# systemd timers: nowocześniejsza alternatywa dla crona

**Poziom: średnio**

**Dotyczy: dystrybucje z systemd (RHEL/Rocky 9–10 domyślnie, Debian/Ubuntu, SLES)**

**Czas: ~20 min**

*Jednostki typu timer to sposób systemd na uruchamianie zadań o określonej porze — to samo, co robi `cron`, ale z logami w journalu, zależnościami między usługami i możliwością ręcznego odpalenia tym samym poleceniem, którym zarządzasz każdą inną usługą.*

**Skróty użyte na tej stronie (3)**

| Skrót | Co znaczy |
|---|---|
| **RHEL** | Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa |
| **SLES** | SUSE Linux Enterprise Server: płatna dystrybucja Linuksa firmy SUSE, konkurent Rocky/RHEL |
| **SSH** | Secure Shell: szyfrowane zdalne logowanie do serwera |

# Fiszka

**TYLKO ODCZYT**

## Wypisanie aktywnych timerów: `systemctl list-timers`

```
systemctl list-timers
```

Pokazujemy wszystkie **włączone** timery w systemie, kiedy uruchomią się następnym razem i kiedy uruchomiły się ostatnio.

Przykład:

```
NEXT                        LEFT     LAST                         PASSED
UNIT                     ACTIVATES
Fri 2026-09-18 03:00:00 CEST 6h left Thu 2026-09-17 03:00:01 CEST 18h ago backup-
nocny.timer      backup-nocny.service
Sat 2026-09-19 00:00:00 CEST 15h left Fri 2026-09-18 00:00:01 CEST 9h ago
logrotate.timer          logrotate.service
```

□□ **systemctl** = system control: sterowanie usługami systemd

□□ TYLKO ODCZYT

## Wypisanie timerów łącznie z nieaktywnymi: `systemctl list-timers --all`

```
systemctl list-timers --all
```

Dokładamy do listy timery wyłączone albo takie, które jeszcze nigdy się nie uruchomiły — kolumna `NEXT` pokazuje wtedy `n/a`. To pierwsze miejsce, w którym sprawdzamy, czy timer w ogóle istnieje w systemie, zanim zaczniemy szukać dalej.

□□ **NEXT** = Near-End Crosstalk: przesłuch zbliżny, czyli ile sygnału z jednej pary przecieka do sąsiedniej pary na tym samym końcu kabla · **systemctl** = system control: sterowanie usługami systemd

□□ TYLKO ODCZYT

## Sprawdzenie stanu konkretnego timera: `systemctl status NazwaJednostki.timer`

```
systemctl status backup-nocny.timer
```

Przykład:

```
● backup-nocny.timer - Uruchom kopię zapasową co noc
     Loaded: loaded (/etc/systemd/system/backup-nocny.timer; enabled)
     Active: active (waiting) since Thu 2026-09-18 06:00:00 CEST
    Trigger: Fri 2026-09-19 03:00:00 CEST; 20h left
   Triggers: ● backup-nocny.service
```

`Active: waiting` to normalny stan timera między uruchomieniami — nie jest błędem, timer po prostu czeka na swoją porę. `Trigger:` pokazuje dokładny czas następnego odpalenia.

**systemctl** = system control: sterowanie usługami systemd

**ZMIENIA SYSTEM**

## Włączenie timera na stałe i uruchomienie go teraz: `systemctl enable --now NazwaJednostki.timer`

```
sudo systemctl enable --now backup-nocny.timer
```

Włączamy timer, żeby startował automatycznie przy każdym uruchomieniu systemu (`enable`), i od razu zaczynamy go pilnować w tej sesji (`--now`). Bez `--now` timer zadziała dopiero po restarcie — łatwo o tym zapomnieć i zdziwić się, że „nic się nie uruchamia", chociaż `enable` się powiodło.

Przykład:

```
Created symlink /etc/systemd/system/timers.target.wants/backup-nocny.timer →
/etc/systemd/system/backup-nocny.timer.
```

**systemctl** = system control: sterowanie usługami systemd

**ZMIENIA SYSTEM**

## Ręczne uruchomienie zadania: `systemctl start NazwaUslugi.service`

```
sudo systemctl start backup-nocny.service
```

Uruchamiamy usługę powiązaną z timerem natychmiast, poza harmonogramem — dokładnie tak samo, jak startujemy dowolną inną usługę. To duża przewaga nad cronem: nie trzeba kopiować komendy z crontaba do terminala, żeby sprawdzić, czy działa — uruchamiamy tę samą jednostkę, którą uruchomiłby timer.

**systemctl** = system control: sterowanie usługami systemd

**ZMIENIA SYSTEM**

## Kompletny przykład: para `.service` + `.timer`

```
sudo tee /etc/systemd/system/backup-nocny.service <<'EOF'
[Unit]
Description=Kopia zapasowa katalogu danych

[Service]
Type=oneshot
ExecStart=/usr/local/bin/backup.sh
EOF
```

```
sudo tee /etc/systemd/system/backup-nocny.timer <<'EOF'
[Unit]
Description=Uruchom kopię zapasową co noc

[Timer]
OnCalendar=*-*-* 03:00:00
Persistent=true
RandomizedDelaySec=300

[Install]
WantedBy=timers.target
EOF
```

Pole po polu: `Type=oneshot` mówi systemd, że usługa uruchamia się, wykonuje pracę i kończy (nie zostaje w tle) — typowy typ dla zadań cyklicznych. `ExecStart` to komenda do wykonania, zawsze ścieżką bezwzględną. W pliku timera `OnCalendar` ustala godzinę, `Persistent=true` dogania przegapione uruchomienie po przestoju, `RandomizedDelaySec` rozmywa dokładny moment startu, a `WantedBy=timers.target` w sekcji `[Install]` to cel, do którego dopina się timer przy `enable` — bez tej linii `enable` nie miałby do czego przypiąć symlinka.

Nazwa pliku `.timer` musi być identyczna z nazwą pliku `.service` (różni się tylko rozszerzeniem) — to jest domyślne powiązanie; timer o innej nazwie wymaga jawnego `Unit=` w sekcji `[Timer]`.

```
sudo systemctl daemon-reload
sudo systemctl enable --now backup-nocny.timer
```

Po dopisaniu albo zmianie plików jednostek zawsze `daemon-reload` — inaczej systemd wciąż operuje na starej wersji wczytanej do pamięci.

**tee** = zapisz strumień wejścia do pliku i jednocześnie wypisz go na ekran · **systemctl** = system control: sterowanie usługami systemd

TYLKO ODCZYT

## Ustawienie najprostszego zapisu: `OnCalendar=daily`

Przykład:

```
OnCalendar=daily
```

Odpowiednik `0 0 * * *` z crona — raz na dobę, o północy. systemd rozumie też `weekly`, `monthly`, `yearly`, `hourly`, `minutely` jako gotowe słowa kluczowe.

TYLKO ODCZYT

## Ustawienie pełnego formatu daty i godziny: `OnCalendar=*-*-* 03:30:00`

Przykład:

```
OnCalendar=*-*-* 03:30:00
```

Format ogólny to `ROK-MIESIĄC-DZIEŃ GODZINA:MINUTA:SEKUNDA`, każde pole z gwiazdką jako „dowolna wartość". Ten zapis znaczy „każdego dnia o 3:30:00" — dokładniejszy odpowiednik `0 30 3 * * *` z crona, bo systemd dorzuca też sekundy.

TYLKO ODCZYT

## Ustawienie z dniem tygodnia: `OnCalendar=Mon *-*-* 06:00`

Przykład:

```
OnCalendar=Mon *-*-* 06:00
```

Dzień tygodnia jako angielski skrót (`Mon`, `Tue`...) na początku wyrażenia, przed datą. Można podać kilka dni: `Mon,Wed,Fri *-*-* 06:00` albo zakres `Mon..Fri *-*-* 06:00`.

TYLKO ODCZYT

## Ustawienie wyzwalacza liczonego od startu systemu: `OnBootSec`

Przykład:

```
OnBootSec=15min
```

Zamiast konkretnej godziny w kalendarzu, ten timer uruchamia się określony czas **po starcie systemu** — piętnaście minut po każdym boocie, niezależnie od tego, o której godzinie faktycznie nastąpił.

**□□ TYLKO ODCZYT**

## Ustawienie wyzwalacza liczonego od poprzedniego uruchomienia: `OnUnitActiveSec`

Przykład:

```
OnUnitActiveSec=1h
```

Uruchamia zadanie godzinę po **zakończeniu poprzedniego uruchomienia tej samej usługi**, a nie o stałej porze zegarowej. Jeśli zadanie samo trwa 20 minut, odstępy między startami i tak wynoszą godzinę licząc od końca, nie od początku — inaczej niż w cronie, gdzie odstępy są stałe względem zegara niezależnie od czasu trwania.

**□□ TYLKO ODCZYT**

## Ustawienie odpowiednika anacrona: `Persistent=true`

Przykład:

```
Persistent=true
```

Jeśli maszyna była wyłączona w momencie, w którym timer miał się uruchomić, `Persistent=true` odpala zaległe zadanie zaraz po starcie systemu, zamiast bezpowrotnie je pomijać — dokładnie to, co dla crona załatwia osobny program `anacron`. Tu jest to jedna linijka w pliku timera, bez dodatkowego demona.

**□□ TYLKO ODCZYT**

## Rozmycie dokładnego momentu uruchomienia: `RandomizedDelaySec`

Przykład:

```
RandomizedDelaySec=300
```

Dodaje losowe opóźnienie od 0 do podanej liczby sekund względem wyliczonego czasu uruchomienia. Przydatne, gdy ten sam timer istnieje na wielu maszynach naraz (flota serwerów) — żeby uniknąć sytuacji, w której wszystkie odpalają backup albo zapytanie do tej samej usługi w dokładnie tej samej sekundzie.

**◫◫ TYLKO ODCZYT**

## Ustawienie tolerancji dokładności: `AccuracySec`

Przykład:

```
AccuracySec=1s
```

Domyślna dokładność timera to minuta — systemd celowo grupuje uruchomienia w oknach czasowych, żeby oszczędzać energię (mniej budzenia procesora). `AccuracySec=1s` wymusza dokładność co do sekundy, kosztem tej oszczędności; przydatne przy zadaniach, dla których liczy się dokładny moment.

**◫◫ TYLKO ODCZYT**

## Sprawdzenie wyrażenia przed wdrożeniem: `systemd-analyze calendar`

```
systemd-analyze calendar "Mon *-*-* 06:00"
```

Parsujemy wyrażenie `OnCalendar` bez tworzenia żadnego pliku i od razu widzimy, kiedy dokładnie się uruchomi — łącznie z kilkoma kolejnymi terminami. To jest sposób, żeby sprawdzić składnię **przed** wdrożeniem, zamiast czekać dzień i patrzeć, czy zadanie faktycznie odpaliło się o właściwej porze.

Przykład:

```
  Original form: Mon *-*-* 06:00
Normalized form: Mon *-*-* 06:00:00
    Next elapse: Mon 2026-09-21 06:00:00 CEST
       (in UTC): Mon 2026-09-21 04:00:00 UTC
       From now: 2 days left
```

**◫◫ TYLKO ODCZYT**

## Wypisanie pełnej treści jednostki z nakładkami: `systemctl cat NazwaJednostki.timer`

```
systemctl cat backup-nocny.timer
```

Wypisujemy efektywną treść pliku jednostki, łącznie z ewentualnymi plikami drop-in nałożonymi na oryginał — jedno polecenie zamiast szukania plików ręcznie w `/etc/systemd/system/` i `/usr/lib/systemd/system/`.

Przykład:

```
# /etc/systemd/system/backup-nocny.timer
[Unit]
Description=Uruchom kopię zapasową co noc

[Timer]
OnCalendar=*-*-* 03:00:00
Persistent=true
RandomizedDelaySec=300

[Install]
WantedBy=timers.target
```

**systemctl** = system control: sterowanie usługami systemd

**TYLKO ODCZYT**

## Przegląd logów konkretnego uruchomienia: `journalctl -u NazwaUslugi.service`

```
journalctl -u backup-nocny.service -n 50
```

Logi usługi uruchomionej przez timer trafiają do journala tak samo jak logi każdej innej usługi systemd — bez osobnego pliku, bez konfigurowania niczego. To jedna z głównych przewag nad cronem, gdzie trzeba było ręcznie przekierowywać wyjście, żeby cokolwiek zostało zapisane.

Przykład:

```
wrz 18 03:00:00 bkp01 systemd[1]: Started Kopia zapasowa katalogu danych.
wrz 18 03:00:04 bkp01 backup.sh[9021]: Kopia zakończona: /var/backups/dane-2026-09-
```

```
18.tar.gz
wrz 18 03:00:04 bkp01 systemd[1]: backup-nocny.service: Deactivated successfully.
```

**journalctl** = journal control: czytanie dziennika systemd

**TYLKO ODCZYT**

## Timery użytkownika: `systemctl --user`

```
systemctl --user list-timers
systemctl --user enable --now moj-timer.timer
```

Każdy zwykły użytkownik może mieć własne jednostki timer, bez roota, przechowywane w `~/.config/systemd/user/`. Działają tylko, gdy trwa sesja użytkownika (chyba że włączono `loginctl enable-linger`), i są jedynym sposobem na przećwiczenie timerów bez uprawnień administracyjnych.

**systemctl** = system control: sterowanie usługami systemd

**TYLKO ODCZYT**

## Wszystkie dyrektywy sekcji `[Timer]`

| Dyrektywa | Co robi |
|---|---|
| `OnCalendar=` | uruchomienie o konkretnej dacie/godzinie w formacie kalendarzowym |
| `OnBootSec=` | uruchomienie N czasu po starcie systemu |
| `OnUnitActiveSec=` | uruchomienie N czasu po zakończeniu poprzedniego uruchomienia |
| `Persistent=true` | dogania przegapione uruchomienie po przestoju maszyny |
| `RandomizedDelaySec=` | losowe opóźnienie startu w podanym zakresie |
| `AccuracySec=` | tolerancja dokładności uruchomienia (domyślnie 1 minuta) |
| `Unit=` | wskazuje jednostkę do uruchomienia, jeśli nazwa różni się od `.timer` |

**Uważaj**

- Sam plik `.timer` nic nie robi bez powiązanego `.service` — `enable` timera nie uruchamia usługi od razu, tylko planuje jej przyszłe uruchomienia.
- Po każdej zmianie pliku jednostki: `systemctl daemon-reload`, inaczej systemd działa na starej wersji z pamięci.
- `OnCalendar` liczy w strefie czasowej systemu; `OnBootSec` / `OnUnitActiveSec` to wyzwalacze monotoniczne, niezależne od zegara — patrz ramka pojęcia niżej.

# Z życia wzięte

**Z ŻYCIA WZIĘTE**

Poniedziałek, 7:00: po nocnej awarii prądu bkp01 wraca do życia, ale w `systemctl list-timers` nocny backup wisi z `NEXT` ustawionym dopiero na jutro. Timer włączyliśmy kiedyś samym `enable`, bez `--now`, więc systemd nigdy nie zaplanował pierwszego uruchomienia po tym restarcie usługi. Dopisanie `Persistent=true` do pliku timera i ponowne `enable --now` sprawiają, że zaległy backup rusza od razu, zamiast czekać do kolejnej nocy.

# Po co to istnieje

**Pojęcie: jednostka timer i powiązana jednostka service.** Jednostka typu `.timer` to plik konfiguracyjny systemd, który sam w sobie niczego nie uruchamia — jego jedynym zadaniem jest w odpowiednim momencie **aktywować inną jednostkę**, najczęściej jednostkę `.service` o tej samej nazwie bazowej. To rozdzielenie na dwa pliki — jeden mówi „co zrobić" (`.service`), drugi „kiedy to zrobić" (`.timer`) — pozwala uruchomić to samo zadanie zarówno ręcznie (`systemctl start nazwa.service`), jak i cyklicznie (`systemctl enable --now nazwa.timer`), bez duplikowania definicji komendy w dwóch miejscach.

Cron istnieje od dekad i wystarcza do prostych zadań, ale systemd, odkąd stał się standardowym systemem inicjalizacyjnym większości dystrybucji (RHEL/Rocky, Debian/Ubuntu, SLES), przy okazji dał administratorom alternatywny mechanizm harmonogramowania, spójny z resztą narzędzi do zarządzania usługami. Zamiast osobnego demona `cron` z własnym formatem plików, harmonogram staje się kolejnym rodzajem jednostki — obiektem, który rozumie `systemctl status`, `systemctl cat`, `journalctl -u` i cały pozostały warsztat, którego i tak już używasz do usług.

# Analogia

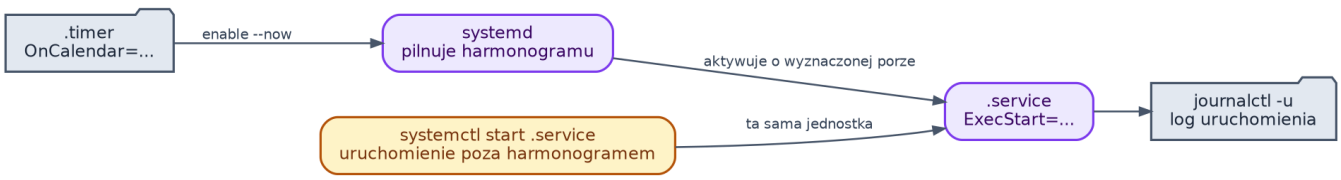
Timer systemd to **budzik podłączony do inteligentnego domu**, w odróżnieniu od crona, który jest zwykłym mechanicznym budzikiem na baterie.

Zwykły budzik (`cron`) po prostu dzwoni o ustawionej porze. Nie wie, czy prąd wysiadł w nocy, nie zostawia zapisu, kiedy dzwonił poprzednio, a żeby go sprawdzić „na sucho", trzeba poczekać do wyznaczonej godziny. Budzik podłączony do systemu domowego (timer) robi to samo, ale dodatkowo zapisuje każde zadziałanie w dzienniku domowym (`journalctl`), potrafi dogonić przegapiony alarm po zaniku prądu (`Persistent=true`) i pozwala nacisnąć przycisk „uruchom teraz" bez czekania (`systemctl start`) — bo alarm i urządzenie wykonawcze są od siebie oddzielone, a nie zlutowane w jedną całość.

Ta wygoda ma cenę: zwykły budzik nastawiasz jednym pokrętłem, budzik od inteligentnego domu wymaga skonfigurowania dwóch osobnych elementów — czujnika czasu i urządzenia wykonawczego — zanim cokolwiek zadziała.

# Jak to działa

**Rozwiń: jak to działa pod spodem**




*Plik .timer aktywuje przez systemd powiązaną jednostkę .service, a ta sama usługa daje się też uruchomić ręcznie, poza harmonogramem (kliknięcie otwiera pełny rozmiar)*

> **Pojęcie: kalendarz systemd (OnCalendar).** Własny format zapisu daty i godziny używany przez systemd w dyrektywie `OnCalendar=`, odrębny od pięciu pól crona. Ogólna postać to `DZIEŃ-TYGODNIA ROK-MIESIĄC-DZIEŃ GODZINA:MINUTA:SEKUNDA`, gdzie każdy segment może być gwiazdką (dowolna wartość), listą (`Mon,Wed,Fri`), zakresem (`Mon..Fri`) albo krokiem (`*-*-* 0/4:00:00` — co cztery godziny). Segment dnia tygodnia jest opcjonalny. Gotowe słowa kluczowe (`daily`, `weekly`, `hourly`) to skróty rozwijane wewnętrznie do tej samej pełnej postaci — `systemd-analyze calendar daily` pokaże, na co się rozwijają.

> **Pojęcie: wyzwalacz monotoniczny.** W odróżnieniu od `OnCalendar`, który odnosi się do konkretnego punktu na zegarze ściennym, wyzwalacze monotoniczne (`OnBootSec`, `OnUnitActiveSec`, `OnStartupSec`, `OnUnitInactiveSec`) liczą czas **od pewnego zdarzenia w systemie**, a nie od północy czy konkretnej daty: od startu systemu, od zakończenia poprzedniego uruchomienia tej samej jednostki, od startu demona systemd. Timer z `OnUnitActiveSec=1h` zawsze odczeka godzinę od **zakończenia** poprzedniego przebiegu, nawet jeśli ten przebieg sam trwał pół godziny — więc realny odstęp między startami rośnie wraz z czasem trwania zadania. To fundamentalnie inne zachowanie niż `cron`, gdzie `*/60 * * * *` uruchamia zadanie o stałych momentach zegarowych bez względu na to, jak długo trwało poprzednie wykonanie — i bez `flock` mogłoby się nałożyć samo na siebie.

**Przewaga nad cronem — uczciwie, punkt po punkcie.** Logi trafiają automatycznie do journala (`journalctl -u`), bez ręcznego przekierowania wyjścia. Jednostka `.service` może deklarować zależności przez `After=`, `Requires=` — zadanie backupu może czekać, aż wystartuje sieć albo zamontuje się dysk, czego cron nie rozumie. `Persistent=true` daje za darmo to, co dla crona wymaga osobnego programu `anacron`. Każda jednostka działa we własnej cgroup, więc `CPUQuota=`/`MemoryMax=` ograniczają zasoby tak samo jak dla dowolnej usługi. I wreszcie: `systemctl start nazwa.service` uruchamia dokładnie to, co uruchomiłby timer.

**Uczciwie też wada.** Prosty cronjob to jedna linijka w jednym pliku. Ten sam efekt w systemd wymaga dwóch osobnych plików (`.service` i `.timer`), `daemon-reload` po każdej zmianie i odrobinę więcej ceremonii — dla drobnego, jednorazowego skryptu bywa to nieproporcjonalny narzut, i wtedy zwykły cronjob jest po prostu szybszym wyborem.

**`systemd-analyze calendar` przed wdrożeniem, nie po.** Wyrażenie `OnCalendar` łatwo pomylić — zamiana kolejności dnia tygodnia i daty, zapomniana gwiazdka, literówka w nazwie dnia. `systemd-analyze calendar "wyrażenie"` sprawdza składnię i pokazuje kilka najbliższych terminów natychmiast, bez tworzenia jednostki i bez czekania do jutra, żeby się przekonać, czy timer w ogóle odpali się o zamierzonej porze.

# Krok po kroku

**1. Utwórz jednostki jako zwykły użytkownik (bez roota).**

```
mkdir -p ~/.config/systemd/user
tee ~/.config/systemd/user/test-timer.service <<'EOF'
[Unit]
Description=Testowe zadanie timera
```

```
[Service]
Type=oneshot
ExecStart=/usr/bin/date +\%s
EOF
```

**2. Dodaj plik timera z krótkim interwałem.**

```
tee ~/.config/systemd/user/test-timer.timer <<'EOF'
[Unit]
Description=Uruchamiaj testowe zadanie co minutę

[Timer]
OnCalendar=*:0/1
AccuracySec=1s

[Install]
WantedBy=timers.target
EOF
```

**3. Przeładuj, włącz i sprawdź.**

```
systemctl --user daemon-reload
systemctl --user enable --now test-timer.timer
systemctl --user list-timers
```

```
NEXT                        LEFT     LAST LAST PASSED UNIT              ACTIVATES
Fri 2026-09-18 09:15:00 CEST 42s left n/a  n/a         test-timer.timer  test-timer.service
```

**4. Poczekaj minutę i obejrzyj log.**

```
sleep 65
journalctl --user -u test-timer.service -n 5
```

```
wrz 18 09:15:00 app01 systemd[2011]: Started Testowe zadanie timera.
wrz 18 09:15:00 app01 date[9911]: 1789635300
wrz 18 09:15:00 app01 systemd[2011]: test-timer.service: Deactivated successfully.
```

# Częste błędy

**Timer włączony, ale zadanie nigdy się nie uruchamia.**

```
Active: active (waiting)
```

`systemctl status` na timerze pokazuje `waiting` i wygląda „zdrowo", ale to samo w sobie nic nie mówi o tym, czy jednostka `.service` w ogóle istnieje albo ma poprawną nazwę. Sprawdzenie: `systemctl list-timers` — kolumna `ACTIVATES` musi pokazywać istniejącą jednostkę; `systemctl cat nazwa.service` potwierdza, że plik faktycznie tam jest.

**Zapomniany `daemon-reload` po edycji pliku.**

Zmieniasz `OnCalendar` w pliku `.timer`, zapisujesz, a `systemd-analyze calendar` na nowym wyrażeniu pokazuje poprawny termin — mimo to timer nadal odpala się o starej porze. systemd trzyma w pamięci wersję jednostki wczytaną przy ostatnim `daemon-reload`, nie czyta pliku na bieżąco. Naprawa: `systemctl daemon-reload`, potem ewentualnie `systemctl restart nazwa.timer`.

**Literówka w `OnCalendar` przechodzi bez ostrzeżenia.**

```
Failed to parse calendar event: Mon *-*-* 6:00:00
```

`systemctl daemon-reload` przy błędnej składni czasem tylko cicho loguje ostrzeżenie do journala, zamiast rzucić błędem na terminalu. Naprawa: zawsze `systemd-analyze calendar "wyrażenie"` przed wpisaniem go do pliku, nie po.

**Timer użytkownika przestaje działać po wylogowaniu.**

Timer `systemctl --user` uruchomiony i przetestowany działa idealnie, dopóki trwa sesja SSH — po rozłączeniu przestaje się odpalać, bo systemd kończy sesję użytkownika razem z jego procesami. Naprawa: `loginctl enable-linger nazwa_uzytkownika`, żeby sesja użytkownika (i jego jednostki `--user`) przetrwała wylogowanie.

**Pomylenie `OnCalendar` z wyzwalaczem monotonicznym.**

Ktoś ustawia `OnUnitActiveSec=1d`, spodziewając się „codziennie o tej samej porze co dziś", a po tygodniu godzina uruchomienia stopniowo się przesuwa. `OnUnitActiveSec` liczy od zakończenia poprzedniego przebiegu, więc każde opóźnienie w wykonaniu (dłuższy przebieg, restart usługi, przeciążony system) kumuluje się w czasie. Do stałej pory zegarowej służy wyłącznie `OnCalendar`.

# □□ Ćwiczenia

> Wszystko poniżej wykonujesz jako zwykły użytkownik, przez `systemctl --user` — nic nie wymaga roota ani zmian w systemowych katalogach.

# Ćwiczenie 1 — timer użytkownika od zera

Powtórz kroki z sekcji *Krok po kroku* powyżej, a następnie zmień `OnCalendar=*:0/1` na `OnCalendar=*:0/2` (co dwie minuty) bez usuwania jednostki.

```
systemctl --user daemon-reload
systemctl --user restart test-timer.timer
systemctl --user list-timers
```

Dlaczego samo `daemon-reload` nie wystarczyło i potrzebny był też `restart` timera?

**Odpowiedź**

`daemon-reload` wczytuje nową treść pliku jednostki do pamięci systemd, ale nie przelicza automatycznie już zaplanowanego czasu następnego uruchomienia dla działającego timera — ten pozostaje aktywny ze starym harmonogramem, dopóki się nie zrestartuje. `systemctl restart nazwa.timer` zatrzymuje i uruchamia timer na nowo, co wymusza przeliczenie kolejnego terminu według zaktualizowanej dyrektywy `OnCalendar`.

# Ćwiczenie 2 — sprawdzenie wyrażenia przed wdrożeniem

```
systemd-analyze calendar "Mon..Fri *-*-* 08,12,16:00:00"
systemd-analyze calendar "*-*-01 00:00:00"
```

Bez wpisywania niczego do pliku jednostki: co robi każde z tych dwóch wyrażeń i kiedy nastąpi najbliższe uruchomienie drugiego z nich?

**Odpowiedź**

Pierwsze uruchamia zadanie trzy razy dziennie (8:00, 12:00, 16:00) w dni robocze od poniedziałku do piątku. Drugie uruchamia się raz w miesiącu, pierwszego dnia o północy — `systemd-analyze calendar` pokaże w polu `Next elapse` dokładną datę najbliższego pierwszego dnia miesiąca liczoną od bieżącej chwili.

# Pytania kontrolne

**1. Czym różni się `OnCalendar=daily` od `OnUnitActiveSec=1d`?**

**Odpowiedź**

`OnCalendar=daily` to wyzwalacz kalendarzowy — uruchamia zadanie zawsze o północy, według zegara ściennego, niezależnie od tego, kiedy uruchomiło się poprzednio. `OnUnitActiveSec=1d` to wyzwalacz monotoniczny — uruchamia zadanie dobę po zakończeniu poprzedniego przebiegu, więc jeśli poprzedni przebieg się opóźnił, godzina kolejnego uruchomienia przesuwa się razem z nim.

**2. Masz plik `.timer` bez odpowiadającego pliku `.service` o tej samej nazwie bazowej. Co się stanie po `systemctl enable --now`?**

**Odpowiedź**

Domyślnie systemd próbuje aktywować jednostkę `.service` o identycznej nazwie bazowej co plik `.timer`. Jeśli taki plik nie istnieje (i nie ma jawnego `Unit=` wskazującego na inną nazwę), aktywacja jednostki w wyznaczonym momencie zakończy się błędem — timer sam w sobie się „uruchomi" (zaloguje zdarzenie), ale nie będzie miał czego wykonać.

**3. Dlaczego `Persistent=true` bywa nazywane „odpowiednikiem anacrona"?**

**Odpowiedź**

Bo rozwiązuje ten sam problem: maszyna wyłączona w momencie, w którym zadanie miało się uruchomić. `anacron` dla crona i `Persistent=true` dla timera systemd oba sprawdzają po starcie systemu, czy dane zadanie przegapiło swój termin, i jeśli tak — uruchamiają je od razu, zamiast czekać do następnego zaplanowanego terminu.

**4. Kiedy zwykły cronjob (jedna linijka w crontabie) jest lepszym wyborem niż para plików `.service`/`.timer`?**

**Odpowiedź**

Przy prostych, jednorazowych albo mało istotnych zadaniach, gdzie narzut dwóch plików i `daemon-reload` nie jest niczym uzasadniony — cron daje ten sam efekt jedną linijką, bez dodatkowej ceremonii. Timer systemd zaczyna się opłacać, gdy potrzebujesz logów w journalu, zależności między usługami, `Persistent=true` albo łatwego ręcznego uruchomienia tym samym poleceniem co reszta usług.

**Powiązane:** cron i crontab: zadania cykliczne · systemd i systemctl: zarządzanie usługami · journalctl: przeszukiwanie dziennika systemd · Dlaczego zadanie nie zadziałało: diagnostyka crona i timerów

**Źródła:** `man 5 systemd.timer` · `man 7 systemd.time` · `man 1 systemd-analyze` · `man 5 systemd.service`

---

Revision #15
Created 2026-09-18 06:57:05 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:44 UTC by Tomasz Konieczny