Bootable Containers

Vortrag im Linux-Cafe Nürnberg am 7.10.2026
Vortragender: Aldo Brießmann

Version des Vortrags: 1.1
Lizenz: CC BY-NC-SA 4.0 (https://creativecommons.org/licenses/by-nc-sa/4.0/)
Code und Materialien: https://codeberg.org/aldo112/vortrag-bootable-containers

Inhalt des Vortrags

- Einführung zu Container+Container-Images
- Fokus auf Fedora, keine anderen Linux-Distros mit bootc
- Vorführung bootc-System und Updates

Einführung: Container(-Images)

- Container-Image: Anwendung mit Abhängigkeiten „paketiert“
  - Format: Containerfile (Beispiel siehe Backup-Slides und Git-Repo)
  - Tools: Docker, Podman, Buildah, …
- Haupt-Verwendung von Containern: Abgekapselter Betrieb von Services
  - OS-Kernel wird vom Host-System genutzt

Einführung **Bootable** Containers

- **Bootable** Container: Container-Image mit Kernel und weiteren Komponenten (z.B. Init-System) drin, **benötigt nur noch Bootloader (z.B. Grub) außerhalb**
- Fedora-Varianten: (verschiedene bootc-Images)
  - CoreOS: „Server“-Variante von Fedora, bootc nutzbar
  - experimentelle „universal blue“-Systeme: Aurora (all-purpose), Bazzite (Gaming), Bluefin (workstation), uCore (Server)
  - ***Zukünftig geplant****: offizielle Fedora Atomic Desktops: Silverblue (Gnome), Kionite (KDE) und weitere*

Setup und Betrieb Bootc-System

- **Setup**: ähnlich wie bei anderen Linux-Distributionen
  - Beim Setup/Update wird der Inhalt des Container-Images auf der System-Partition (unter /ostree/) gespeichert und ein Bootloader installiert, und beim nächsten Boot wird der Container-Image-Inhalt als Root-Dateisystem gemountet
  - Persistente Ordner (z.B. /home, /root, /etc) liegen in eigener Partition (btrfs-subvolumes oder unter /var) und werden wie üblich gemountet
- **Updates**: bootc-Image-Download+Reboot
  - Zu vorheriger Version wechseln bei Problemen: Wechsel auf vorheriges Image mit Reboot und Auswahl im Bootloader und/oder Kommando

    `sudo bootc rollback`

Programm-Installation bei Bootc-Systemen

- **User-Anwendungen**: grundsätzlich Flatpak(/AppImage/Snap) → Programme liegen im Home-Verzeichnis des Users, nicht im Host-System-Speicher
- **CLI-Anwendungen** (für User und root): mit „`toolbox`“ in eigenem Container → ebenfalls im Home-Verzeichnis
- **Server-Anwendungen**: in bootc-Image einbauen (nur für viele gleiche Systeme sinnvoll) oder containerisiert betreiben (z.B. mit Podman-Quadlets)

Live-Demo 1

- Bootc-System (Fedora Bluefin)
- Anwendungs-Installationen:
  - z.B. Flatpak → „Bazar“-App-Store
  - CLI: mit dem CLI-Tool `toolbox` einen Container erstellen:
    - z.B. Fedora 44: `toolbox create --distro fedora --release 44`
    - Container-Shell öffnen: `` `toolbox enter` ``
    - In Toolbox-Shell/-Container nach Bedarf als Root Pakete installieren
    - Eigenes Home-Verzeichnis ist im Container automatisch verfügbar

Fedora uCore Setup

- Fedora-Coreos-Image (!) booten
- Ignition-File vorbereiten (min. Passwort+SSH-Key), siehe Materialien
- `sudo coreos-installer install /dev/sda --ignition-file $IGNITION_FILE`
  - Wichtig: falls ohne Ignition aufgesetzt, muss ein SSH-Key oder Passwort o.ä. separat eingerichtet werden
  - uCore-Ignition-File enthält Systemd-Service, der nach dem ersten Start vom Fedora-CoreOS-Image auf das uCore-Image wechselt

Verwendung von eigenen Images

- Voraussetzungen:
  - **Container-Image-Registry** (Docker Hub/quay.io/selfhosted/…), muss beim Update erreichbar sein
    - Andere Option: Image lokal bauen und nutzen
  - Möglichkeit zum Bauen von Images (lokal oder in Pipeline)
- Wechsel: Umstellung auf eigenes Image + Reboot

Live-Demo 2: Image-Bau

- Containerfile+Befehle siehe Materialien
- Vorgehensweise:
  1) Container-Image-Bau und -Push
  2) (optional: Test mit bcvk)
  3) Wechsel des Ziel-Systems auf neues Image und Reboot
-

Praxis-Erfahrungen

- Wegwerf-Root-Dateisystem macht ggf. Workarounds nötig (z.B. Service in Container betreiben statt einer normalen Installation)
  - Postinstall-Scripte von Paketen funktionieren teils nicht!
  - `rpm-ostree ...` im Containerfile kann /boot-Mount kaputtmachen
  - Hardware mit speziellen Boot-Anforderungen (z.B. Pi5) geht teils nicht oder nur kompliziert
- Updates gehen schneller
- Höhere Resistenz gegen „Kaputtbasteln“
  - (aber nicht voll, einige üblicherweise persistente Ordner bleiben erhalten)

Fazit

- Bootable containers sind sehr nützlich für:
  - generische Systeme ohne tiefgreifende Software- und Hardware-Eigenheiten
  - Server-/Desktop-Einsatz für viele (nahezu) gleiche Systeme
- Bei ordentlicher Verwendung von eigenen Images ist eine Container-Image-Registry (Cloud oder selbstgehostet) und ein automatisierter Image-Bau sehr empfehlenswert
- Für Bastel-Systeme sind Workarounds nötig, dann ist ein Nicht-Bootc-System ggf. einfacher zu handhaben

Weiterführende Themen

- Andere Linux-Distributionen mit bootable Containers
- Auto-Image-Update (Systemd-Service)
- Boot-Image-Anpassung (`image-builder` → noch experimentell)
- Automatisierter Image-Bau mit Pipeline
- Optimierungen beim Image-Bau (z.B. Image-Größe)
- Rollout+Betrieb von Server-Anwendungen auf bootc-Systemen
- Bootc-Image-Test mit `bcvk` → noch experimentell
- rpm-ostree („Vorgänger“ von bootc)

Links

- Code und Materialien zur Präsentation: https://codeberg.org/aldo112/vortrag-bootable-containers
- Allgemeine Informationen: https://bootc.dev
- Firmen hinter bootc: https://bootc.dev/about/adopters/
- Bootc-Images für mehrere Distros: https://github.com/bootcrew/mono
- Universal Blue: https://universal-blue.org/
  - Bluefin: https://projectbluefin.io/
  - Ucore: https://projectucore.org/ → https://projectucore.org/docs/first-install/
- Toolbox/“Toolbx“: https://containertoolbx.org/