Stateless Systems, Factory Reset, Golden Master Systems and systemd

LinuxCon Europe, Duesseldorf

October 2014

Factory Reset?

Factory Reset?

The procedure to bring a system back into the state that is was shipped in.

Stateless System?

Stateless System?

A system where every single boot-up is as if a factory reset was just completed.

Golden Master?

The one master image a factory reset returns the state to.

Golden Master?

The one master image a factory reset returns the state to.

The same image is usually shared between a multitude of systems.

Where do you want this?

Where do you want this?

Containers,

Where do you want this?

Containers, servers,

Where do you want this?

Containers, servers, laptops/desktops/tablets,

Where do you want this?

Containers, servers, laptops/desktops/tablets, mobile,

Where do you want this?

Containers, servers, laptops/desktops/tablets, mobile, embedded

Where do you want this?

Containers, servers, laptops/desktops/tablets, mobile, embedded

Verifiable setups

Where do you want this?

Containers, servers, laptops/desktops/tablets, mobile, embedded

Verifiable setups

Apps

Tons of prior art: Android, ChromeOS, CoreOS, virtualization infrastructure, and many embedded systems

Our goal with working on this in the systemd context: to solve this in a modular and generic way, for all usecases

Our goal with working on this in the systemd context: to solve this in a modular and generic way, for all usecases

Right in the OS itself.

Let's seperate state from OS resources!

Let's seperate state from OS resources!

/etc: configuration

Let's seperate state from OS resources!

/etc: configuration

/var: state

Let's seperate state from OS resources!

/etc: configuration

/var: state

/usr: vendor OS resources

Let's seperate state from OS resources!

/etc: configuration

/var: state

/usr: vendor OS resources

(after the /usr merge)

Flushing /etc, /var, just keeping /usr: full factory reset

Flushing just /var, keeping /usr and /etc: keeping settings, but dropping collected state

Booting with /var empty?

Booting with /var empty?

Mostly just works, just a few more tmpfiles rules

Booting with /var empty?

Mostly just works, just a few more tmpfiles rules

What's tmpfiles again?

What's tmpfiles again?

```
d /var 0755 - - -

L /var/run - - - - ../run

d /var/log 0755 - - -
f /var/log/wtmp 0664 root utmp -
f /var/log/btmp 0600 root utmp -

d /var/cache 0755 - - -

d /var/lib 0755 - - -
d /var/spool 0755 - - -
```

Booting with /etc empty?

Booting with /etc empty?

More complex

Booting with /etc empty?

More complex

Software is more allergic if configuration files in /etc are missing

Booting with /etc empty?

More complex

Software is more allergic if configuration files in /etc are missing

User database!

Booting with /etc empty?

More complex

Software is more allergic if configuration files in /etc are missing

User database!

Core OS components shipped by systemd are fixed

Booting with /etc empty?

More complex

Software is more allergic if configuration files in /etc are missing

User database!

Core OS components shipped by systemd are fixed

Exception in the core OS: dbus, PAM

tmpfiles to the rescue:

```
C /etc/pam.d
C /etc/nsswitch.conf
```

Introducing: /usr/share/factory/etc

sysusers to the rescue:

```
u root    0     "Super User" /root
u nobody  65534 "Nobody"     -
g adm     -     -            -
g wheel   -     -            -
g kmem    -     -            -
g lock    -     -            -
g tty     5     -            -
g utmp    -     -            -
g audio   -     -            -
g cdrom   -     -            -
g dialout -     -            -
g disk    -     -            -
g input   -     -            -
g lp      -     -            -
g tape    -     -            -
g video   -     -            -
g users   -     -            -
```

systemd-nspawn –volatile=no -b -D /srv/mycontainer systemd-nspawn –volatile=no -b -D /srv/mycontainer

systemd-nspawn –volatile=state -b -D /srv/mycontainer

```
systemd-nspawn –volatile=no -b -D /srv/mycontainer
systemd-nspawn –volatile=state -b -D /srv/mycontainer
systemd-nspawn –volatile=yes -b -D /srv/mycontainer
```

Updating

Updating

/usr can be updated offline

Updating

/usr can be updated offline

On next boot, /etc and /var are updated

Updating

/usr can be updated offline

On next boot, /etc and /var are updated

ConditionNeedsUpdate=

Updating

/usr can be updated offline

On next boot, /etc and /var are updated

ConditionNeedsUpdate=

ldconfig, sysusers, udev hwdb, . . .

Updating

/usr can be updated offline

On next boot, /etc and /var are updated

ConditionNeedsUpdate=

ldconfig, sysusers, udev hwdb, ...

All atomic

Double Buffering

Double Buffering

Multiple /usr trees around!

RPM?

Classic Distributions?

Timeframe?

Apps!

Apps!

/usr: os, runtime, framework

Apps!

/usr: os, runtime, framework

/opt/*appname*: app

OS: a /usr one can boot up a system with

OS: a /usr one can boot up a system with

Runtime: a /usr one can run executables against

OS: a /usr one can boot up a system with

Runtime: a /usr one can run executables against

Framework: a /usr one can build executables with

OS, Runtime, Framework, Instance, Apps

OS, Runtime, Framework, Instance, Apps

All in multiple versions on the same system

OS, Runtime, Framework, Instance, Apps

All in multiple versions on the same system

btrfs subvolumes

btrfs???

Clear naming Scheme for subvolumes

Clear naming Scheme for subvolumes

usr:*vendorid*:*architecture*:*version*

Clear naming Scheme for subvolumes

usr:*vendorid*:*architecture*:*version*

root:*name*:*vendorid*:*architecture*

Clear naming Scheme for subvolumes

usr:*vendorid*:*architecture*:*version*

root:*name*:*vendorid*:*architecture*

runtime:*vendorid*:*architecture*:*version*

Clear naming Scheme for subvolumes

usr:*vendorid*:*architecture*:*version*

root:*name*:*vendorid*:*architecture*

runtime:*vendorid*:*architecture*:*version*

framework:*vendorid*:*architecture*:*version*

Clear naming Scheme for subvolumes

usr:*vendorid*:*architecture*:*version*

root:*name*:*vendorid*:*architecture*

runtime:*vendorid*:*architecture*:*version*

framework:*vendorid*:*architecture*:*version*

app:*vendorid*:*runtime*:*architecture*:*version*

Namespaces!

Namespaces!

Dynamic views on the system, for containers and apps

Namespaces!

Dynamic views on the system, for containers and apps

Multiple root subvolumes sharing the same usr subvolume!

Namespaces!

Dynamic views on the system, for containers and apps

Multiple root subvolumes sharing the same usr subvolume!

Multiple app subvolumes sharing the same runtime subvolume!

Delivery:

Delivery:

btrfs send/recv deltas via http

Delivery:

btrfs send/recv deltas via http

Same for OS, runtimes, frameworks and apps

OS installation:

OS installation:

1. Create GPT table with ESP + btrfs

OS installation:

1. Create GPT table with ESP + btrfs
2. Deserialize usr tree into btrfs

OS installation:

1. Create GPT table with ESP + btrfs
2. Deserialize usr tree into btrfs
3. Install bootloader into ESP

OS installation:

1. Create GPT table with ESP + btrfs
2. Deserialize usr tree into btrfs
3. Install bootloader into ESP
4. Profit!

http://0pointer.net/blog/projects/stateless.html

http://0pointer.net/blog/
revisiting-how-we-put-together-linux-systems.html

That's all, folks!