Lucas Nussbaum
lucas.nussbaum@univ-lorraine.fr

Licence professionnelle ASRALL
Administration de systèmes, réseaux et applications à base de logiciels libres

License: GNU General Public License version 3 or later
or Creative Commons BY-SA 3.0 Unported
(see README.md)
Outline

1. Introduction
2. Behind the scenes: cgroups
3. Managing services
4. Analyzing startup performance
5. Exploring the system status
6. Configuring services by writing unit files
7. Timer units
8. Socket activation
9. Logging with journald
10. Containers integration
11. Networking with systemd-networkd
12. Migration from sysvinit
13. Conclusions
Init system

- First process started by the kernel (pid 1)
- Responsible for *bringing up the rest of userspace*
  - Mounting filesystems
  - Starting services
  - . . .
- Also the parent for orphan processes
- Traditional init system on Linux: *sysVinit*
  - Inherited from Unix System V
  - With additional tools (insserv, startpar) to handle dependencies and parallel initialization
systemd

▶ Written (since 2010) by Lennart Poettering (Red Hat) and others
▶ Now the default on most Linux distributions
▶ Shifts the scope from starting all services (sysVinit) to managing the system and all services

▶ Key features:
  ♦ Relies on cgroups for
    ★ Services supervision
    ★ Control of services execution environment
  ♦ Declarative syntax for unit files \(\sim\) more efficient/robust
  ♦ Socket activation for parallel services startup
  ♦ Nicer user interface (systemctl & friends)
▶ Additional features: logging, timer units (cron-like), user sessions handling, containers management
Behind the scenes: cgroups

- Abbreviated from *control groups*
- Linux kernel feature
- Limit, account for and isolate *processes and their resource usage* (CPU, memory, disk I/O, network, etc.)
- Related to *namespace isolation*:
  - Isolate processes from the rest of the system
  - *Chroots on steroids*
  - PID, network, UTS, mount, user, etc.
- LXC, Docker ≈ cgroups + namespaces (+ management tools)
cgroups and systemd

- Each service runs in its own cgroup
- Enables:
  - Tracking and killing all processes created by each service
  - Per-service accounting and resources allocation/limitation
- Previously, with sysVinit:
  - No tracking of which service started which processes
    - PID files, or hacks in init scripts: `pidof`, `killall`, `pgrep`
    - Hard to completely terminate a service (left-over CGI scripts when killing Apache)
  - No resources limitation (or using `setrlimit` (= `ulimit`), which is per-process, not per-service)
- Also isolate user sessions ↳ kill all user processes (not by default)
- More information: Control Groups vs. Control Groups and Which Service Owns Which Processes?
systemd-cgls: visualizing the cgroups hierarchy

```
- /sbin/init
  -system.slice
    -apache2.service
      -1242 /usr/sbin/apache2 -k start
      -9880 /usr/sbin/apache2 -k start
      -9881 /usr/sbin/apache2 -k start
    -system-getty.slice
      -getty@tty1.service
        -1190 /sbin/agetty --noclear tty1 linux
      -getty@tty2.service
        -24696 /sbin/agetty --noclear tty2 linux
    -system-postgresql.slice
      -postgresql@9.4-main.service
        -1356 postgres: checkpointer process
        -1357 postgres: writer process
        -1358 postgres: wal writer process
        -1359 postgres: autovacuum launcher process
        -1360 postgres: stats collector process
    -gdm.service
      -1209 /usr/sbin/gdm3
      -1238 /usr/bin/Xorg :0 -novtswitch -background none -noreset -verbose 3 -auth /run/user/1000/gdm/auth /run/user/1000/gdm
  -user.slice
    -user-1000.slice
      -session-1.scope
        -1908 gdm-session-worker [pam/gdm-password]
        -1917 /usr/bin/gnome-keyring-daemon --daemonize --login
        -1920 gnome-session
        -1966 /usr/bin/dbus-launch --exit-with-session gnome-session
```
systemd-cgtop: per-service resources usage

<table>
<thead>
<tr>
<th>Path</th>
<th>Tasks</th>
<th>%CPU</th>
<th>Memory</th>
<th>Input/s</th>
<th>Output/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>92</td>
<td>68.8</td>
<td>-</td>
<td>0B</td>
<td>243.9K</td>
</tr>
<tr>
<td>/system.slice</td>
<td>-</td>
<td>65.8</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/ModemManager.service</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/NetworkManager.service</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/accounts-daemon.service</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/apache2.service</td>
<td>3</td>
<td>0.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/atd.service</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/avahi-daemon.service</td>
<td>2</td>
<td>0.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/colorsd.service</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/system-postgresql.slice</td>
<td>8</td>
<td>66.0</td>
<td>-</td>
<td>340.4K</td>
<td>112.4M</td>
</tr>
<tr>
<td>/system.slice/system-postgresql.slice/postgresql@9.4-main.service</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/systemd-journald.service</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/system.slice/systemd-logind.service</td>
<td>1</td>
<td>0.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/user.slice</td>
<td>13</td>
<td>1.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/user.slice/user-1001.slice</td>
<td>-</td>
<td>1.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/user.slice/user-1001.slice/session-2.scope</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/user.slice/user-1001.slice/session-4.scope</td>
<td>6</td>
<td>1.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>/user.slice/user-1001.slice/session-6.scope</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Requires enabling CPUAccounting, BlockIOAccounting, MemoryAccounting
Managing services with systemctl

- What is being manipulated is called a **unit**: services (.service), mount points (.mount), devices (.device), sockets (.socket), etc.

- Basic commands:

<table>
<thead>
<tr>
<th>sysVinit</th>
<th>systemd</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>service foo start</td>
<td>systemctl start foo</td>
<td>restart if already running auto-start at next boot disable auto-start</td>
</tr>
<tr>
<td>service foo stop</td>
<td>systemctl stop foo</td>
<td></td>
</tr>
<tr>
<td>service foo restart</td>
<td>systemctl restart foo</td>
<td></td>
</tr>
<tr>
<td>service foo reload</td>
<td>systemctl reload foo</td>
<td></td>
</tr>
<tr>
<td>service foo condrestart</td>
<td>systemctl condrestart foo</td>
<td></td>
</tr>
<tr>
<td>update-rc.d foo enable</td>
<td>systemctl enable foo</td>
<td></td>
</tr>
<tr>
<td>update-rc.d foo disable</td>
<td>systemctl disable foo</td>
<td></td>
</tr>
<tr>
<td></td>
<td>systemctl is-enabled foo</td>
<td></td>
</tr>
</tbody>
</table>

- There’s auto-completion (**apache2** and **apache2.service** work)

- Several services can be specified:

  ```
  systemctl restart apache2 postgresql
  ```
systemd and runlevels

- With sysVinit, runlevels control which services are started automatically
  - 0 = halt; 1 = single-user / minimal mode; 6 = reboot
  - Debian: no difference by default between levels 2, 3, 4, 5
  - RHEL: 3 = multi-user text, 5 = multi-user graphical

- systemd replaces runlevels with targets:
  - Configured using symlinks farms in /etc/systemd/system/target.wants/
  - systemctl enable/disable manipulate those symlinks
  - systemctl mask disables the service and prevents it from being started manually
  - The default target can be configured with systemctl get-default/set-default
  - More information: The Three Levels of "Off"
Default targets (\texttt{bootup(7)})

\begin{verbatim}
local-fs-pre.target
  \v
  (various mounts and
  fsck services...) (various swap
  devices...) (various cryptsetup
  devices...) (various low-level
  services: udevd,
  API VFS mounts:
  tmpfiles, random
  mqueue, configfs,
  seed, sysctl, ...)
  local-fs.target swap.target cryptsetup.target
  \v
  \v
  sysinit.target
  \v
  \v
  timers.target
  \v
  \v
  basic.target
  \v
  \v
  display-manager.service
  (various system
  services)
  \v
  \v
  emergency.service
  \v
  \v
  graphical.target
\end{verbatim}
Analyzing startup performance

- Fast boot matters in some use-cases:
  - Virtualization, Cloud:
    - Almost no BIOS / hardware checks \(\leadsto\) only software startup
    - Requirement for infrastructure elasticity
  - Embedded world

- **systemd-analyze time:** summary
  
  Startup finished in 4.883 s (kernel) + 5.229 s (userspace) = 10.112 s

- **systemd-analyze blame:** worst offenders
  
  2.417s systemd-udev-settle.service
  2.386s postgresql@9.4-main.service
  1.507s apache2.service
  240ms NetworkManager.service
  236ms ModemManager.service
  194ms accounts-daemon.service
systemd-analyze plot

- Similar to bootchartd, but does not require rebooting with a custom `init=` kernel command-line
systemd-analyze critical-chain

- Shows services in the critical path

graphical.target @5.226s
  └ multi-user.target @5.226s
      └ exim4.service @5.144s +81ms
          └ postgresql.service @5.142s +1ms
              └ postgresql@9.4-main.service @2.755s +2.386s
                  └ basic.target @2.743s
                      └ timers.target @2.743s
                          └ systemd-tmpfiles-clean.timer @2.743s
                              └ sysinit.target @2.742s
                                  └ networking.service @2.589s +153ms
                                      └ local-fs.target @2.587s
                                          └ run-user-117.mount @3.877s
                                              └ local-fs-pre.target @223ms
                                                  └ systemd-remount-fs.service @218ms +4ms
                                                      └ keyboard-setup.service @157ms +61ms
                                                          └ systemd-udevd.service @154ms +2ms
                                                              └ systemd-tmpfiles-setup-dev.service @113ms +33ms
                                                                  └ kmod-static-nodes.service @102ms +10ms
                                                                      └ system.slice @96ms
                                                                          └ --.slice @94ms
Exploring the system status

- Listing units with `systemctl list-units` (or just `systemctl`):
  - active units: `systemctl`
  - List only services: `systemctl -t service`
  - List units in failed state: `systemctl --state failed`

- Whole system overview: `systemctl status`
systemctl status service

- avahi-daemon.service - Avahi mDNS/DNS-SD Stack
  Loaded: loaded (/lib/systemd/system/avahi-daemon.service; enabled)
  Active: active (running) since Wed 2015-04-01 21:49:28 CEST; 27s ago
  Main PID: 2858 (avahi-daemon)
  Status: "avahi-daemon 0.6.31 starting up."
  CGroup: /system.slice/avahi-daemon.service
    └─2858 avahi-daemon: running [grep.local]
       └─2859 avahi-daemon: chroot helper

Includes:

- Service name and description, state, PID
- Free-form status line from systemd-notify(1) or sd_notify(3)
- Processes tree inside the cgroup
- Last lines from journald (syslog messages and stdout/stderr)
Configuring services by writing unit files

- With **sysVinit**: shell scripts in `/etc/init.d/`
  - Long and difficult to write
  - Redundant code between services
  - Slow (numerous `fork()` calls)

- With **systemd**: declarative syntax (.desktop-like)
  - Move intelligence from scripts to systemd
  - Covers most of the needs, but shell scripts can still be used
  - Can use includes and overrides (`systemd-delta`)
  - View config file for a unit: `systemctl cat atd.service`
  - Or just find the file under `/lib/systemd/system/` (distribution’s defaults) or `/etc/systemd/system` (local overrides)
Simple example: atd

[Unit]
Description=Deferred execution scheduler
# Pointer to documentation shown in systemctl status
Documentation=man:atd(8)

[Service]
# Command to start the service
ExecStart=/usr/sbin/atd -f
IgnoreSIGPIPE=false  # Default is true

[Install]
# Where "systemctl enable" creates the symlink
WantedBy=multi-user.target
Common options

- Documented in `systemd.unit(5)` ([Unit]), `systemd.service(5)` ([Service]), `systemd.exec(5)` (execution environment)

- Show all options for a given service:
  `systemctl show atd`

- Sourcing a configuration file:
  `EnvironmentFile=-/etc/default/ssh`
  `ExecStart=/usr/sbin/sshd -D $SSHD_OPTS`

- Using the `$MAINPID` magic variable:
  `ExecReload=/bin/kill -HUP $MAINPID`

- Auto-restart a service when crashed: (≈ runit / monit)
  `Restart=on-failure`

- Conditional start:
  `ConditionPathExists=!/etc/ssh/sshd_not_to_be_run`
  Conditions on architecture, virtualization, kernel cmdline, AC power, etc.
Options for isolation and security

▶ Use a network namespace to isolate the service from the network:
  PrivateNetwork=yes

▶ Use a filesystem namespaces:
  ♦ To provide a service-specific /tmp directory:
    PrivateTmp=yes
  ♦ To make some directories inaccessible or read-only:
    InaccessibleDirectories=/home
    ReadOnlyDirectories=/var

▶ Specify the list of capabilities(7) for a service:
  CapabilityBoundingSet=CAP_CHOWN CAP_KILL
  Or just remove one:
  CapabilityBoundingSet=~CAP_SYS_PTRACE

▶ Disallow forking:
  LimitNPROC=1
Options for isolation and security (2)

- Run as user/group: User=, Group=

- Run service inside a chroot:
  
  RootDirectory=/srv/chroot/foobar
  ExecStartPre=/usr/local/bin/setup-foobar-chroot.sh
  ExecStart=/usr/bin/foobard
  RootDirectoryStartOnly=yes

- Control CPU shares, memory limits, block I/O, swapiness:
  
  CPUShares=1500
  MemoryLimit=1G
  BlockIOWeight=500
  BlockIOReadBandwidth=/var/log 5M
  ControlGroupAttribute=memory.swapiness 70

- More information: Converting sysV init scripts to systemd service files, Securing your services, Changing roots, Managing resources
Timer units

- Similar to cron, but with all the power of systemd (dependencies, execution environment configuration, etc)

- **Realtime (wallclock) timers**: calendar event expressions
  
  - Expressed using a complex format (see systemd.time(7)), matching timestamps like: Fri 2012-11-23 11:12:13
  
  - Examples of valid values: hourly (= *-*-* *:00:00), daily (= *-*-* 00:00:00), *:2/3 (= *-*-* *:02/3:00)

- **Monotonic timers**, relative to different starting points:
  
  - 5 hours and 30 mins after system boot: OnBootSec=5h 30m
  
  - 50s after systemd startup: OnStartupSec=50s
  
  - 1 hour after the unit was last activated: OnUnitActiveSec=1h (can be combined with OnBootSec or OnStartupSec to ensure that a unit runs on a regular basis)
Timer units example

► `myscript.service:`

```
[Unit]
Description=MyScript

[Service]
Type=simple
ExecStart=/usr/local/bin/myscript
```

► `myscript.timer:`

```
[Unit]
Description=Runs myscript every hour

[Timer]
# Time to wait after booting before we run first time
OnBootSec=10min
# Time between running each consecutive time
OnUnitActiveSec=1h
Unit=myscript.service

[Install]
WantedBy=multi-user.target
```
Timer units example (2)

- Start timer:
  ```bash
  systemctl start myscript.timer
  ```

- Enable timer to start at boot:
  ```bash
  systemctl enable myscript.timer
  ```

- List all timers:
  ```bash
  systemctl list-timers
  ```
Socket activation

- systemd listens for connection on behalf of service until the service is ready, then passes pending connections

- Benefits:
  - No need to express ordering of services during boot:
    - They can all be started in parallel → faster boot
    - And they will wait for each other when needed (when they will talk to each other), thanks to socket activation
  - Services that are seldomly used do not need to keep running, and can be started on-demand

- Not limited to network services: also D-Bus activation and path activation

- More information: Converting inetd Service, Socket Activation for developers (+ follow-up)
Socket activation example: dovecot

dovecot.socket:

[Unit]
Description=Dovecot IMAP/POP3 email server activation socket

[Sockets]
# dovecot expects separate IPv4 and IPv6 sockets
BindIPv6Only=ipv6-only
ListenStream=0.0.0.0:143
ListenStream=[::]:143
ListenStream=0.0.0.0:993
ListenStream=[::]:993
KeepAlive=true

[Install]
WantedBy=sockets.target

dovecot.service:

[Unit]
Description=Dovecot IMAP/POP3 email server
After=local-fs.target network.target

[Service]
Type=simple
ExecStart=/usr/sbin/dovecot -F
NonBlocking=yes

[Install]
WantedBy=multi-user.target
Socket activation example: sshd

▶ sshd.socket:

[Unit]
Description=SSH Socket for Per-Connection Servers

[Socket]
ListenStream=22
Accept=yes

[Install]
WantedBy=sockets.target

▶ sshd@.service:

[Unit]
Description=SSH Per-Connection Server

[Service]
ExecStart=-/usr/sbin/sshd -i
StandardInput=socket
Socket activation example: sshd (2)

- sshd@.service means that this is an instantiated service
- There’s one instance of sshd@.service per connection:

```
# systemctl --full | grep ssh
sshd@172.31.0.52:22-172.31.0.4:47779.service loaded active running
sshd@172.31.0.52:22-172.31.0.54:52985.service loaded active running
sshd.socket loaded active listening
```

- Instanciated services are also used by getty
- See **Serial console** and **Instanciated services**
Logging with journald

- Component of systemd
- Captures syslog messages, kernel log messages, initrd and early boot messages, messages written to stdout/stderr by all services
  - Forwards everything to syslog
- Structured format (key/value fields), can contain arbitrary data
  - But viewable as syslog-like format with journalctl
- Indexed, binary logs; rotation handled transparently
- Can replace syslog (but can also work in parallel)
- Not persistent across reboots by default – to make it persistent, create the /var/log/journal directory, preferably with:
  
  ```
  install -d -g systemd-journal /var/log/journal
  setfacl -R -nm g:adm:rx,d:g:adm:rx /var/log/journal
  ```
- Can log to a remote host (with systemd-journal-gatewayd)
Example journal entry

_SERVICE=systemd-logind.service
MESSAGE=User harald logged in
MESSAGE_ID=422bc3d271414bc8bc9570f222f24a9
_EXE=/lib/systemd/systemd-logind
_COMM=systemd-logind
_CMDLINE=/lib/systemd/systemd-logind
_PID=4711
_UID=0
_GID=0
_SYSTEMD_CGROUP=/system/systemd-logind.service
_CGROUPS=cpu:/system/systemd-logind.service
_PRIORITY=6
_BOOT_ID=422bc3d271414bc8bc95870f222f24a9
_MACHINE_ID=c686f3b205dd48e0b43ceb6eda479721
_HOSTNAME=waldi
LOGIN_USER=500
Using journalctl

- View the full log: `journalctl`
- Since last boot: `journalctl -b`
- For a given time interval: `journalctl --since=yesterday`
  or `journalctl --until="2013-03-15 13:10:30"`
- View it in the verbose (native) format: `journalctl -o verbose`
- Filter by systemd unit: `journalctl -u ssh`
- Filter by field from the verbose format: `journalctl _SYSTEMD_UNIT=ssh.service`
  `journalctl _PID=810`
- Line view (≈ `tail -f`): `journalctl -f`
- Last entries (≈ `tail`): `journalctl -n`
- Works with bash-completion
- See also: Journald design document, Using the Journal
Containers integration

- General philosophy: integrate management of services from machines (VMs and containers) with those of the host
  - `systemd-machined`: tracks machines, provides an API to list, create, register, kill, terminate machines, transfer images (tar, raw, Docker)
  - `machinectl`: command-line utility to manipulate machines
  - Other tools also have containers support:
    - `systemctl -M mycontainer restart foo`
    - `systemctl list-machines`: provides state of containers
    - `journalctl -M mycontainer`
    - `journalctl -m`: combined log of all containers

- `systemd` has its own mini container manager: `systemd-nspawn`

- Other virtualization solutions can also talk to `machined`

- More information: [Container integration](#)
Networking with systemd-networkd

- Replacement for /etc/network/interfaces, on servers and VMs
  - Not really for Network Manager on desktops and laptops
- Supports setting IP configuration, configuring bridges, vlans, bonding, tunnels, etc
- Configuration files with a [Match] section to match on MAC address, driver, udev path, type, hostname, etc
  - **foo.link**: link-level configuration – MAC address, interface name, MTU, rate, Duplex mode, Wake on Lan
  - **foo.netdev**: creation of virtual network devices (bridges, bonds, vlans, IPIP or GRE tunnels, VXLAN, tun, tap, veth)
  - **foo.network**: network devices configuration: IP (static or DHCP, gateway, additional routes, DNS), addition to bridge
- More information: systemd-networkd(8), systemd.link(5), systemd.network(5), systemd.netdev(5)
Example 1: DHCP, additional route

▶ For higher performance, systemd includes a DHCP client

# /etc/systemd/network/ethernet.network
[Match]
Name=eth0

[Network]
DHCP=yes

[Route]
Gateway=192.168.1.253
Destination=10.0.0.0/8
Example 2: static addressing and VLAN

# /etc/systemd/network/vlan1.netdev
# [Match] section is optional in netdev files
[NetDev]
Name=vlan1
Kind=vlan

[VLAN]
Id=1

# /etc/systemd/network/ethernet.network
[Match]
Name=eth0

[Network]
DHCP=yes
VLAN=vlan1 # will create vlan1 on this device

# /etc/systemd/network/vlan1.network
[Match]
Name=vlan1

[Network]
Address=192.168.1.1/24
# etc/systemd/network/bridge0.netdev
[NetDev]
Name=bridge0
Kind=bridge

# etc/systemd/network/bridge0.network
[Match]
Name=bridge0

[Network]
Address=192.168.1.1/24
DHCPServer=yes # systemd has its own, very basic, DHCP server

# etc/systemd/network/tap.netdev
[NetDev]
Name=tap0
Kind=tap

# etc/systemd/network/tap.network
[Match]
Name=bridge0

[NetDev]
Bridge=bridge0
Migration from sysvinit

- **systemd hooks into LSB init scripts:** service foo start|stop|...
  and /etc/init.d/foo redirect to systemctl

- **systemd-sysv-generator creates wrapper units for LSB scripts:**
  
  - Using LSB dependencies
  
  - Services are described as *LSB: foo*
    
    - List all generated services:
      
      systemctl list-units | grep LSB:
Generated wrapper service file for apache2

$ systemctl cat apache2.service
# /run/systemd/generator.late/apache2.service
# Automatically generated by systemd-sysv-generator

[Unit]
Description=LSB: Apache2 web server
Before=runlevel2.target runlevel3.target runlevel4.target runlevel5.target shutdown.target
After=local-fs.target remote-fs.target network-online.target systemd-journald-dev-log.socket nss-lookup.target
Wants=network-online.target
Conflicts=shutdown.target

[Service]
Type=forking
KillMode=process
[...]
ExecStart=/etc/init.d/apache2 start
ExecStop=/etc/init.d/apache2 stop
ExecReload=/etc/init.d/apache2 reload
More stuff

- New cross-distro configuration files: /etc/hostname, /etc/locale.conf, /etc/sysctl.d/*.conf, /etc/tmpfiles.d/*.conf

- Tools to manage hostname, locale, time and date: hostnamectl, localectl, timedatectl

- Support for watchdogs

- Handling of user sessions
  - Each within its own cgroup
  - Multi-seat support
  - loginctl to manage sessions, users, seats
Conclusions

- systemd revisits the way we manage Linux systems
  - *If we redesigned services management from scratch, would it look like systemd?*

- For service developers: easier to support systemd than sysVinit
  - No need to fork, to drop privileges, to write a pid file
  - Just output logs to stdout (redirected to syslog, with priorities)

- Some parts still have rough edges, or are still moving targets, but are promising: journal, containers, networking