mkosi-initrd: initrds built from system packages

Zbigniew Jędrzejewski-Szmek
zbyszek@in.waw.pl

ASG Berlin 2024, 26.9.2024

Why?

Why?

Current approach to initrds

Why?

Current approach to initrds

- take files from host fs,
  `ldd` to resolve dependencies

Why?

Current approach to initrds

- take files from host fs,
  `ldd` to resolve dependencies
- the packaging layer is duplicated

Why?

Current approach to initrds

- take files from host fs,
  `ldd` to resolve dependencies
- the packaging layer is duplicated
- lots of CPU cycles burnt during each kernel update

Why?

Current approach to initrds

- take files from host fs,
  `ldd` to resolve dependencies
- the packaging layer is duplicated
- lots of CPU cycles burnt during each kernel update

- at runtime: custom logic (e.g. dracut's initqueue)

Why?

Current approach to initrds

- take files from host fs,
  `ldd` to resolve dependencies
- the packaging layer is duplicated
- lots of CPU cycles burnt during each kernel update

- at runtime: custom logic (e.g. dracut's initqueue)
- custom tools (e.g. scripts to bring up LVM, dracut modules)

Why?

Current approach to initrds

- take files from host fs,
  `ldd` to resolve dependencies
- the packaging layer is duplicated
- lots of CPU cycles burnt during each kernel update

- at runtime: custom logic (e.g. dracut's initqueue)
- custom tools (e.g. scripts to bring up LVM, dracut modules)
- "custom" execution environment

Why?

Current approach to initrds

- take files from host fs,
  `ldd` to resolve dependencies
- the packaging layer is duplicated
- lots of CPU cycles burnt during each kernel update

- at runtime: custom logic (e.g. dracut's initqueue)
- custom tools (e.g. scripts to bring up LVM, dracut modules)
- "custom" execution environment
- complexity (in particular when dracut is used with systemd)

Why?

Current approach to initrds

- take files from host fs,
  `ldd` to resolve dependencies
- the packaging layer is duplicated
- lots of CPU cycles burnt during each kernel update

- at runtime: custom logic (e.g. dracut's initqueue)
- custom tools (e.g. scripts to bring up LVM, dracut modules)
- "custom" execution environment
- complexity (in particular when dracut is used with systemd)

- very little sharing of initrd logic between distros

Why?

Why?

See Wednesday's keynote:
"Reproducible and Immutable OS Images

Why?

See Wednesday's keynote:
"Reproducible and Immutable OS Images (with NixOS)"

Why?

See Wednesday's keynote:
"Reproducible and Immutable OS Images (with NixOS)"

UKI

Why?

See Wednesday's keynote:
"Reproducible and Immutable OS Images (with NixOS)"

UKI
=> combined kernel+initrd (and signed!)

Why?

See Wednesday's keynote:
"Reproducible and Immutable OS Images (with NixOS)"

UKI
=> combined kernel+initrd (and signed!)
=> initrd must be built by the vendor/distro

Why?

See Wednesday's keynote:
"Reproducible and Immutable OS Images (with NixOS)"

UKI
=> combined kernel+initrd (and signed!)
=> initrd must be built by the vendor/distro
=> no local modifications

Why?

See Wednesday's keynote:
"Reproducible and Immutable OS Images (with NixOS)"

UKI
=> combined kernel+initrd (and signed!)
=> initrd must be built by the vendor/distro
=> no local modifications
=> a system designed for local modifications is not useful

Why?

See Wednesday's keynote:
"Reproducible and Immutable OS Images (with NixOS)"

UKI
=> combined kernel+initrd (and signed!)
=> initrd must be built by the vendor/distro
=> no local modifications
=> a system designed for local modifications is not useful

if we are building in a package builder, let's build directly from distro packages
(we *could* build from files in the fs, but why?)

mkosi

"**Ma**K**e** **O**perating **S**ystem **I**mage"

Build OS images from distro packages (debs, rpms, ...)

mkosi

"**Ma**K**e** **O**perating **S**ystem **I**mage"

Build OS images from distro packages (debs, rpms, ...)

Uses `apt`, `dnf`, `dnf5`, `zypper`, `pacman`

mkosi

"**Ma**K**e** **O**perating **S**ystem **I**mage"

Build OS images from distro packages (debs, rpms, ...)

Uses `apt`, `dnf`, `dnf5`, `zypper`, `pacman`

Now uses `systemd-repart` → fully unprivileged operation

mkosi

"**Ma**K**e** **O**perating **S**ystem **I**mage"

Build OS images from distro packages (debs, rpms, ...)

Uses `apt`, `dnf`, `dnf5`, `zypper`, `pacman`

Now uses `systemd-repart` → fully unprivileged operation

Profiles and [Match] sections → flexibility

mkosi

"**Ma**K**e** **O**perating **S**ystem **I**mage"

Build OS images from distro packages (debs, rpms, ...)

Uses `apt`, `dnf`, `dnf5`, `zypper`, `pacman`

Now uses `systemd-repart` → fully unprivileged operation

Profiles and [Match] sections → flexibility

Sam Leonard, "Improving systemd's integration testing infrastructure"
Jelle van der Waa, "Creating Arch Linux images using mkosi"

mkosi

"**Ma**K**e** **O**perating **S**ystem **I**mage"

Build OS images from distro packages (debs, rpms, ...)

Uses `apt`, `dnf`, `dnf5`, `zypper`, `pacman`

Now uses `systemd-repart` → fully unprivileged operation

Profiles and [Match] sections → flexibility

Sam Leonard, "Improving systemd's integration testing infrastructure"
Jelle van der Waa, "Creating Arch Linux images using mkosi"
Daan De Meyer, "A re-introduction to mkosi — A Tool for Generating OS Images"
"mkosi: Building Bespoke Operating System Images" @ ASG 2023
"systemd-repart: Building Discoverable Disk Images" @ ASG 2023

Why distro packages?

Why distro packages?

- **less** things

Why distro packages?

- **less** things
- we use package dependency resolution mechanism

Why distro packages?

- **less** things
- we use package dependency resolution mechanism
- we let rpm/deb/pacman handle 98% of the installation

Why distro packages?

- **less** things
- we use package dependency resolution mechanism
- we let rpm/deb/pacman handle 98% of the installation
- we don't pull files from the host

Why distro packages?

- **less** things
- we use package dependency resolution mechanism
- we let rpm/deb/pacman handle 98% of the installation
- we don't pull files from the host
- images can be reproducible

Why distro packages?

- **less** things
- we use package dependency resolution mechanism
- we let rpm/deb/pacman handle 98% of the installation
- we don't pull files from the host
- images can be reproducible
- images are the same for everyone

Why distro packages?

- **less** things
- we use package dependency resolution mechanism
- we let rpm/deb/pacman handle 98% of the installation
- we don't pull files from the host
- images can be reproducible
- images are the same for everyone
- images signed by vendor

Why distro packages?

- **less** things
- we use package dependency resolution mechanism
- we let rpm/deb/pacman handle 98% of the installation
- we don't pull files from the host
- images can be reproducible
- images are the same for everyone
- images signed by vendor
- systemd does the heavy lifting in the initrd

Nitty-gritty

How the idea of mkosi-initrd has evolved

How the idea of mkosi-initrd has evolved

0. PoC, config for mkosi, works on a Thinkpad

How the idea of mkosi-initrd has evolved

0. PoC, config for mkosi, works on a Thinkpad

1. Goal: conquer the world — iscsi, fcoe, nfs, raid, kdump, networking!!

How the idea of mkosi-initrd has evolved

...ctd., example

How the idea of mkosi-initrd has evolved

...ctd., example

iscsi: iscsi-initiator-utils $\rightarrow$ 4 binaries, 6 service files

How the idea of mkosi-initrd has evolved

...ctd., example

iscsi: iscsi-initiator-utils → 4 binaries, 6 service files

/usr/lib/dracut/modules.d/95iscsi/cleanup-iscsi.sh
/usr/lib/dracut/modules.d/95iscsi/iscsiroot.sh
/usr/lib/dracut/modules.d/95iscsi/module-setup.sh
/usr/lib/dracut/modules.d/95iscsi/mount-lun.sh
/usr/lib/dracut/modules.d/95iscsi/parse-iscsiroot.sh
→ approx. 1000 lines of bash code generating bash code to wrap the binaries

How the idea of mkosi-initrd has evolved

...ctd., example

iscsi: iscsi-initiator-utils → 4 binaries, 6 service files

/usr/lib/dracut/modules.d/95iscsi/cleanup-iscsi.sh
/usr/lib/dracut/modules.d/95iscsi/iscsiroot.sh
/usr/lib/dracut/modules.d/95iscsi/module-setup.sh
/usr/lib/dracut/modules.d/95iscsi/mount-lun.sh
/usr/lib/dracut/modules.d/95iscsi/parse-iscsiroot.sh
→ approx. 1000 lines of bash code generating bash code to wrap the binaries

```
# iscsi-init.service
[Service]
ExecStart=/usr/bin/sh -c
  'echo "InitiatorName=`/usr/sbin/iscsi-iname`"
  > /etc/iscsi/initiatorname.iscsi'
```

How the idea of mkosi-initrd has evolved

...ctd., example

iscsi: iscsi-initiator-utils → 4 binaries, 6 service files

/usr/lib/dracut/modules.d/95iscsi/cleanup-iscsi.sh
/usr/lib/dracut/modules.d/95iscsi/iscsiroot.sh
/usr/lib/dracut/modules.d/95iscsi/module-setup.sh
/usr/lib/dracut/modules.d/95iscsi/mount-lun.sh
/usr/lib/dracut/modules.d/95iscsi/parse-iscsiroot.sh
→ approx. 1000 lines of bash code generating bash code to wrap the binaries

```
# iscsi-init.service
[Service]
ExecStart=/usr/bin/sh -c
  'echo "InitiatorName=`/usr/sbin/iscsi-iname`"
  > /etc/iscsi/initiatorname.iscsi'
```

```
$ dracut --list-modules | wc -l
119
```

How the idea of mkosi-initrd has evolved

...ctd.

How the idea of mkosi-initrd has evolved

...ctd.

2. Updated goal: make “new initrds” work correctly for simple cases, build the infrastructure

How the idea of mkosi-initrd has evolved

...ctd.

2. Updated goal: make “new initrds” work correctly for simple cases, build the infrastructure

3. Allow easy local use, preprare for centralized builds

Initrds and ... exitrds

Initrds and ... exitrds

In the dracut approach, the initrd saves a compressed subset of itself in memory to unpack and execute shutdown.

Initrds and ... exitrds

In the dracut approach, the initrd saves a compressed subset of itself in memory to unpack and execute shutdown.

Why waste memory? Why execute old code?

Initrds and ... exitrds

In the dracut approach, the initrd saves a compressed subset of itself in memory to unpack and execute shutdown.

Why waste memory? Why execute old code?

"exitrd" — code to execute at shutdown to clean up the root file system

Initrds and ... exitrds

In the dracut approach, the initrd saves a compressed subset of itself in memory to unpack and execute shutdown.

Why waste memory? Why execute old code?

"exitrd" — code to execute at shutdown to clean up the root file system

systemd has this covered: `systemd-shutdown` `systemd-standalone-shutdown.rpm`

Next steps

Consequences of centralized builds

If we use pre-built images, how to deliver differentiated code?

Consequences of centralized builds

If we use pre-built images, how to deliver differentiated code?

0. initrd variants
1. "addons" → checked via SecureBoot db / shim
2. systemd-sysexts → checked via kernel keyring
3. systemd-confexts
4. credentials → encrypted via TPM|local key

steps towards world domination

steps towards world domination

mkosi-initrd is now part of mkosi,
installed as `/usr/lib/kernel/install.d/50-mkosi.install`

steps towards world domination

mkosi-initrd is now part of mkosi,
installed as `/usr/lib/kernel/install.d/50-mkosi.install`

support for openssl “signing engines”, sbsign, pesign

steps towards world domination

mkosi-initrd is now part of mkosi,
installed as `/usr/lib/kernel/install.d/50-mkosi.install`

support for openssl “signing engines”, sbsign, pesign
(open pull request for) offline signing

steps towards world domination

mkosi-initrd is now part of mkosi,
installed as `/usr/lib/kernel/install.d/50-mkosi.install`

support for openssl “signing engines”, sbsign, pesign
(open pull request for) offline signing

mkosi can build sysexts, but it’s still PoC

steps towards world domination

mkosi-initrd is now part of mkosi,
installed as `/usr/lib/kernel/install.d/50-mkosi.install`

support for openssl “signing engines”, sbsign, pesign
(open pull request for) offline signing

mkosi can build sysexts, but it's still PoC

mkosi / mkosi-initrd can detect the CPU vendor and build µcode
initrd => `.ucode` section

The “big initrd” problem

The “big initrd” problem

the problem is general: drivers, firmware, composefs, ...

The “big initrd” problem

the problem is general: drivers, firmware, composefs, ...

copying is fast, decompression is slow => avoid decompression

The “big initrd” problem

the problem is general: drivers, firmware, composefs, ...

copying is fast, decompression is slow => avoid decompression

the kernel may allow zero-copy donation of fs image

Summary: short list of tools & concepts

Summary: short list of tools & concepts

mkosi
mkosi-initrd | 50-mkosi.install
ukify | 60-ukify.install
kernel-install
systemd-measure
pesign | sbsign
addons | sysexts | confexts | credentials
sd-stub
reproducible builds
UKIs | multi-profile UKIs | incremental builds | offline signing
cpio | squashfs | EROFS | zero-copy unpacking
immutable & signed initrd

Links

https://github.com/systemd/mkosi
https://www.freedesktop.org/software/systemd/man/systemd-sysext.html
https://gitlab.com/cryptsetup/cryptsetup/-/wikis/DMVerity
https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/verity.html
https://www.kernel.org/doc/html/latest/filesystems/overlayfs.html

These slides: https://github.com/keszybz/mkosi-initrd-talk/raw/main/asg2024-mkosi-initrd.pdf

Links

https://github.com/systemd/mkosi
https://www.freedesktop.org/software/systemd/man/systemd-sysext.html
https://gitlab.com/cryptsetup/cryptsetup/-/wikis/DMVerity
https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/verity.html
https://www.kernel.org/doc/html/latest/filesystems/overlayfs.html

These slides: https://github.com/keszybz/mkosi-initrd-talk/raw/main/asg2024-mkosi-initrd.pdf

QUESTIONS? / EOF

Objections?

Objections?

- systemd was already used in the initrd

Objections?

- systemd was already used in the initrd
- the first thing systemd does is to set up the environment

Objections?

- systemd was already used in the initrd
- the first thing systemd does is to set up the environment
- having tools that support running in a custom environment is hence not useful

Objections?

- systemd was already used in the initrd
- the first thing systemd does is to set up the environment
- having tools that support running in a custom environment is hence not useful
- after removing custom logic we don't need to add anything back

Objections?

- systemd was already used in the initrd
- the first thing systemd does is to set up the environment
- having tools that support running in a custom environment is hence not useful
- after removing custom logic we don't need to add anything back
- the ecosystem is moving away from scripts towards compiled daemons

Objections?

- systemd was already used in the initrd
- the first thing systemd does is to set up the environment
- having tools that support running in a custom environment is hence not useful
- after removing custom logic we don't need to add anything back
- the ecosystem is moving away from scripts towards compiled daemons
- most of the code is in shared libraries, which are installed in full because of link dependencies

Objections?

- systemd was already used in the initrd
- the first thing systemd does is to set up the environment
- having tools that support running in a custom environment is hence not useful
- after removing custom logic we don't need to add anything back
- the ecosystem is moving away from scripts towards compiled daemons
- most of the code is in shared libraries, which are installed in full because of link dependencies
- error handling, timeouts, retries, localized messages, event-driven logic, netlink, D-bus, all are much easier with "real" code