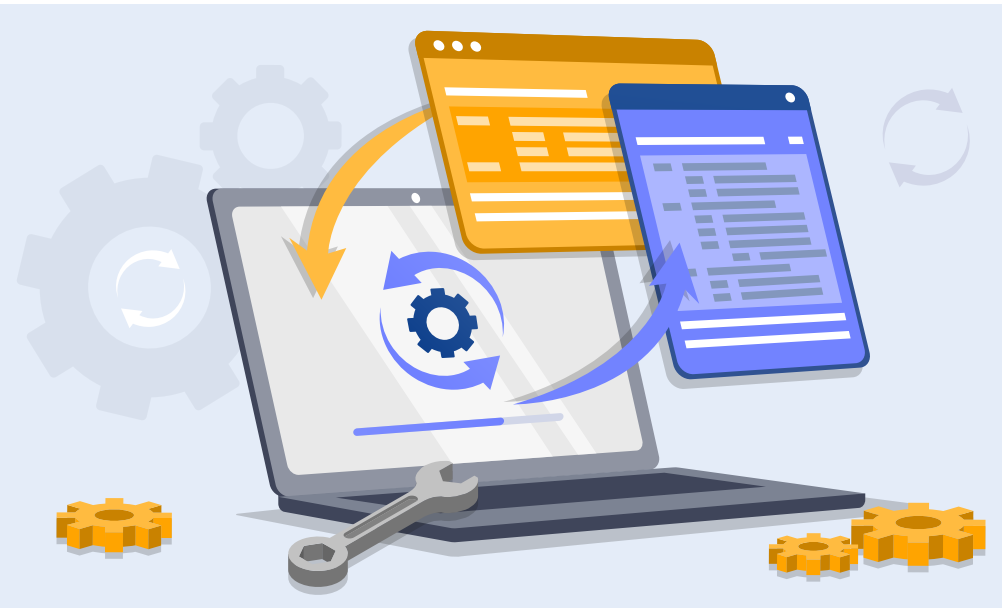


IoT-Geräte-Updates automatisieren

Das Aktualisieren von IoT-Geräten ist für deren Hersteller eine komplexe Herausforderung. Unterstützung bieten Rollback-Mechanismen, Over-the-Air-Updates und Tools wie Yocto.

Von Josef Holzmayr

Die Zahl der vernetzten Geräte steigt weltweit ununterbrochen: Schätzungen prognostizieren für Ende 2022 bereits über 16 Milliarden Geräte im Internet of Things (IoT) (siehe ix.de/z4kv). Smart Home und Automatisierung, aber auch Datenaufzeichnung und Sicherheitsfunktionen zählen zu den Haupteinsatzgebieten. Ein essenzieller Baustein in der Pflege dieser Geräte sind Software-Updates, die großflächig automatisiert und verlässlich eingespielt werden sollen. Allerdings behandeln viele Hersteller Updates immer noch sehr stiefmütterlich. Die europäische Richtlinie ETSI 303 645 soll diesen Zustand nun endlich beenden (siehe ix.de/z4kv). Abschnitt 5.3 erläutert ausführlich die Anforderungen an die Hersteller.

An dieser Stelle kommen Over-the-Air-Updates (OTA) ins Spiel, wobei der Name nicht allzu wörtlich zu verstehen ist: Darunter fasst man das Ausrollen von Updates aus der Ferne und meist ohne Benutzerinteraktion am eigentlichen Gerät zusammen – unabhängig von der tatsächlich genutzten Verbindungsart. Die folgenden Begriffe sind dabei gebräuchlich:

- Artifact – ein Softwarepaket, etwa ein zur Verteilung bereitgestelltes Update;
- Deployment – Verteilen und Einspielen von Artifacts;
- Unattended – Einspielen eines Updates oder Softwarepaketes ohne Benutzerinteraktion.

iX-TRACT

- IoT-Geräte vervielfachen aufgrund ihrer großen Verbreitung die potenzielle Angriffsfläche von Unternehmen.
- Eine von IoT-Herstellern gern vernachlässigte Aufgabe ist das automatisierte Bereitstellen und Ausliefern von Updates. Eine europäische Richtlinie soll Abhilfe schaffen.
- Mit A/B-Mechanismen, Over-the-Air-Updates und Distributions-Build-Systemen wie Yocto lassen sich IoT-Geräte dauerhaft und zuverlässig mit Aktualisierungen versorgen.

Besonders beim Unattended Deployment unterscheiden sich OTA-Updates damit grundsätzlich von Updates, die Benutzer aktiv herunterladen und einspielen. Eine Sonderform sind Aktualisierungen, die zwar automatisiert heruntergeladen, aber erst nach Bestätigung durch den Benutzer angewendet werden. Dies dient meist dazu, unerwünschte Ausfallzeiten zu vermeiden, da diese im Automatisierungsumfeld gravierende Folgen haben können. Der Albtraum von der stillstehenden Fertigungslinie ist hier nahe an der Realität.

Damit dieser Fall nicht eintritt, müssen mehrere Abläufe perfekt aufeinander abgestimmt sein. Das Kernstück ist dabei der A/B-Mechanismus: Er ermöglicht im Fall eines Fehlers während des Updates die Rückkehr, das Rollback zum funktionsfähigen Zustand vor dem Update.

Abwechslung tut not

Das Kernstück des A/B-Mechanismus sind zwei Systempartitionen, A und B genannt (siehe Abbildung 1). Sie wechseln beim Einspielen eines Systemupdates zwischen den Zuständen aktiv und inaktiv, wobei das System die aktive Partition momentan zum Starten verwendet. Der Wechsel bedeutet auch, dass alle Daten auf den Systempartitionen als transient zu behandeln sind, da sie bei einem Update im Ganzen ersetzt werden. Als letzte Bausteine benötigt ein solcher Mechanismus daher immer zumindest eine persistente Datenpartition sowie einen Bootprozess, der das Umschalten der Partitionen beherrscht.

Der in Abbildung 2 gezeigte Ablauf eines Systemupdates beginnt im aktiven, laufenden System. Das neue System wird auf die inaktive Partition geschrieben und anschließend ein für den Bootprozess sichtbarer Merker zum Partitionswechsel gesetzt. Beim nächsten Neustart werden aktive und inaktive Partition getauscht, womit das System im neuen Zustand startet. Läuft der Bootvorgang erfolgreich durch, löscht der Updatemechanismus den Merker und macht den Wechsel damit permanent. Schlägt der Systemstart fehl und wird nicht durch den Updatemechanismus bestätigt, wechselt der Bootloader beim nächsten Neustart die Partitionen zum vorherigen Zustand zurück und ersetzt den Update- durch einen Fehlermerker. Durch diese Rückkehr kann das System sicher wieder starten und über den Fehlermerker ein fehlgeschlagenes Update erkennen.

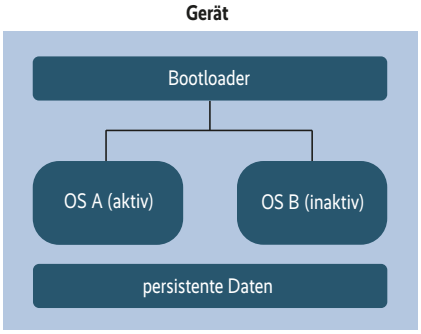
Das Umschalten der Partitionen sowie das Setzen und Lesen der Merker zur

Boot-Zeit kann unterschiedliche Formen annehmen. Im einfachsten Fall lässt sich der Bootloader um die erforderliche Logik erweitern, etwa durch ein Skript oder einen Patch. Alle populären Bootloader wie U-Boot, GRUB und zu einem gewissen Grad (U)EFI stellen weitgehend vorbereitete und gut dokumentierte Vorgehensweisen bereit. Dieser Ansatz hat allerdings den Nachteil, dass ein essenzieller Teil des Updatemechanismus im softwarebasierten Bootloader hinterlegt ist. Dadurch lässt sich dieser meist nicht auf vergleichbare Weise mit Updates versorgen, da er keine vorgelagerte Stufe besitzt, A/B-Mechanismen abzubilden. Der Bootloader mutiert damit zum Single Point of Failure.

Die SoC-Hersteller (System-on-a-Chip, siehe Kasten) haben die Notwendigkeit robuster Updates erkannt. Eine OTA-Lösung muss üblicherweise eine Anpassung der Bootloader-Integration vornehmen, um solche meist sehr herstellerspezifischen Features zu nutzen. Ein Beispiel ist Nvidias Tegra-Plattform, die A/B-Updates bis zur Bootloader-Ebene hinab ermöglicht.

Build-Systeme helfen automatisieren

Ein A/B-Schema erfordert zwei grundsätzliche Teile der Integration: Partitionierung und Bootprozess. Beides sind vergleichsweise invasive Änderungen am Gesamtsystem, die einen gewissen Aufwand erfordern. Solche Änderungen an einem (Root-File-)System sind fehlerträchtig und sollten so weit wie möglich automatisiert werden. Hier kommen die Distributions-Build-Systeme ins Spiel. Sie übernehmen üblicherweise das Zusammenstellen der gewählten oder durch Abhängigkeiten benötigten Softwarepakete, deren Anpassung und erstellen daraus die gewünschten Dateisysteme und gegebenenfalls Images.



Von den beiden Systempartitionen ist eine aktiv, die andere dient zum Aufspielen der Aktualisierungen (Abb. 1).

Da die OTA-Integration mit all diesen Schritten koordiniert werden muss, ist der Wert einer OTA-Software nicht allein durch die Features gegeben, sondern zu einem substanziellen Teil auch dadurch, wie reibungslos sie mit dem verwendeten Build-System zusammenarbeitet. Alle später im Artikel behandelten OTA-Vertreter stellen für das Yocto Project – das derzeit populärste Build-System – Metalayer bereit, die den Einsatz in eigenen Projekten beherrschen und stark vereinfachen. Mit dem Thema Yocto hat sich iX schon mehrfach beschäftigt [1, 2].

Auswahl der Werkzeuge

Ursprünglich waren Updatemechanismen ein Feld für Homebrew-Lösungen: Jeder kocht sein eigenes Süppchen (der Autor dieses Artikels bekennt sich hier selbst schuldig). Aus den vielen daraus entstandenen Ansätzen sollen drei vorgestellt werden, die folgende Bedingungen erfüllen:

- A/B-Mechanismus;
- Linux-Plattformen als Zielsystem;
- nicht an einen Hardware- oder IC-Hersteller gebunden;
- Open Source, kommerzieller Support erhältlich.

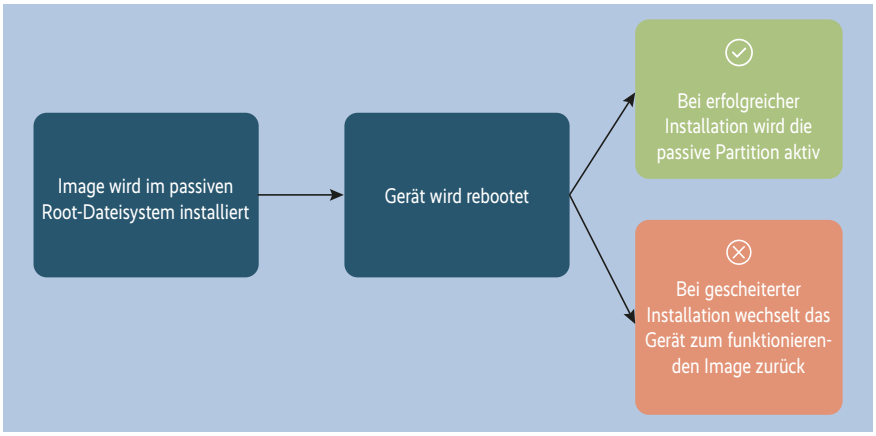
Varianten für Mikrocontroller bleiben hier außen vor, da hier die Situation bislang noch sehr unübersichtlich und von herstellerspezifischen Produkten dominiert ist.

Das von Stefano Babic schon 2013 ins Leben gerufene SWupdate ist wohl der bereits am längsten verfügbare Ansatz. Es punktet mit einer Vielzahl von Handlern für alle denkbaren Aufgaben. Der Fokus liegt auf einem mächtigen Client, der bis hin zur webbasierten Verwaltungsoberfläche alles mitbringt, was Entwickler sich wünschen. Der Einsatz als OTA-Updater steht zwar nicht im Vordergrund, ist aber in Kombination mit dem für Updates von IoT-Geräten konzipierten Hawkbit-Server der Eclipse Foundation etabliert.

Die seit 2015 von einer aktiven Community um Entwickler der Firma Pengutronix gepflegte Software RAUC ist ein Open-Source-Vertreter mit Augenmerk auf Freiheit. Über mehrere Slots in einem Update lassen sich verschiedene in Beziehung stehende Updates bündeln und in einem Rutsch ausliefern, was besonders für Systeme mit mehreren beteiligten Softwarepaketen interessant ist. Wie SWupdate ist RAUC ein Client-only-Projekt, das für volle OTA mit einem Hawkbit-Server kombiniert wird.

Ebenfalls seit 2015 aktiv, wird Mender von Northern.tech als Open Core bereitgestellt. Bestehend aus Client und Backend bietet es eine Ende-zu-Ende-Lösung mit einer integrierten Verwaltung der Updatepakete und der Koordination des Ausrollens. Derzeit ist es die einzige Software mit Features, die ein koordiniertes Zusammenspiel aus Client und Backend erfordern, beispielsweise ein Preload der Updates mit synchronisiertem Start durch den Operator.

Alle drei Pakete können Updates über verschiedene Algorithmen signieren und bieten APIs zur Kontrolle des Updateprozesses. SWupdate stellt eine Schnittstelle per Unix Domain Socket bereit, RAUC und Mender nutzen D-Bus. Letzteres bietet darüber hinaus eine Backend-seitige RESTful API.



Nach einem erfolgreichen Reboot ins aktualisierte Image tauschen die beiden Partitionen die Rollen (Abb. 2).

OTA-Updates – ein Ablauf in mehreren Schritten

Grundsätzlich ähneln sich bei allen drei Kandidaten die Schritte, ein Gerät mit OTA-Updates zu versorgen. Im Folgenden sollen sie am Beispiel Mender kurz umrissen werden, da dies keine Installation eines Backends benötigt, sondern mit einem kostenlosen Trial-Account funktioniert. Diese Schritte sind nicht im Detail ausformuliert, sondern sollen nur die Konzepte verdeutlichen. Sie sind in

System-on-a-Chip

Die überwiegende Mehrzahl der in IoT-Geräten verbauten CPUs gehört zur Kategorie der Systems-on-a-Chip. Der Begriff verdeutlicht, dass ein Großteil der für ein funktionierendes Gesamtsystem benötigten Baugruppen auf einem Chip vereint sind: ein oder mehrere CPU-Kerne, Caches, Speichercontroller, Timer und Schnittstellen sowie oft ein substanzieller Teil des Energiemanagements und der Takterzeugung. Bei Systemen, die ein Display ansteuern, beispielsweise Mobiltelefonen, sind die Grafikeinheiten üblicherweise ebenfalls integriert.

Ein SoC benötigt je nach Ausbau und Leistungsfähigkeit zum Betrieb nur eine Stromversorgung, RAM und Flashspeicher. Der Markt wird stark von ARM dominiert, nahezu alle Linux-tauglichen SoCs basieren auf dieser Architektur. Alternativen wie x86, RISC-V oder auch MIPS und PowerPC sind Nischenerscheinungen. Eng verwandt, aber nicht mit SoCs zu verwechseln sind die Systems-on-Module, kurz SoM. Diese vereinen ein SoC mit Speicherbausteinen und Stromversorgung auf einer Leiterplatte, die als Basissystem fungiert. Der Begriff wird allerdings fließend verwendet, und je nach Kontext kann alles von einem aus zwei Dice bestehenden SoC bis hin zum Raspberry Pi gemeint sein.

ähnlicher Form auch bei SWupdate/RAUC und Hawkbit nötig.

Am Anfang steht immer ein Linux-Image für das Zielgerät, meist mit Yocto gebaut. Das Listing zeigt eine YAML-Datei, die einen vollständigen Yocto-Build über das Tool `kas` definiert. Der Aufruf

```
kas build ota.yml
```

stößt den Build an. Achtung: Wie jeder Yocto-Build kann dies mehrere Stunden und viele Gigabyte Festplattenplatz erfordern. Nach Abschluss finden sich im Verzeichnis /tmp/deploy/images/raspberrypi4 diverse Ausgabedateien, wobei zum Verständnis nur zwei davon notwendig sind. Die .sdimg-Datei dient als Startpunkt – im Sprachgebrauch von Herstellern die Provisionierung von Geräten. Diese Art von Images beinhaltet das Partitionslayout sowie den vorbereiteten Bootloader und wird im Falle des Raspberry Pi üblicherweise auf die SD-Karte geschrieben, zum Beispiel mit dem balena Etcher. Die .mender-Datei ist ein Artifact, das sich in einem laufenden System als Update einspielen lässt.

Das gebootete System verhält sich wie ein normales Linux. Um mit OTA-Updates versorgt zu werden, muss es im Backend registriert werden. Im einfachsten Fall für Mender folgt man der interaktiven Registrierung per `mender setup` auf der Kommandozeile. Es empfiehlt sich, die Frage nach dem verkürzten Polling-Intervall mit `Y` zu bestätigen. In realen Projekten werden mehrere Minuten bis Stunden benötigt, was während der Evaluation oder Entwicklung unangenehm träge ist.

Nun muss die Anfrage des Geräts auf der Verwaltungsoberfläche des Backends bestätigt werden. Im Anschluss erscheint der Raspberry Pi im Dashboard, und das zeigt den aktuellen Softwarezustand des Systems. Diese Darstellung heißt im IoT-Jargon meist Inventory.

Zum Ausrollen eines Updates muss man es in das Repository des Backend hochladen. Hier kommt die oben erwähnte .mender-Datei ins Spiel. Sie enthält ein per OTA verteilbares Linux-System. Es definiert als Metadaten ein oder mehrere kompatible Geräte sowie die enthaltene Softwareversion. Theoretisch könnte man das im ersten Build erstellte Artifact verwenden. Da es allerdings dieselbe Softwareversion trägt und dasselbe System enthält wie das bereits auf der SD-Karte laufende Linux, ist dieses Update wenig sinnvoll.

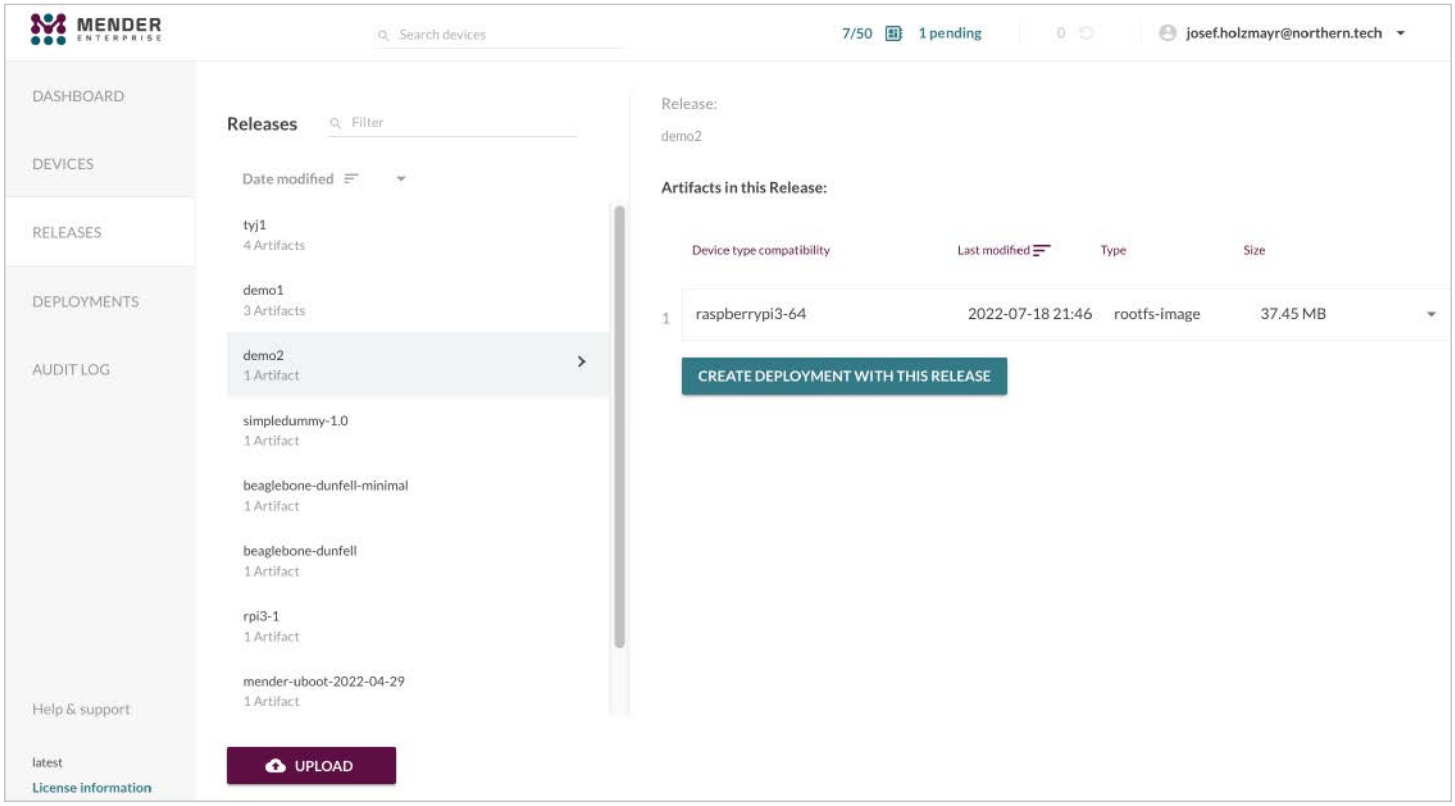
Eine einfache Änderung besteht darin, die Variable `MENDER_ARTIFACT_NAME` in der kas-Datei auf einen beliebigen anderen Wert zu ändern und den Build anschließend neu zu starten. Falls gewünscht, kann man die Zeile

```
IMAGE_INSTALL:append = " "
```

um ein zu installierendes Paket ergänzen. Zur Verdeutlichung der neuen Softwareversion kann beispielsweise der Kommandozeilenrechner bc installiert werden. Dazu fügt man dem append-

Listing: ota.yml

```
header:
  version: 11

distro: poky

machine: raspberrypi4

defaults:
  repos:
    refspec: dunfell

repos:
  poky:
    url: https://git.yoctoproject.org/git/poky
    layers:
      meta:
      meta-poky:
      meta-yocto-bsp:

  meta-openembedded:
    url: https://git.openembedded.org/meta-openembedded
    layers:
      meta-oe:

  meta-mender:
    url: https://github.com/mendersoftware/meta-mender.git
    layers:
      meta-mender-core:
      meta-mender-demo:
      meta-mender-raspberrypi:

  meta-raspberrypi:
    url: https://github.com/agherzan/meta-raspberrypi.git

local_conf_header:
  base: |
    CONF_VERSION = "1"
    PACKAGE_CLASSES = "package_ipk"
    INHERIT += "mender-full"
    INIT_MANAGER = "systemd"

  raspberrypi: |
    RPI_USE_U_BOOT = "1"
    ENABLE_UART = "1"
    MENDER_BOOT_PART_SIZE_MB = "100"
    IMAGE_INSTALL:append = " kernel-image kernel-devicetree"
    IMAGE_FSTYPES:remove = " rpi-sdimg"

  heise_developer: |
    MENDER_ARTIFACT_NAME = "heisedeveloper1"
    IMAGE_INSTALL:append = ""

target:
  - core-image-minimal
```



Nach dem Build wird das Update im Backend registriert und anschließend ausgeliefert (Abb. 3).

String die Variable `bc` hinzu, wobei das Leerzeichen davor immer erhalten bleiben muss. Yocto kann bei fortgesetzten Builds große Teile wiederverwenden, deshalb ist hier jetzt mit deutlich weniger Zeitaufwand zu rechnen. Anschließend stehen je eine neue .sdimg- und .mender-Datei bereit, letztere lässt sich ins Backend hochladen.

Als letzter Schritt wird ein Deployment erstellt. Dazu wählt man in der Detailansicht der Releases das eben hochgeladene Artifact aus und folgt dem Dialog „create deployment with this release“ (siehe Abbildung 3). Nach Bestätigung wird das Update im nächsten Polling-Zyklus eingereiht und innerhalb einiger Minuten inklusive Neustart installiert.

Zusammenfassung

OTA-Updates sind im Mainstream angekommen, sowohl auf Seite der Nachfrage als auch des Angebots. Welche Lösung im eigenen Projekt am besten passt, bedarf einer detaillierten Evaluation, da diese Entscheidung den kompletten Lebenszyklus eines Produktes begleitet. Dabei sollten nicht nur die Features auf dem Gerät selbst entscheiden, da selbst bei kleineren jährlichen Stückzahlen schnell Flottengrößen erreicht werden, die nicht mehr manuell zu verwalten sind. Im Sinne eines nachhaltigen Projekts ist es daher unerlässlich, die Verteilung und Verwaltung von Updates als integralen Teil der Produktpflege zu betrachten, da nur so die künftige Versorgung mit notwendigen, zum Teil sicherheitskritischen Updates zu garantieren ist. (avr@ix.de)

Distributions-Build-Systeme

Im Unterschied zu Desktop-Systemen nutzen eingebettete Systeme selten eine der bekannten Standard-Linux-Distributionen wie Ubuntu, Debian, Fedora, sondern eine mehr oder minder spezifisch angepasste Distribution. Hier existieren viele verschiedene Tools auch als Distributions-Build-Systeme, die aus unterschiedlichen Quellen eine Distribution erzeugen können. Sie lassen sich in zwei Kategorien unterteilen: binär- und Source-basiert.

Aufseiten der binärbasierten stehen beispielsweise das von Canonical angebotene Ubuntu Core oder auch die auf Debian basierenden Projekte Isar und ELBE. Sie nutzen die Repositorys der Ursprungsdistributionen und ergänzen sie durch Mechanismen zum Erzeugen von Images und zum Anpassen einzelner Pakete. Zu den Source-basierten Build-Systemen zählen unter anderem Yocto, OpenEmbedded, buildroot und ptxdist. Sie kompilieren die Distribution komplett aus den Quellen, was zwar eine sehr hohe Flexibilität bietet, aber auch sehr zeitaufwendig sein kann.

Quellen

[1] Josef Holzmayr; Das Linux-Ökosystem fürs IIoT; iX Special 2018, S. 130
[2] Tam Hanna; Zweite Runde für den Distributionsbaukasten Yocto; iX 6/2016, S. 74
[3] Weiterführende Informationen siehe ix.de/z4kv

JOSEF HOLZMAYR

arbeitet als Head of Developer Relations bei Mender.io.