systemd for users

Michael „sECuRE“ Stapelberg
2014-06-20, GPN14

Mein Hintergrund

- anfangs skeptisch (für ca. 1 Jahr)
- eine handvoll Patches für systemd
- einer der systemd-maintainer in Debian

  (Bugs, unit reviews, dh-systemd, …)

Status quo

- systemd zuerst 2010-04-30 vorgestellt
- als Standard-Initsystem bereits in:

  Fedora, RHEL, Arch Linux, Mageia, OpenSUSE, …
- bald ebenfalls in:

  Debian ≥ 8 (jessie)
- 2014-02-14: Upstart wird offiziell aufgegeben

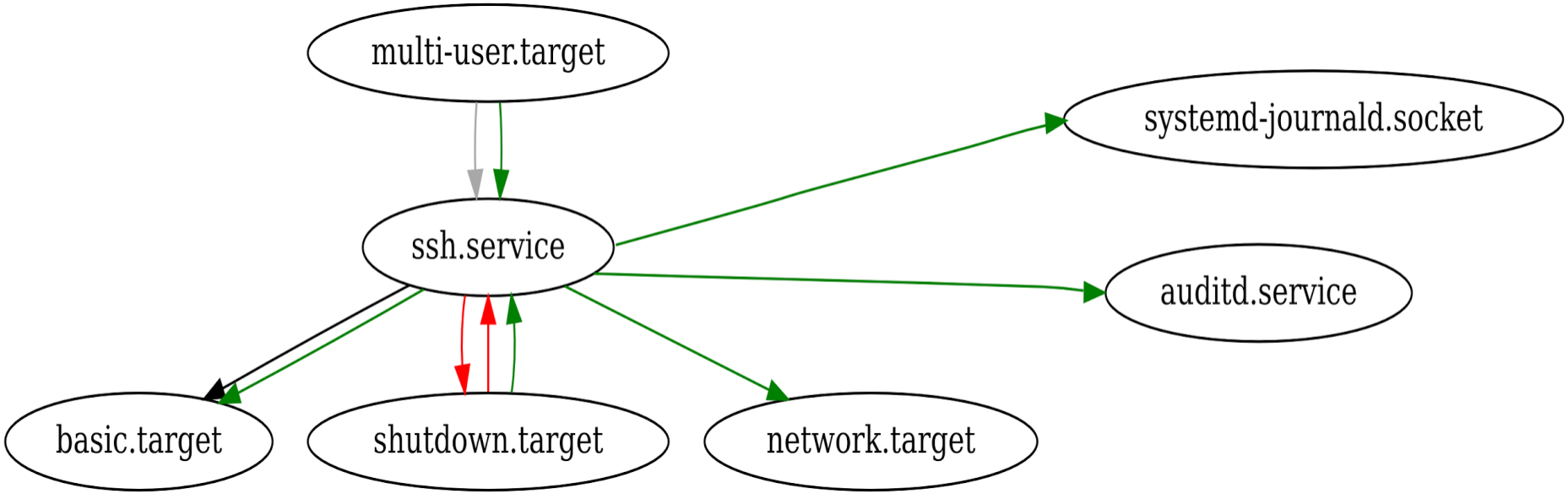
Agenda

- start/stop/status genauer angesehen
- Was sind Units und Targets? Wie hängen sie zusammen?
- Debugging, Journal
- logind, Inhibitoren
- Wir schreiben ein Service-File
- sd_notify, …: APIs für Service-Autoren

start/stop/status

- siehe `man systemctl(1)`
- <*demo: status in detail*>

```
$ systemd-analyze dot ssh.service | dot -Tsvg -o ssh.svg
```



Requires: unit wird auch aktiviert

Wants: wie Requires, aber unit darf failen

After: unit A nach B starten

Conflicts: unit wird gestoppt

Units

- service files: Pendant zu sysvinit scripts

  (= wie wird der Dienst gestartet?)
- socket files: socket activation

  (= welche Sockets soll systemd bereitstellen?)
- timer files: kleines Subset von cron

  (= periodische Ausführung)

Units (2)

- mount/swap files: generiert aus /etc/fstab

  (= welche Dateisysteme sollen gemounted werden?)
- path files: path-based activation (inotify)
- snapshot units: snapshot, depended auf momentan laufenden Units

Targets

- Äquivalent zu runlevels, z.B. multi-user.target (2), graphical.target (≥ 2), shutdown.target, …
- siehe /lib/systemd/system/*.target
- effektiv gruppieren Targets mehrere Units mit korrekten Abhängigkeiten

start/stop/status (2)

- jede Aktion erstellt einen Job
- `systemctl list-jobs`

```
JOB UNIT                        TYPE  STATE
1   graphical.target            start waiting
2   multi-user.target           start waiting
51  fail2ban.service            start waiting
56  netctl@ethernet.service start running
58  network.target              start waiting
63  sshd.service                start waiting
8 jobs listed.
```

udev

- systemd nutzt einen udev-monitor, wird benachrichtigt über neue devices
- mount unit braucht z.B. dev-mapper-myvolume.device
- (udev nutzt systemd um z.B. `RUN=` auszuführen)

Debugging (1)

- `systemctl status`
- Unit enabled? Abhängigkeiten korrekt?
- passende Doku zur Version? (zu neue Features)

Debugging (2)

- Kernel-Parameter, im Bootloader (GRUB) angegeben
- `systemd.log_level=debug`

  `systemd.log_target=kmsg`
- `systemd.unit=rescue.target`
- `systemctl enable debug-shell.service`, C-A-F9
- `init=/bin/bash` falls gar nichts mehr geht

Journal

- Alternative zu syslog
  - strukturierte, indizierte Logs
  - standardmäßig Ringbuffer
  - syslog kann man „hinter“ journald nutzen
- `journalctl -u <unit>`
- `journalctl -b` (nur Logs vom aktuellen Boot)

logind

- logind trackt sessions, bietet zugriff auf shutdown/sleep, handled power/sleep-keys, inhibitoren, (multi-seat)
- z.B. login via ssh: session **nicht** in ssh.service cgroup (via `pam_systemd`-Modul)

logind: inhibitoren

- `systemd-inhibit my-backup-script`
- `systemd-inhibit --list`
- desktop environments sollten handle-power-key etc. inhibiten wenn sie selbst power key handlen

Wir schreiben ein service-File

```
cat /etc/systemd/system/autowake.service
[Unit]
Description=automatically wakes up NAS when it’s being mounted
[Service]
ExecStart=/home/michael/gocode/src/autowake/autowake
[Install]
WantedBy=multi-user.target
```

Wir schreiben ein service-File (2)

- gültige Config voraussetzen

```
[Service]
ExecStartPre=/usr/sbin/lighttpd -t -f /etc/lighttpd.conf
ExecStart=/usr/sbin/lighttpd -D -f /etc/lighttpd.conf
```

Wir schreiben ein service-File (3)

- reload implementieren (kein default!)

```
[Service]
ExecStart=/usr/sbin/sshd -D
ExecReload=/bin/kill -HUP $MAINPID
```

Wir schreiben ein service-File (4)

- privilege dropping, private /tmp

```
[Service]
ExecStart=/usr/lib/colord/colord
User=colord
PrivateTmp=yes
```

Wir verändern ein service-File

- original in `/lib/systemd/system/nginx.service`
- `mkdir /etc/systemd/system/nginx.service.d/`
- `echo "[Service]\nIOSchedulingClass=realtime" >/etc/systemd/system/nginx.service.d/io.conf`
- `systemctl daemon-reload`
- `systemctl restart nginx.service`

sd_notify

- für readiness, watchdog, status, …
- ```
  #include <systemd/sd-daemon.h>

  // …initialize…
  sd_notify(false, "READY=1");
  while (true) {
      sd_notify(false, "WATCHDOG=1");
      // …do work…
  }
  ```
  ```

socket activation (1)

- systemd erstellt sockets, vererbt sie an daemon
- → socket wird sehr früh erstellt
- → systemd bemerkt `connect()`, startet daemon
- → daemon bemerkt aktivität, liest, verarbeitet
- → clients müssen dependency nicht angeben

socket activation (2)

```
#include <systemd/sd-daemon.h>
int fds = sd_listen_fds(true);
for (int fd = 0; fd < fds; fd++) {
    event_loop_add(SD_LISTEN_FDS_START + fd);
}
if (fds == -1) {
    // setup sockets: bind, listen, etc.
}
// …do work…
```

journald API

```
unsigned long pid = (unsigned long)getpid();


sd_journal_print(LOG_INFO,
  "Hello World, this is PID %lu!", pid);


sd_journal_send("MESSAGE=my pid is %lu!", pid,
                "PRIORITY=%i", LOG_INFO,
                NULL);
```

“the missing API”

- wenn man in einer systemd-only-Umgebung arbeitet (z.B. im Büro) fallen viele Notwendigkeiten weg
- daemonization (double-fork), pid files
- syslog/logging
- privilege dropping/isolation

Danke!

- man 5 systemd.unit
- http://www.freedesktop.org/wiki/Software/systemd/
- #systemd on FreeNode