systemd-Start analysieren

Durchgeblickt

Michael Plura

Der Umstieg auf *systemd* bescherte Linux das nützliche Werkzeug *systemd-analyze*, mit dem man nicht nur den Bootvorgang analysieren kann.

Seit Urzeiten sehen Systemverwalter ihren Unix-Systemen beim Starten zu. Meldungen von Kernel und Init-System fliegen über den Bildschirm – und wenn es mal stockt, reicht meist ein kurzer Blick in die letzte Zeile, um den Übeltäter zu identifizieren. Mit ausgefeilten Werkzeugen wie *bootchart* untersucht man den Startprozess anschließend genauer.

Mit der Einführung von *systemd* ist für die Analyse des Bootvorgangs das in Python entwickelte *systemd-analyze* vorgesehen. Es sammelt diverse Statuswerte und statistische Informationen des *systemd*-Frameworks, um daraus eine vereinfachte, an *bootchart* erinnernde Liste oder aussagekräftige Grafiken zu erstellen. Zusätzlich kann es alle Unit-Dateien (mit der Konfiguration der Systemobjekte) auf korrekte Syntax überprüfen.

Obwohl zur Systemanalyse gedacht, sollte *systemd-analyze* möglichst mit eingeschränkten Benutzerrechten laufen. Ohne Parameter oder als

```
systemd-analyze time
```

aufgerufen, gibt es die Startzeiten von Kernel- und Userspace aus. Damit ist nicht die wirklich vergangene Zeit gemeint, die das System braucht, bis alle Dienste voll funktionsfähig sind, sondern die Zeit, die *systemd* benötigt, um Sockets einzurichten und den Startvorgang aller Dienste einzuleiten – ein System ist nach dieser Zeit also nicht unbedingt betriebsbereit.

Der Parameter *blame* gibt eine Liste aller laufenden Units aus, sortiert nach der Zeit, die diese zum Initialisieren benötigt haben. Diese Angaben sind mit Vorsicht zu genießen, denn auch wenn eine Unit auf eine andere warten muss, zählt *systemd* das als Wartezeit. Um die Liste auf das Wesentliche reduziert auszugeben, verwendet man stattdessen *critical-chain*. Rote Zeilen in der Ausgabe zeigen Dienste an, die zu Wartezeiten geführt haben. Typische Ausreißer sind hier der *NetworkManager* oder das Netz selbst. Hohe Werte von etlichen Sekunden deuten auf Komplikationen hin, etwa beim Hochfahren des Netzes:

```
network-online.target @34.567s
```

Um trotzdem schnell zu starten, kann man in diesem Fall den Wert *--timeout =30*" in der Unit *NetworkManager-wait-online.service* (im Verzeichnis */lib/systemd/system/*) beispielsweise auf 10 Sekunden setzen. Alternativ kann der Administrator zunächst das Warten deaktivieren:

```
systemctl disable NetworkManager-wait-online.service
```

Die Übersicht behalten

Eine vollständige Übersicht des aktuellen Status des *systemd*-Frameworks und aller Units gibt *dump* aus. Die – leicht mehrere Zehntausend Zeilen lange – Liste soll laut den *systemd*-Entwicklern nicht von Anwendungen ausgewertet werden, weil sich ihr Format jederzeit ändern kann.

Per *plot* erzeugt man eine Bootchart vergleichbare Grafik im SVG-Format, in der alle gestarteten Dienste angeordnet sind:

```
systemd-analyze plot > start.svg
```

Die so angelegte SVG-Datei zeigt der Browser übrigens wesentlich besser und schneller an als manche Bildbetrachter. Ähnlich vorzeigefähige Resultate erzeugt (nach mehreren Sekunden)

```
systemd-analyze dot | dot -Tsvg > systemd.svg
```

Hierbei handelt es sich um einen Abhängigkeitsgraphen des gesamten *systemd*-Systems. Wegen der enormen Ausmaße der Zeichnung ist es allerdings sinnvoll, nur einen Teil des Systems zeichnen zu lassen:

```
systemd-analyze dot 'Network*' | dot -Tsvg > Network.svg
```

Neben diesen hübschen Funktionen für DevOps lässt sich *systemd-analyze* auch zum ernsthaften Debugging einsetzen. Den gewünschten Detailgrad (Log Level) von *systemd* setzt man permanent in der Konfigurationsdatei */etc/systemd/system.conf* oder im laufenden Betrieb durch

```
systemd-analyze set-log-level debug
systemd-analyze set-log-target console
```

Informationen werden anschließend auf der Konsole ausgegeben. Wer beispielsweise mit *SystemCallFilter* arbeitet, kann sich die Systemaufrufe der *syscall*-Gruppen anzeigen lassen, allerdings nur auf aktuellen *systemd*-Installationen ab Version 233 (nicht auf Debian Stretch mit *systemd* 232):

```
systemd-analyze syscall-filter @basic-io @file-system
```

Alle Unit-Dateien und deren interne Abhängigkeiten lassen sich auf Korrektheit überprüfen:

```
systemd-analyze verify
```

Auch Unit-Dateien im aktuellen Verzeichnis, per Pfadangabe angehängte Units oder solche in $SYSTEMD_UNIT_PATH (dieser Wert überschreibt den in *systemd* einkompilierten) lassen sich mit *verify* testen. Ein Systemverwalter sollte den Befehl tunlichst nicht mit *root*-Rechten verwenden, sondern mit denen eines unprivilegierten Benutzers, weil der Befehl sonst alle generierten Unit-Dateien in */run/systemd/generator.** löscht. Ein aktuelles Debian Stretch verliert dadurch alle gemounteten Dateisysteme, die per *systemd-fstab-generator* von der */etc/fstab* in generierte Unit-Dateien übertragen wurden. Der Fehler ist seit Mai 2017 bekannt. (tiw)

Michael Plura

ist freier Autor und lebt in Schweden.