LINUTRONIX

April 27, 2026

ELBE – oder wie man seine eigene Embedded-Distribution bauen kann

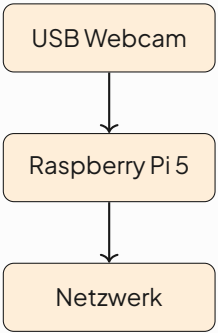
LINUTRONIX

Das Szenario

Aufgabe

USB-Webcam am Raspberry Pi 5 streamt Live-Video über das Netzwerk – kein Display, automatischer Start beim Boot.

- Setup funktioniert bereits
- Jetzt: reproduzierbar deployen



LINUTRONIX

Naheliegende Lösung: Raspberry Pi OS

1. Raspberry Pi OS flashen
2. Pakete installieren

```
sudo apt install gstreamer1.0-tools \
  gstreamer1.0-plugins-good v4l-utils
```

Funktioniert einmal

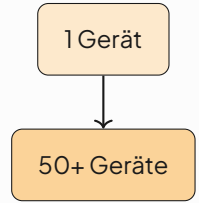
Skaliert nicht

LINUTRONIX

Das Problem mit „einfach installieren”

Deployment

- Welche Paketversion lief eigentlich?
- Wie rolle ich das auf 50 Geräte aus?
- Was passiert in 2 Jahren?



Skalierungsproblem

LINUTRONIX

Was fehlt?

Qualität & Workflow

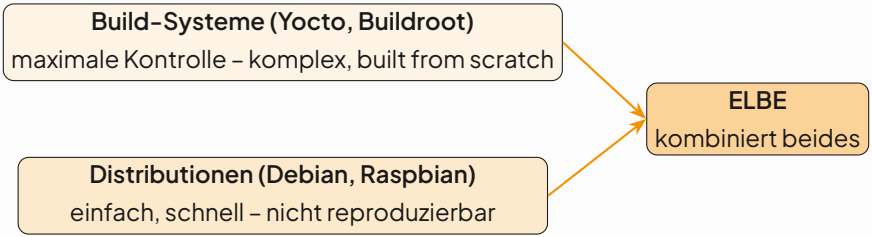
- Reproduzierbare Images
- Wartbarkeit nach Jahren
- Keine händische Einzelarbeit
- CI/CD-tauglich

Sicherheit & Compliance

- Schnelle Security-Updates
- SBOM – Software Bill of Materials

LINUTRONIX

Embedded Linux – Zwei Ansätze



Früher: viele Embedded-Plattformen nicht in Mainstream-Linux unterstützt. Heute: ARM, RISC-V, x86 u. v. m. direkt in Debian verfügbar. ELBE macht genau diese Pakete für Embedded-Produkte nutzbar.

LINUTRONIX

Binärdistributionen – Stärken und Lücke

Stärken

- Weit verbreitet und gut bekannt
- Langfristige Sicherheitspflege
- Große Paketauswahl

Die Lücke

Fehlende Werkzeuge zur Automatisierung:

- Eigene Images erstellen
- Binärpakete und eigenes kombinieren

LINUTRONIX

Vergleich: Embedded Build-Systeme

System	Basis	Ansatz
Yocto / OpenEmbedded	Eigene Distro	Alles selbst kompilieren
Buildroot	Eigene Distro	Minimale Cross-Toolchain
PTXdist	Eigene Distro	Toolchain-basiert
ELBE	**Debian**	**Pakete wiederverwenden**

LINUTRONIX

Was ist ELBE?

Embedded **L**inux **B**uild **E**nvironment

- Werkzeug zur Erstellung von embedded Linux Images
- Entwickelt und gepflegt von Linutronix
- Open Source – GPL v3
- Debian Paket für ELBE selbst

LINUTRONIX

ELBE – Features (1)

Grundlage

- Debian-Images aus Debian-Paketen
- Pakete von Debian-Derivaten (Ubuntu, Raspbian, …)
- Baut auf Debian-Werkzeugen auf

Plattformen & Compliance

- Fremde Architekturen: ARM, RISC-V, x86
- Reproduzierbare Images (mit bekannten Grenzen)
- Lizenz-Compliance: Binärpakete, Quellen, Lizenztexte
- Flexibles Deployment: Hardware, VM, Container

LINUTRONIX

ELBE – Features (2)

Automatisierung

- Eine einzige Eingabedatei: XML
- Nicht-interaktiv – CI/CD-tauglich
- SBOM-Erzeugung (CycloneDX)

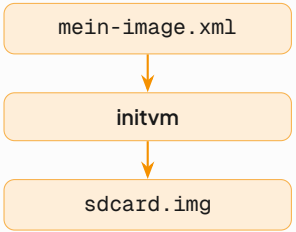
Ausgabe-Formate

- Bootfähige Images: SD-Karte, Raw-Flash, NFS-Root
- Flexible Anpassung: Finetuning, Archive
- Applikationsentwicklung

LINUTRONIX

Grundprinzip: initvm

- Beim ersten nutzen erstellt man eine virtuelle Maschine – die **initvm**
- `elbe` auf dem Host kommuniziert mit `elbe-daemon` in der VM
- initvm wird über libvirt verwaltet
- Builds für fremde Architekturen via QEMU-User
- Isolierte Build-Umgebung für Root-Filesysteme
- Image-Erstellung benötigt Root-Rechte
  → VM schützt den Host



Kommando

```
elbe initvm create elbe initvm submit
```

LINUTRONIX

XML stark vereinfacht – wo gehört was hin?

<RootFileSystem>

- **<project>** → Debian, Mirror, Version
- **<target>** → Zielsystem
  - **<pkg-list>** → Pakete (GStreamer, Kernel, …)
  - **<images>** → SD-Karte Layout
  - **<fstab>** → Mountpoints
  - **<finetuning>** → Commands + Setup

</RootFileSystem>

Das ist **stark vereinfacht**! In der Praxis: deutlich mehr Details (Keys, Archive, Bootconfig, …)

LINUTRONIX

ELBE Beispiel

Ziel

Bootfähiges Debian-Trixie-Image für den Raspberry Pi – gebaut mit ELBE auf einem x86-Host.

Image-Format

SD-Karte: Boot + Root

Spezifikation

- Architektur: **aarch64**
- Release: **trixie**
- Bootloader: `raspi-firmware`
- Kernel: `linux-image-rpi-2721`

LINUTRONIX

XML – project-Abschnitt

```
<project>
  <mirror>
    <primary_host>deb.debian.org</primary_host>
    <primary_path>/debian</primary_path>
    <primary_proto>http</primary_proto>
    <url-list>
      <url>
        <binary>http://deb.debian.org/debian-security
          trixie-security main</binary>
      </url>
      ...
    </url-list>
  </mirror>
</project>
```

LINUTRONIX

XML – Pakete (Auszug)

```
<pkg-list>
  <!-- Kernel + Firmware -->
  <pkg>linux-image-rpi-2721</pkg>
  <pkg>raspi-firmware</pkg>

  <!-- Netzwerk -->
  <pkg>wpasupplicant</pkg>

  <!-- Zugriff -->
  <pkg>openssh-server</pkg>
</pkg-list>
```

- Automatische Abhängigkeitsauflösung

LINUTRONIX

XML – Image

```
<images>
  <msdoshd>
    <name>sdcard.img</name>

    <partition>
      <size>256MB</size>
      <label>firmware</label>
    </partition>

    <partition>
      <size>remain</size>
      <label>root</label>
      <bootable/>
    </partition>
  </msdoshd>
</images>
```

LINUTRONIX

XML – Finetuning und Archive

```
  <finetuning>
    <!-- APT-Cache aufraeumen -->
    <rm>/var/cache/apt/archives/*.deb</rm>
    <!-- SSH-Key-Generierung beim ersten Boot -->
    <command>systemctl enable ssh</command>
    <!-- Eigenen User anlegen -->
    <command>
      useradd -m -G sudo -s /bin/bash user
    </command>
  </finetuning>
</target>
```

LINUTRONIX

Build-Ablauf – 3 Schritte

1. `elbe` als Debian-Paket installiert
   elbe-rfs.org → Quickstart-Guide
2. initvm einrichten

```
elbe initvm create
```

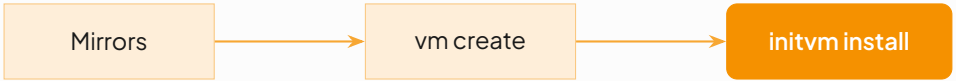
3. Image bauen

```
elbe initvm submit raspi5.xml
```

LINUTRONIX

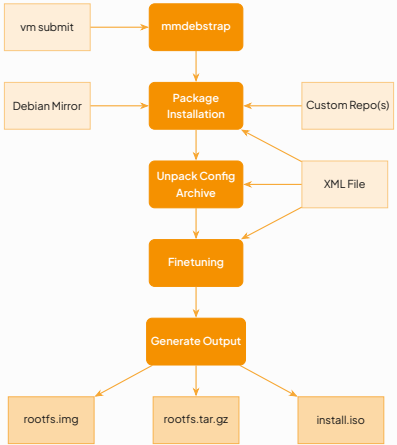
Build-Ablauf – Internals

`elbe initvm create`



LINUTRONIX

Build-Ablauf – Internals



LINUTRONIX

Ergebnis-Dateien

Images

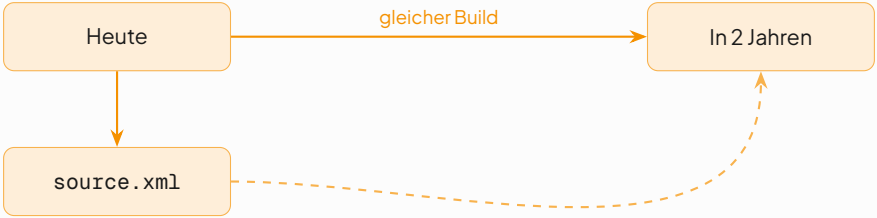
- `sdcard.img` – Ziel-Image
- `rootfs.tar.gz` – Root-Dateisystem

Dokumentation & Reproduzierbarkeit

- `source.xml` – vollständiges XML inkl. Paketversionen
- Lizenztexte aller Pakete
- SBOM (CycloneDX)

LINUTRONIX

Reproduzierbarkeit



Einschränkung

Aktuell funktional reproduzierbar – binär identische Images sind in Arbeit.

LINUTRONIX

Praxisbeispiel: MJPEG-Stream per GStreamer

Szenario

USB-Webcam am Raspberry Pi 5 streamt Live-Video als MJPEG über TCP Port 8080 – kein Desktop, automatischer Start beim Boot.

Alles in einer XML-Datei

`webcam-streamer.xml` beschreibt den kompletten Stack: Pakete, Konfiguration und Autostart.

Stack

- Raspberry Pi 5, arm64
- Debian Trixie
- GStreamer + V4L2
- Systemd-Autostart
- SSH-Zugang

LINUTRONIX

Schritt 1: GStreamer-Pakete aus Debian (1/4)

```
<pkg-list>
  <!-- Kernel und Firmware -->
  ...
  <!-- Basis-System -->
  ...
  <!-- GStreamer -->
  <pkg>gstreamer1.0-tools</pkg>
  <pkg>gstreamer1.0-plugins-base</pkg>
  <!-- v4l2src, jpegenc, multipartmux, tcpserversink -->
  <pkg>gstreamer1.0-plugins-good</pkg>
  <pkg>v4l-utils</pkg>
</pkg-list>
```

- Kein Kompilieren – alle Pakete direkt aus Debian

LINUTRONIX

Schritt 2: Systemd-Unit im Overlay (2/4)

Datei `overlay/etc/systemd/system/webcam-stream.service`:

```
[Unit]
Description=Webcam Stream
After=network.target
Wants=network.target

[Service]
User=pi
ExecStart=/usr/bin/gst-launch-1.0 -e \
  v4l2src device=/dev/video0 ! videoconvert ! \
  jpegenc quality=85 ! \
  multipartmux boundary=frame ! \
  tcpserversink host=0.0.0.0 port=8080
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

LINUTRONIX

Schritt 3: Overlay einbinden und Service aktivieren (3/4)

Overlay-Verzeichnis wird direkt referenziert:

```
<archivedir>overlay</archivedir>
```

Finetuning im XML:

```
<finetuning>
  <command>useradd -m -G video,sudo -s /bin/bash pi</command>
  <command>systemctl enable webcam-stream</command>
</finetuning>
```

- Das Overlay-Verzeichnis liegt neben der XML – kein Einbetten nötig
- Alle Dateien, Units und Konfigurationen – sauber getrennt

LINUTRONIX

Schritt 4: Image bauen und flashen (4/4)

```
elbe initvm submit webcam-streamer.xml
```

Ergebnis

- `sdcard.img` flashen, einschieben – fertig
- GStreamer startet automatisch beim Boot
- Stream erreichbar auf Port 8080

Reproduzierbarkeit

- `source.xml` fixiert Paketversionen
- Gleicher Build in zwei Jahren möglich
- CI/CD-fähig: ein Kommando

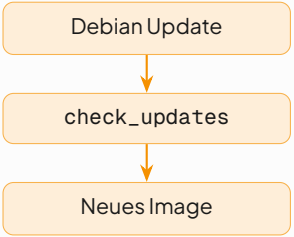
LINUTRONIX

ELBE und Updates

- Debian veröffentlicht regelmäßig Sicherheits-Updates
- Kein manuelles Patchen
- Image neu gebaut mit Fixes von Debian

Workflow

1. `elbe check_updates ./source.xml`
2. `elbe initvm submit ./source.xml`



LINUTRONIX

Einsatzszenarien in der Industrie

Industrie-Steuerungen

Langlebige Produkte (10+ Jahre), vorhersehbare Paket-Updates, IEC-62443–Compliance mit zum Beispiel IGLOS (iglos.com).

Medizintechnik

Rückverfolgbarkeit, Versionskontrolle und dokumentierte Build-Prozesse für Zertifizierungsprozesse (IEC 62304)

Automotive / Embedded Gateways

Schnelle Security-Patches durch direkten Einsatz von Debian Security Updates

LINUTRONIX

Fazit

Die Kraft von Debian nutzen

XML = komplettes System

Build heute → identisch in 2 Jahren

LINUTRONIX

Ressourcen

ELBE

- elbe-rfs.org
- github.com/linutronix/elbe
- https://github.com/rootfx-io/elbe-cookbook-recipes

Debian

- www.debian.org
- Snapshots: snapshot.debian.org