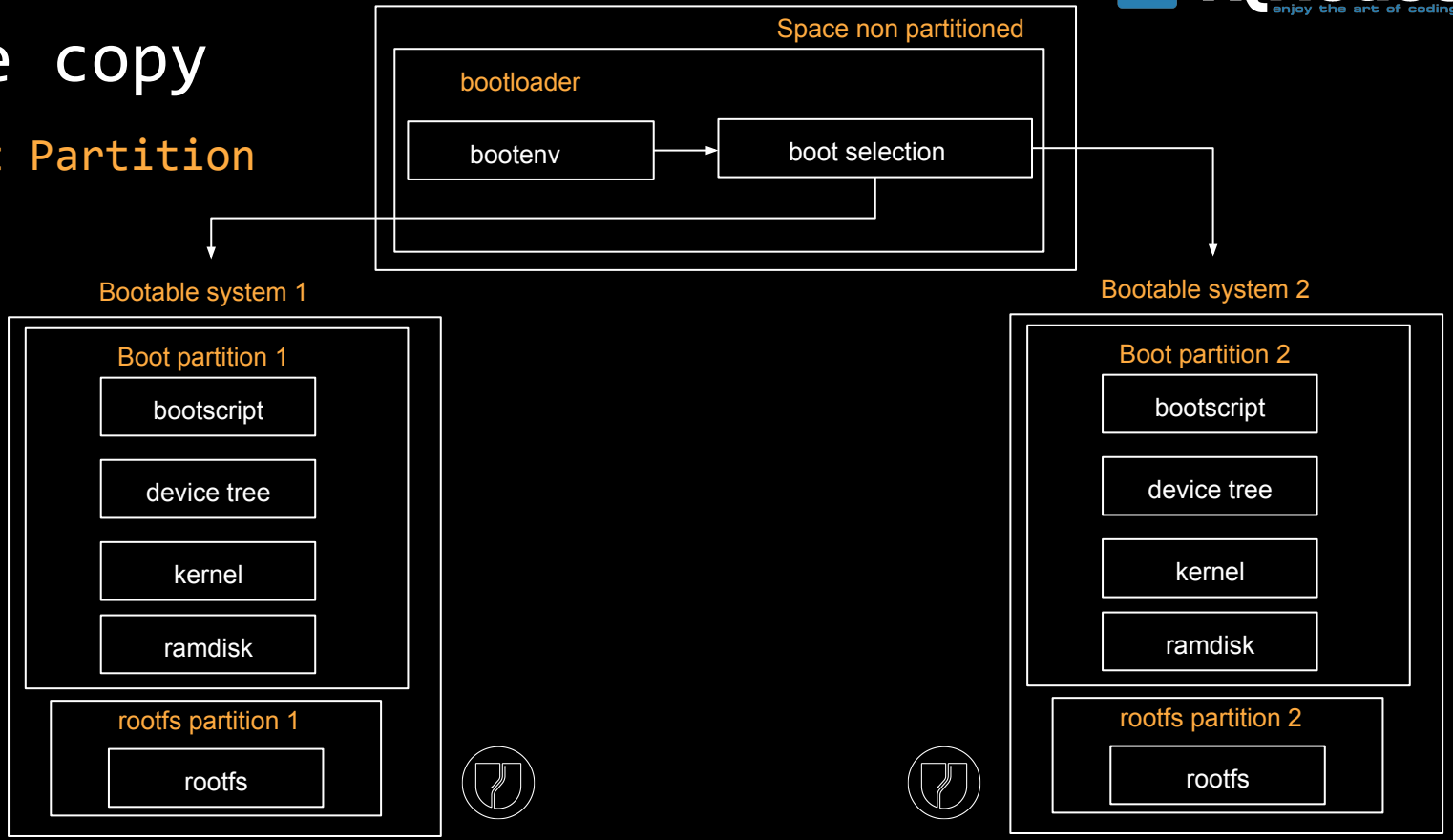
SWUpdate: single image format




```
software =
{
        version = "0.1.0";
        warp = {
        hardware-compatibility: [ "1.0"];
        Images: (
                        {
                filename = “rootfs.ext4.gz”;
                device = /dev/mmcblk0p2”:
                type = “raw”
                compressed = true:
                }
        );
        scripts:(
                    {
                filename = “update.sh”;
                type = “schellscript”;
                sha256 = “faaaa30c….”
                     }
        );
         }
}
```

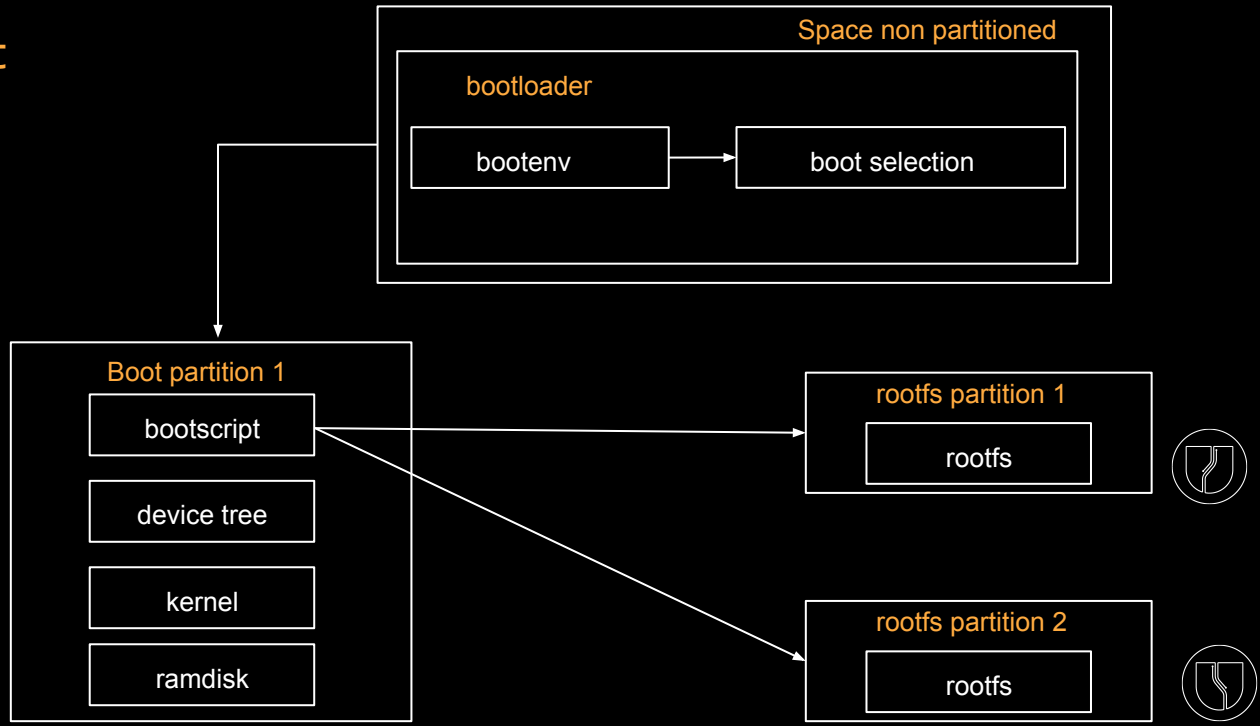
Double copy

Dual Boot Partition



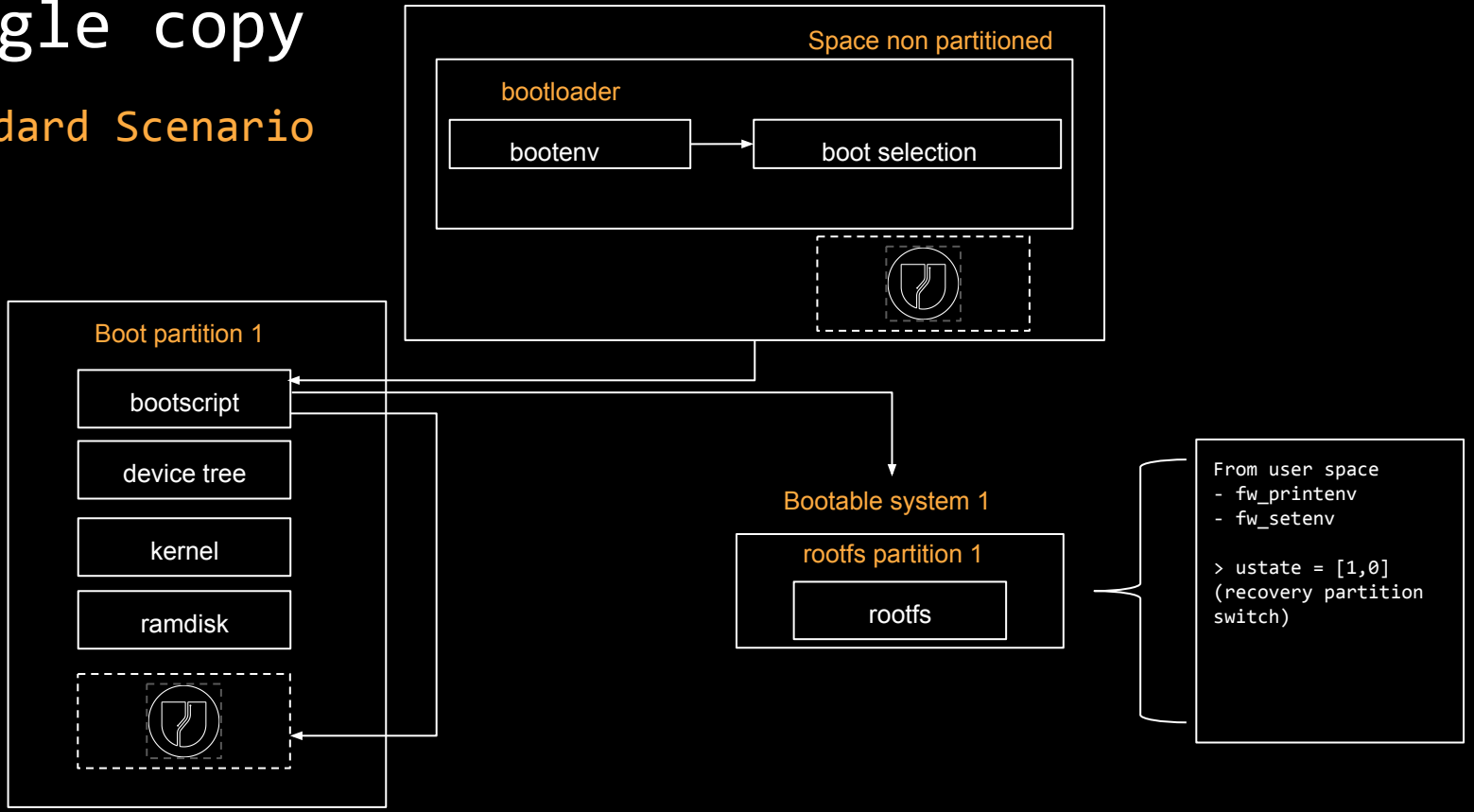
Double copy

Single Boot Partition



Single copy

Standard Scenario



Security notes

- SWUpdate combines signed sw-description with the verification of hashes for each single image.
  - RSA PKCS#1 (public/private)
  - CMS PKCS#7 (certificates)
- This means that only sighed sw-description, generated by a verified source, can be trusted by the installer.
  - sw-description.sig
  - Public.pem can be passed to SWUpdate daemon (on the device)
- sw-description contains hashes for each sub-image to verify that each delivered subimage really belongs to the release.
  - Each image inside sw-description must have the attribute “sha256”

Security notes: sign and configuration

```
#!/bin/bash

MODE="RSA"
PRODUCT_NAME="myproduct"
CONTAINER_VER="1.0"
IMAGES="rootfs kernel"
FILES="sw-description sw-description.sig $IMAGES"

#if you use RSA
if [ x"$MODE" == "xRSA" ]; then
    openssl dgst -sha256 -sign priv.pem sw-description >
sw-description.sig
else
    openssl cms -sign -in  sw-description -out sw-description.sig
-signer mycert.cert.pem \
        -inkey mycert.key.pem -outform DER -nosmimecap -binary
fi
for i in $FILES;do
        echo $i;done | cpio -ov -H crc >
${PRODUCT_NAME}_${CONTAINER_VER}.swu
```

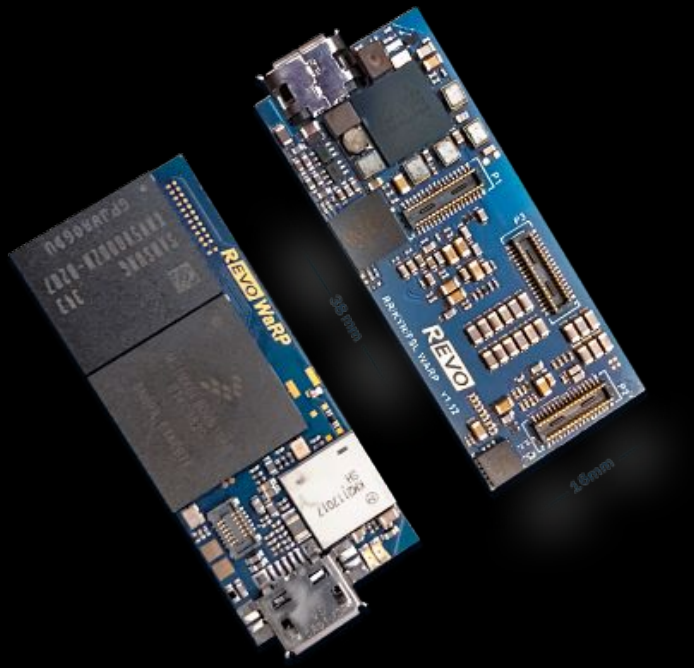
```
software =
{
        version = "0.1.0";

        hardware-compatibility: [ "revC"];

        images: (
                {
                    filename =
"core-image-full-cmdline-beaglebone.ext3";
                    device = "/dev/mmcblk0p2";
                    type = "raw";
                    sha256 =
"43cdedde429d1ee379a7d91e3e7c4b0b9ff952543a91a55bb2221e5c72cb
342b";
                }
        );
        scripts: (
                {
                    filename = "install.sh";
                    type = "shellscript";
                    sha256 =
"f53e0b271af4c2896f56a6adffa79a1ffa3e373c9ac96e00c4cfc577b9be
a5f1";
                 }
        );
}
```

Security notes (2)

- SWUpdate supports encrypted images
  - SWUpdate allows to symmetrically encrypt update images using the 256 bit AES block cipher in CBC mode
  - encrypted = true parameter in sw-description

```
software =
{
        version = "0.0.1";
        images: ( {
                        filename = "core-image-full-cmdline-beaglebone.ext3.enc";
                        device = "/dev/mmcblk0p3";
                        encrypted = true;
                }
        );
}
```

Case Study: Warp board



- Small wearable reference platform
- Community: www.warpx.io
- Support for SWUpdate for OS updates
- Single image
  - From bootloader, flash stand alone SWUpdate OS Image on the eMMC
    - (UMS): dd img file
    - mmc read ${initrd_addr} 0x2000 0xAA80
  - Boot the SWUpdate OS image
  - Load module for USB over ethernet
  - From a host use browser and upload the SWU image

Part 2: Eclipse hawkBit

The Eclipse Foundation has been very active in promoting significative projects for the IoT, in particular under the umbrella of the Eclipse IoT community.

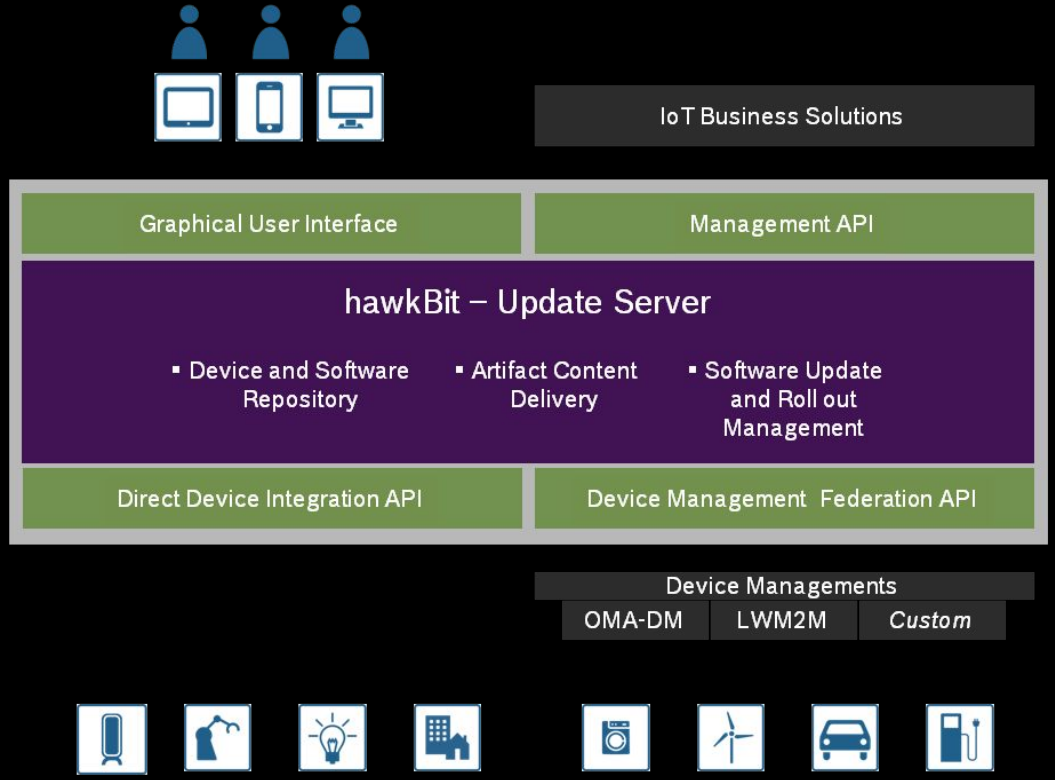
Eclipse IoT is an ecosystem of companies and individuals that are working together to establish an Internet of Things based on open technologies.

https://iot.eclipse.org, https://eclipse.org/hawkbit/

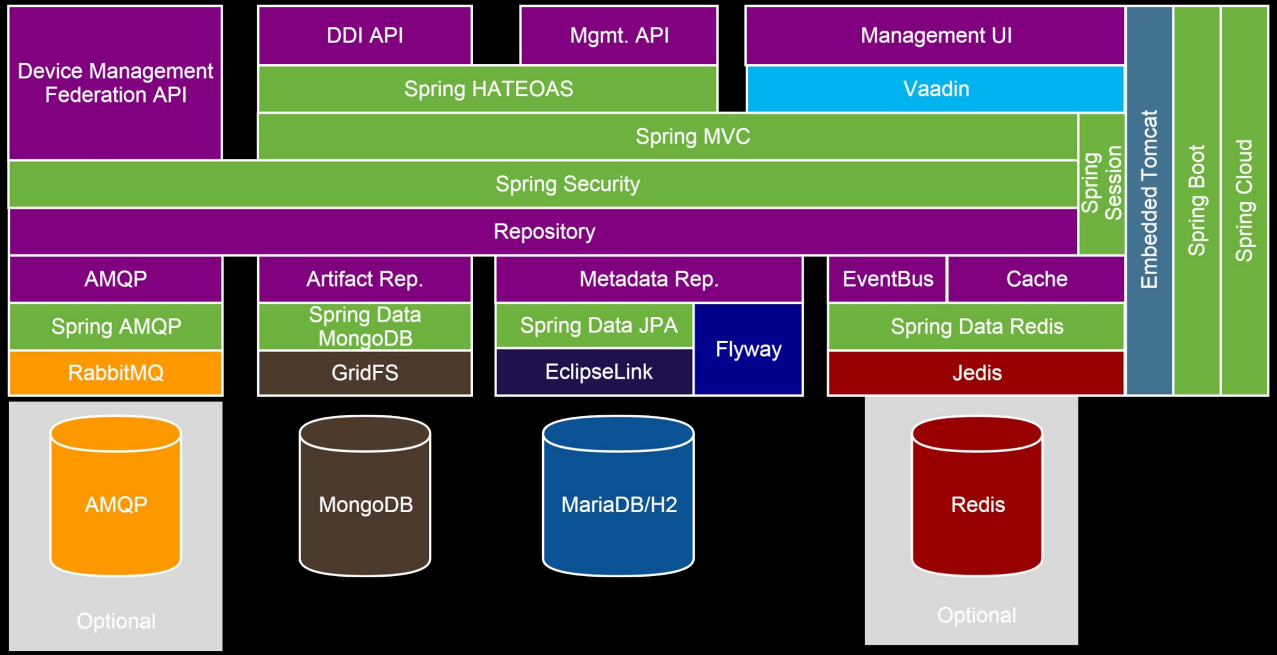
One of the (many) projects is hawkBit *"to create a domain independent back end solution for rolling out software updates to constrained edge devices connected to IP based networking infrastructure"*

hawkBit overview

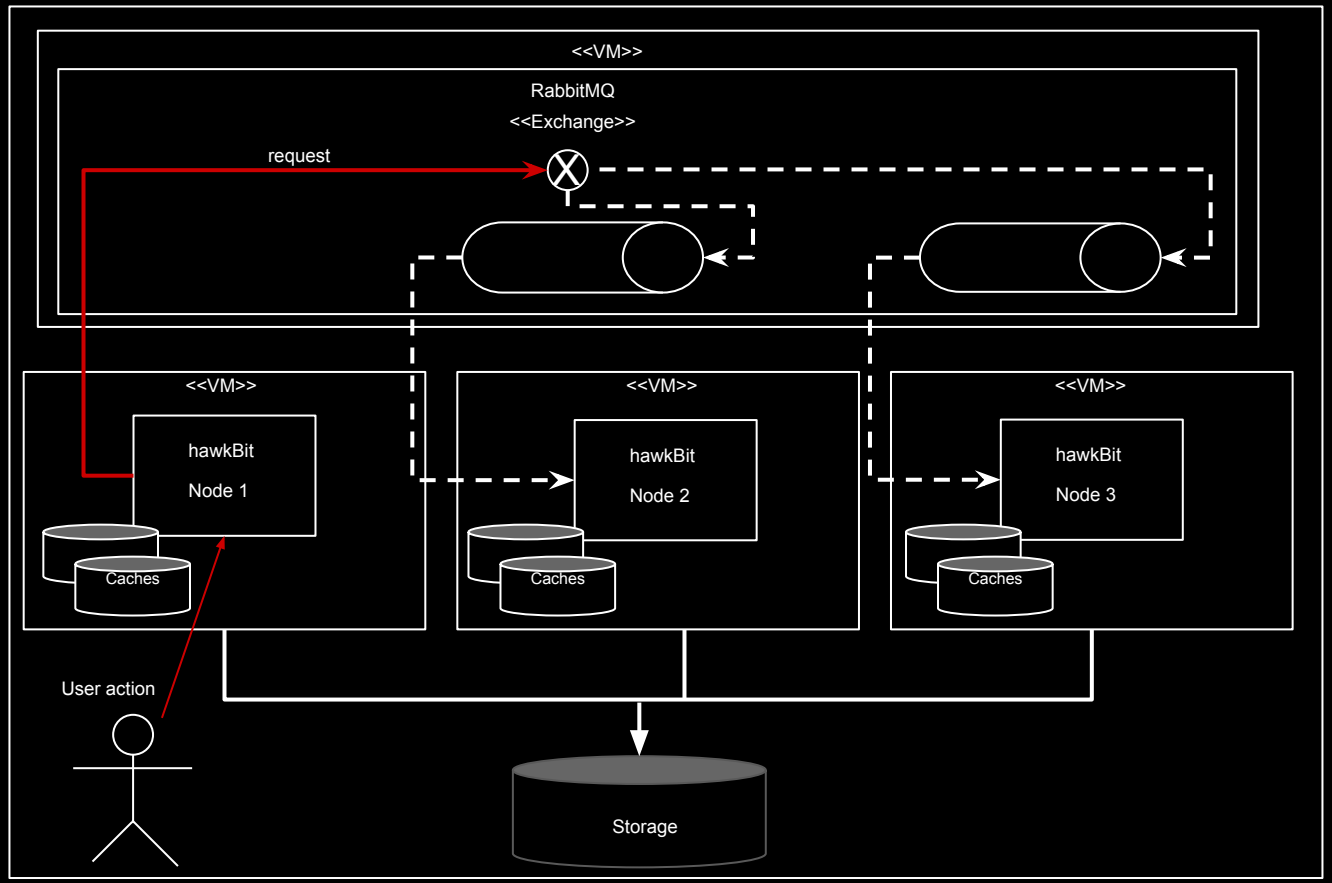
- User/Applications
  - UI
  - MGMT (API)
- Devices
  - DDI (HTTP/REST/JSON)
  - DMF (AMQP)



hawkBit Architecture



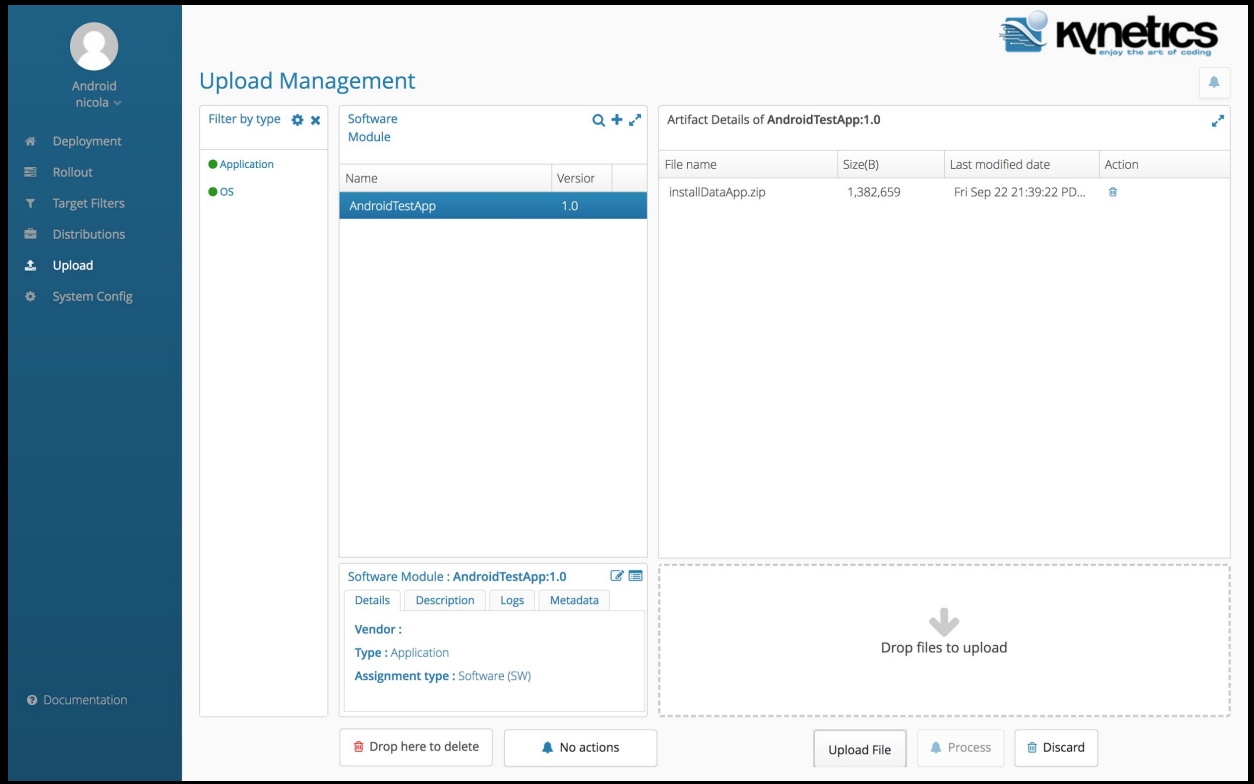
Clustering


<<VM>>
RabbitMQ
<<Exchange>>
request
<<VM>>
hawkBit
Node 1
Caches
<<VM>>
hawkBit
Node 2
Caches
<<VM>>
hawkBit
Node 3
Caches
User action
Storage

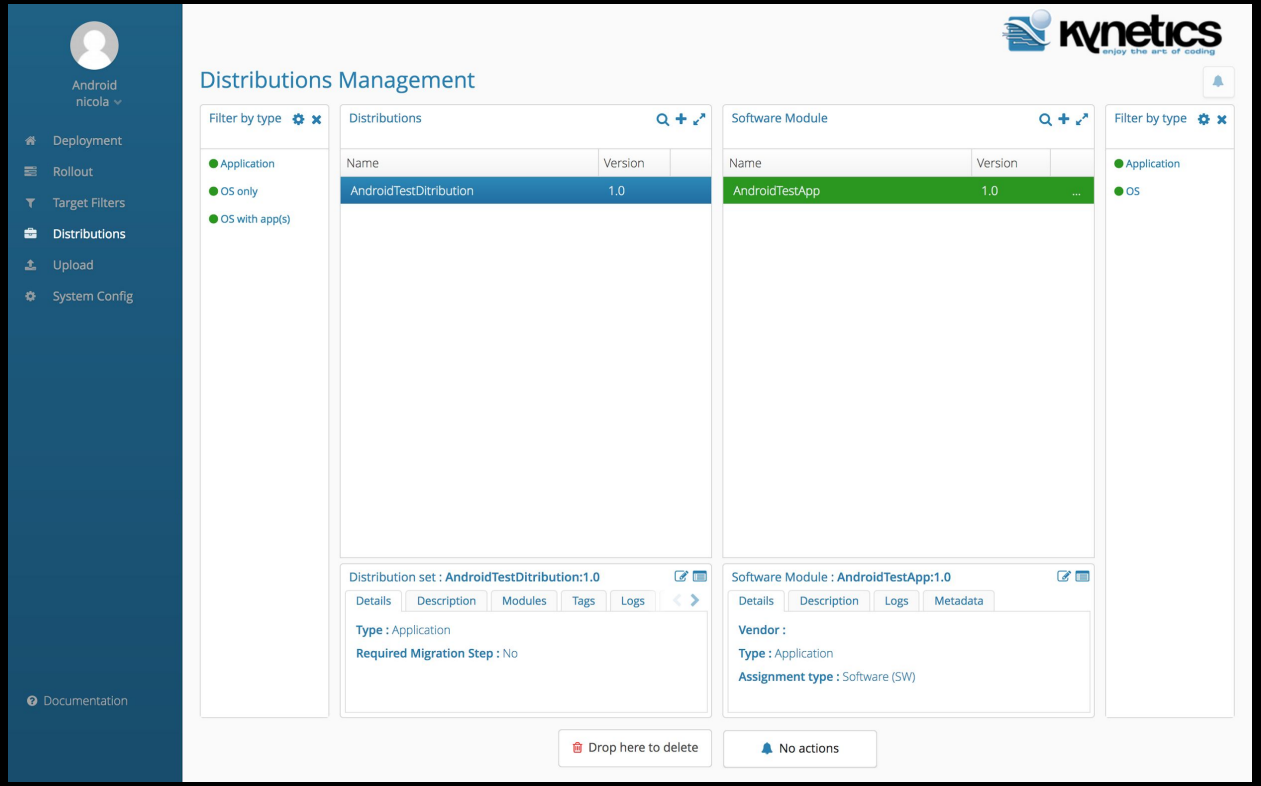
hawkBit: workflow of a rollout campaign

- Prepare the update file and upload it
- Create a Software Module and add an artifact to it
- Create a Distribution
- Rollout a distribution to Targets
- Targets features:
  - Attributes (i.e HW revision, custom)
  - Tags (for grouping purposes)
  - Others like device description, what installed, logs, etc..
- Rollouts can be managed by groups
  - TAG filter
  - Group threshold

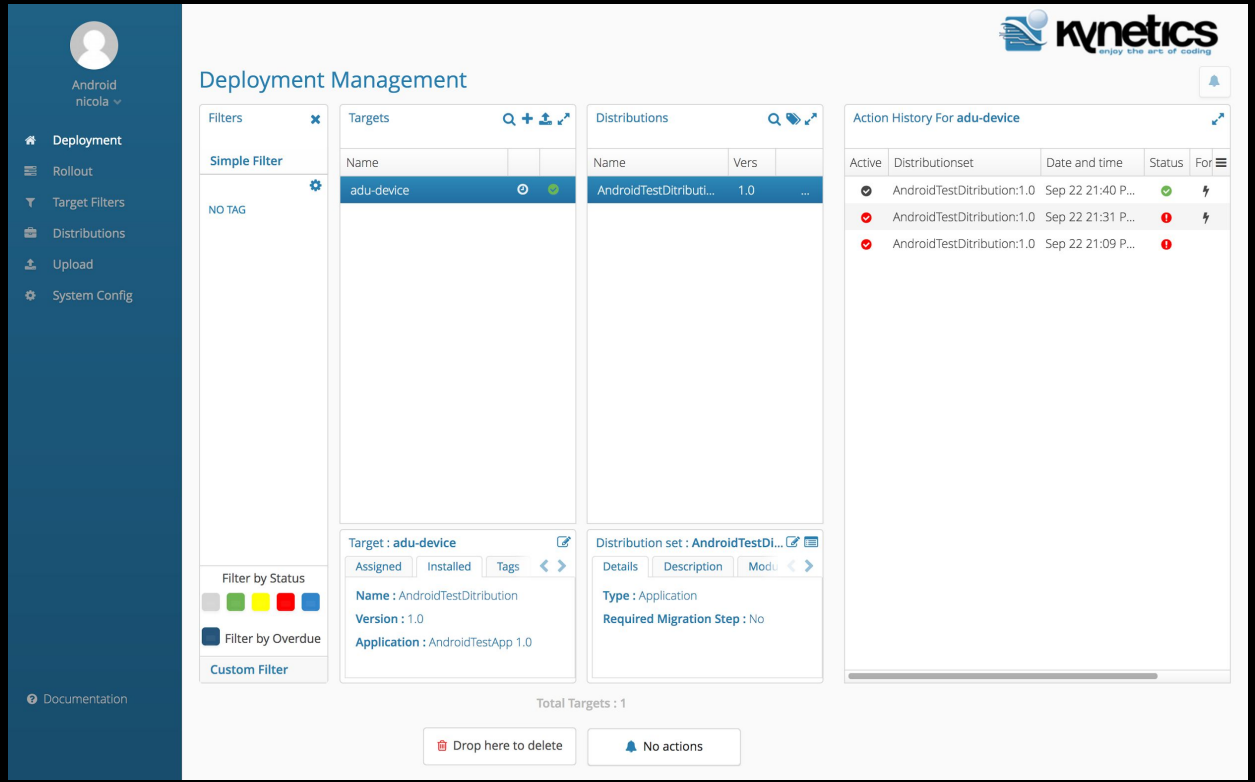
Artifacts and Modules



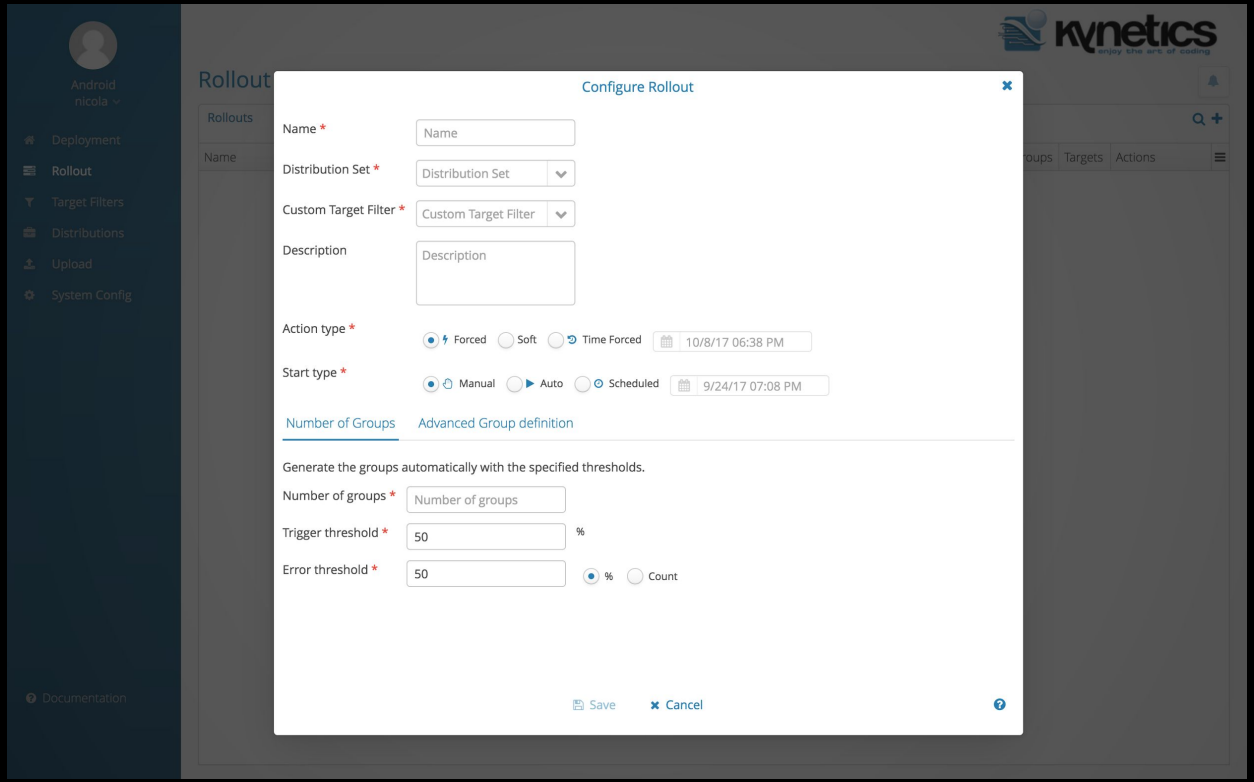
Distributions



Deploy Management



Rollout Configuration



Update Factory

Platform to manage and deliver software update artifacts which are deployed on single copy Linux and Android devices, featuring recovery mode

Or simply….

“Manage and Deploy Android-like software updates on Embedded Linux!”

Update Factory Architecture

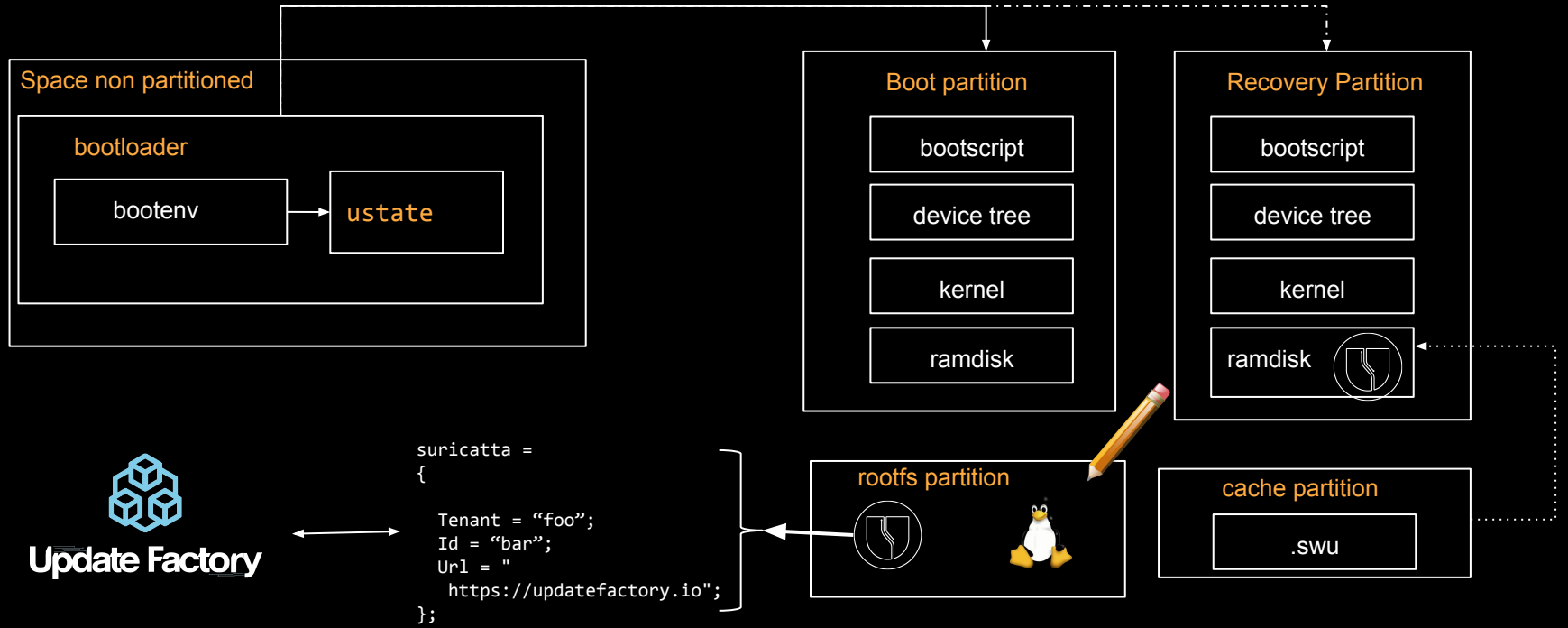
- Service on the embedded device
  - Gnu/Linux featuring SWUpdate
  - Android Service featuring Update Server API
- Update Server featuring hawkBit™
- IAM Server
- Artifact Repository
- Metadata Repository
- MsgBroker

Android “like” behaviour on Embedded Linux

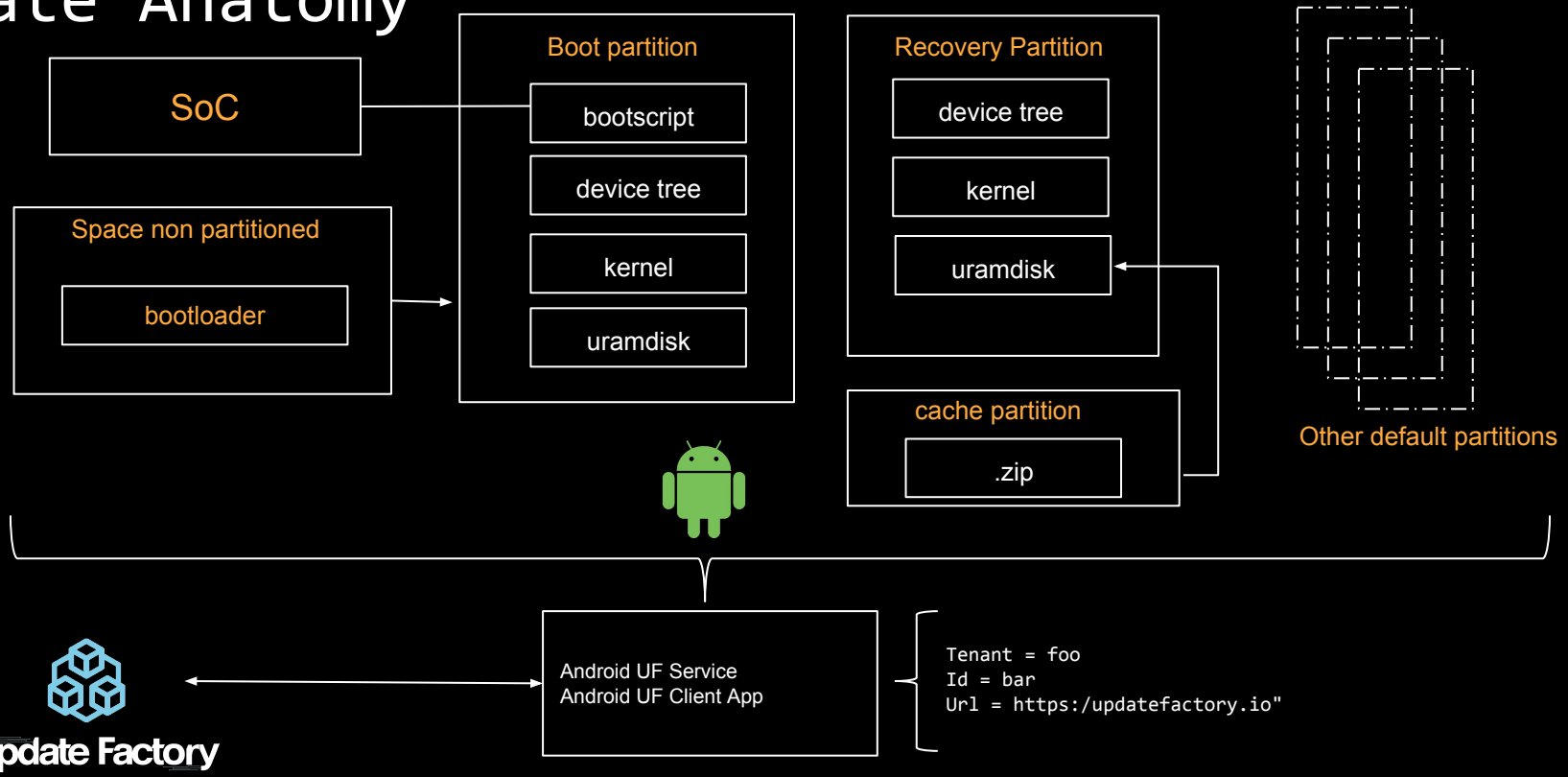
Update Factory implements all the missing bits to have an Android-like OTA mechanism on an Embedded Linux OS

- Device to cloud communication
- Recovery partition
- Recovery ramdisk
- Recovery bootscript
- Bootloader coordination (boot selector using *ustate* env var)
- Device updating status to the cloud

UF Update Anatomy



Update Anatomy



Update Factory goals

- Support medium scale general purpose CPU-SOC deployments
- Android like OTA update strategy for Embedded Linux based on single image approach
- Create a neutral platform to support both Linux and Android devices
- Provide a solid integration with Yocto to facilitate the adoption
- Remote Update Management Platform as a service
- Free Tier

Links

- https://www.kynetics.com/update-factory
- https://docs.updatefactory.io/
- https://github.com/Kynetics/meta-updatefactory
- http://warpx.io/blog/tutorial/easy-os-upgrades-swupdate

- https://eclipse.org/hawkbit/
- https://sbabic.github.io/swupdate
- https://android.googlesource.com/platform/bootable/recovery/+/android-8.0.0_r4/recovery.cpp#167

Thank you.

Contacts:

USA Kynetics LLC 2040 Martin Ave, Santa Clara CA 95050 Ph: +1 (408) 475 7760	Italy Kynetics Srl Via G. Longhin, Padova (PD) 35129 Ph: +39 (049) 781 1091
info@kynetics.com | www.kynetics.com	