Engineer

Daemon

ein ohne Bedienung im Hintergrund laufendes Programm

Was es ist

Ein Daemon ist ein langlebiger Prozess, den niemand von Hand startet: Das Betriebssystem oder ein Dienstmanager startet ihn, er hat kein Terminal und bleibt aktiv, um Anfragen zu beantworten oder einen Teil des Rechners zu überwachen. Die Mehrzahl lautet Daemons. Viele tragen das Wort als abschließendes d im Namen: `sshd` nimmt SSH-Verbindungen an, `crond` führt zeitgesteuerte Aufgaben aus, `systemd-journald` sammelt Logs. Windows nennt dieselbe Art Programm einen Dienst. Das tut auch systemd, das heute die meisten Daemons unter Linux überwacht.

Nach dem klassischen Unix-Rezept, das das Linux-Handbuch `daemon(7)` weiterhin in fünfzehn nummerierten Schritten beschreibt, löst sich das Programm selbst von seiner Umgebung. Es schließt geerbte Dateideskriptoren, setzt Signalhandler zurück, spaltet mit fork einen Prozess ab, verlässt mit `setsid()` die Terminalsitzung und spaltet sich erneut ab, damit es nie wieder ein Terminal erhalten kann. Der erste Kindprozess endet, sodass PID 1 den verbleibenden Prozess übernimmt. Dieser leitet seine Standarddatenströme nach `/dev/null`, wechselt nach `/`, schreibt eine PID-Datei und gibt Berechtigungen ab. Der Aufruf `daemon()` der C-Bibliothek erledigt einen Teil davon. Das Handbuch rät von ihm ab, weil er nur eine Teilmenge der Schritte umsetzt.

Der im selben Handbuch beschriebene moderne Daemon, wie ihn sowohl systemd als auch Apples launchd erwarten, kehrt das Rezept um. Das Programm bleibt im Vordergrund, weil der Supervisor es bereits abgekoppelt gestartet hat, mit einer sauberen Umgebung, dem richtigen Nutzer und dem richtigen Arbeitsverzeichnis. Es schreibt seine Logzeilen auf die Standardfehlerausgabe, wo der Dienstmanager sie sammelt, meldet seine Bereitschaft, Anfragen zu bedienen, beendet sich bei SIGTERM sauber und lädt bei SIGHUP seine Konfiguration neu. Apples Anleitung für launchd-Jobs ist bei der alten

Gewohnheit unmissverständlich: Ein Job darf sich nicht selbst zum Daemon machen. Wenn er sich mit fork abspaltet und endet, schließt launchd daraus, dass der Prozess gestorben ist.

Zwei verwandte Begriffe werden damit verwechselt. Eine von cron oder einem systemd-Timer gestartete zeitgesteuerte Aufgabe beginnt, erledigt ihre Arbeit und endet. Ein Daemon bleibt dagegen aktiv. Außerdem teilt macOS den Begriff auf: Ein Launch-Daemon bedient das gesamte System und wird beim Booten geladen, während ein Launch-Agent zu einem Nutzer gehört und nur läuft, solange dieser angemeldet ist. Linux zieht dieselbe Grenze mit den nutzerspezifischen systemd-Instanzen.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Daemon?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Server, E-Mail, Konten und Updates werden im Alltag betreut, damit die Dinge, auf die alle angewiesen sind, unauffällig funktionieren.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 10 PostgreSQL-/MS-SQL-Server auf Ubuntu — 10 bereitgestellte und funktionsfähig gehaltene Datenbankserver

10 PostgreSQL-/MS-SQL-Server auf Ubuntu Kontakt aufnehmen →

Warum es wichtig ist

Jeder Daemon auf einem Server ist eine dauerhafte Verpflichtung. Er belegt Speicher, ob ihn jemand nutzt oder nicht, und jemand muss ihn patchen, sobald sein Anbieter eine Korrektur ausliefert. Lauscht er auf einem Netzwerkport, gehört er jede Betriebsstunde zur Angriffsfläche. Die nützliche Frage lautet daher selten, welchen Daemon man installieren sollte, sondern ob die Arbeit überhaupt einen dauerhaft laufenden Prozess braucht.

Drei Formen decken die meiste Arbeit ab. Etwas nach Zeitplan, etwa eine Sicherung, ein Bericht oder eine Zertifikatserneuerung, braucht keinen dauerhaft laufenden Prozess: Ein Timer startet es, es läuft zu Ende und verursacht zwischen den Läufen keine Kosten. Etwas, das gelegentliche Anfragen beantwortet, kann per Socket aktiviert werden. Der Dienstmanager hält den lauschenden Socket und startet den Daemon bei der ersten Verbindung. Der Socket bleibt bei Neustarts gebunden, sodass eingehende Anfragen warten, statt abgewiesen zu werden. Nur Arbeit, die ständig reagieren muss, etwa ein Webserver, eine

Datenbank oder ein Mail Transfer Agent, rechtfertigt einen ständig aktiven Prozess. Auf einem kleinen Rechner entscheidet diese Wahl, ob alles in den Arbeitsspeicher passt.

Mit jedem Daemon eines Drittanbieters auf dem Weg in den Produktivbetrieb kommt eine zweite Entscheidung. Vier Fragen klären sie: Bleibt er auf Wunsch im Vordergrund? Schreibt er auf die Standardfehlerausgabe oder besteht er auf eigenen Logdateien? Kann er dem Supervisor seine Bereitschaft melden, statt nur gestartet zu sein? Und braucht er root, oder kann der Dienstmanager ihm genau die eine privilegierte Ressource übergeben, die er benötigt, etwa einen Socket auf Port 443? Ein Daemon, der alle vier Fragen gut beantwortet, lässt sich über Jahre mit wenig Aufwand betreiben. Einer, der daran scheitert, bringt ein Wrapper-Skript, eigene Logrotation und eine Ausnahme im nächsten Sicherheitsreview mit.

Wie es in der Praxis aussieht

Die Auslieferung der Website über Gemini und Gopher kostete zwei dauerhaft laufende Prozesse, keinen zweiten Build. Als die Website auf beide Protokolle gespiegelt wurde, schrieb der HTML-Build beim dokumentierten Implementierungsstand auch 777 Gopher-Menüs und ebenso viele Gemini-Dokumente. Zwei kleine Daemons auf demselben Host übernahmen die Auslieferung. Ein separater Buildvergleich hielt fest, dass sich durch das Hinzufügen der Ausgabeformate null HTML-Bytes änderten. Das Ergebnis beschreibt keine späteren redaktionellen oder visuellen Änderungen.

Der Tor-Onion-Spiegel zeigt die Netzwerkseite: Sein Daemon baut ausgehende Tor-Circuits auf. Deshalb verbindet sich nichts von außen hinein, und kein Firewallport wird geöffnet. Die Rolle zur Einrichtung des Onion-Spiegels installiert den Daemon aus dem Paket der Distribution, schreibt eine Konfiguration mit abgeschaltetem ausgehendem Proxyport und verweigert das Überschreiben einer Identität, die sie nicht selbst erzeugt hat. Ein Schritt lieferte eine allgemeine Lehre: Der Schlüssel lag richtig, der Konvergenzlauf meldete keine Änderung, und die neue Adresse blieb unveröffentlicht, weil zu diesem Zeitpunkt nichts in der Konfiguration auf das neue Verzeichnis verwies. Die Rolle schlägt jetzt fehl, wenn eine Identität auf dem Datenträger liegt, aber kein Dienst sie benennt.

Die Berichterstattung zeigt, was ein stiller Daemon verbergen kann. Sechs nur lesende Kollektoren, die den Host als Markdown darstellen wurden überarbeitet, als klar wurde, dass eine blinde Erfassung und ein gesunder Host dieselbe Seite ausgaben: Ein Spiegel, dessen Daemon sein Logformat geändert hatte, meldete dieselbe Null wie ein Spiegel ohne Besucher. Die Kollektoren stellen jetzt unter anderem gelesene Journalzeilen den erkannten Journalzeilen gegenüber. Ein Daemon, der seine Ausgabe ändert, wird so als Lücke sichtbar statt als Ruhe.

Wo es schiefgeht

Der Dienst startet und stoppt eine Sekunde später wieder. Ein klassischer Daemon spaltet sich mit fork ab und lässt seinen Elternprozess nach dem Rezept enden. Unter systemds Standard `Type=simple` gilt der von systemd gestartete Prozess als Hauptprozess. Sein Ende bedeutet deshalb, dass der Dienst gestoppt hat. Mit dem Standard `KillMode=control-group` wird der eigentlich weiterlaufende Kindprozess zusammen mit allem anderen in der Kontrollgruppe der Unit beendet. Starten Sie das Programm mit seiner Vordergrundoption. Für die wenigen unveränderbaren Programme deklarieren Sie `Type=forking` mit `PIDFile=`, einen Typ, von dem das Handbuch für neuen Code abrät.

Clients verbinden sich zu früh und werden abgewiesen. Mit `Type=simple` gilt ein Dienst für systemd als gestartet, sobald es den Prozess abgespaltet hat. Danach eingeordnete Units starten also, während der Daemon noch seine Konfiguration liest. Das Diensthandbuch ergänzt, dass `systemctl start` bei diesem Typ sogar dann Erfolg meldet, wenn die Programmdatei fehlt oder der konfigurierte Nutzer nicht existiert. Bevorzugen Sie `Type=exec`, das wartet, bis die Programmdatei tatsächlich ausgeführt wurde. Lassen Sie alles, wovon andere abhängen, seine Bereitschaft unter `Type=notify` selbst melden:

```
#include <systemd/sd-daemon.h>

int main(void) {
    load_config();
    open_listeners();
    sd_notify(0, "READY=1"); /* units ordered after this one start now */
    return serve_forever();
}
```

Ein Stoppskript beendet einen fremden Prozess. Ein sich abspaltender Daemon schreibt seine Prozess-ID in eine Datei, damit Skripte ihm später Signale senden können. Nach einem Absturz bleibt die Datei liegen. Der Kernel vergibt dieselbe

Nummer an einen unbeteiligten Prozess, und der nächste Stopp sendet SIGTERM an das falsche Programm. Das Diensthandbuch rät, PID-Dateien in modernen Projekten ganz zu vermeiden. Ein Supervisor, der den Prozess gestartet hat, verfolgt ihn über seine Kontrollgruppe. Dabei gibt es keine Datei, die veralten kann.

Ein Daemon stirbt beim Start und hinterlässt keine Spur. Das alte Rezept leitet Standardausgabe und Standardfehlerausgabe nach `/dev/null` . Ein Fehler vor dem Öffnen der eigenen Logdatei verschwindet dadurch. Lassen Sie unter einem Supervisor beide Datenströme unangetastet: Was der Daemon auf die Standardfehlerausgabe schreibt, landet im Journal. Dort liest `journalctl -u` es Unit für Unit.

Eine ausgenutzte Lücke liefert den ganzen Rechner aus. Ein Daemon, der als root läuft, weil er beim Start Port 80 brauchte, behält root für jede danach bediente Anfrage. Lassen Sie den Dienstmanager den privilegierten Socket binden und übergeben. Überlassen Sie auch die Abgabe von Berechtigungen dem Supervisor, wie `daemon(7)` empfiehlt. Einstellungen wie `NoNewPrivileges=` und `ProtectSystem=` begrenzen, worauf ein kompromittierter Prozess zugreifen kann.

Weiterführende Lektüre

- daemon(7) — die fünfzehn Schritte des klassischen Rezepts und das Vorgehen eines modernen Daemons.
- daemon(3) — die Abkürzung der C-Bibliothek und der im eigenen Handbuch dokumentierte Sitzungsleiterfehler.
- sd_notify(3) — wie ein Daemon systemd seine Bereitschaft, das Neuladen oder das Beenden meldet.
- systemd.socket(5) — Socket-Aktivierung: Der Manager hält den Port und startet den Daemon bei Bedarf.
- setsid — die POSIX-Definition des Aufrufs, der einen Prozess von seinem Terminal löst.
- Launch-Daemons und -Agents erstellen — Apples Regeln für launchd-Jobs, auch warum diese niemals einen Kindprozess abspalten und dann enden dürfen.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Daemon?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/daemon/