Mkosi-initrd (in Fedora)

Zbigniew Jędrzejewski-Szmek

zbyszek@in.waw.pl

DevConf Brno 2023, 18.6.2023

(instead of) Intro

(instead of) Intro

see Vitaly Kuzentsov's
"Confidential VMs in the cloud"
Christophe de Dinechin's
"Confidential Computing, from host to workload"

(instead of) Intro

see Vitaly Kuzentsov's
"Confidential VMs in the cloud"
Christophe de Dinechin's
"Confidential Computing, from host to workload"

signed code only

(instead of) Intro

see Vitaly Kuzentsov's
"Confidential VMs in the cloud"
Christophe de Dinechin's
"Confidential Computing, from host to workload"

signed code only
=> kernel+initrd combined into a UKI (and signed!)

(instead of) Intro

see Vitaly Kuzentsov's
"Confidential VMs in the cloud"
Christophe de Dinechin's
"Confidential Computing, from host to workload"

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

(instead of) Intro

see Vitaly Kuzentsov's
"Confidential VMs in the cloud"
Christophe de Dinechin's
"Confidential Computing, from host to workload"

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

(instead of) Intro

see Vitaly Kuzentsov's
"Confidential VMs in the cloud"
Christophe de Dinechin's
"Confidential Computing, from host to workload"

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

(instead of) Intro

see Vitaly Kuzentsov's
"Confidential VMs in the cloud"
Christophe de Dinechin's
"Confidential Computing, from host to workload"

signed code only
=> kernel+initrd combined into a UKI (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?)

Current approach to initrds

Current approach to initrds

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

Current approach to initrds

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

Current approach to initrds

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

Current approach to initrds

- local builds 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)

Current approach to initrds

- local builds 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)

Current approach to initrds

- local builds 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)
- different execution environment

Current approach to initrds

- local builds 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)
- different execution environment
- complexity (in particular when dracut is used with systemd)

Current approach to initrds

- local builds 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)
- different execution environment
- complexity (in particular when dracut is used with systemd)

- very little sharing of initrd logic between distros

mkosi

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

Build OS images from distro packages (`dnf`, `apt`, `pacman`)

mkosi

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

Build OS images from distro packages (`dnf`, `apt`, `pacman`)

Now uses `systemd-repart` → (partially) unprivileged operation

dnf5

Profiles and [Match] sections

[mkosi 15, not released yet]

What is mkosi-initrd?

What is mkosi-initrd?

Just a few config files for mkosi ;)

`https://github.com/systemd/mkosi-initrd`

Benefits

- **less** things

Benefits

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

Benefits

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

Benefits

- **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

Benefits

- **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

Benefits

- **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

Benefits

- **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 can be easily signed

Benefits

- **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 can be easily signed
- systemd does the heavy lifting in the initrd

Drawbacks

We get fully functional initrds, but...

Drawbacks

We get fully functional initrds, but...

The initrds are **bigger**.

Drawbacks

We get fully functional initrds, but...

The initrds are **bigger**.
Most of the difference is caused by kernel modules

Drawbacks

We get fully functional initrds, but...

The initrds are **bigger**.
Most of the difference is caused by kernel modules

Only some subset of installations is supported

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

Consequences of centralized builds

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

0. initrd variants
1. credentials

Consequences of centralized builds

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

0. initrd variants
1. credentials
2. systemd-sysexts

Consequences of centralized builds

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

0. initrd variants
1. credentials
2. systemd-sysexts
3. systemd-confexts

Consequences of centralized builds

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

0. initrd variants
1. credentials
2. systemd-sysexts
3. systemd-confexts
4. “addons”

1st extension mechanism: credentials

Credentials

Credentials

A generic mechanism:

data $\rightarrow$
file (`/etc/credstore/data`) | other storage $\rightarrow$
service has `LoadCredential=data` $\rightarrow$
manager passes the credential $\rightarrow$
service sees `$CREDENTIALS_DIRECTORY/data`

Credentials

A generic mechanism:

data →
file (`/etc/credstore/data`) | other storage →
service has `LoadCredential=data` →
manager passes the credential →
service sees `$CREDENTIALS_DIRECTORY/data`

file | pipe |
qemu `SMBIOS` | `fw_cfg` |
kernel command-line |
boot loader |
inherited credential

Credential encryption

(with machine keys)

Credential encryption

(with machine keys)

data →
`systemd-creds encrypt` →
file | other storage →
`LoadCredential`**`Encrypted=`** →
manager decrypts →
service sees `$CREDENTIALS_DIRECTORY/data`

Credential encryption

(with machine keys)

data →
`systemd-creds encrypt` →
file | other storage →
`LoadCredential`**`Encrypted=`** →
manager decrypts →
service sees `$CREDENTIALS_DIRECTORY/data`

Encryption with:
- `/var/lib/systemd/credential.secret`
- TPM2
- both

`https://systemd.io/CREDENTIALS`

2nd extension mechanism: confexts

confexts (and sysexts)

"Configuration Extensions"
"System Extensions"

sysext — partial system image that is `overlayfs`ed on the host system: `/usr` and `/opt`

confexts (and sysexts)

"Configuration Extensions"
"System Extensions"

sysext — partial system image that is `overlayfs`ed on the host system: `/usr` and `/opt`

confext — partial system image that is `overlayfs`ed on the host system: `/etc`

Discoverable Disk Images

The Discoverable Partitions Specification
(recognition of file system role by part-type UUID)

Discoverable Disk Images

The Discoverable Partitions Specification
(recognition of file system role by part-type UUID)

`systemd-dissect <image>`

Discoverable Disk Images

The Discoverable Partitions Specification
(recognition of file system role by part-type UUID)

```
systemd-dissect <image>
```

```
systemd-dissect --mount <image> <path>
```

Discoverable Disk Images

The Discoverable Partitions Specification
(recognition of file system role by part-type UUID)

```
systemd-dissect <image>

systemd-dissect --mount <image> <path>

systemd-dissect --mtree <image>
```

systemd-dissect <image>

```
zbyszek@x1c ~/s/systemd-work (fix-root-resize-new)> sudo build/systemd-dissect mkosi.output/final
      Name: final
      Size: 1.4G
 Sec. Size: 512
     Arch.: x86-64

Image UUID: eb23b587-59fa-4259-83f4-65bc0cbac54e
OS Release: NAME=Fedora Linux
            VERSION=38 (Thirty Eight)
            ID=fedora
            VERSION_ID=38
            SUPPORT_END=2024-05-14

    Use As: ✓ bootable system for UEFI
            ✓ bootable system for container
            ✗ portable service
            ✗ initrd
            ✗ extension for system
            ✗ extension for initrd
            ✗ extension for portable service

RW DESIGNATOR     PARTITION U PARTITION LABEL       FSTYPE               ARCHITECTURE VERITY GROWFS NODE
ro usr            c166fc47-e4 usr-x86-64            erofs                x86-64       no         no /dev/loop1p2
rw esp            6bb89bb2-e9 esp                   vfat                 -            -          no /dev/loop1p1
ro usr-verity     e2968b7e-ea usr-x86-64-verity     DM_verity_hash       x86-64       -          no /dev/loop1p3
ro usr-verity-sig 5f52932e-a3 usr-x86-64-verity-sig verity_hash_signature x86-64       -          no /dev/loop1p4
```

3rd extension mechanism: “addons”

“Addons”

UKI-like binaries with kernel parameters

How to deliver differentiated code?

How to deliver differentiated code?

0. initrd variants

How to deliver differentiated code?

0. initrd variants
1. credentials $\rightarrow$ encrypted via TPM

How to deliver differentiated code?

0. initrd variants
1. credentials $\rightarrow$ encrypted via TPM
2. systemd-sysexts $\rightarrow$ checked via kernel keyring
3. systemd-confexts

How to deliver differentiated code?

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

Unified Kernel Images

Recent work

Lots of work on tooling for Unified Kernel Images (UKIs)
- ukify: python-pefile, config files, sbsign/pesign, addons, SBAT
- `systemd-measure` to precalculate PCR measurement after boot
- `kernel-install/60-ukify.install` (`initrd_generator=ukify`)
- `kernel-install/90-uki-copy.install` (`layout=uki`)

- `bootctl kernel-identify / kernel-inspect`

Recent work – other areas

- `mkosi-initrd` has a growing test suite (booting different storage types)
- `mkosi` supports builds as an unprivileged user
- Fedora 38 Change for Unified Kernel Images for VMs
- New kernel package split in Fedora (`kernel-modules-core`)
- GRUB2 might get support for UKIs (`https://github.com/osteffenrh/grub2`)
- Fedora 39 Change for `mkosi-initrd` (`https://fedoraproject.org/wiki/Changes/mkosi-initrd`)

Links

https://github.com/systemd/mkosi
https://github.com/systemd/mkosi-initrd
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/devconf2023-mkosi-initrd-fedora.pdf

Links

https://github.com/systemd/mkosi
https://github.com/systemd/mkosi-initrd
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/devconf2023-mkosi-initrd-fedora.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