Engineer

Демон

програма, що працює у фоні без нагляду

Що це таке

Демон — це довгоживучий процес, який людина не запускає власноруч: його стартує операційна система або менеджер служб, він не має термінала й залишається працювати, щоб відповідати на запити чи стежити за якоюсь частиною машини. Англійською слово пишеться daemon, і багато демонів носять його в назві як кінцеву літеру d: `sshd` приймає SSH-з'єднання, `crond` виконує завдання за розкладом, `systemd-journald` збирає журнали. Windows називає таку саму програму службою, і так само її називає systemd, який сьогодні керує більшістю демонів у Linux.

Класичний рецепт Unix, який сторінка посібника Linux `daemon(7)` досі викладає у п'ятнадцяти пронумерованих кроках, змушує програму відокремитися самотужки. Вона закриває успадковані файлові дескриптори, скидає обробники сигналів, робить fork, викликає `setsid()`, щоб покинути сесію термінала, і робить fork ще раз, щоб термінал до неї вже не повернувся. Перший нащадок завершується, тож PID 1 усиновлює вцілілий процес, а той спрямовує стандартні потоки в `/dev/null`, переходить у `/`, записує PID-файл і відмовляється від привілеїв. Функція `daemon()` із бібліотеки C робить частину цієї роботи, і посібник радить її уникати саме тому, що вона реалізує лише частину кроків.

Новий стиль, який той самий посібник називає new-style daemons і якого чекають і systemd, і launchd від Apple, перевертає рецепт. Програма лишається на передньому плані, бо менеджер служб уже запустив її відокремленою, з чистим оточенням, потрібним користувачем і потрібним робочим каталогом. Вона пише журнальні рядки в стандартний потік помилок, stderr, звідки їх забирає менеджер служб, повідомляє, коли готова приймати роботу, отримавши SIGTERM, коректно завершується, а отримавши SIGHUP, перечитує конфігурацію. Посібник Apple про завдання

launchd різко висловлюється про стару звичку: завдання не має демонізувати себе, а якщо воно робить fork і завершується, launchd вирішує, що процес помер.

Із демоном часто плутають двох сусідів. Завдання за розкладом, яке запускає cron або таймер systemd, стартує, виконує роботу й завершується, а демон лишається. А macOS ділить слово надвоє: launch daemon обслуговує всю систему й завантажується під час старту, тоді як launch agent належить одному користувачеві й працює лише тоді, коли той увійшов у систему. Linux проводить ту саму межу за допомогою окремих екземплярів systemd для кожного користувача.

Застосуйте поняття до рішення

Застосуйте «Що таке демон?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Щоденний догляд за серверами, поштою, обліковими записами й оновленнями, щоб те, на що всі покладаються, працювало тихо й без сюрпризів.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Підтримувала 10 серверів PostgreSQL і MS SQL — 10 серверів баз даних розгорнуто й підтримувано в справному стані

Підтримувала 10 серверів PostgreSQL і MS SQL Зв'язатися з нами →

Чому це важливо

Кожен демон на сервері — це постійне зобов’язання. Він займає пам’ять, хай ним користуються чи ні, і хтось мусить його оновлювати щоразу, коли постачальник випускає виправлення. Якщо він слухає мережевий порт, то входить у поверхню атаки щогодини, поки працює. Тож корисне питання рідко звучить «який демон поставити?», а значно частіше «чи потрібен цій роботі постійний процес узагалі?».

Більшість роботи вкладається в три форми. Те, що відбувається за розкладом, як-от резервна копія, звіт чи поновлення сертифіката, не потребує постійного процесу: таймер запускає його, воно доходить до кінця, а між запусками його вартість нульова. Те, що обслуговує нечасті запити, можна активувати через сокет: менеджер служб володіє слухаючим сокетом і запускає демон під час першого з’єднання, а оскільки сокет лишається прив’язаним під час перезапуску, нові запити чекають у черзі замість отримати відмову. Лише робота, яка мусить реагувати безперервно, як

вебсервер, база даних чи поштовий сервер, заслуговує на процес, що працює завжди. На маленькій машині від цього вибору залежить, чи все вміститься в пам'ять.

Друге рішення постає щоразу, коли демон, написаний кимось іншим, має потрапити в робоче середовище, і його вирішують чотири питання. Чи лишається він на передньому плані, якщо попросити? Чи пише журнал у stderr, чи наполягає на власних файлах журналу? Чи вміє повідомити менеджерові служб, що вже готовий, а не лише що запустився? І чи потрібен йому root, чи менеджер служб може передати йому ту єдину привілейовану річ, яка йому потрібна, наприклад сокет на порту 443? Демон, що добре відповідає на всі чотири, дешево експлуатувати роками. Той, що не проходить ці перевірки, приходить зі скриптом-обгорткою, власною ротацією журналів і винятком у наступному аудиті безпеки.

Як це виглядає на практиці

На власній платформі компанії ці рішення видно у трьох роботах.

Обслуговування сайту через Gemini і Gopher потребувало двох постійних процесів, а не другого збирання. Коли сайт віддзеркалили в обидва протоколи, та сама збірка, що записує HTML, на зафіксованому етапі реалізації також створювала 777 меню Gopher і стільки ж документів Gemini, а їх обслуговування взяли на себе два невеликі демони на тому самому хості. Окреме порівняння збірок зафіксувало нуль змінених байтів HTML під час додавання форматів виводу; цей результат не описує пізніших редакторських чи візуальних змін.

Дзеркало в Tor показує мережевий бік того самого вибору: його демон будує вихідні ланцюги, тож ззовні до нього ніхто не підключається і в брандмауері не відкривається жоден порт. Роль, яка розгортає onion-дзеркало, встановлює демон із пакета самого дистрибутива, пише конфігурацію з вимкненим вихідним проксі-портом і відмовляється перезаписувати ідентичність onion-сервісу, яку не сама згенерувала. Один крок дав урок для демонів загалом: ключ ліг правильно, прогін повідомив, що змін немає, а нову адресу так і не опублікували, бо на той момент жоден рядок конфігурації не вказував на новий каталог. Тепер роль зупиняється з помилкою, щойно на диску з'являється ідентичність, на яку не вказує жодна служба.

Звітність показує, що може приховати тихий демон. Шість збирачів, які лише читають і подають хост у вигляді Markdown, переробили, коли стало ясно, що сліпий збір і здоровий хост дають однакову сторінку: дзеркало, чий демон змінив формат журналу, показувало той самий нуль, що й дзеркало, яке ніхто не відвідав. Тепер збирачі серед іншого звітують, скільки рядків журналу прочитали і скільки з них розпізнали, тож демон, який змінив те, що пише, проявляється як розрив, а не як спокій.

Де виникають проблеми

Служба запускається і за секунду знову зупиняється. Демон старого зразка робить fork і дає батьківському процесу завершитися, як і велить рецепт. За типового `Type=simple` головним процесом вважається той, який запустив systemd, тож його завершення означає зупинку служби, а за типового `KillMode=control-group` нащадок, що мав працювати далі, гине разом з усією контрольною групою юніта. Запускайте програму з прапорцем переднього плану. Для тих небагатьох, які змінити неможливо, оголосіть `Type=forking` разом із `PIDFile=`, пам’ятаючи, що посібник радить уникати цього типу в новому коді.

Клієнти підключаються зарано й отримують відмову. За `Type=simple` systemd вважає службу запущеною, щойно відгалузив процес, тож юніти, впорядковані після неї, стартують, поки демон ще читає конфігурацію. Посібник зі служб додає, що за цього типу `systemctl start` повідомляє про успіх, навіть коли виконуваного файлу немає або налаштованого користувача не існує. Обирайте `Type=exec`, який чекає, доки програму справді буде виконано, а для всього, від чого залежать інші, нехай демон сам повідомляє про готовність через `Type=notify`:

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

int main(void) {
    load_config();
    open_listeners();
    sd_notify(0, "READY=1"); /* юніти, що чекають на цей, стартують зараз */
    return serve_forever();
}
```

Скрипт зупинки вбиває чужий процес. Демон, що робить fork, записує свій ідентифікатор процесу у файл, щоб скрипти могли пізніше надіслати йому сигнал. Після аварії файл лишається, ядро віддає той самий номер зовсім

іншому процесові, і наступна зупинка надсилає SIGTERM не тій програмі. Посібник зі служб радить у сучасних проєктах узагалі обходитися без PID-файлів. Менеджер служб, який сам запустив процес, стежить за ним через його контрольну групу, тож файлу, що міг би застаріти, у нього просто немає.

Демон гине під час старту й не лишає слідів. Старий рецепт спрямовує стандартний вивід і stderr у `/dev/null`, тож повідомлення про помилку, яке з'явилося до того, як демон відкрив власний файл журналу, зникає. Під менеджером служб обидва потоки слід лишити як є: усе, що демон пише в stderr, потрапляє в журнал, де `journalctl -u` показує його окремо для кожного юніта.

Одна вразливість віддає всю машину. Демон, який працює від root, бо під час старту йому був потрібен порт 80, зберігає права root для кожного наступного запиту. Нехай менеджер служб прив'яже привілейований сокет і передасть його демонові, а скидання привілеїв залиште менеджерові, як радить `daemon(7)`; параметри на кшталт `NoNewPrivileges=` і `ProtectSystem=` звужують те, до чого дотягнеться зламаний процес.

Що почитати далі

- daemon(7) — п'ятнадцять кроків класичного рецепта і те, що натомість робить демон нового стилю.
- daemon(3) — скорочений шлях бібліотеки C і вада з лідером сесії, яку фіксує її власний посібник.
- sd_notify(3) — як демон повідомляє systemd, що він готовий, перечитує конфігурацію або зупиняється.
- systemd.socket(5) — активація через сокет, коли порт тримає менеджер, а демон стартує на вимогу.
- setsid — визначення POSIX для виклику, що відокремлює процес від термінала.
- Creating Launch Daemons and Agents — правила Apple для завдань launchd, зокрема чому завдання не може робити fork і завершуватися.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке демон?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/daemon/