systemd

Ein Überblick

OpenLab Augsburg

Christine Koppelt

15. Januar 2017

Wer bin ich

Softwareentwicklerin

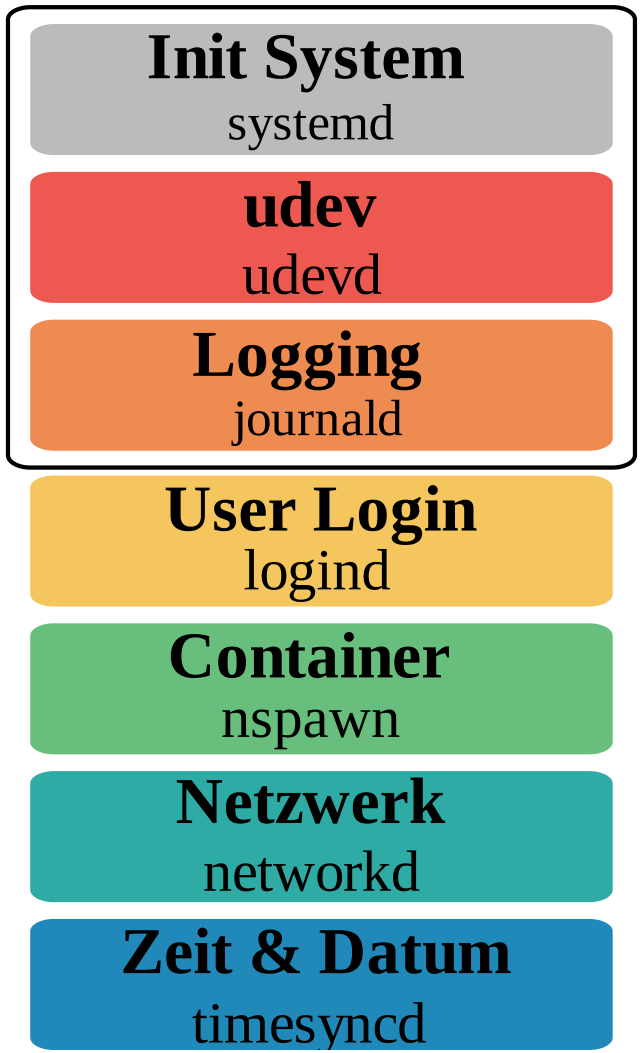
Linux seit 2006

systemd seit 2015

Ziel von systemd

"systemd is in the process of becoming a comprehensive, integrated and modular platform providing everything needed to bootstrap and maintain an operating system's userspace."

Quelle: http://0pointer.de/blog/projects/why.html - Blog von Lennart Pöttering


Init System
systemd
udev
udevd
Logging
journald
User Login
logind
Container
nspawn
Netzwerk
networkd
Zeit & Datum
timesyncd

Allgemeines

- Wurde 2010 veröffentlicht
- Im Wesentlichen von RedHat Mitarbeitern entwickelt
  - Lennart Pöttering, Kai Sievert
- LGPL lizenziert
- Mittlerweile default System- und Servicemanagment Framework in den meisten Linux Distributionen
  - Debian, Arch, Fedora, Ubuntu, Red Hat Enterprise Linux, NixOS
  - Nicht: Gentoo (optional), Android, Devuan

Kritik an systemd

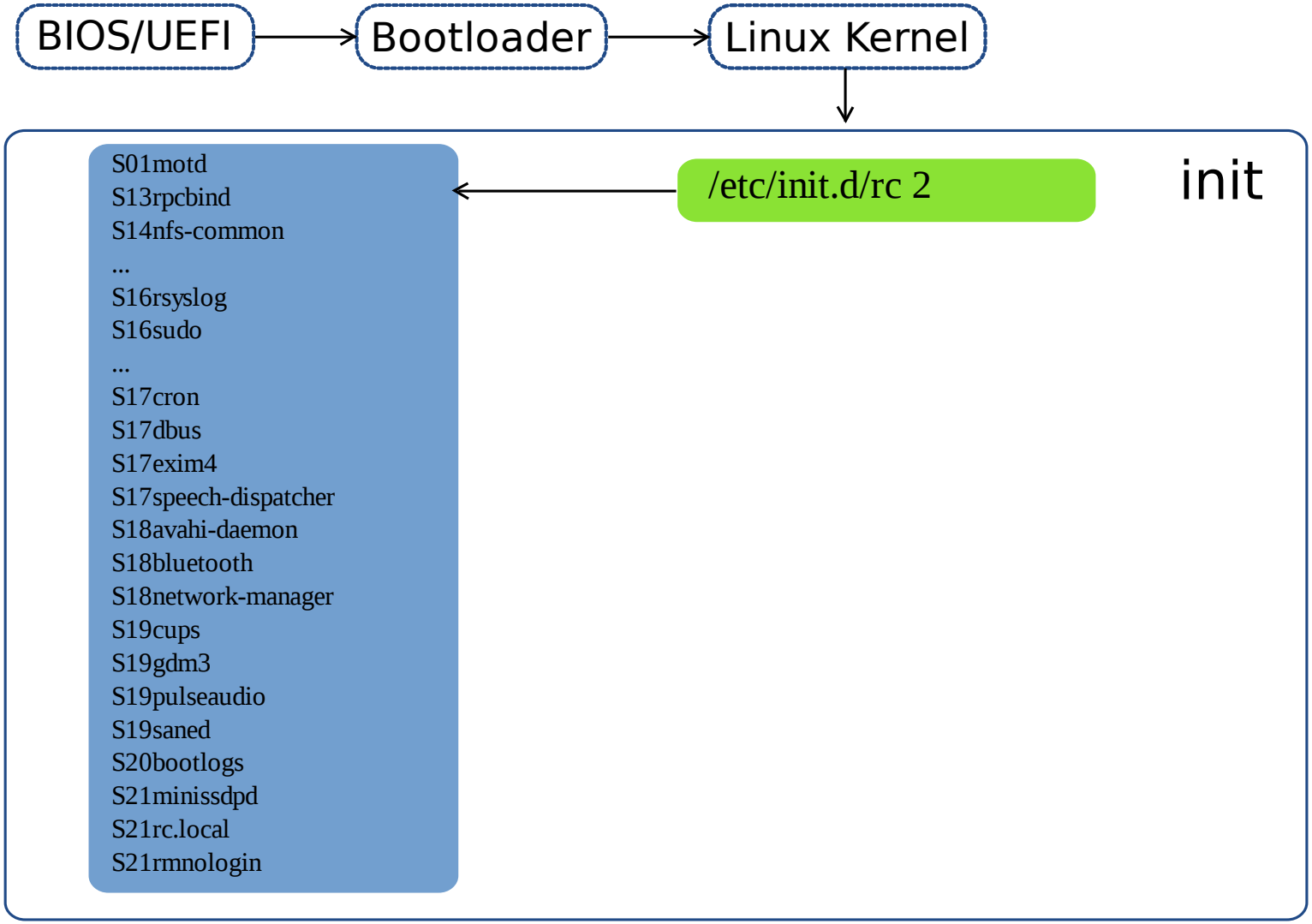
- Monolithische Architektur, keine APIs zwischen den Komponenten
  - Minimal Build: systemd + udev + journald + einige utilities
- Kopplung von systemd Komponenten an Software
  - Beispiel: Gnome -> logind
- Schlechte Kommunikation

Alternativen zu systemd

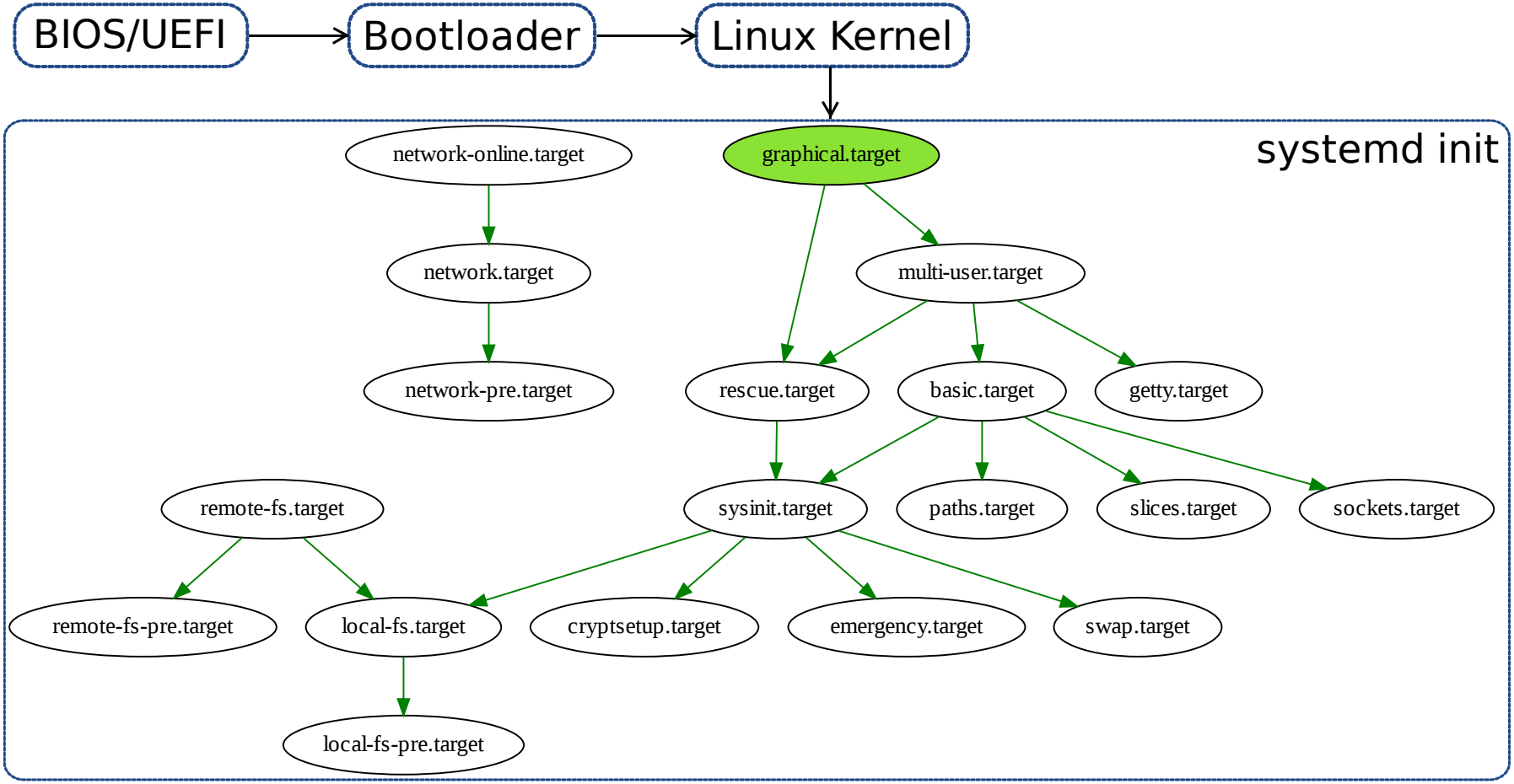
- OpenRC (Gentoo, Alpine Linux)
  - Service Manager, implementiert von Gentoo Entwicklern
  - baut auf sysvinit auf
  - ermöglicht Cross-Service Dependencies
- weitere: http://without-systemd.org/wiki/index.php/Init

Init Prozess
&
Systemstatus

Start mit SysVinit



Start mit systemd



systemd init

- Units
  - "Organisationseinheit" von systemd
  - Verschiedene Typen von Units (z.B. Services, Targets)
- Targets
  - spezieller Unit-Typ
  - Gruppierung von Units und Synchronisationspunkten
  - Können parallel gestartet werden

Typen von Units

- services
  - Dienst, z.B. sshd, nginx
- sockets
  - Socket basierte Aktivierung eines Services
- mounts/swap
  - Mount Points des Dateisystems, werden aus /etc/fstab generiert
- timers
  - Entsprechen Cron-Jobs

Analysieren des Systems

Start Kernel & Userspace

```
systemd-analyze
systemd-analyze critical-chain
```

Default Target

```
systemctl get-default
```

Status von Units

```
systemctl list-units
systemctl list-units --type=target
systemctl list-units --state=failed
systemctl status docker
```

Servicemanagement

Verwalten von Services

- SysVInit

```
service httpd start/stop
```

- systemd

```
systemctl start/stop httpd.service
```

- Unterstützt werden nur Standard Optionen (start, stop, restart)
- Services können so konfiguriert werden, dass sie im Fehlerfall automatisch neu gestartet werden

Units vs. init-Skripte

- Neue, deklarative Syntax
  - init-Skripte funktionieren großteils weiterhin
- System Defaults: `[/usr]/lib/systemd`
  - Sollten nicht geändert werden
- Eigene Konfigurationen `/etc/systemd/system`
  - wird gegenüber dem Default bevorzugt
  - Überschreiben der Defaults möglich

```
systemd-delta
```

- Typ wird durch Dateiendung des Konfigurationsfiles angezeigt
  - `myapplication.service`
- System vs. User Services
  - User Services `~/.config/systemd/user/`

Anlegen eines neuen Service

/etc/systemd/system/testserver.service

```
[Unit]
Description=Demo HTTP server
After=network.target

[Service]
ExecStart=/usr/bin/python /tmp/demohttpserver.py 1234
Restart=always
RestartSec=10

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

Manuell Starten

```
systemctl start testserver
```

Status prüfen

```
systemctl status testserver
```

Konfigurieren eines Services

Überprüfen der unit-Files auf Korrektheit

```
systemd-analyze verify
```

Automatisches Starten aktivieren

```
systemctl enable foo.service
```

Manueller Reload nach Änderungen erforderlich

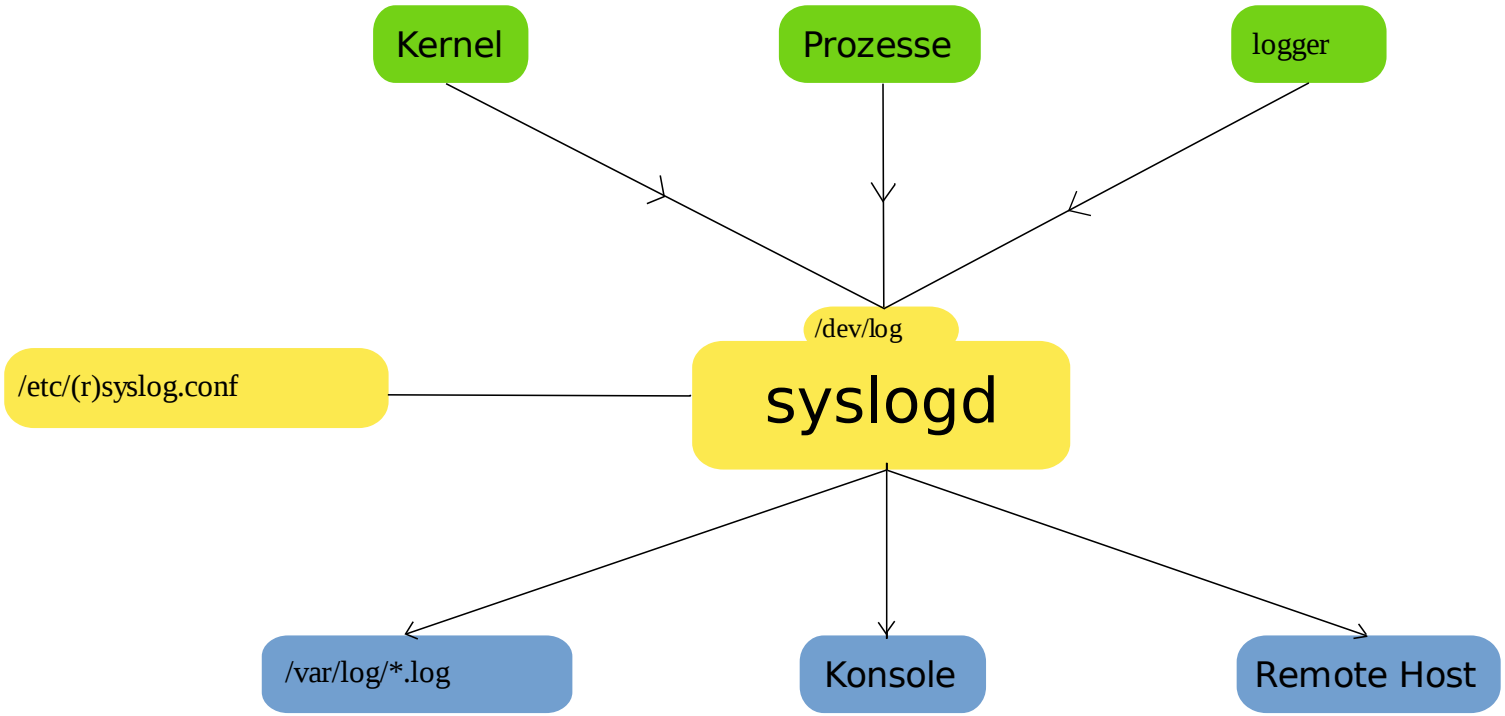
```
systemctl daemon-reload
```

Konfigurationsmöglichkeiten

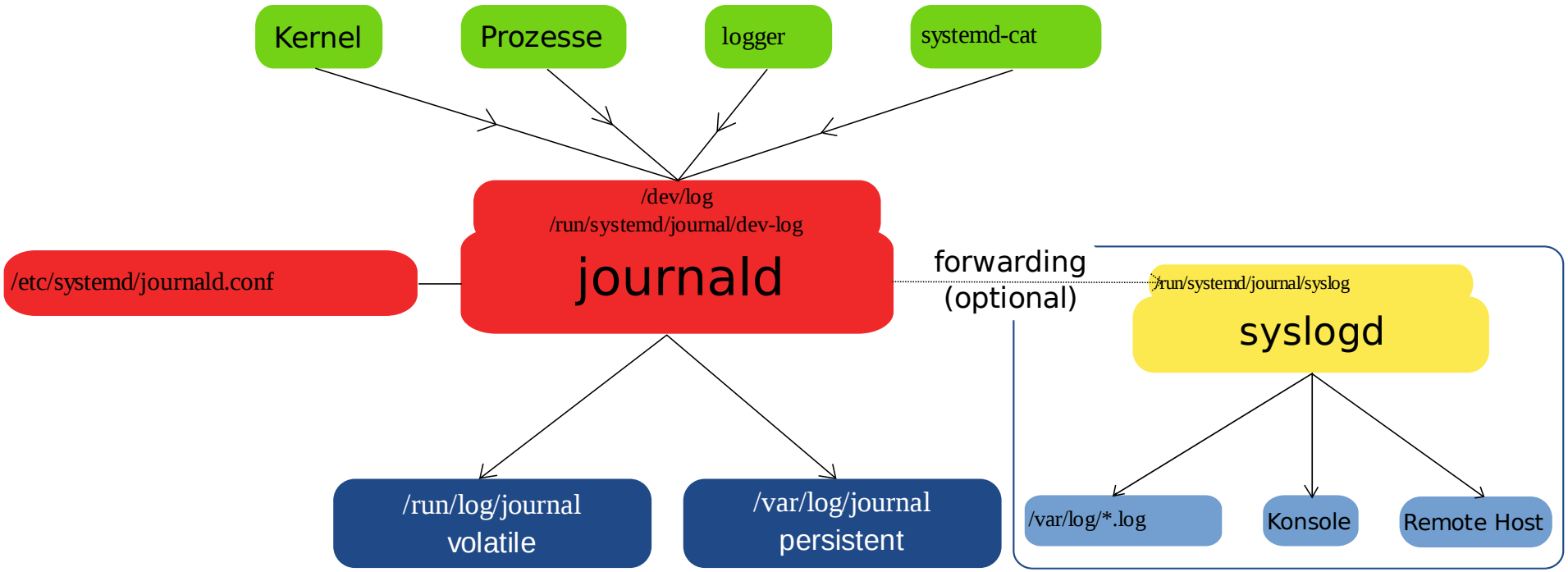
- Angabe von Startbedingungen, beispielweise
  - Virtualisierte Umgebung
  - Rechnerarchitektur
  - Verzeichnisse/Dateien vorhanden
- Maximaler Ressourcenverbrauch
  - Arbeitsspeicher
  - Device Zugriff
  - CPU Zeit
- Überwachungsfunktionen
  - Neustart eines Services im Fehlerfall
  - Timeouts für Start/Stop

Logging

syslog


Kernel
Prozesse
logger
/dev/log
/etc/(r)syslog.conf
syslogd
/var/log/*.log
Konsole
Remote Host

journal


Kernel
Prozesse
logger
systemd-cat
/dev/log
/run/systemd/journal/dev-log
journald
/etc/systemd/journald.conf
forwarding
(optional)
/run/systemd/journal/syslog
syslogd
/run/log/journal
volatile
/var/log/journal
persistent
/var/log/*.log
Konsole
Remote Host

journal: Eigenschaften

- Binärformat
  - strukturierte Daten
  - indiziert => schneller analysierbar
  - nicht mehr mit Texttools analysierbar
  - spezielle Tools für Analyse erforderlich
- default: nicht persistent
- Optional: Forwarding an syslog daemon
- Speicherung umfangreicher Metadaten
- Trusted Fields

Journal lesen: journalctl

```
journalctl
journalctl -f
```

Unit

```
journalctl -u sshd
journalctl -u httpd.service
```

die letzten n Zeilen

```
journalctl -n 100
journalctl -u httpd.service -n 100
journalctl _PID=2100 -n 100
```

Autovervollständigung

```
journalctl <TAB>
```

Journal lesen: journalctl

Zeiträume

```
journalctl -u httpd.service --since today
journalctl --since="2015-09-21 09:00:00" --until="2015-09-22 13:59:59"
journalctl --since "20 min ago"
```

Bootlog

```
journalctl -b
journalctl -b -1
```

Ausgabeformat

```
journalctl -o verbose
journalctl -o json
```

Schreiben von Journal-Einträgen

- Ausgaben von systemd Prozessen landen automatisch im jounal und syslog
- Kommandozeilentools
  - systemd-cat

```
echo 'hello' | systemd-cat
echo 'hello' | systemd-cat -p info
echo 'hello' | systemd-cat -p warning
echo 'hello' | systemd-cat -p emerg
```

  - logger

```
logger -p notice Hello
```

- Schreiben mittels syslog
- Schreiben mittels journald API Bindings

Journal: Bindings

- Offiziell: Python, Erlang
- Viele weitere: Ruby, Go, Haskell, Lua, Perl ...
- Eingeschlafen: (node, php)

Beispiel: Integration in das Python Logging Framework

```
log = logging.getLogger('custom_logger_name')
log.propagate = False
log.addHandler(journal.JournalHandler())
log.warn("Some message: %s", detail)
```

Logeintrag syslog

```
logger ein logeintrag mittels logger
```

syslog

```
Sep 24 22:43:44 dev ck: ein logeintrag mittels logger
```

- Format: `TIMESTAMP HOSTNAME CONTENT`
- Timestamp im Format `Mmm DD HH:MM:SS`, z.B. Sep 25 18:42:23
  - Jahr? Zeitzone?
- Hostname: kurz oder FQDN
  - Eindeutigkeit?
- Content: meist `Prozessname[PID]: Message`
  - Kaum vorgegebene Struktur
- Syslog Protokoll RFC 3164

Logeintrag journal

- Format nicht standardisiert, Änderungen möglich
- Trusted Fields
  - Beginnen mit _
  - Metadaten zum aufrunden Prozess und dem aufrufenen Kommando
  - Werden vom Journal Daemon hinzugefügt, können vom loggenden Programm nicht geändert werden
- User Fields
  - Können vom loggenden Client gesetzt werden
  - Message, Message Id, Log Level, Code Location, Fehlertyp

Beispiel Logeintrag

```
logger ein logeintrag mittels logger
```

journal

```
Do 2015-09-24 22:43:44.070670 CEST
_BOOT_ID=648d7d2635754b94b17e90a13ab4422e
_MACHINE_ID=d4da9ca42888e9dae063f4f852049ef1
_HOSTNAME=dev
PRIORITY=5
_TRANSPORT=syslog
SYSLOG_FACILITY=1
_CAP_EFFECTIVE=0
_GID=1000
_AUDIT_SESSION=1
_AUDIT_LOGINUID=1000
_SYSTEMD_OWNER_UID=1000
_SYSTEMD_SLICE=user-1000.slice
_UID=1000
_SYSTEMD_CGROUP=/user.slice/user-1000.slice/session-1.scope
_SYSTEMD_SESSION=1
_SYSTEMD_UNIT=session-1.scope
_COMM=logger
SYSLOG_IDENTIFIER=ck
MESSAGE=ein logeintrag mittels logger
_PID=14313
_SOURCE_REALTIME_TIMESTAMP=1443127424070670
```

Konfiguration: /etc/systemd/journald.conf

- Art der Speicherung (keine, in-Memory, auf der Platte)
- Komprimierung (ja/nein)
- Kryptographische Signierung
- Schreibintervall auf die Platte
- Maximal belegter Plattenplatz
- Logrotation (Zeit oder Speicherplatzbasiert)
- Weiterleitung von Log Meldungen an syslog/Konsole/alle Benutzer
- Max. Loglevel das berücksichtigt werden soll
  - emerg, alert, crit, err, warning, notice, info, debug
  - Ein Integer Wert zwischen 0..7

Fragen? Fragen!