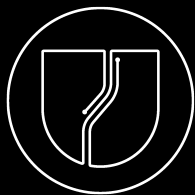


Update Remotely IoT Devices using Eclipse hawkBit and SWUpdate

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:

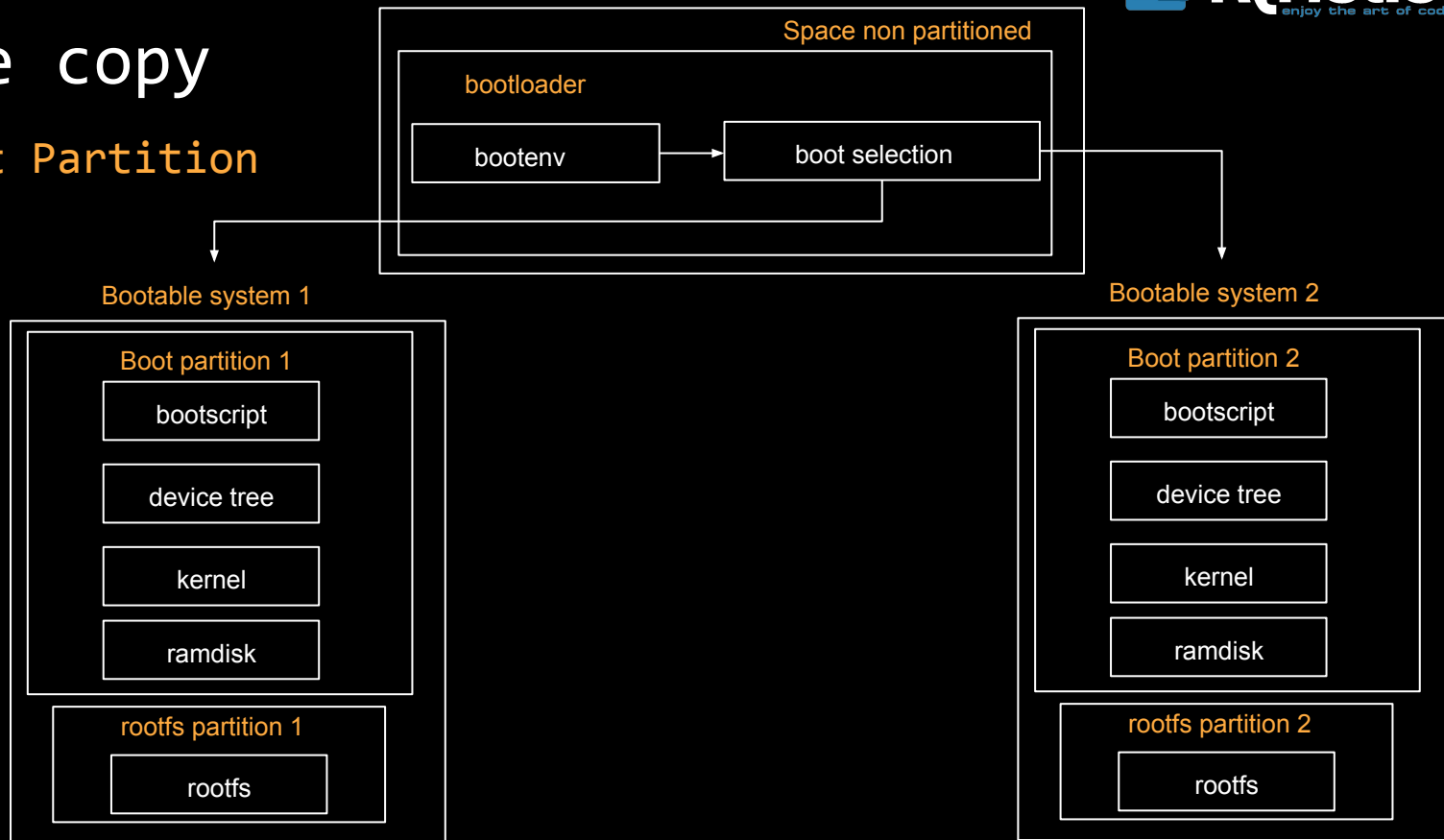
- » The device features **two** copy of the Application/OS/RootFS
- » Each copy must contain the kernel, the root file system, and each further component that can be updated.
- » Cooperation with the boot loader is necessary, to decide which copy should be booted

› 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*.

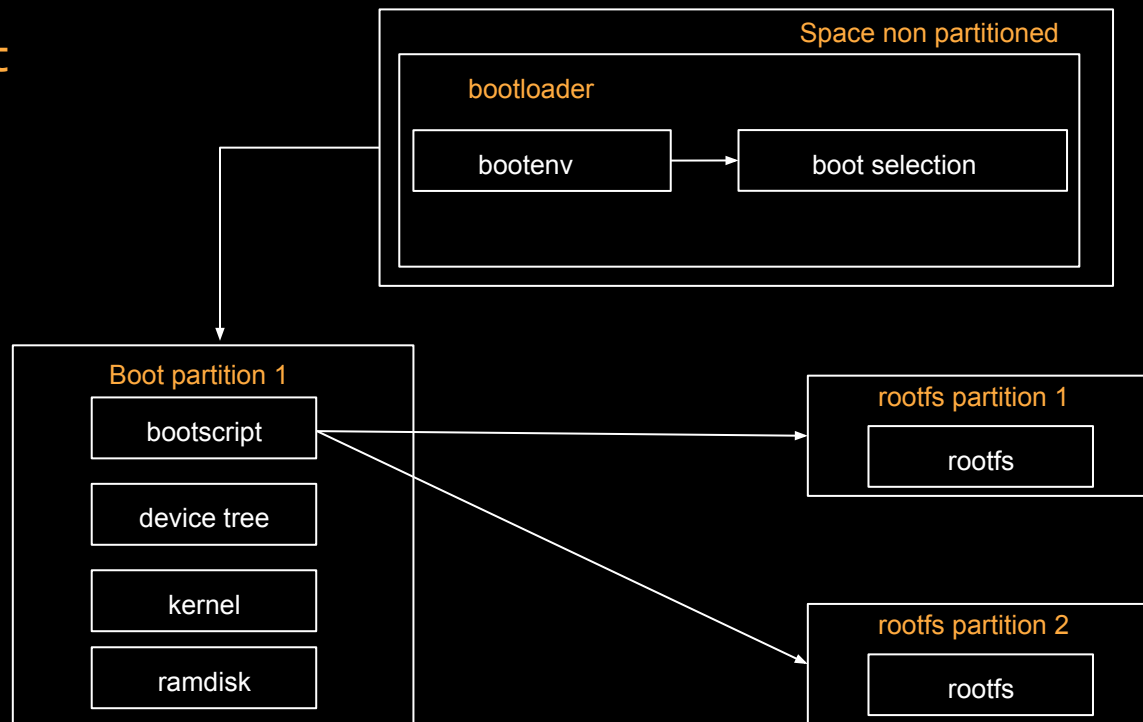
Double copy

Dual Boot Partition



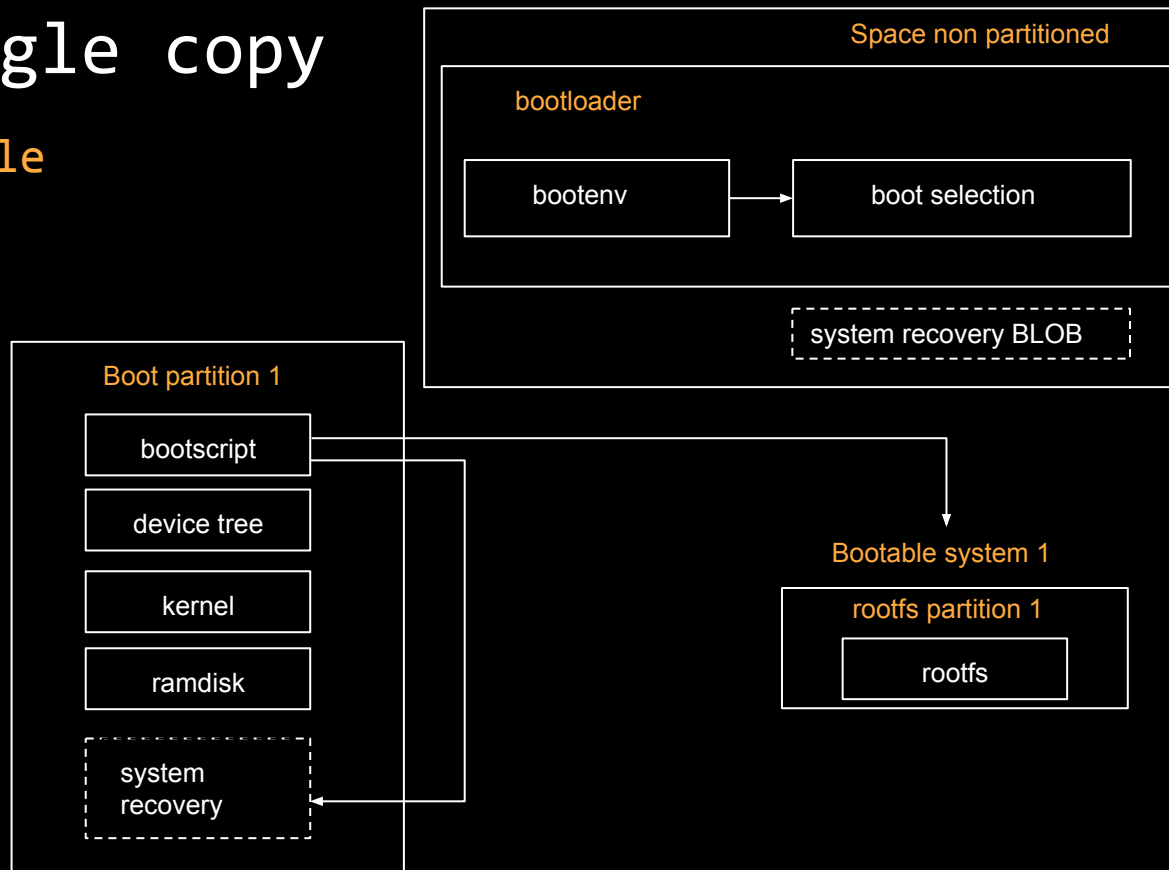
Double copy

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 (tftpbboot / 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 "$rval" -eq "x100"; then
+    echo "----- run fastboot here";
+else
+    if itest.s "$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 (**Suricata** embedded client)

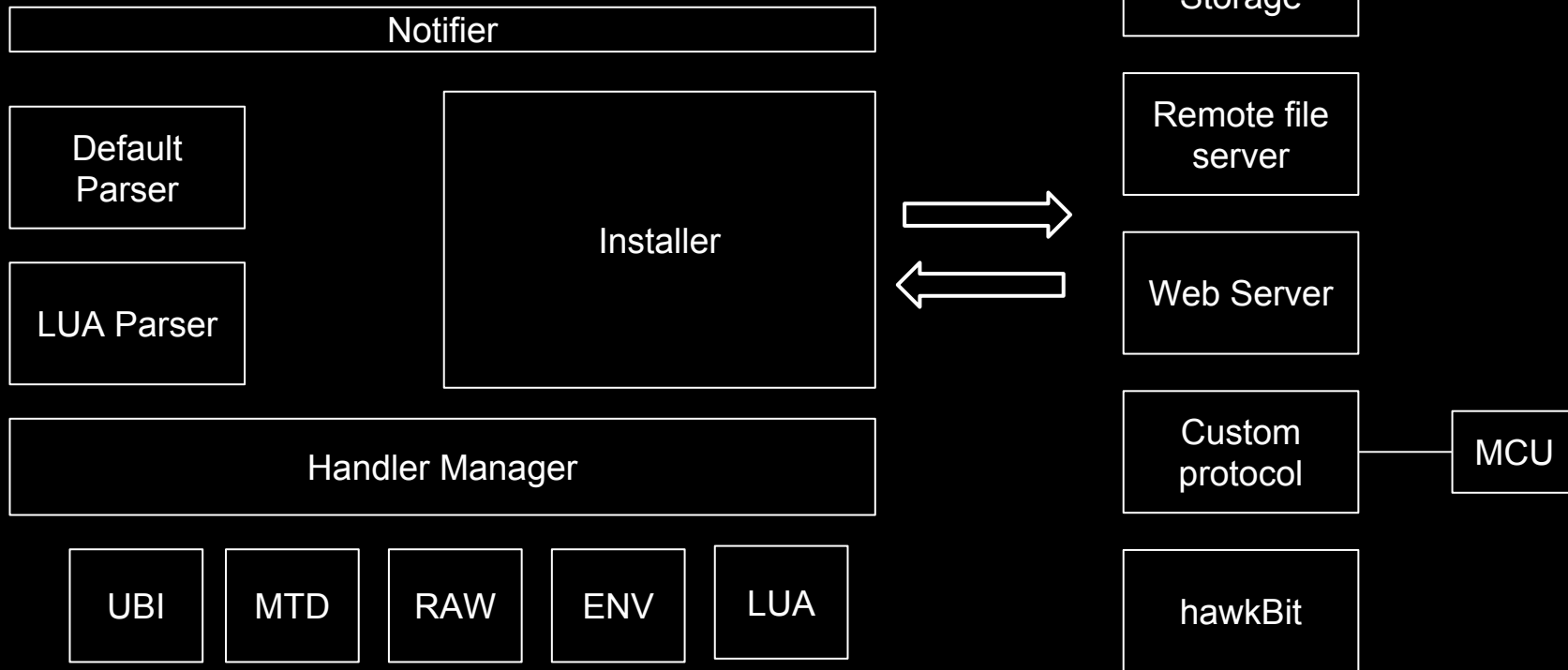
› Signature and encryption of update files

› Handlers

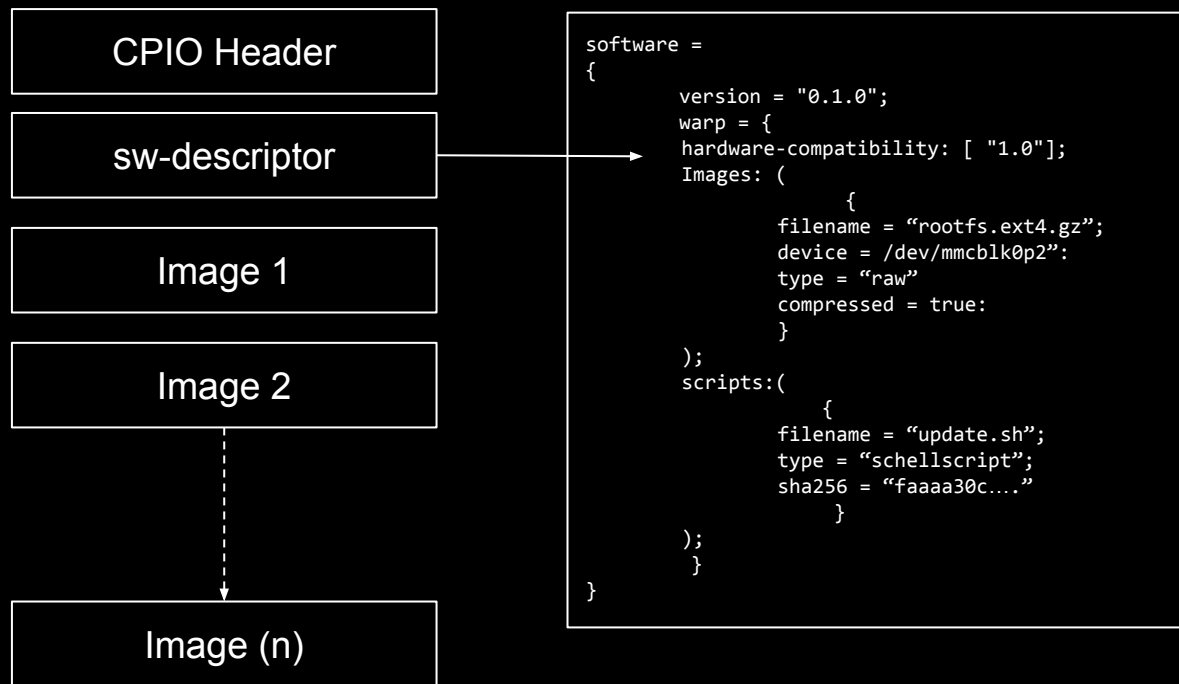
- › U-boot for reading environment variables
- › Shell pre/post install scripts (also LUA)
- › Default config parser using libconfig (to parse update description file)

SWUpdate: Architecture

START, RUN, SUCCESS, FAILURE, DOWLOAD, DONE

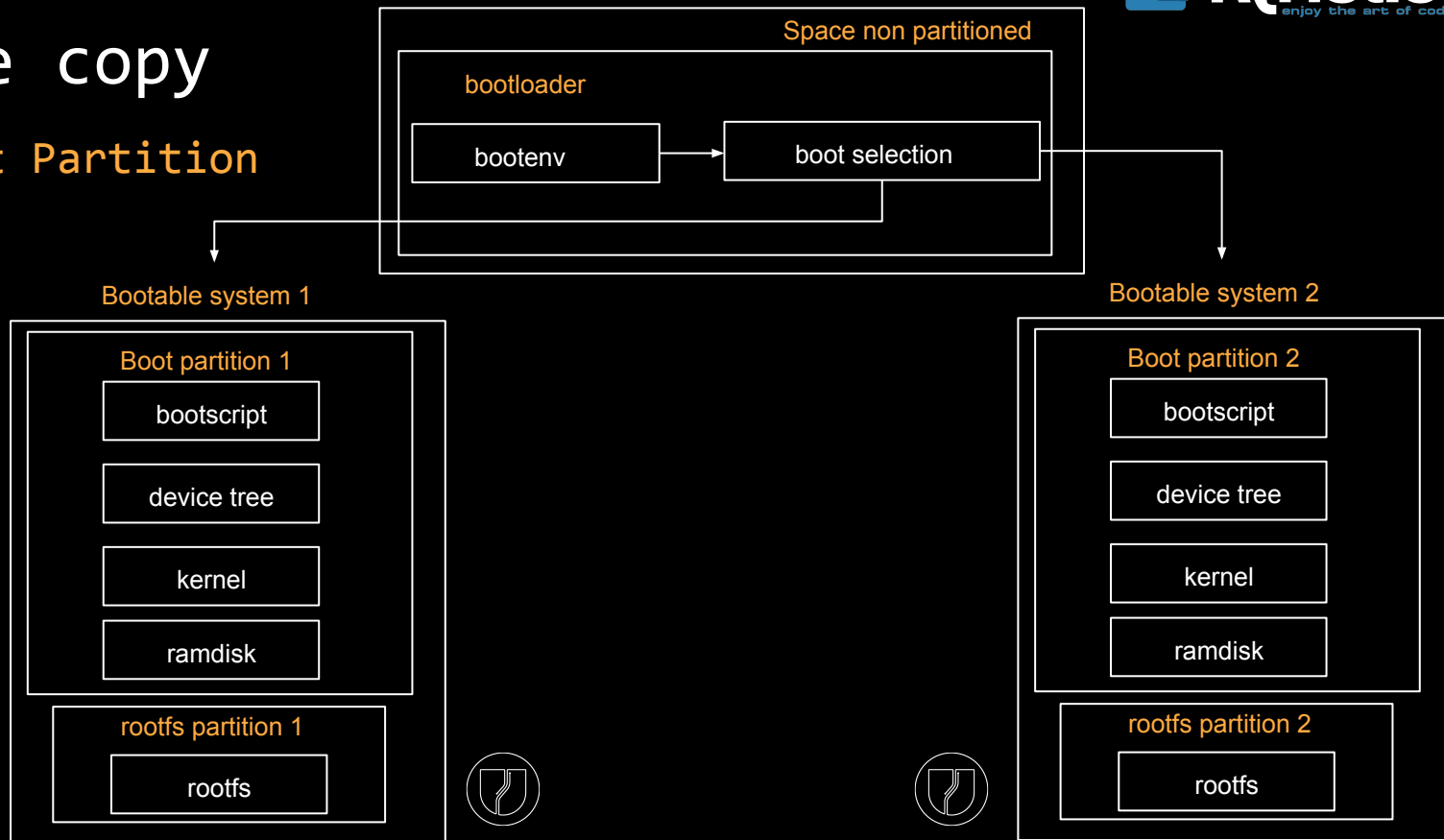


SWUpdate: single image format



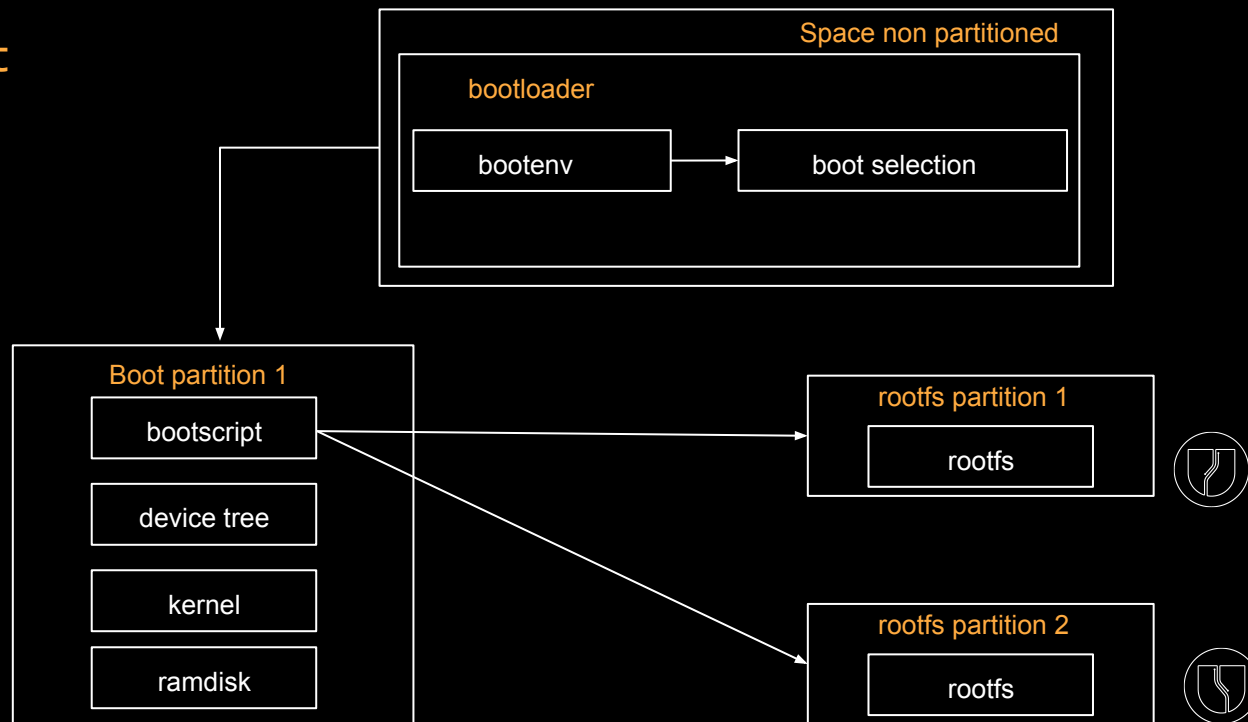
Double copy

Dual Boot Partition



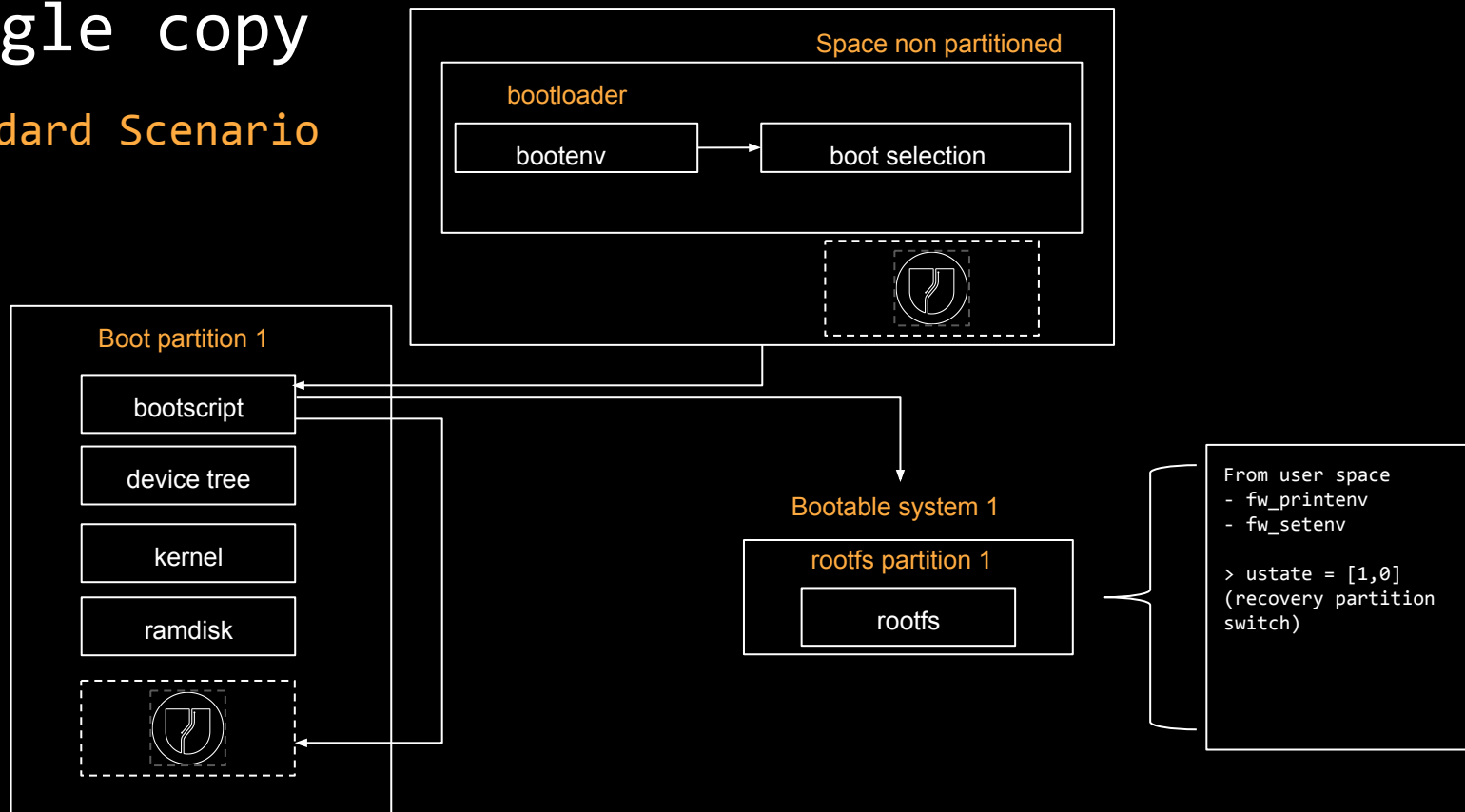
Double copy

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 signed 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 significant 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/hawkbite/>



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”

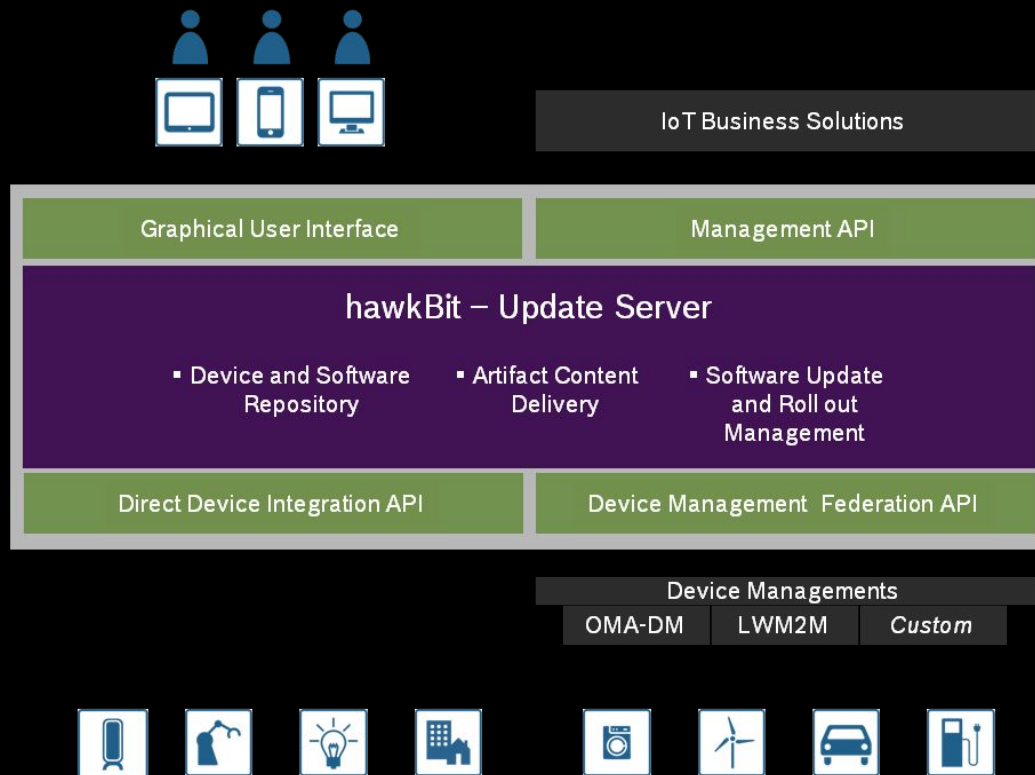
hawkBit overview

› User/Applications

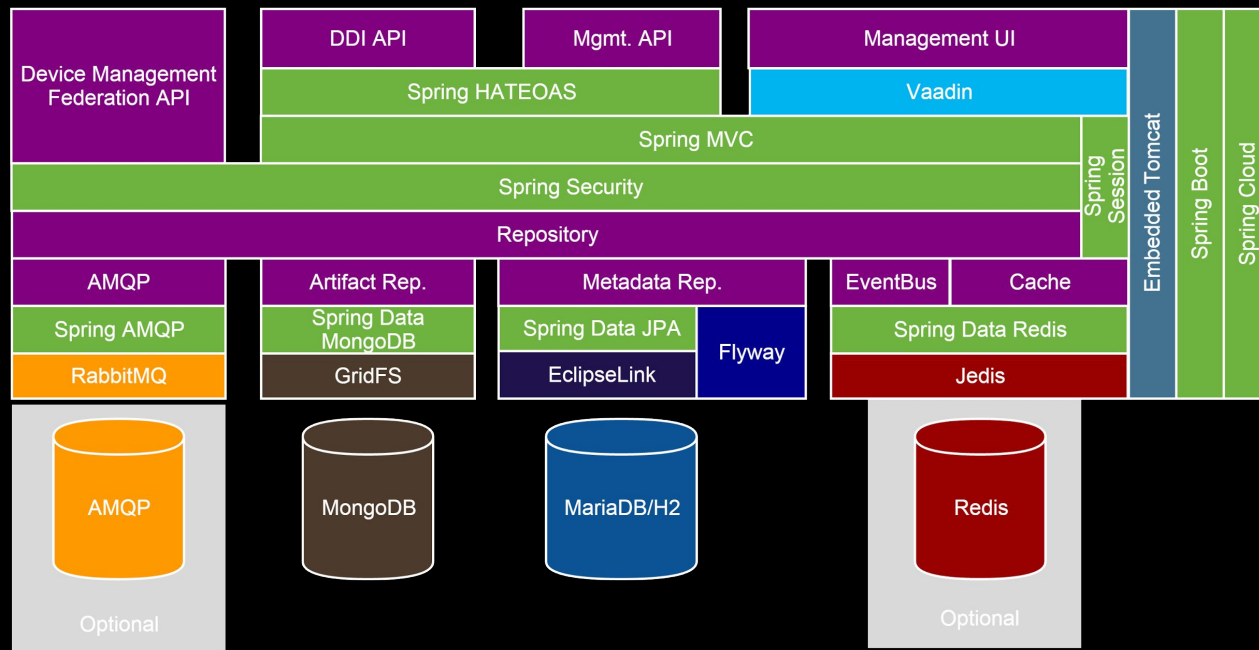
- › UI
- › MGMT (API)

› Devices

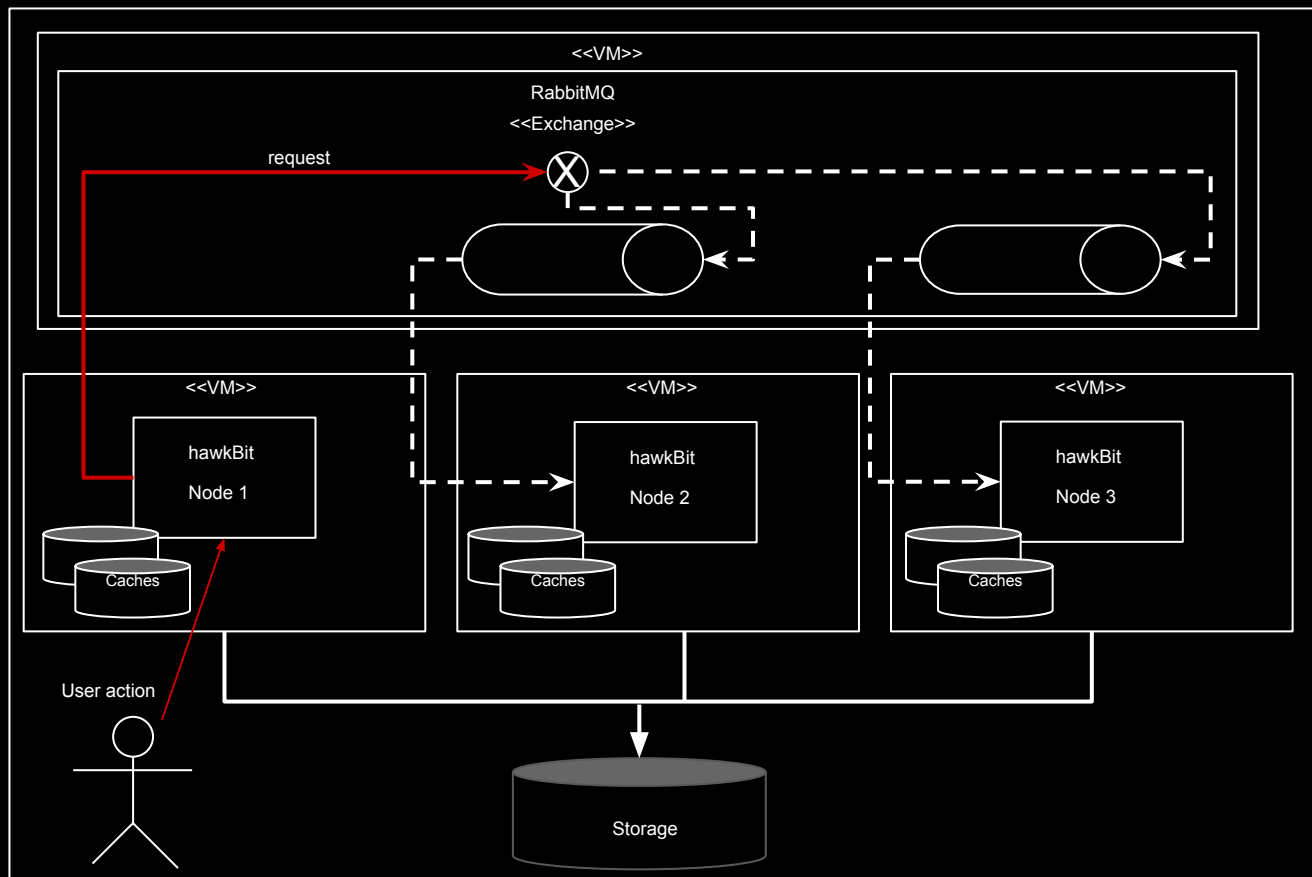
- › DDI
(HTTP/REST/JSON)
- › DMF (AMQP)



hawkBit Architecture



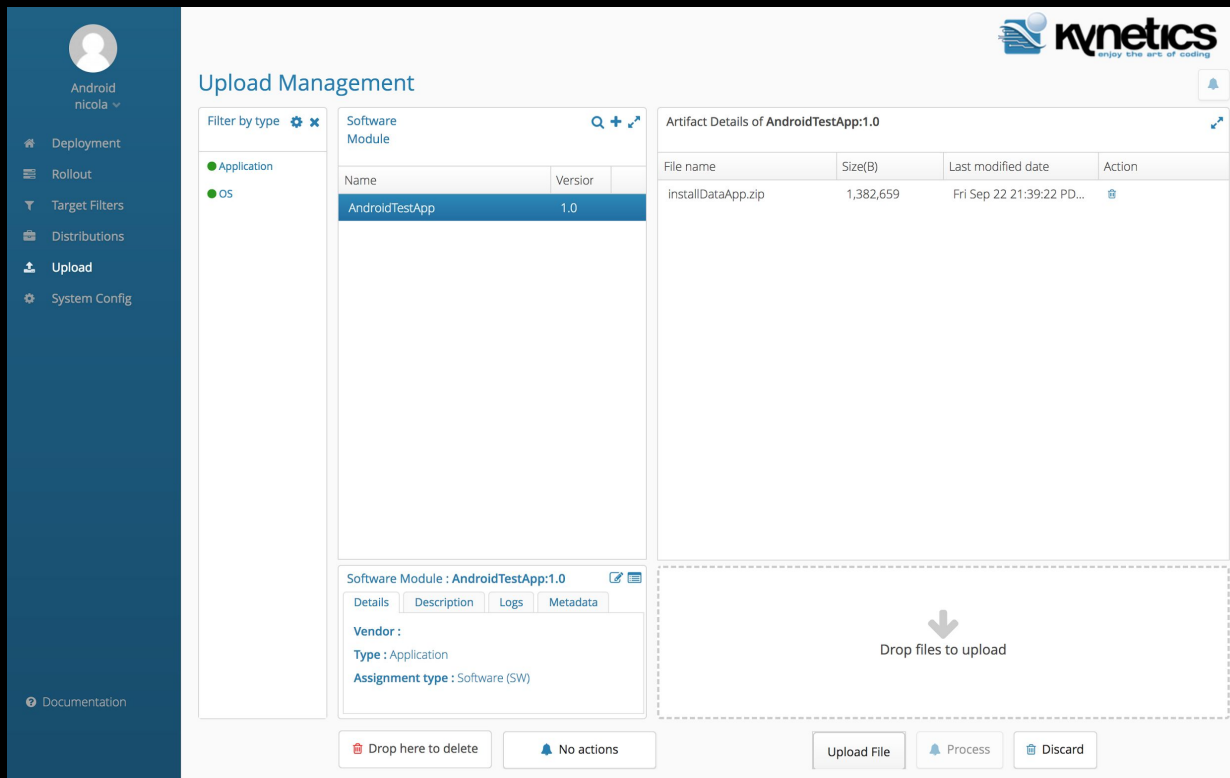
Clustering



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



The screenshot shows the Kynetics Upload Management interface. On the left is a dark blue sidebar with a user profile for 'Android nicola' and a menu with options: Deployment, Rollout, Target Filters, Distributions, Upload (highlighted), and System Config. At the bottom of the sidebar is a 'Documentation' link. The main content area is titled 'Upload Management' and features a 'Filter by type' section with 'Application' and 'OS' filters. Below this is a table of 'Software Module' entries. The first entry, 'AndroidTestApp', is selected and highlighted in blue. Below the table, there is a section for 'Software Module : AndroidTestApp:1.0' with tabs for 'Details', 'Description', 'Logs', and 'Metadata'. The 'Details' tab is active, showing 'Vendor:', 'Type : Application', and 'Assignment type : Software (SW)'. To the right of the table is a section titled 'Artifact Details of AndroidTestApp:1.0' containing a table with columns 'File name', 'Size(B)', 'Last modified date', and 'Action'. The table lists one artifact: 'installDataApp.zip' with a size of '1,382,659' and a last modified date of 'Fri Sep 22 21:39:22 PD...'. Below this table is a large dashed box with a downward arrow and the text 'Drop files to upload'. At the bottom of the interface are three buttons: 'Drop here to delete', 'No actions', and 'Upload File'. To the right of the 'Upload File' button are two more buttons: 'Process' and 'Discard'.

Android
nicola

- Deployment
- Rollout
- Target Filters
- Distributions
- Upload**
- System Config

Documentation

Upload Management

Filter by type

- Application
- OS

Name	Version
AndroidTestApp	1.0

Software Module : AndroidTestApp:1.0

Details Description Logs Metadata

Vendor:

Type : Application

Assignment type : Software (SW)

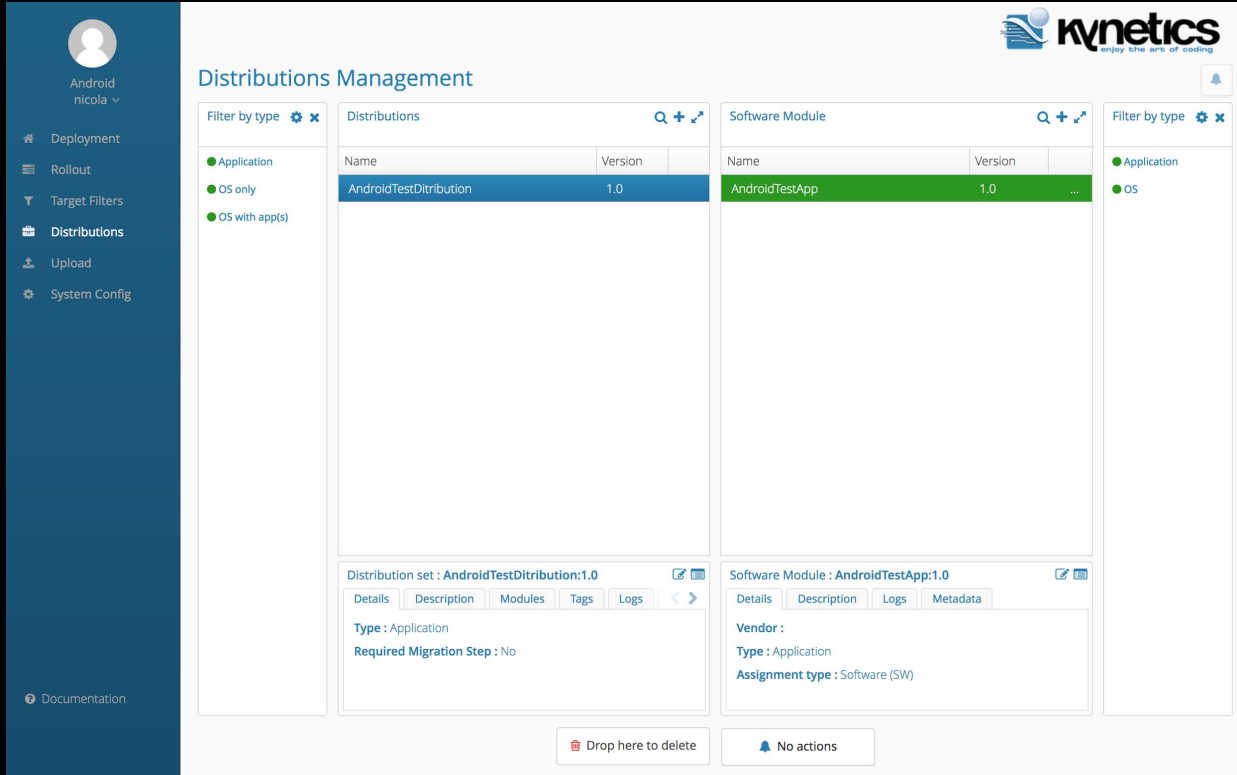
Artifact Details of AndroidTestApp:1.0

File name	Size(B)	Last modified date	Action
installDataApp.zip	1,382,659	Fri Sep 22 21:39:22 PD...	

Drop files to upload

Drop here to delete No actions Upload File Process Discard

Distributions



The screenshot shows the 'Distributions Management' interface in the Kynetics application. On the left is a dark blue sidebar with a user profile 'Android nicola' and a menu containing: Deployment, Rollout, Target Filters, **Distributions**, Upload, and System Config. At the bottom of the sidebar is a 'Documentation' link. The main content area has a white background with the Kynetics logo and a notification bell in the top right. The title 'Distributions Management' is centered at the top of the main area. Below the title are two panels: 'Distributions' and 'Software Module'. Each panel has a 'Filter by type' dropdown (set to 'Application') and a table. The 'Distributions' table has columns 'Name' and 'Version', with one entry: 'AndroidTestDitribution' (note the typo) with version '1.0'. The 'Software Module' table has columns 'Name' and 'Version', with one entry: 'AndroidTestApp' with version '1.0'. Below each table is a details section. For the 'Distributions' entry, the details show 'Distribution set : AndroidTestDitribution:1.0' and tabs for 'Details', 'Description', 'Modules', 'Tags', and 'Logs'. The 'Details' tab is active, showing 'Type : Application' and 'Required Migration Step : No'. For the 'Software Module' entry, the details show 'Software Module : AndroidTestApp:1.0' and tabs for 'Details', 'Description', 'Logs', and 'Metadata'. The 'Details' tab is active, showing 'Vendor :', 'Type : Application', and 'Assignment type : Software (SW)'. At the bottom of the interface are two buttons: 'Drop here to delete' and 'No actions'.

Distributions Management

Distributions

Name	Version
AndroidTestDitribution	1.0

Software Module

Name	Version
AndroidTestApp	1.0

Distribution set : AndroidTestDitribution:1.0

Details | Description | Modules | Tags | Logs

Type : Application
Required Migration Step : No

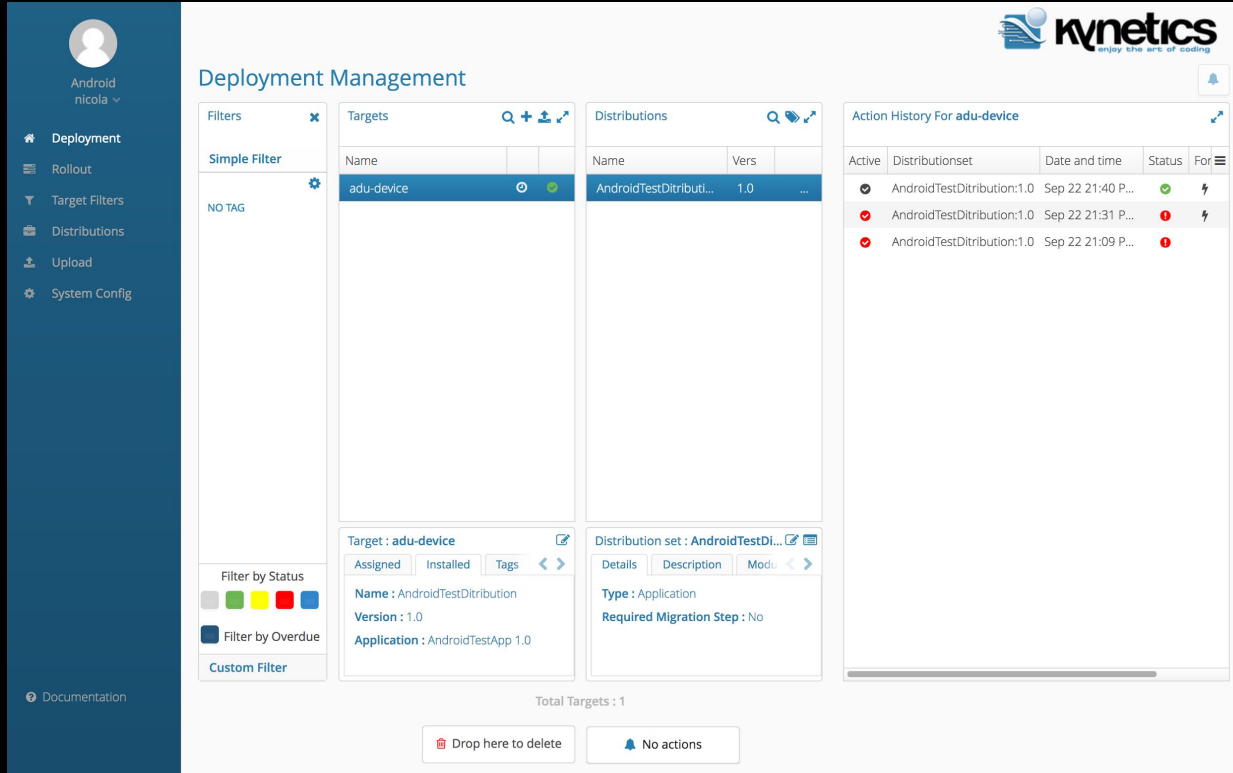
Software Module : AndroidTestApp:1.0

Details | Description | Logs | Metadata

Vendor :
Type : Application
Assignment type : Software (SW)

Drop here to delete | No actions

Deploy Management



The screenshot shows the Kynetics Deployment Management interface. On the left is a sidebar with a user profile (Android nicola) and navigation links: Deployment, Rollout, Target Filters, Distributions, Upload, and System Config. The main area is titled 'Deployment Management' and contains three panels: Filters, Targets, and Distributions.

Filters Panel: Shows 'Simple Filter' with 'NO TAG' and a 'Filter by Status' section with color-coded buttons (grey, green, yellow, red, blue) and a 'Filter by Overdue' button. A 'Custom Filter' button is at the bottom.

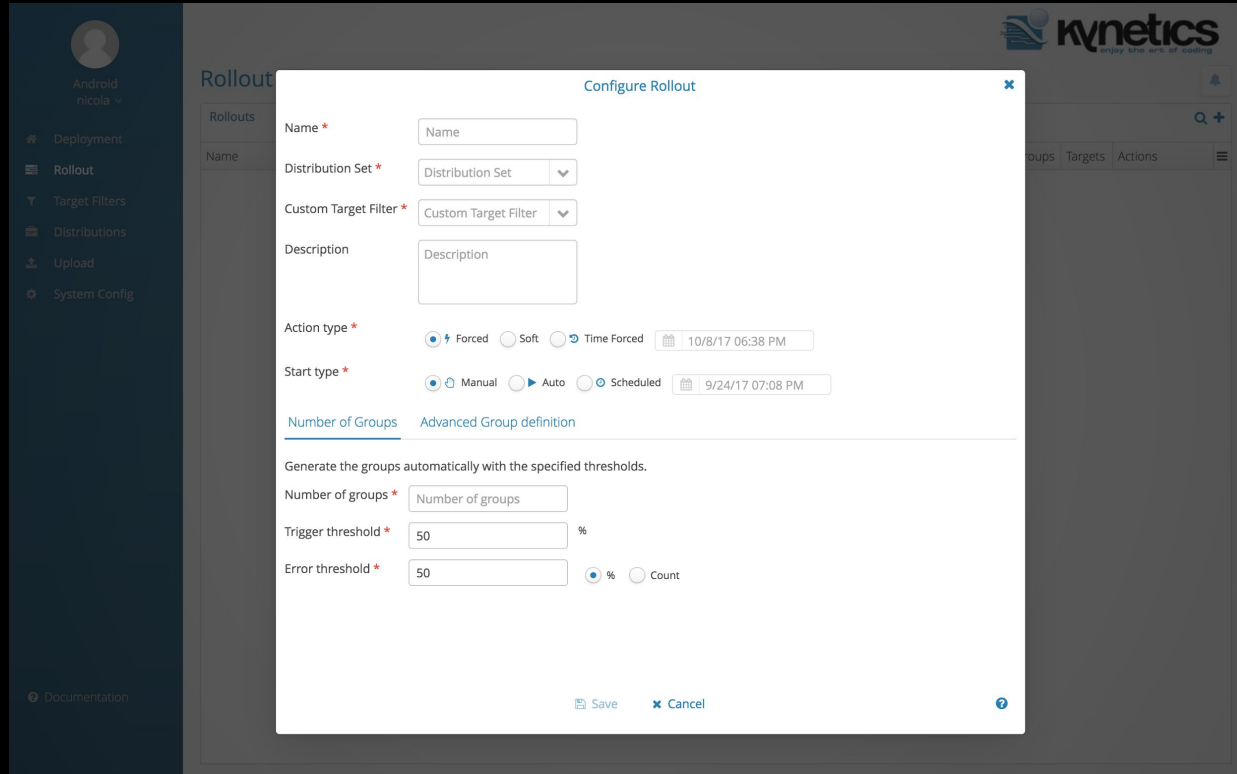
Targets Panel: Displays a table with one target: 'adu-device'. Below the table, details for the target are shown: 'Target: adu-device', 'Assigned', 'Installed', 'Tags', 'Name: AndroidTestDistribution', 'Version: 1.0', and 'Application: AndroidTestApp 1.0'.

Distributions Panel: Displays a table with one distribution: 'AndroidTestDistributi...' with version '1.0'. Below the table, details for the distribution set are shown: 'Distribution set: AndroidTestDi...', 'Details', 'Description', 'Modu', 'Type: Application', and 'Required Migration Step: No'.

Action History Panel: Titled 'Action History For adu-device', it shows a table with columns: Active, Distributionset, Date and time, Status, and For. It lists three actions, all with a status of 'Failed' (red lightning bolt icon).

At the bottom of the main area, it says 'Total Targets: 1' and provides two buttons: 'Drop here to delete' and 'No actions'.

Rollout Configuration



The screenshot shows the 'Configure Rollout' dialog box in the Kynetics application. The dialog is titled 'Configure Rollout' and has a close button (X) in the top right corner. It contains the following fields and options:

- Name ***: A text input field with the placeholder 'Name'.
- Distribution Set ***: A dropdown menu with 'Distribution Set' selected.
- Custom Target Filter ***: A dropdown menu with 'Custom Target Filter' selected.
- Description**: A text input field with the placeholder 'Description'.
- Action type ***: Three radio buttons: 'Forced' (selected), 'Soft', and 'Time Forced'. A date/time picker shows '10/8/17 06:38 PM'.
- Start type ***: Three radio buttons: 'Manual' (selected), 'Auto', and 'Scheduled'. A date/time picker shows '9/24/17 07:08 PM'.
- Number of Groups**: A tabbed interface with 'Number of Groups' selected and 'Advanced Group definition' as an alternative tab.
- Generate the groups automatically with the specified thresholds.**: A section with three input fields:
 - Number of groups ***: A text input field with the placeholder 'Number of groups'.
 - Trigger threshold ***: A text input field with '50' and a '%' symbol.
 - Error threshold ***: A text input field with '50' and two radio buttons: '%' (selected) and 'Count'.
- Buttons**: 'Save' and 'Cancel' buttons at the bottom, and a help icon (?) in the bottom right corner.

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

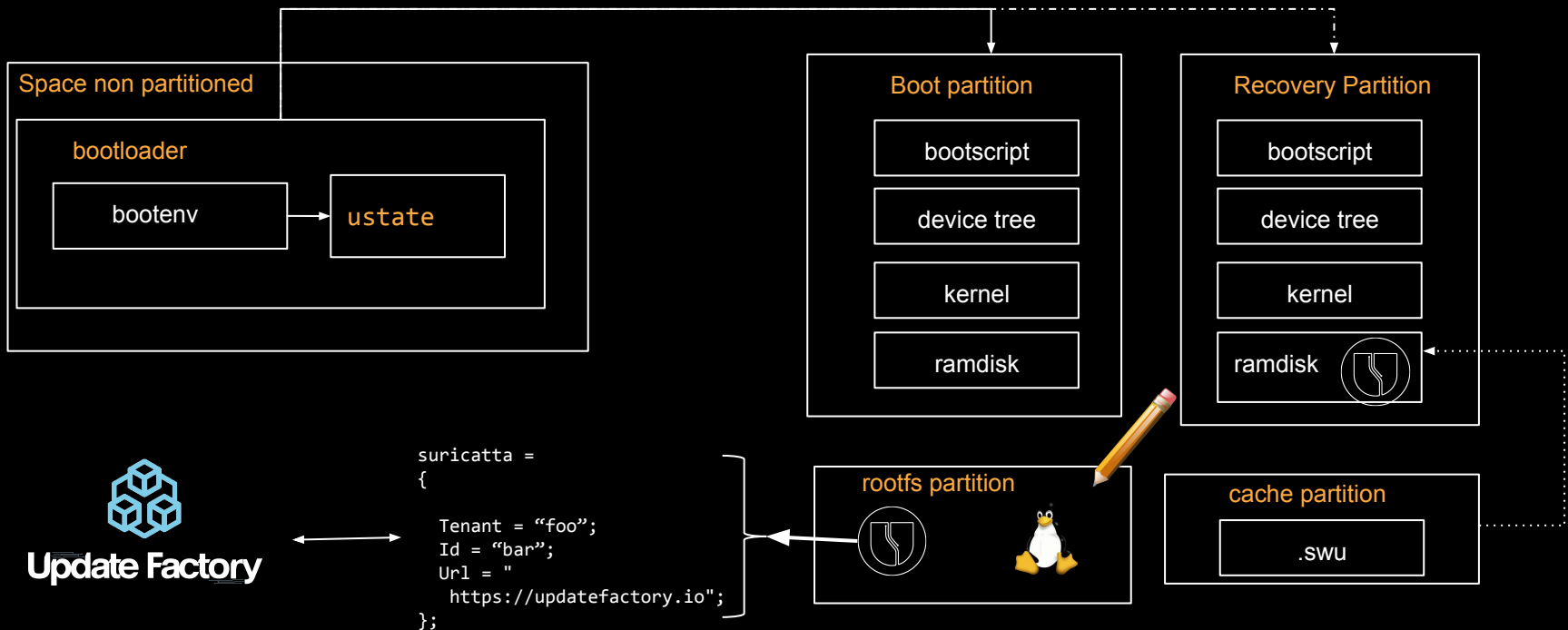
- › Service on the embedded device
 - › Gnu/Linux featuring **SWUpdate**
 - › Android Service featuring Update Server API
- › Update Server featuring **hawkBit™**
- › IAM Server
- › Artifact Repository
- › Metadata Repository
- › MsgBroker

Android “like” behaviour on Embedded Linux

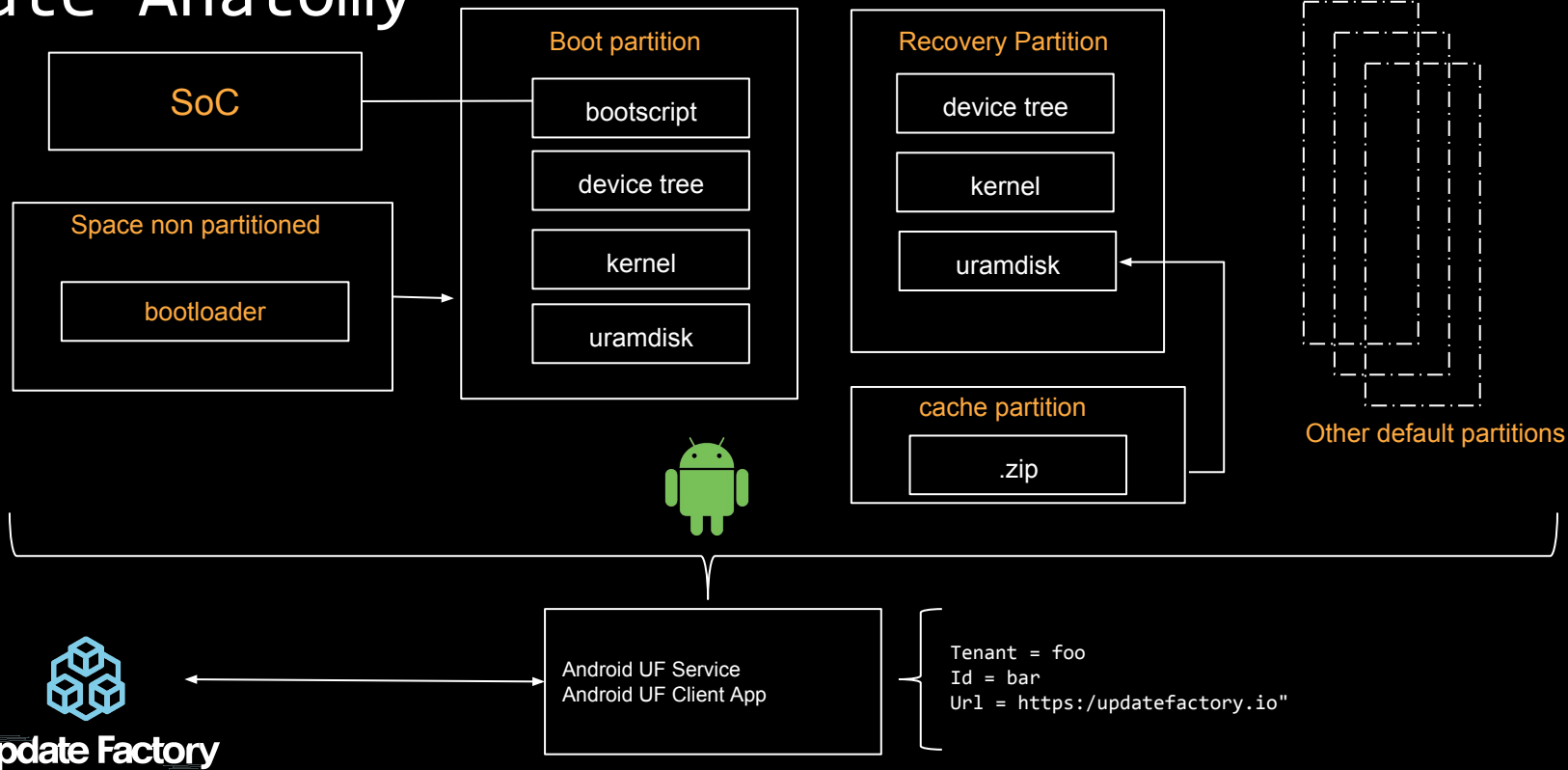
Update Factory implements all the missing bits to have an Android-like OTA mechanism on an Embedded Linux OS

- › 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/hawkbite/>
- › <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