Engineer

Daemon

et program som kjører i bakgrunnen, uten tilsyn

Hva det er

En daemon er en prosess med lang levetid som ingen starter for hånd: operativsystemet eller en tjenestebehandler (*service manager*) starter den, den har ingen terminal, og den holder seg i live for å svare på forespørsler eller holde øye med en del av maskinen. Mange daemoner bærer en d til slutt i navnet sitt: `sshd` tar imot SSH-tilkoblinger, `crond` kjører planlagte jobber, og `systemd-journald` samler logger. Windows kaller den samme typen program en tjeneste (*service*), og det gjør også systemd, som har tilsyn med de fleste daemoner på Linux i dag.

Den tradisjonelle Unix-oppskriften, som manualsiden `daemon(7)` på Linux fortsatt legger fram i femten nummererte trinn, lar programmet løsrive seg selv. Det lukker arvede fildeskriptorer, nullstiller signalhåndtererne, forker, kaller `setsid()` for å forlate terminalens økt (*session*), og forker en gang til, slik at det aldri kan få en terminal igjen. Det første barnet avslutter, slik at PID 1 adopterer den gjenlevende prosessen, som peker standardstrømmene sine mot `/dev/null`, flytter til `/`, skriver en PID-fil og gir fra seg privilegiene. Kallet `daemon()` i C-biblioteket gjør en del av dette, og manualen fraråder det, fordi det bare implementerer et utvalg av trinnene.

Den nye typen daemon (*new-style daemon*) som den samme manualen beskriver, den typen både systemd og launchd fra Apple forventer, snur oppskriften på hodet. Programmet blir i forgrunnen, fordi tilsynsprosessen (*supervisor*) allerede har startet det løsrevet, med et rent miljø, riktig bruker og riktig arbeidskatalog. Det skriver logglinjene sine til feilstrømmen stderr, der tjenestebehandleren samler dem, sier fra når det er klart til å betjene forespørsler, avslutter ryddig ved SIGTERM og laster konfigurasjonen sin på nytt ved SIGHUP. Apples veiledning om launchd-jobber er rett på sak om den gamle vanen: en jobb må ikke gjøre seg selv til daemon, og hvis den forker og avslutter, konkluderer launchd med at prosessen er død.

To naboer blir forvekslet med den. En planlagt jobb, kjørt av cron eller en timer i systemd, starter, gjør arbeidet sitt og avslutter, mens en daemon blir værende. Og macOS deler ordet i to: en *launch daemon* betjener hele systemet og lastes ved oppstart, mens en *launch agent* tilhører én bruker og bare kjører mens vedkommende er logget inn. Linux trekker den samme grensen med systemd-instanser per bruker.

Bruk begrepet i en beslutning

Bruk «Hva er en daemon?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Daglig ettersyn av servere, e-post, kontoer og oppdateringer, slik at det alle er avhengige av, forblir kjedelig.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Drev 10 PostgreSQL- og MS SQL-servere på Ubuntu — 10 databaseservere produksjonssatt og holdt i god stand

Drev 10 PostgreSQL- og MS SQL-servere på Ubuntu Ta kontakt →

Hvorfor det er viktig

Hver daemon på en server er en stående forpliktelse. Den holder på minne enten noen bruker den eller ikke, og noen må patche den hver gang leverandøren sender ut en rettelse. Lytter den på en nettverksport, er den en del av angrepsflaten hver time den kjører. Det nyttige spørsmålet er derfor sjelden hvilken daemon som skal installeres, men om arbeidet i det hele tatt trenger en prosess som alltid er der.

Tre former dekker det meste av arbeidet. Noe som skjer etter en tidsplan, som en sikkerhetskopi, en rapport eller en fornyelse av et sertifikat, trenger ingen fast prosess: en timer starter det, det kjører ferdig, og mellom kjøringene koster det ingenting. Noe som svarer på sporadiske forespørsler, kan socketaktiveres: tjenestebehandleren eier den lyttende socketen og starter daemonen ved første tilkobling, og socketen forblir bundet over omstarter, så innkommende forespørsler settes i kø i stedet for å bli avvist. Bare arbeid som må reagere hele tiden, en webserver, en database, en e-postserver (*mail transfer agent*), fortjener en prosess som alltid kjører. På en liten maskin avgjør det valget om alt får plass i minnet.

Enda en beslutning følger med hver tredjeparts daemon på vei til produksjon, og fire spørsmål avgjør den. Blir den i forgrunnen når den blir bedt om det? Logger den til stderr, eller insisterer den på egne loggfiler? Kan den fortelle tilsynsprosessen at den er klar, og ikke bare at den er startet? Og trenger den

root, eller kan tjenestebehandleren gi den den ene privilegerte tingen den trenger, som en socket på port 443? En daemon som svarer godt på alle fire, er billig å drive i årevis. En som stryker på dem, kommer med et innpakningsskript, sin egen loggrotasjon og et unntak i neste sikkerhetsgjennomgang.

Slik ser det ut i praksis

Tre deler av selskapets egen plattform viser de beslutningene i praksis.

Å betjene nettstedet over Gemini og Gopher kostet to fast kjørende prosesser, ikke et separat bygg. Da nettstedet ble speilet til begge protokollene, skrev det samme bygget som skriver HTML-en, også 777 Gopher-menyer og like mange Gemini-dokumenter på det dokumenterte implementeringstidspunktet. To små daemoner på samme vert tok seg av å betjene dem. En separat sammenligning av bygg viste null endrede HTML-byte da utdataformatene ble lagt til; dette resultatet beskriver ikke senere redaksjonelle eller visuelle endringer.

Tor-speilet viser nettverkssiden: daemonen bygger utgående kretser, så ingenting ringer inn, og ingen port i brannmuren åpnes. Rollen som setter opp onion-speilet installerer daemonen fra distribusjonens egen pakke, skriver en konfigurasjon der den utgående proxyporten er slått av, og nekter å overskrive en identitet den ikke selv genererte. Ett trinn ga en generell lærdom: nøkkelen kom riktig på plass, converge-kjøringen meldte ingen endring, og den nye adressen ble aldri publisert, fordi ingenting i konfigurasjonen på det tidspunktet navnga den nye katalogen. Som omtalen sier, «rollen feiler nå på en identitet på disk som ingen tjeneste navngir».

Rapporteringen viser hva en stille daemon kan skjule. Seks skrivebeskyttede innsamlere som gjengir verten som Markdown, ble bygd om da det ble klart at en blind innsamling og en sunn vert skrev ut den samme siden. Omtalen sier det slik: «Et speil der daemonen har endret loggformat, gir den samme nullen som et speil ingen besøkte.» Innsamlerne rapporterer nå blant annet journallinjer lest mot journallinjer gjenkjent, så en daemon som endrer det den skriver, viser seg som et hull og ikke som ro.

Hvor det går galt

Tjenesten starter og stopper igjen et sekund senere. En daemon av den gamle typen forker og lar foreldreprosessen avslutte, slik oppskriften sier. Under standardinnstillingen `Type=simple` i systemd er prosessen systemd startet,

hovedprosessen, så når den avslutter, betyr det at tjenesten har stoppet, og med standardinnstillingen `KillMode=control-group` blir barnet som skulle fortsette, drept sammen med alt annet i kontrollgruppen til enheten. Kjør programmet med flagget for forgrunn. For de få som ikke kan endres, deklarerer du `Type=forking` med `PIDFile=` , en type manualen fraråder for ny kode.

Klienter kobler til for tidlig og blir avvist. Med `Type=simple` regner systemd tjenesten som startet i det øyeblikket prosessen er forket, så enheter som er ordnet etter den, starter mens daemonen fortsatt leser konfigurasjonen sin. Tjenestemanualen legger til at `systemctl start` melder suksess under den typen selv når den kjørbare filen mangler, eller når kontoen tjenesten skal kjøre som, ikke finnes. Foretrekk `Type=exec` , som venter til den kjørbare filen faktisk er kjørt, og for alt andre er avhengige av, lar du daemonen selv melde fra om at den er klar, under `Type=notify` :

```
#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();
}
```

Et stoppskript dreper en fremmed. En daemon som forker, skriver prosess-ID-en sin til en fil, slik at skript kan sende den signaler senere. Etter et krasj blir filen liggende, systemkjernen gir det samme nummeret til en prosess som ikke har noe med saken å gjøre, og neste stopp sender SIGTERM til feil program. Rådet i tjenestemanualen er å unngå PID-filer helt i moderne prosjekter. En tilsynsprosess som startet prosessen, følger den gjennom kontrollgruppen dens og har ingen fil som kan bli utdatert.

En daemon dør ved oppstart og etterlater seg ingen spor. Den gamle oppskriften peker både standard utdata og feilstrømmen mot `/dev/null` , så en feil som oppstår før daemonen åpner sin egen loggfil, forsvinner. Under en tilsynsprosess lar du begge strømmene være i fred: det daemonen skriver til stderr, havner i journalen, der `journalctl -u` leser det enhet for enhet.

Én utnyttet feil gir bort hele maskinen. En daemon som kjører som root fordi den trengte port 80 ved oppstart, beholder root for hver forespørsel den betjener etterpå. La tjenestebehandleren binde den privilegerte socketen og sende den inn, og overlat det å gi fra seg privilegier til tilsynsprosessen, slik `daemon(7)` anbefaler. Innstillinger som `NoNewPrivileges=` og `ProtectSystem=` snevrer inn hva en kompromittert prosess kan røre.

Les mer

- daemon(7) — de femten trinnene i den klassiske oppskriften, og hva en daemon av den nye typen gjør i stedet.
- daemon(3) — snarveien i C-biblioteket, og svakheten rundt sesjonslederen som dens egen manualside skriver ned.
- sd_notify(3) — hvordan en daemon forteller systemd at den er klar, laster på nytt eller stopper.
- systemd.socket(5) — socketaktivering, der tjenestebehandleren holder porten og starter daemonen ved behov.
- setsid — POSIX-definisjonen av kallet som løsriver en prosess fra terminalen sin.
- Creating Launch Daemons and Agents — reglene Apple setter for launchd-jobber, blant dem hvorfor en jobb aldri må forke og avslutte.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en daemon?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/daemon/