Servicemanagement mit systemd

Augsburger Linux-Infotag

Christine Koppelt

22. April 2017

Wer bin ich

Softwareentwicklerin

Linux seit 2006

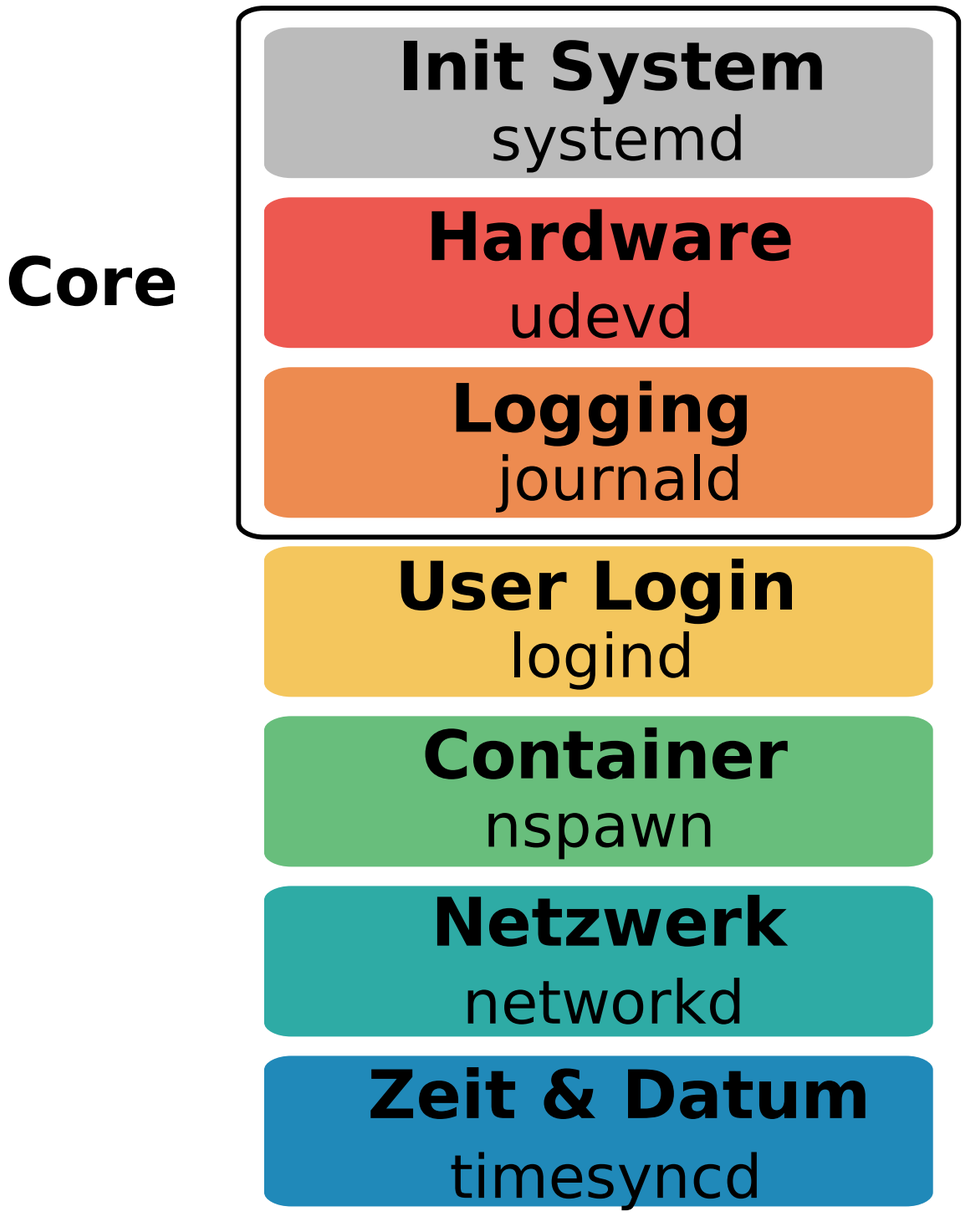
systemd seit 2015

NixOS Contributor seit 2016

Was ist 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


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

Warum systemd?

Etabliertes System

- 2010 veröffentlicht
- Default in den meisten Linux Distributionen
  - Debian, Arch, Fedora, Ubuntu, Red Hat Enterprise Linux, NixOS
  - Nicht: Gentoo (optional), Android, Devuan

Warum systemd?

Funktionalität: Systemstart

- Beschleunigter Systemstart durch paralleles Starten von Services
- Automatische Ermittlung von Reihenfolge und Abhängigkeiten

Warum systemd?

Funktionalität: Servicemanagment

- standardisiertes Interface (start, stop, restart, reload, status)
- Kontrollierte Umgebung für services mittels cgroups
- automatischer Neutstart abgestürzter Dienste
- On-Demand Aktivierung von Services
- Vereinfachte, deklarative Syntax für Konfigurationsdateien

Warum systemd?

Funktionalität: Logging

- umfangreiche Tools zur Analyse der Logs
- Metadaten zu jedem Logeintrag

Kritik an systemd

Abkehr von den Unix-Prinzipien

- *Schreibe Computerprogramme so, dass sie nur eine Aufgabe erledigen und diese gut machen.*
- *Schreibe Programme so, dass sie zusammenarbeiten.*
- *Schreibe Programme so, dass sie Textströme verarbeiten, denn das ist eine universelle Schnittstelle.*

Douglas McIlroy - Erfinder der Unix Pipes

Kritik an systemd

Kein Community Projekt

- Im Wesentlichen von RedHat Mitarbeitern entwickelt
  - Lennart Pöttering, Kai Sievert

Kritik an systemd

Monolithische Architektur

- Keine stabilen, dokumentierten APIs zwischen den Komponenten
- Minimal Build: systemd + udev + journald + einige utilities

Kritik an systemd

Nur für Linux-Systeme

- Problematisch: Kopplung von systemd Komponenten an Software
  - Beispiel: Gnome -> logind

Kritik an systemd

Kommunikationsstil

- Der Hauptgrund für seine Umbeliebtheit?

Alternativen zu systemd

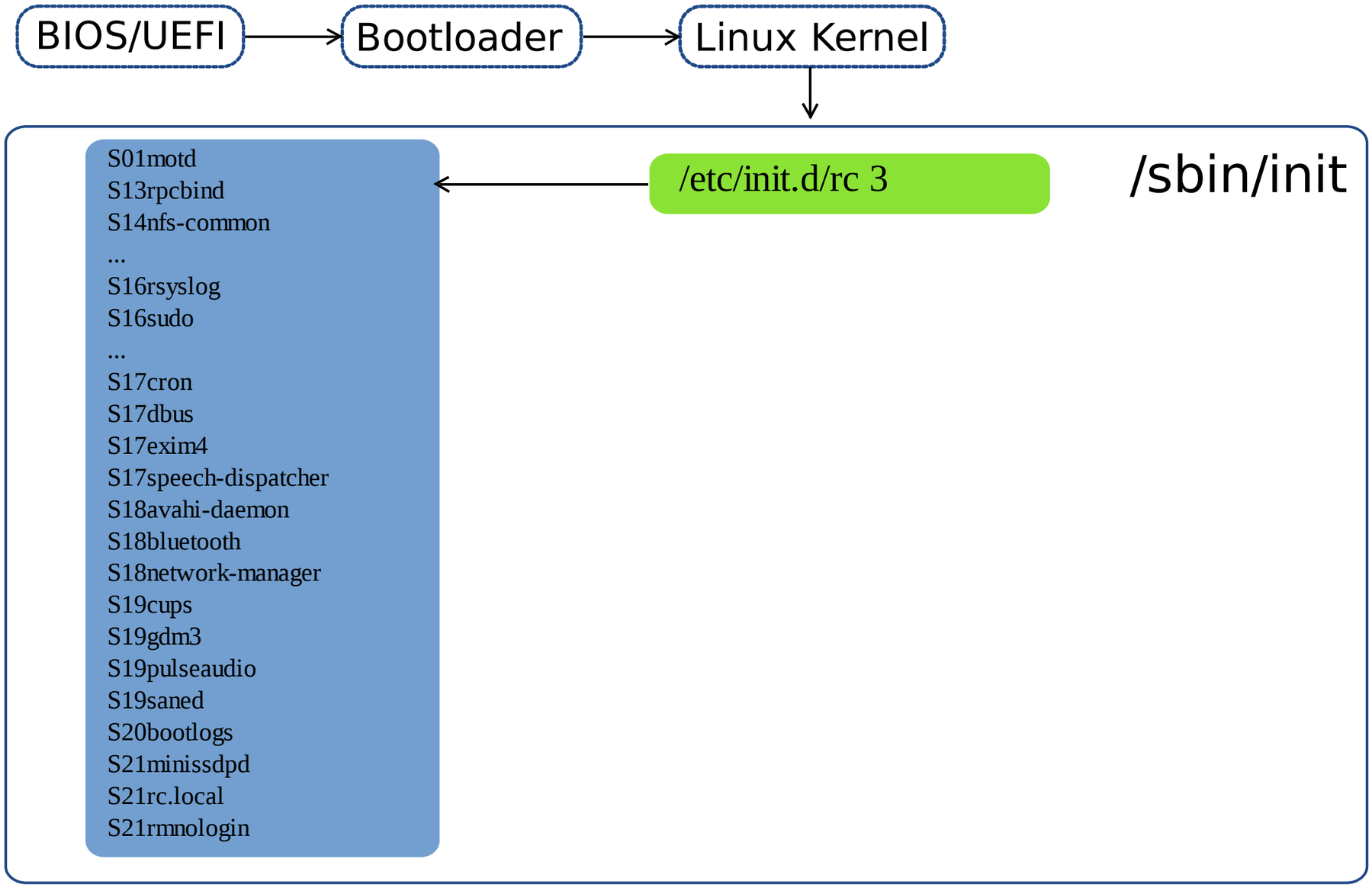
- OpenRC (Gentoo, Alpine Linux)
  - Service Manager, implementiert von Gentoo Entwicklern
  - baut auf sysvinit auf
  - automatische Verwaltung von serviceübergreifenden Abhängigkeiten
- weitere: http://without-systemd.org/wiki/index.php/Init

Init Prozess

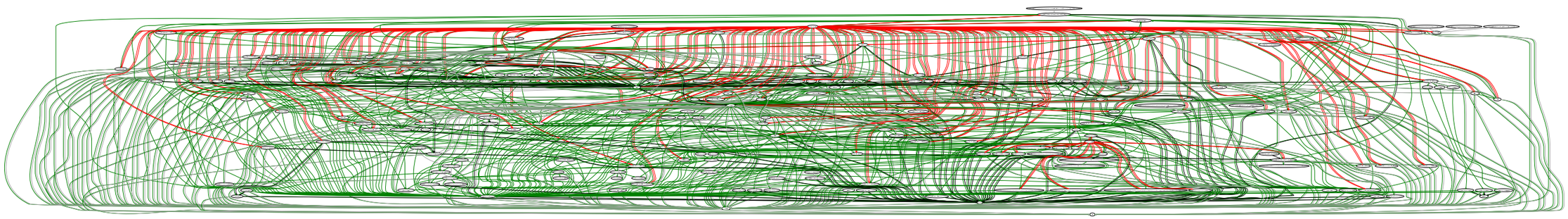
&

Systemstatus

Start mit SysVinit

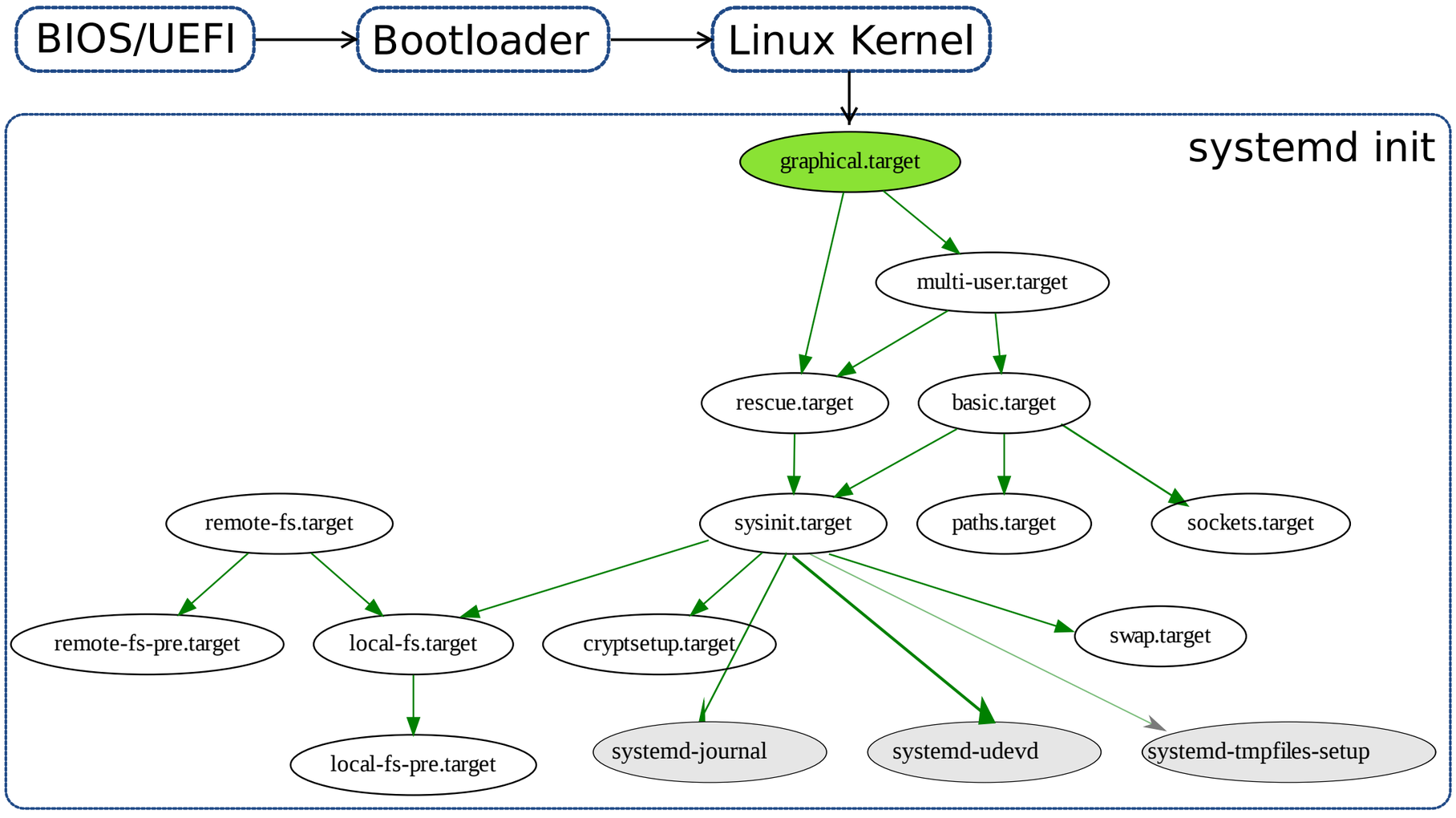


Start mit systemd



(Just kidding)

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
  - Hintergrunddienst, 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
- nspawn
  - Starten von nspawn Containern

Analysieren des Systems

Systemstarts

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

Status von Units

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

Analyse von Abhängigkeiten

```
systemctl list-dependencies
```

Übung

- Wie lange hat Dein System für den letzten Bootvorgang benötigt?
- Gib alle Timer Units aus
- Liste alle Units auf, die gestartet sein müssen, bevor `systemd-journald` gestartet wird
- Liste alle Units auf, die von einem gestarteten Netzwerk abhängen

Für die Beantwortung der Fragen kannst du die Befehle `systemctl` und `systemd-analyze` verwenden.

Verwalten & Anlegen von Units

Units

- Jede Unit liegt in einer eigenen Datei
- Typ wird durch Dateiendung angezeigt
  - `my_app.service`
- Ablageort
  - Default Speicherort: `[/usr]/lib/systemd`
  - Eigene Units `/etc/systemd/system`
    - wird gegenüber dem Default bevorzugt
    - Überschreiben der Defaults möglich
  - Userspezifische Units: `~/.config/systemd/user/`
- Eine Unit Datei hat einen allgemeinen und optional einen typspezifischen Teil
- Eigene, deklarative Syntax
- Dokumentation: `man systemd.unit` und `man systemd.<typ>`

Beispiel

```
# /lib/systemd/system/docker.socket
[Unit]
Description=Docker Socket for the API
PartOf=docker.service

[Socket]
ListenStream=/var/run/docker.sock
SocketMode=0660
SocketUser=root
SocketGroup=docker

[Install]
WantedBy=sockets.target
```

Verwalten von Units

```
systemctl start/stop a_service.service
```

- Jede Service Unit unterstützt Standard Operationen
  - **start**: Startet Unit
  - **stop**: Stoppt Unit
  - **restart**: Startet Unit neu. Falls sie noch nicht läuft, wird sie gestartet
  - **reload**: Startet Unit neu und lädt aktualisierte Konfigurationsdateien
  - **daemon-reload**: Startet Unit neu und lädt aktualisierte Unit Datei
  - **status**: Zeigt Status, zugehörige Prozesse, PID Pfad zur Doku und Unit Datei an
  - **enable**: Aktiviert automatischen Start
  - **disable**: Deaktiviert automatischen Start
  - **is-enabled**: Prüft ob Unit aktiv ist

Service Units

- Bexonderheiten
  - Starten mittels udev Regeln
  - Neustart im Fehlerfall
  - Kombination mit Socket oder Timer Units
- Default: Start nach dem basic.target
- weitgehend abwärtskompatibel zu SysV init-Skripten

Anlegen eines neuen Service

/etc/systemd/system/testserver.service

```
[Unit]
Description=Demo HTTP server

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

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

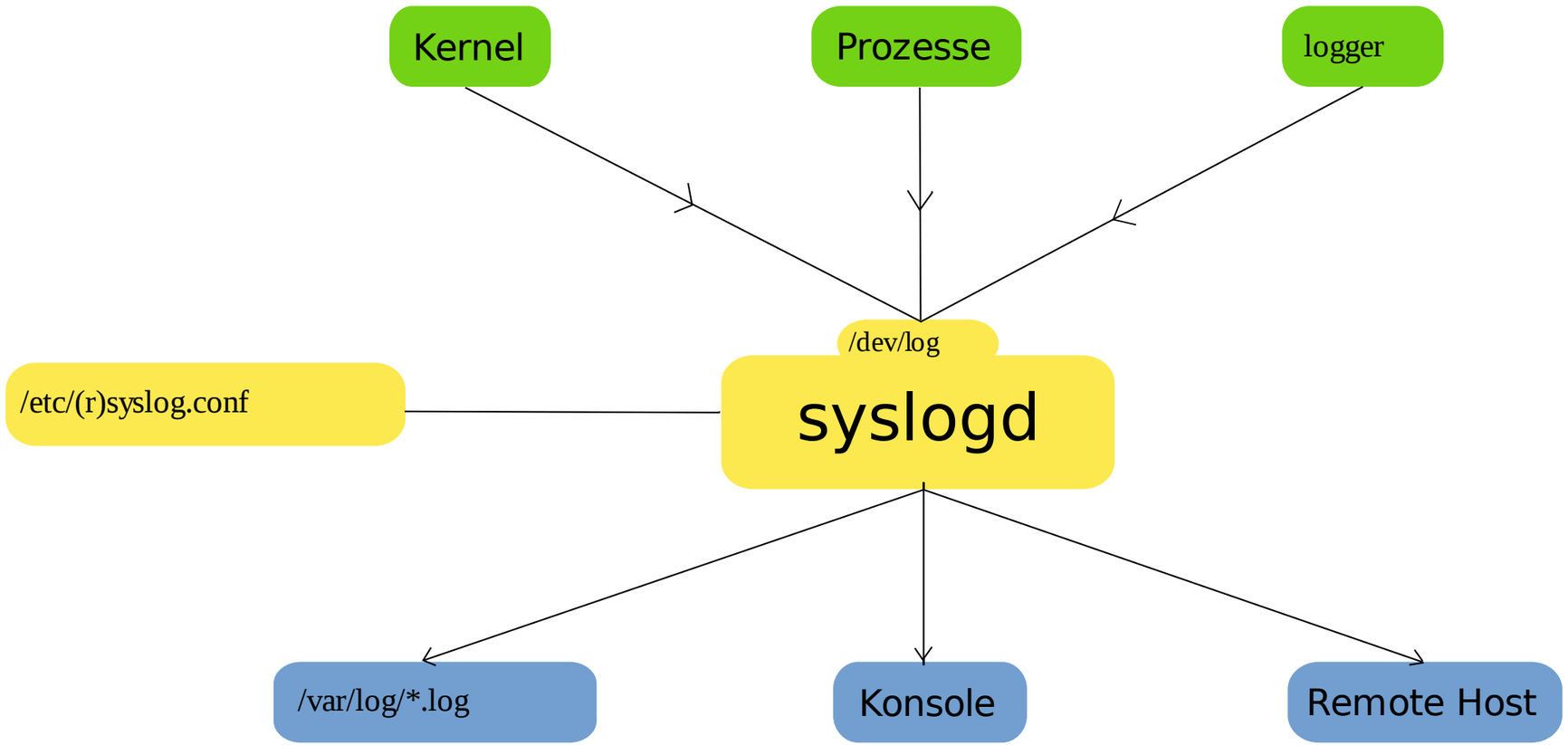
- Angabe von Startbedingungen, beispielweise
  - Virtualisierte Umgebung
  - Rechnerarchitektur
  - Verzeichnisse/Dateien vorhanden
- Maximaler Ressourcenverbrauch
  - Arbeitsspeicher
  - Device Zugriff
  - CPU Zeit
- Überwachungsfunktionen
  - Neustart eines Services im Fehlerfall
  - Timeouts für Start/Stop

Übung

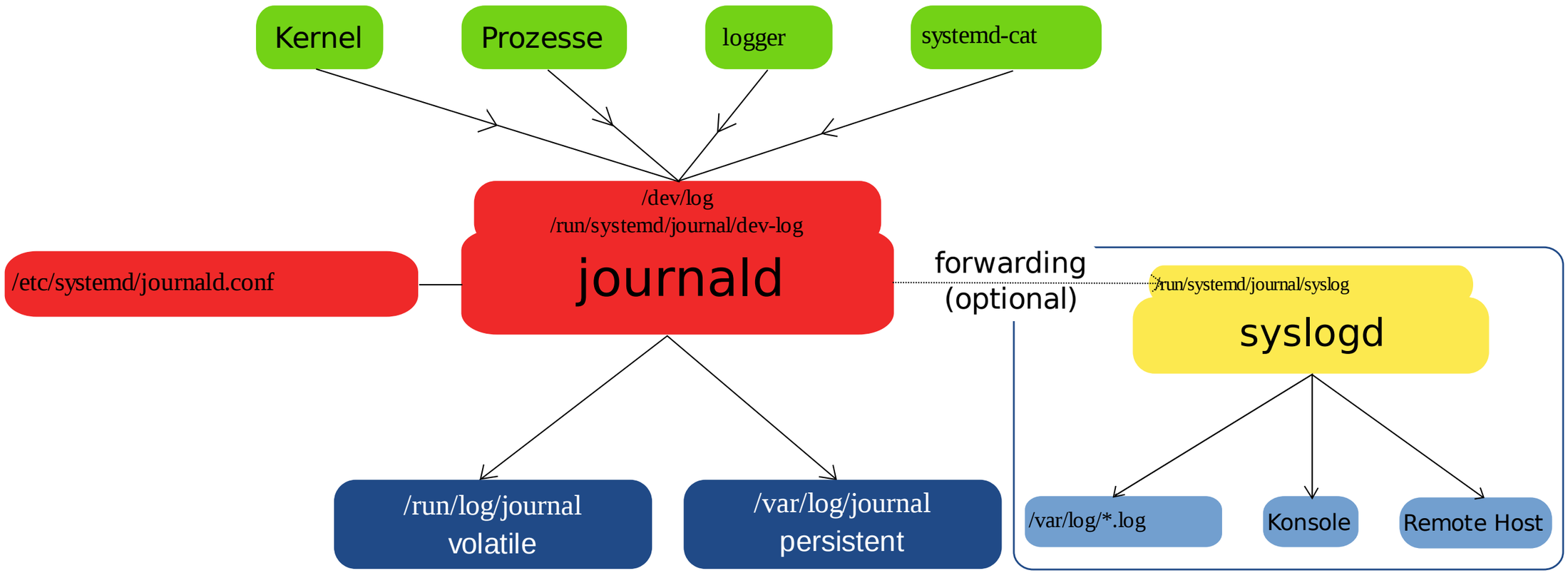
- Die Datei https://github.com/cko/systemdworkshop/blob/master/demohttpserver.py beinhaltet eine sehr einfache Serveranwendung. Diese kann mit `python demohttpserver.py 1234` gestartet werden. Mittels der Url http://localhost:1234/ping kann geprüft werden ob die Anwendung läuft, mittels http://localhost:1234/kill kann sie gestoppt werden. Erstelle zu der Anwendung ein Unit-File, starte und aktiviere die Unit und sorge dafür, dass sie nach einem Aufruf von http://localhost:1234/kill automatisch wieder neu gestartet wird.

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

Übung

- Durchsuche und verfolge die Logmeldungen aus der vorherigen "demohttpserver" Übung. Starte und stoppe den Server hierfür mehrmals.
- Logge einige Meldungen mittels `systemd-cat` und verfolge die Logmeldungen. Benutze einen eigenen `identifier` als Filter.

Fragen? Fragen!