Engineer

Daemon

ett program som körs i bakgrunden, utan tillsyn

Vad det är

En daemon är en process som lever länge utan att någon människa startar den för hand: operativsystemet eller en tjänstehanterare startar den, den har ingen terminal, och den förblir igång för att besvara förfrågningar eller hålla uppsikt över någon del av maskinen. I svensk text behåller ordet oftast sin engelska stavning, i plural daemoner, och många av dem bär ordet i sitt namn som ett avslutande d: `sshd` tar emot SSH-anslutningar, `crond` kör jobb på bestämda tider och `systemd-journald` samlar in loggar. Windows kallar samma slags program för en tjänst, service, och det gör även systemd, som övervakar de flesta daemoner på Linux i dag.

Det traditionella Unix-receptet, som Linux-manualen `daemon(7)` fortfarande lägger fram i femton numrerade steg, låter programmet koppla loss sig självt. Det stänger ärvda fildeskriptorer, återställer signalhanterare, forkar, anropar `setsid()` för att lämna terminalens session och forkar igen, så att det aldrig kan få tillbaka en terminal. Det första barnet avslutas så att PID 1 adopterar den överlevande processen, som riktar sina standardströmmar mot `/dev/null`, byter katalog till `/`, skriver en PID-fil och släpper sina privilegier. C-bibliotekets anrop `daemon()` gör en del av detta, och manualen avråder från det eftersom det bara genomför en delmängd av stegen.

Den nya sortens daemon som samma manual beskriver, den sort som både systemd och Apples launchd förväntar sig, vänder receptet ut och in. Programmet stannar i förgrunden, eftersom övervakaren redan har startat det fristående, med en ren miljö, rätt användare och rätt arbetskatalog. Det skriver sina loggrader till stderr, standardströmmen för fel, där tjänstehanteraren samlar in dem, meddelar när det är redo att ta emot förfrågningar, avslutas rent vid SIGTERM och läser om sin konfiguration vid SIGHUP. Apples guide till launchd-jobb är rak om den gamla vanan: ett jobb får inte göra sig självt till daemon, och om det forkar och avslutas drar launchd slutsatsen att processen har dött.

Två grannar blandas ihop med den. Ett tidsstyrt jobb, som körs av cron eller en systemd-timer, startar, gör sitt arbete och avslutas, medan en daemon stannar kvar. Och macOS delar ordet i två: en launch daemon tjänar hela systemet och laddas vid uppstart, medan en launch agent hör till en enda användare och bara körs medan den användaren är inloggad. Linux drar samma gräns med de instanser av systemd som körs per användare.

Använd begreppet i ett beslut

Använd ”Vad är en daemon?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Daglig skötsel av servrar, e-post, konton och uppdateringar, så att det som alla förlitar sig på förblir odramatiskt.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Drev 10 PostgreSQL- och MS SQL-servrar på Ubuntu — 10 databasservrar som driftsattes och hölls i gott skick

Drev 10 PostgreSQL- och MS SQL-servrar på Ubuntu Hör av er →

Varför det spelar roll

Varje daemon på en server är ett stående åtagande. Den håller minne oavsett om någon använder den, och någon måste patcha den varje gång leverantören släpper en rättelse. Lyssnar den på en nätverksport är den dessutom en del av attackytan under varje timme som den körs. Den användbara frågan är därför sällan vilken daemon man ska installera, utan om arbetet över huvud taget behöver en process som alltid finns på plats.

Tre former täcker det mesta. Det som sker på bestämda tider, som en säkerhetskopiering, en rapport eller förnyelsen av ett certifikat, behöver ingen ständigt körande process: en timer startar det, det körs till slut, och mellan körningarna kostar det ingenting. Det som besvarar enstaka förfrågningar kan socketaktiveras: tjänstehanteraren äger den lyssnande socketen och startar daemonen vid den första anslutningen, och socketen förblir bunden över omstarter, så att inkommande förfrågningar köas i stället för att avvisas. Bara arbete som måste reagera hela tiden, en webbserver, en databas, en server som för e-post vidare, förtjänar en process som alltid är uppe. På en liten maskin avgör det valet om allt får plats i minnet.

Ett andra beslut följer med varje daemon från tredje part som är på väg mot produktion, och fyra frågor avgör det. Stannar den i förgrunden när den ombeds? Loggar den till stderr, eller insisterar den på egna loggfiler? Kan den tala om för övervakaren att den är redo, och inte bara att den har startat? Och behöver den

root, eller kan tjänstehanteraren ge den det enda privilegierade som den behöver, t.ex. en socket på port 443? En daemon som svarar bra på alla fyra är billig att köra i åratal; en som inte gör det kommer med ett wrapperskript, en egen loggrotation och ett undantag i nästa säkerhetsgranskning.

Så ser det ut i praktiken

Tre delar av företagets egen plattform visar de besluten fattade i det öppna.

Att servera webbplatsen över Gemini och Gopher kostade två ständigt körande processer, inte ett andra bygge. När webbplatsen speglades till båda protokollen skrev bygget som skriver HTML-koden också 777 Gopher-menyer och lika många Gemini-dokument i det dokumenterade implementeringsskedet, och två små daemoner på samma värddator tog hand om att servera dem. En separat jämförelse av byggen registrerade noll ändrade HTML-byte när utdataformaten lades till; resultatet beskriver inte senare redaktionella eller visuella ändringar.

Spegeln som onion-tjänst i Tor visar nätverkssidan: dess daemon bygger utgående kretsar, så ingenting ansluter inåt och ingen port i brandväggen öppnas. Rollen som sätter upp onion-spegeln installerar daemonen från distributionens eget paket, skriver en konfiguration där den utgående proxyporten är avstängd och avstår från att skriva över en identitet som den inte själv har genererat. Ett steg gav en lärdom som gäller allmänt: nyckeln hamnade rätt, konvergenskörningen rapporterade ingen ändring, och den nya adressen publicerades aldrig, eftersom ingenting i konfigurationen vid den tidpunkten namngav den nya katalogen. Rollen fallerar nu så snart en identitet ligger på disk utan att någon tjänst namnger den.

Rapporteringen visar vad en tyst daemon kan dölja. Sex insamlare som bara läser och återger servern som Markdown byggdes om när det stod klart att en blind insamling och en frisk server gav samma sida: en spegel vars daemon hade ändrat sitt loggformat rapporterade samma nolla som en spegel som ingen besökte. Insamlarna rapporterar nu bl.a. lästa journalrader mot igenkända journalrader, så att en daemon som ändrar vad den skriver syns som en lucka och inte som lugn.

Var det går fel

Tjänsten startar och stannar igen en sekund senare. En daemon av den gamla sorten forkar och låter sin förälder avslutas, precis som receptet säger. Under systemd:s standardinställning `Type=simple` är den process som systemd startade

huvudprocessen, så när den avslutas har tjänsten stannat, och med standardinställningen `KillMode=control-group` dödas barnet som skulle ha fortsatt tillsammans med allt annat i enhetens kontrollgrupp. Kör programmet med dess flagga för förgrunden; för de få som inte går att ändra, deklarera `Type=forking` med `PIDFile=` , en typ som manualen avråder från för ny kod.

Klienter ansluter för tidigt och avvisas. Med `Type=simple` betraktar systemd tjänsten som startad i samma ögonblick som processen har forkats, så enheter som är ordnade efter den startar medan daemonen fortfarande läser sin konfiguration. Manualen för `systemd.service` tillägger att `systemctl start` rapporterar framgång med den typen även när programfilen saknas eller den konfigurerade användaren inte finns. Föredra `Type=exec` , som väntar tills programfilen verkligen har körts, och för allt som andra är beroende av, låt daemonen själv meddela att den är redo med `Type=notify` :

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

int main(void) {
    load_config();
    open_listeners();
    sd_notify(0, "READY=1"); /* enheter ordnade efter den här startar nu */
    return serve_forever();
}
```

Ett stoppskript dödar en främling. En forkande daemon skriver sitt process-ID till en fil så att skript kan skicka signaler till den senare. Efter en krasch ligger filen kvar, kärnan ger samma nummer till en orelaterad process, och nästa stopp skickar SIGTERM till fel program. Manualens råd är att helt undvika PID-filer i moderna projekt. En övervakare som har startat processen följer den genom dess kontrollgrupp och har ingen fil som kan bli inaktuell.

En daemon dör vid uppstarten och lämnar inga spår. Det gamla receptet riktar standard output och stderr mot `/dev/null` , så ett fel som uppstår innan daemonen har öppnat sin egen loggfil försvinner. Under en övervakare ska båda strömmarna lämnas i fred: det som daemonen skriver till stderr hamnar i journalen, där `journalctl -u` läser det enhet för enhet.

En utnyttjad bugg lämnar över hela maskinen. En daemon som körs som root för att den behövde port 80 vid uppstarten behåller root för varje förfrågan som den betjänar efteråt. Låt tjänstehanteraren binda den privilegierade socketen

och skicka in den, och överlåt åt övervakaren att släppa privilegierna, som `daemon(7)` rekommenderar; inställningar som `NoNewPrivileges=` och `ProtectSystem=` begränsar vad en komprometterad process kan komma åt.

Läs mer

- daemon(7) — de femton stegen i det klassiska receptet, och vad en daemon av den nya sorten gör i stället.
- daemon(3) — genvägen i C-biblioteket, och bristen kring sessionsledaren som dess egen manual dokumenterar.
- sd_notify(3) — hur en daemon talar om för systemd att den är redo, läser om sin konfiguration eller stannar.
- systemd.socket(5) — socketaktivering, där hanteraren håller porten och startar daemonen vid behov.
- setsid — POSIX-definitionen av det anrop som kopplar loss en process från dess terminal.
- Creating Launch Daemons and Agents — Apples regler för launchd-jobb, bl.a. varför ett jobb aldrig får forka och avslutas.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är en daemon?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/daemon/