Systemd-Units und Unit-Dateien verstehen

Einleitung

Zunehmend übernehmen Linux-Distributionen das

systemd Init-System. Diese leistungsstarke Softwaresuite kann viele Aspekte Ihres Servers verwalten, von Diensten über gemountete Geräte bis hin zu Systemzuständen.

In systemd bezieht sich eine

Unit auf jede Ressource, die das System zu bedienen und zu verwalten weiß. Dies ist das primäre Objekt, mit dem die systemd-Tools umgehen können. Diese Ressourcen werden mithilfe von Konfigurationsdateien, den sogenannten Unit-Dateien, definiert.

In diesem Leitfaden stellen wir Ihnen die verschiedenen Units vor, die systemd verwalten kann. Wir werden auch einige der vielen Direktiven behandeln, die in Unit-Dateien verwendet werden können, um die Art und Weise zu gestalten, wie diese Ressourcen auf Ihrem System gehandhabt werden.

Was bieten Ihnen Systemd-Units?

Units sind die Objekte, die systemd verwalten kann. Sie sind im Grunde eine standardisierte Darstellung von Systemressourcen, die von der Suite von Daemons verwaltet und von den bereitgestellten Dienstprogrammen manipuliert werden können. Man kann sagen, dass Units Diensten oder Jobs in anderen Init-Systemen ähneln. Eine Unit hat jedoch eine viel breitere Definition, da sie zur Abstraktion von Diensten, Netzwerkressourcen, Geräten, Dateisystem-Mounts und isolierten Ressourcenpools verwendet werden kann.

Einige der Funktionen, die Units einfach implementieren können, sind:

- **Socket-basierte Aktivierung:** Sockets, die mit einem Dienst verbunden sind, werden am besten aus dem Daemon selbst herausgelöst, um separat behandelt zu werden. Dies bietet eine Reihe von Vorteilen, wie z.B. die Verzögerung des Starts eines Dienstes, bis zum ersten Zugriff auf den zugehörigen Socket.
- **Bus-basierte Aktivierung:** Units können auch über die von D-Bus bereitgestellte Bus-Schnittstelle aktiviert werden. Eine Unit kann gestartet werden, wenn ein zugehöriger Bus veröffentlicht wird.
- **Pfad-basierte Aktivierung:** Eine Unit kann basierend auf der Aktivität in oder der Verfügbarkeit von bestimmten Dateisystempfaden gestartet werden. Dies nutzt *inotify*.
- **Geräte-basierte Aktivierung:** Units können auch bei der ersten Verfügbarkeit von zugehöriger Hardware durch die Nutzung von *udev*-Ereignissen gestartet werden.
- **Implizite Abhängigkeitszuordnung:** Der Großteil des Abhängigkeitsbaums für Units kann von systemd selbst erstellt werden. Sie können immer noch Abhängigkeits- und Reihenfolgeinformationen hinzufügen, aber der Großteil der Arbeit wird Ihnen abgenommen.

- **Instanzen und Vorlagen:** Vorlagen-Unit-Dateien können verwendet werden, um mehrere Instanzen derselben allgemeinen Unit zu erstellen. Dies ermöglicht leichte Variationen oder Geschwister-Units, die alle dieselbe allgemeine Funktion bereitstellen.
- **Einfache Sicherheitshärtung:** Units können durch das Hinzufügen einfacher Direktiven recht gute Sicherheitsfunktionen implementieren. Sie können beispielsweise keinen oder nur Lesezugriff auf Teile des Dateisystems festlegen, Kernel-Fähigkeiten einschränken und privaten

  ```
  /tmp
  ```

  - und Netzwerkzugriff zuweisen.

- **Drop-Ins und Snippets:** Units können leicht erweitert werden, indem Snippets bereitgestellt werden, die Teile der Unit-Datei des Systems überschreiben. Dies erleichtert den Wechsel zwischen Standard- und angepassten Unit-Implementierungen.

Wo befinden sich Systemd-Unit-Dateien?

Die Dateien, die definieren, wie systemd eine Unit behandeln wird, können an vielen verschiedenen Orten gefunden werden, von denen jeder unterschiedliche Prioritäten und Auswirkungen hat.

`/lib/systemd/system`: Die Systemkopie der Unit-Dateien wird im Allgemeinen in diesem Verzeichnis aufbewahrt. Wenn Software Unit-Dateien auf dem System installiert, ist dies der Ort, an dem sie standardmäßig platziert werden. Sie sollten die Dateien in diesem Verzeichnis

nicht bearbeiten.

`/etc/systemd/system`: Wenn Sie die Funktionsweise einer Unit ändern möchten, ist dies der beste Ort dafür. Unit-Dateien, die sich in diesem Verzeichnis befinden, haben Vorrang vor allen anderen Speicherorten auf dem Dateisystem.

`/etc/systemd/system/unit.name.d/`: Wenn Sie nur bestimmte Direktiven aus der System-Unit-Datei überschreiben möchten, können Sie Snippets in einem Unterverzeichnis bereitstellen. Für eine Unit namens

```
example.service
```

könnte ein Unterverzeichnis namens `example.service.d` erstellt werden. Innerhalb dieses Verzeichnisses kann eine Datei, die auf

```
.conf
```

endet, verwendet werden, um Attribute zu überschreiben oder zu erweitern.

`/run/systemd/system`: Es gibt auch einen Ort für Laufzeit-Unit-Definitionen. Dateien in diesem Verzeichnis haben eine Priorität zwischen denen in `/etc/systemd/system` und `/lib/systemd/system`. Der systemd-Prozess selbst verwendet diesen Ort für dynamisch zur Laufzeit erstellte Unit-Dateien. Alle in diesem Verzeichnis vorgenommenen Änderungen gehen bei

einem Neustart des Servers verloren.

Arten von Units

Systemd kategorisiert Units nach dem Typ der Ressource, die sie beschreiben. Der einfachste Weg, den Typ einer Unit zu bestimmen, ist ihr Typ-Suffix. Die folgende Liste beschreibt die verfügbaren Unit-Typen:

- **.service:** Beschreibt, wie ein Dienst oder eine Anwendung auf dem Server verwaltet wird.
- **.socket:** Beschreibt einen Netzwerk- oder IPC-Socket oder einen FIFO-Puffer, den systemd für die Socket-basierte Aktivierung verwendet.
- **.device:** Beschreibt ein Gerät, das von *udev* oder dem *sysfs*-Dateisystem für die systemd-Verwaltung vorgesehen wurde.
- **.mount:** Definiert einen Mountpunkt im System, der von systemd verwaltet wird.
- **.automount:** Konfiguriert einen Mountpunkt, der automatisch eingehängt wird.
- **.swap:** Beschreibt Swap-Speicher auf dem System.
- **.target:** Dient als Synchronisationspunkt für andere Units beim Hochfahren oder bei Zustandsänderungen.
- **.path:** Definiert einen Pfad, der für die Pfad-basierte Aktivierung verwendet werden kann.
- **.timer:** Definiert einen von systemd verwalteten Timer, ähnlich einem Cron-Job für verzögerte oder geplante Aktivierung.
- **.snapshot:** Wird automatisch vom Befehl `systemctl snapshot` erstellt. Ermöglicht die Wiederherstellung des aktuellen Systemzustands nach Änderungen.
- **.slice:** Ist mit Linux Control Group (cgroup) Knoten verbunden und ermöglicht die Zuweisung oder Beschränkung von Ressourcen für alle Prozesse, die der Slice zugeordnet sind.
- **.scope:** Werden automatisch von systemd aus Informationen erstellt, die von seinen Bus-Schnittstellen empfangen werden. Werden zur Verwaltung von extern erstellten Systemprozessen verwendet.

Anatomie einer Unit-Datei

Die interne Struktur von Unit-Dateien ist mit Abschnitten organisiert. Abschnitte werden durch ein Paar eckiger Klammern „[]“ mit dem Abschnittsnamen darin gekennzeichnet.

Allgemeine Merkmale von Unit-Dateien

Abschnittsnamen sind wohldefiniert und Groß-/Kleinschreibung-sensitiv. Innerhalb dieser Abschnitte wird das Verhalten und die Metadaten der Unit durch einfache Direktiven im Schlüssel-Wert-Format definiert, wobei die Zuweisung durch ein Gleichheitszeichen erfolgt:

```
[Section]
Directive1=value
Directive2=value
```

Um eine Direktive in einer Überschreibungsdatei zurückzusetzen, wird ihr ein leerer String zugewiesen:

```
Directive1=
```

Anweisungen im Abschnitt [Unit]

Der erste Abschnitt in den meisten Unit-Dateien ist der Abschnitt `[Unit]`. Er wird im Allgemeinen zur Definition von Metadaten und zur Konfiguration der Beziehung der Unit zu anderen Units verwendet.

Häufige Direktiven sind:

- **Description=:** Eine kurze, spezifische und informative Beschreibung der Unit.
- **Documentation=:** Eine Liste von URIs für die Dokumentation (Manpages oder URLs).
- **Requires=:** Listet Units auf, von denen diese Unit unbedingt abhängt. Wenn die hier aufgeführten Units nicht erfolgreich aktiviert werden können, schlägt die aktuelle Unit fehl.
- **Wants=:** Ähnlich wie Requires=, aber weniger streng. systemd versucht, die hier aufgeführten Units zu starten, aber ein Fehlschlag hindert die aktuelle Unit nicht an der Ausführung. Dies ist der empfohlene Weg, um die meisten Abhängigkeitsbeziehungen zu konfigurieren.
- **BindsTo=:** Ähnlich wie Requires=, bewirkt aber zusätzlich, dass die aktuelle Unit gestoppt wird, wenn die zugehörige Unit beendet wird.
- **Before=:** Die hier aufgelisteten Units werden erst gestartet, nachdem die aktuelle Unit als gestartet markiert wurde.
- **After=:** Die hier aufgelisteten Units werden gestartet, bevor die aktuelle Unit gestartet wird.
- **Conflicts=:** Listet Units auf, die nicht gleichzeitig mit der aktuellen Unit ausgeführt werden können.
- **Condition…=:** Ermöglicht es, bestimmte Bedingungen zu testen, bevor die Unit gestartet wird. Wenn die Bedingung nicht erfüllt ist, wird die Unit einfach übersprungen.
- **Assert…=:** Ähnlich wie Condition, aber ein negatives Ergebnis führt zu einem Fehlschlag.

Anweisungen im Abschnitt [Install]

Dieser optionale Abschnitt definiert das Verhalten einer Unit, wenn sie aktiviert (

enabled) oder deaktiviert (disabled) wird. Das Aktivieren einer Unit markiert sie für den automatischen Start beim Hochfahren.

- **WantedBy=:** Die gebräuchlichste Methode, um festzulegen, wie eine Unit aktiviert werden soll.

Wenn eine Unit mit dieser Direktive aktiviert wird, wird ein symbolischer Link zu dieser Unit im `.wants`-Verzeichnis der angegebenen Ziel-Unit (z. B. `/etc/systemd/system/multi-user.target.wants/`) erstellt.

- **RequiredBy=:** Ähnlich wie WantedBy=, aber es wird eine erforderliche Abhängigkeit spezifiziert, die bei Nichterfüllung zum Fehlschlag der Aktivierung führt.
- **Alias=:** Ermöglicht es, die Unit auch unter einem anderen Namen zu aktivieren.
- **Also=:** Ermöglicht es, Units als Gruppe zu aktivieren oder zu deaktivieren.
- **DefaultInstance=:** Ein Rückfallwert für den Namen bei Template-Units.

Unit-spezifische Abschnittsanweisungen

Zwischen den Abschnitten

[Unit] und [Install] finden sich typischerweise Abschnitte, die spezifisch für den Unit-Typ sind (z.B. [Service], [Socket], etc.).

Der Abschnitt [Service]

Dieser Abschnitt enthält Konfigurationen, die nur für Dienste gelten.

- **Type=:** Gibt an, wie der Dienst verwaltet wird.
- **simple:** (Standard) Der Hauptprozess wird in `ExecStart=` angegeben.
- **forking:** Der Dienstprozess erzeugt einen Kindprozess und beendet sich selbst. `systemd` verfolgt den Kindprozess.
- **oneshot:** Der Prozess ist kurzlebig; systemd wartet auf sein Ende, bevor es fortfährt.
- **dbus:** Die Unit nimmt einen Namen auf dem D-Bus an.
- **notify:** Der Dienst sendet eine Benachrichtigung, wenn er fertig gestartet ist.
- **idle:** Der Dienst wird erst gestartet, wenn alle anderen Jobs erledigt sind.
- **ExecStart=:** Der vollständige Pfad und die Argumente des Befehls, der zum Starten des Dienstes ausgeführt wird.
- **ExecStop=:** Der Befehl, der zum Stoppen des Dienstes benötigt wird.
- **ExecReload=:** Der Befehl zum Neuladen der Konfiguration des Dienstes.
- **Restart=:** Legt fest, unter welchen Umständen systemd versuchen soll, den Dienst automatisch neu zu starten (z.B. „always“, „on-failure“).
- **RemainAfterExit=:** (Normalerweise mit Type=oneshot) Gibt an, dass der Dienst auch nach dem Beenden des Prozesses als aktiv betrachtet werden soll.
- **PIDFile=:** (Bei Type=forking) Der Pfad zur PID-Datei des zu überwachenden Haupt-Kindprozesses.

Der Abschnitt [Socket]

Jede Socket-Unit muss eine passende Service-Unit haben, die aktiviert wird, wenn der Socket eine Verbindung empfängt.

- **ListenStream=:** Adresse für einen Stream-Socket (TCP).
- **ListenDatagram=:** Adresse für einen Datagramm-Socket (UDP).
- **Accept=:** Bestimmt, ob für jede Verbindung eine neue Instanz des Dienstes gestartet wird (Standard: false).
- **Service=:** Wenn der Name der Service-Unit nicht mit dem der Socket-Unit übereinstimmt, kann er hier angegeben werden.

Der Abschnitt [Mount]

Ermöglicht die Verwaltung von Mountpunkten durch systemd. Mount-Units werden oft direkt aus `/etc/fstab`-Dateien übersetzt.

- **What=:** Der absolute Pfad zur Ressource, die gemountet werden soll.
- **Where=:** Der absolute Pfad des Mountpunktes.
- **Type=:** Der Dateisystemtyp des Mounts.
- **Options=:** Alle Mount-Optionen, die angewendet werden müssen.

Der Abschnitt [Timer]

Wird verwendet, um Aufgaben zu planen. Ersetzt oder ergänzt die Funktionalität von *cron* und *at*.

- **OnCalendar=:** Aktiviert die zugehörige Unit basierend auf einem Kalenderereignis (absolute Zeit), ähnlich wie bei Cron-Jobs.
- **OnBootSec=:** Startet die Unit eine bestimmte Zeit nach dem Systemstart.
- **OnUnitActiveSec=:** Startet die Unit eine bestimmte Zeit, nachdem die zugehörige Unit das letzte Mal aktiviert wurde.
- **Unit=:** Gibt die Unit an, die aktiviert werden soll, wenn der Timer abläuft.
- **Persistent=:** Wenn auf „true“ gesetzt, wird der Timer ausgelöst, wenn das System wieder aktiv wird, falls er während der Inaktivität hätte ausgelöst werden sollen.

Erstellen von Instanz-Units aus Vorlagen-Unit-Dateien

Vorlagen-Unit-Dateien ermöglichen die Erstellung mehrerer Instanzen von Units.

Namen von Vorlagen- und Instanz-Units

Vorlagen-Unit-Dateien sind daran zu erkennen, dass sie ein

@-Symbol nach dem Basis-Unit-Namen und vor dem Unit-Typ-Suffix enthalten.

```
example@.service
```

Wenn eine Instanz aus einer Vorlage erstellt wird, wird ein Instanzbezeichner zwischen das

@-Symbol und den Punkt gesetzt.

```
example@instance1.service
```

Vorlagen-Spezifizierer

Die Stärke von Vorlagen-Unit-Dateien liegt in ihrer Fähigkeit, Informationen dynamisch zu ersetzen. Dies geschieht durch die Verwendung von Variablen-Spezifizierern:

- %n: Der vollständige resultierende Unit-Name.
- %p: Der Präfix-Name der Unit (der Teil vor dem @).
- %i: Der Instanzname (der Bezeichner nach dem @). Dies ist einer der am häufigsten verwendeten Spezifizierer.
- %u: Der Name des Benutzers, der zur Ausführung der Unit konfiguriert ist.
- %H: Der Hostname des Systems.
- %%: Wird verwendet, um ein literales Prozentzeichen einzufügen.

Fazit

Wenn Sie mit systemd arbeiten, kann das Verständnis von Units und Unit-Dateien die Verwaltung erleichtern. Im Gegensatz zu vielen anderen Init-Systemen müssen Sie keine Skriptsprache kennen, um die zum Starten von Diensten oder des Systems verwendeten Init-Dateien zu interpretieren. Die Unit-Dateien verwenden eine recht einfache, deklarative Syntax, die es Ihnen ermöglicht, auf einen Blick den Zweck und die Auswirkungen einer Unit bei der Aktivierung zu erkennen.