init rides the rocket: systemd is here

Olaf Kirch
Director SUSE Linux Enterprise
okir@suse.com

SUSE®
We adapt. You succeed.

Love it or hate it?

1996: Linux Distros adopt SysV-init

2001: LSB standardizes init scripts

2010: Mobile Distros and Parallel Boot

2027: Still in Good Shape?

HOW ABOUT ENDING THIS CARTOON BEFORE I HIT?

Why did you do this to me?!

What's wrong with sysvinit?

- Nothing
- But it could do many things better
  - It's slow
  - It's hard to parallelize
  - Too coarse synchronization points
  - LSB dependencies only do what you need 50% of the time
  - Have you ever tried to kill all processes spawned by a user session?
  - No automatic restart of services
  - No unified logging
  - No unified resource limit handling

Why systemd

- Considered several alternatives, systemd came out on top for several reasons
- It's not Marmite, but it certainly polarizes the users today
  - Some of that is certainly because it's new
- It seems to be the solution most distros are standardizing on

Why is it so hard to like systemd, then?

- It isn't, once you get to know to it better
- It changes a lot of things
  - You need to learn a lot of new commands
    - But let's be honest, many of them do things you could never do with sysvinit
  - Some time-honored features (like `inittab`) are simply gone
  - It is an intrusive change, and fixing up the fallout has been a significant amount of work
  - “Backward compatibility” is not high on the list of development priorities
- That said, some level of backward compatibility is possible, and we're providing that

SUSE backward compatibility

- insserv, chkconfig and /sbin/service will still be supported
- Old style "`rcfoobar start`" redirected to new systemd tools automatically
- LSB compatibility for targets like `$network`... still available
- And of course init scripts are still supported!

Systemd Concepts

Generic concepts

- systemd replaces the traditional init process
  - It's not one, but a collection of DBus services
- Activate services on demand rather than up-front
- One-stop shopping for boot, shutdown and power management
  - integrated logging
  - unified command line tools for almost everything
  - automatic restart of services
  - cgroups and security compartments for everything
  - multi-seat hosts
  - handling of ACPI power management events

Unit files

- Unit files describe *targets* (i.e. synchonization points) and *services* (what used to be init scripts)
- Runlevels are replaced by targets
  - Runlevel 3: multi-user.target
  - Runlevel 5: graphical.target
- Much of what the LSB standard used is modeled in unit files
  - Plus a few more, for instance with LSB you could never say “my service needs to be started before kdm”

Sessions and Seats

- A *seat* is the set of hardware available at one work place (graphics card, keyboard, mouse, usb devices)
- A *session* is created once a user is logged on, using a specific seat
  - Only one session can be active per seat
  - Default seat (for Linux consoles) is **seat0**
- Hardware is assigned to seats
  - This replaces ConsoleKit

Sample service unit file: sshd.service

```
# This is a comment!
[Unit]
Description=OpenSSH Daemon
After=network.target

[Service]
EnvironmentFile=/etc/sysconfig/ssh
ExecStartPre=/usr/sbin/sshd-gen-keys-start
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=always

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

LSB Analogs/Equivalents

```
[Unit]
# Description: ...
# Required-Start: $network

[Service]
# <- All of these used to be
# <- open coded in the init script
# <-
# <-
# <-
Automated Restart: didn't exist previously

[Install]
# Default-Start: 3 5
```

Cgroups for Everything

- Systemd puts each service and each session into a separate cgroup
  - Sessions also get assigned an audit ID matching their cgroup ID
- You can restrict these cgroups in all the way the kernel supports
  - IO bandwidth, memory or CPU consumption, etc

Improved Security for Everything

- Restrict services and sessions using namespaces
  - Linux kernel namespaces are the technology underlying Linux containers
  - blacklist directories
  - require private /tmp directory
  - whitelist devices to which access is granted
- specify user/group to run as
- assign Linux kernel capabilities (CAP_FOOBAR)
- set ulimit values

Overriding defaults for a service

- With sysvinit, if you want to do anything more advanced that enable/disable a service, you need to edit the init script
  - This doesn't go well with security updates
- Systemd supports that nicely
  - Assume you want to modify settings for **foobar.service**
  - Create **/etc/systemd/system/foobar.service.d**
  - Drop a file named **mysettings.conf** in there:

```
[Service]
InaccessibleDirectories=/precious
MemoryLimit=1G
```

Getting Started with Systemd Tools

Very Rough Cheat Sheet

- The following overview is far from exhaustive
- This is just meant as a starting point to help you exploring systemd and its tools

Service status

- You want a list of all started services and their status

```
systemctl
```

- You want the status of service **foobar**:

```
systemctl status foobar.service
```

```
$ systemctl status icecream.service
icecream.service - LSB: icecc
      Loaded: loaded (/etc/init.d/icecream)
      Active: active (running) since Fri, 2013-04-19 09:27:31 CEST; 4 days ago
      CGroup: name=systemd:/system/icecream.service
          ├ 4786 /usr/sbin/icecc-scheduler -d -l /var/log/icecc_sch...
          └ 4791 /usr/sbin/iceccd -d -l /var/log/iceccd --nice 5 -u...

Apr 19 09:27:31 foobar systemd[1]: Starting LSB: icecc...
Apr 19 09:27:31 foobar icecream[4777]: Starting Distribut...
Apr 19 09:27:31 foobar systemd[1]: Started LSB: icecc.
```

Starting and Stopping Services

- `systemctl <verb> foobar.service`
  - Where <*verb*> is one of start, stop, restart, try-restart, reload
- `systemctl kill foobar.service`
  - Kill all processes in the cgroup of this service

Enabling and Disabling Services

- `systemctl <verb> foobar.service`
  - Where <*verb*> is one of enable, disable

We'll still be friends, pstree

```
# systemd-cgls

 system
  ├ 1 /sbin/init showopts
  ├ icecream.service
  │ ├ 4786 /usr/sbin/icecc-scheduler -d -l /var/log/icecc_scheduler
  │ └ 4791 /usr/sbin/iceccd -d -l /var/log/iceccd --nice 5 -u icecream -b /...
  ├ colord.service
  │ └ 1677 /usr/lib/colord
  ├ udisks2.service
  │ └ 1498 /usr/lib/udisks2/udisksd --no-debug
  ├ rtkit-daemon.service
  │ └ 1353 /usr/lib/rtkit/rtkit-daemon
  ├ upower.service
  │ └ 1161 /usr/lib/upower/upowerd
  ├ accounts-daemon.service
  │ └ 1125 /usr/lib/accounts-daemon
  ├ xdm.service
  │ ├  964 /usr/sbin/gdm
  │ ├  966 /usr/lib/gdm/gdm-simple-slave --display-id /org/gnome/DisplayMan...
  │ ├ 1021 /usr/bin/Xorg :0 -background none -verbose -auth /run/gdm/auth-f...
  │ └ 1515 gdm-session-worker [pam/gdm-password]
```

Session Handling

- List all sessions:
  - `loginctl [list-sessions]`
- Show session details:
  - `loginctl session-status <session-number>`
- Forcefully terminate a session:
  - `loginctl kill-session|kill-user|terminate-seat <name>`

Session Handling, continued

```
2 - fcrozat (1000)
           Since: lun. 2013-07-29 11:58:41 CEST; 4h 13min ago
          Leader: 1550 (gdm-session-wor)
            Seat: seat0; vc7
         Display: :0
         Service: gdm-password; type x11; class user
           State: active
          CGroup: systemd:/user/1000.user/2.session
                     1550 gdm-session-worker [pam/gdm-password]
                     1557 /usr/bin/gnome-keyring-daemon --daemonize
                     1560 /usr/bin/gnome-session
```

And it comes with lots more stuff

- See how your configuration differs from the vendor defaults
  - `systemd-delta`
- Analyze boot times and bottlenecks
  - `systemd-analyze`

References

- Systemd on SLE12 :
  - Check our official documentation
- Upstream:
  - http://www.freedesktop.org/wiki/Software/systemd/
  - Check manpages, they are extremely verbose (if something is missing there, it is a bug !)

You are now ready to test systemd in SLE12

Thank you.

SUSE®
We adapt. You succeed.