Zadania zaplanowane, usługi i procesy

Poziom: średniozaawansowany **Dotyczy: Windows 10/11, Windows Server 2016+**

Czas: ~30 min

Trzy warstwy tego, co dzieje się na serwerze bez naszego udziału — zaplanowane zadania, usługi działające w tle i procesy — oraz dziennik zdarzeń jako jedyne wiarygodne źródło odpowiedzi na pytanie „kto to zrobił i kiedy".

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
CIM	Common Information Model: standard opisu obiektów zarządzania w Windows, na nim stoi WMI
gMSA	group Managed Service Account: konto usługi, któremu Windows sam zmienia hasło
WMI	Windows Management Instrumentation: interfejs do odczytu stanu Windowsa
WMS	Warehouse Management System: system zarządzania magazynem

Fiszka

ZMIENIA SYSTEM

Wyszukanie zadania w Task Scheduler o odpowiedniej nazwie

```
# Definiujemy nazwę zadania do wyszukania
$taskName = 'NazwaZadania'

# Pobierz nazwę hosta i skonstruuj nazwę pliku
$hostname = $env:COMPUTERNAME
$filepath = "tasks_$hostname.txt"
```

```
# Wypisz wszystkie zadania wraz z nazwami i ścieżkami, zapisz do pliku
Get-ScheduledTask | Select-Object TaskName, TaskPath | Out-File -FilePath $filepath -
Encoding UTF8

# Oddziel sekcje w pliku
Add-Content -Path $filepath "`nWeryfikacja czy istnieje task o określonej nazwie:`n"

# Sprawdź, czy zadanie o podanej nazwie istnieje, dopisz wynik na końcu pliku lub
stosowny komunikat
$task = Get-ScheduledTask | Where-Object { $_.TaskName -eq $taskName }
if ($task) {
    $task | Out-File -FilePath $filepath -Append -Encoding UTF8
} else {
    Add-Content -Path $filepath "Brak zadania o nazwie $taskName"
}
```

Skrypt uruchamiamy w PowerShell z uprawnieniami Administratora. Zapisuje pliki w aktualnie używanym folderze — przechodzimy najpierw do katalogu docelowego, zanim go uruchomimy. Przydatny, gdy trzeba potwierdzić na wielu serwerach naraz (przez `Invoke-Command` opisane na stronie WinRM: zarządzanie zdalne), czy dane zadanie wdrożeniowe faktycznie istnieje pod oczekiwaną nazwą, zamiast otwierać konsolę Harmonogramu zadań na każdej maszynie z osobna.

Przykład:

```
Weryfikacja czy istnieje task o określonej nazwie:

TaskPath      : \
TaskName      : NazwaZadania
State         : Ready
```

□□ **WinRM** = Windows Remote Management: zdalne zarządzanie Windowsem, podstawa PowerShell Remoting · **hostname** = Hostname: wypisuje nazwę komputera

□□ TYLKO ODCZYT

Wylistowanie wszystkich zadań zaplanowanych: Get-ScheduledTask

```
Get-ScheduledTask | Where-Object State -ne "Disabled" | Format-Table TaskName, TaskPath,
State -AutoSize
```

Zwracamy wszystkie aktywne zadania (pomijając wyłączone) wraz z ich stanem — `Ready` oznacza gotowość do uruchomienia zgodnie z wyzwalaczem, `Running` oznacza trwające właśnie wykonanie.

Przykład:

```
TaskName                TaskPath    State
--------                --------    -----
GoogleUpdateTaskMachine \           Ready
BackupCodzienny         \Firma\     Running
```

◫◫ ZMIENIA SYSTEM

Rejestracja nowego zadania zaplanowanego: Register-ScheduledTask

```
$akcja = New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "-File
SciezkaDoPliku"
$wyzwalacz = New-ScheduledTaskTrigger -Daily -At "03:00"
$ustawienia = New-ScheduledTaskSettingsSet -StartWhenAvailable

Register-ScheduledTask -TaskName "NazwaZadania" -Action $akcja -Trigger $wyzwalacz -
Settings $ustawienia -User "SYSTEM"
```

Budujemy zadanie z trzech osobnych obiektów — akcji (co uruchomić), wyzwalacza (kiedy) i ustawień (jak się zachować, np. przy braku zasilania albo spóźnionym starcie) — i rejestrujemy je jednym poleceniem. `-User "SYSTEM"` uruchamia zadanie na koncie systemowym, bez potrzeby przechowywania hasła — dla zadań wymagających konta z hasłem odsyłamy do strony Poświadczenia i bezpieczne hasła w skryptach.

◫◫ ZMIENIA SYSTEM

Ręczne uruchomienie zadania: Start-ScheduledTask

```
Start-ScheduledTask -TaskName "NazwaZadania"
```

Uruchamiamy zadanie natychmiast, z pominięciem jego wyzwalacza — przydatne przy testowaniu nowo zarejestrowanego zadania bez czekania na najbliższą zaplanowaną porę.

🔒 TYLKO ODCZYT

Sprawdzenie historii wykonań zadania

```
Get-ScheduledTaskInfo -TaskName "NazwaZadania" | Select-Object LastRunTime,
LastTaskResult, NextRunTime
```

> **Pojęcie: LastTaskResult.** Kod zakończenia ostatniego uruchomienia zadania — `0` oznacza sukces, każda inna wartość to kod błędu zwrócony przez uruchomiony program albo skrypt. Sam `Get-ScheduledTask` nie pokazuje tej informacji — trzeba dociągnąć ją osobnym poleceniem `Get-ScheduledTaskInfo`, analogicznie do dociągania nazwy procesu do połączenia sieciowego na stronie Sieć z wiersza poleceń.

Przykład:

```
LastRunTime      LastTaskResult NextRunTime
-----------      -------------- -----------
24.09.2026 03:00              0 25.09.2026 03:00
```

🔒 TYLKO ODCZYT

Pełna historia zdarzeń zadania z dziennika systemowego

```
Get-WinEvent -LogName "Microsoft-Windows-TaskScheduler/Operational" -FilterXPath
"*[EventData[Data[@Name='TaskName']='\NazwaZadania']]"
```

`Get-ScheduledTaskInfo` pokazuje tylko ostatnie uruchomienie — pełną historię z konkretnymi znacznikami czasu każdego startu, zakończenia i ewentualnego błędu trzeba wyciągnąć z dziennika operacyjnego harmonogramu zadań, który domyślnie bywa wyłączony i wymaga włączenia w Podglądzie zdarzeń (`Applications and Services Logs\Microsoft\Windows\TaskScheduler\Operational`).

🔒 TYLKO ODCZYT

Sprawdzenie stanu usługi: Get-Service

```
Get-Service -Name NazwaUslugi
```

Przykład:

```
Status   Name               DisplayName
------   ----               -----------
Running  wuauserv           Windows Update
```

⚠️ ZMIENIA SYSTEM

Zmiana typu uruchomienia usługi: Set-Service

```
Set-Service -Name NazwaUslugi -StartupType Automatic
```

Zmieniamy, czy usługa startuje automatycznie z systemem, ręcznie na żądanie, czy jest wyłączona (`Automatic`, `Manual`, `Disabled`) — bez wpływu na jej bieżący stan działania, jeśli akurat już działa albo stoi.

⚠️ ZMIENIA SYSTEM

Zrestartowanie usługi: Restart-Service

```
Restart-Service -Name NazwaUslugi -Force
```

`-Force` restartuje też usługi zależne, które musiałyby zostać zatrzymane, żeby zatrzymać usługę główną — bez tego parametru polecenie odmawia restartu, jeśli inne usługi deklarują zależność od tej wskazanej.

Przykład błędu bez `-Force`:

```
Restart-Service : Cannot stop service 'NazwaUslugi' because it has dependent services.
```

✅ TYLKO ODCZYT

Sprawdzenie konta, na którym działa usługa: Get-CimInstance Win32_Service

```
Get-CimInstance -ClassName Win32_Service -Filter "Name='NazwaUslugi'" | Select-Object Name, StartName, State
```

> **Pojęcie: Win32_Service.** `Get-Service` nie pokazuje, na jakim koncie faktycznie działa usługa — tę informację (właściwość `StartName`) trzeba pobrać przez WMI/CIM, osobny podsystem zarządzania niż ten, na którym operuje `Get-Service`. Przydatne przy audycie, które usługi działają na koncie `LocalSystem`, a które na dedykowanym koncie usługi wymagającym rotacji hasła — albo na koncie gMSA, opisanym na stronie Konfiguracja kont MSA / gMSA.

Przykład:

```
Name        StartName             State
----        ---------              -----
MSSQLSERVER NT SERVICE\MSSQLSERVER Running
```

WMI = Windows Management Instrumentation: interfejs do odczytu stanu Windowsa · **gMSA** = group Managed Service Account: konto usługi, któremu Windows sam zmienia hasło · **MSA** = Managed Service Account: konto usługi z automatyczną zmianą hasła, dla jednego serwera

TYLKO ODCZYT

Wylistowanie procesów z zużyciem zasobów: Get-Process

```
Get-Process | Sort-Object WorkingSet64 -Descending | Select-Object -First 5 ProcessName, Id, CPU, WorkingSet64
```

Przykład:

```
ProcessName    Id        CPU  WorkingSet64
-----------    --        ---  ------------
sqlservr     5560     892.44     942813184
chrome       8420     412.34     487194112
```

CPU = Central Processing Unit: procesor

ZMIENIA SYSTEM

Zakończenie procesu: Stop-Process

```
Stop-Process -Id NumerPID -Force
```

`-Force` kończy proces bez pytania o potwierdzenie — przydatne w skrypcie automatycznym, ale warto najpierw sprawdzić `-WhatIf`, opisane na stronie Skrypty PowerShell: struktura i dobre praktyki, zanim ubijemy proces, który miał zostać przy życiu.

🔒 TYLKO ODCZYT

Znalezienie procesu trzymającego dany port

```
$port = NumerPortu
$pid = (Get-NetTCPConnection -LocalPort $port -ErrorAction
SilentlyContinue).OwningProcess
Get-Process -Id $pid
```

Łączymy `Get-NetTCPConnection` opisane na stronie Sieć z wiersza poleceń z `Get-Process`, żeby jednym poleceniem odpowiedzieć na pytanie „co dokładnie zajmuje ten port" — częste przy diagnozowaniu błędu `port already in use` zaraz po próbie startu nowej usługi.

Przykład:

```
Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
   1204      42   88320      94512      12.34   4488   0 w3wp
```

🔒 TYLKO ODCZYT

Odpytanie dziennika zdarzeń filtrem: Get-WinEvent -FilterHashtable

```
Get-WinEvent -FilterHashtable @{LogName = 'System'; Id = 1074} | Format-Table -Property
TimeCreated, Message -Wrap
```

> **Pojęcie: Get-WinEvent kontra Get-EventLog.** `Get-EventLog` to starsze polecenie, ograniczone do klasycznych dzienników (`Application`, `System`, `Security`) i wolniejsze przy dużych dziennikach, bo odczytuje zdarzenia sekwencyjnie i filtruje je już po stronie klienta PowerShella. `Get-WinEvent` z `-FilterHashtable` przekazuje filtr (nazwę dziennika, identyfikator zdarzenia, zakres czasu) bezpośrednio do silnika ETW/EventLog systemu Windows, który filtruje zdarzenia przed ich przesłaniem — przy dziennikach liczonych w dziesiątkach tysięcy wpisów różnica w czasie wykonania bywa rzędu wielkości. `Get-WinEvent` obsługuje też nowsze dzienniki aplikacyjne i usługowe (

`Microsoft-Windows-...`), których `Get-EventLog` w ogóle nie widzi.

Przykład:

```
TimeCreated          Message
-----------          -------
24.09.2026 22:14:02  The process C:\Windows\System32\shutdown.exe (SQL01) has initiated
the restart
                     of computer SQL01 on behalf of user DOMENA\nazwa_usera for the
following
                     reason: Other (Planned)
                     Reason Code: 0x80000000
                     Shutdown Type: restart
                     Comment:
```

TYLKO ODCZYT

Filtrowanie po kilku identyfikatorach zdarzeń naraz

```
Get-WinEvent -FilterHashtable @{LogName = 'System'; Id = 1074, 6006, 6008, 41} |
    Sort-Object TimeCreated -Descending |
    Select-Object TimeCreated, Id, Message
```

Łączymy w jednym zapytaniu identyfikatory pokrywające pełny cykl zamknięcia i startu systemu, opisane w tabeli niżej — jedno polecenie pokazuje chronologię: kto zainicjował zamknięcie, czy dziennik zdarzeń zdążył się zatrzymać poprawnie, i czy poprzedni start nie był nieoczekiwanym odzyskaniem po awarii.

TYLKO ODCZYT

Wszystkie identyfikatory zdarzeń związane z restartem i zamknięciem systemu

ID zdarzenia	Dziennik	Znaczenie
`1074`	System	proces albo użytkownik zainicjował zamknięcie lub restart — zawiera nazwę procesu inicjującego, konto użytkownika i podany powód

ID zdarzenia	Dziennik	Znaczenie
`6006`	System	usługa Dziennika zdarzeń zatrzymała się poprawnie — oznacza czyste zamknięcie systemu
`6008`	System	poprzednie zamknięcie systemu było nieoczekiwane — brak zdarzenia `6006` przed startem oznacza zawieszenie, wyłączenie zasilania albo awarię, nie planowe zamknięcie
`41`	System	system uruchomił się ponownie bez czystego zamknięcia (`Kernel-Power`) — najczęściej twardy reset, zanik zasilania albo BSOD, silniejszy sygnał awarii niż samo `6008`

Uważaj

- Zdarzenie `1074` mówi, kto zainicjował zamknięcie z poziomu systemu operacyjnego — nie zarejestruje twardego wyłączenia zasilania ani resetu sprzętowego, na to wskazuje dopiero `41` albo `6008`.
- `Restart-Service -Force` na usłudze z wieloma zależnymi usługami restartuje też je wszystkie — warto sprawdzić `Get-Service -Name NazwaUslugi -RequiredServices` i `-DependentServices` przed użyciem na usłudze współdzielonej przez wiele innych komponentów.
- `Get-WinEvent -FilterHashtable` z błędną nazwą klucza (na przykład literówka `Level` zamiast `LevelDisplayName`) zwraca cichy błąd walidacji parametru, a nie pusty wynik — czytelny komunikat błędu ułatwia poprawkę, o ile nie jest przechwycony przez zbyt szeroki `try/catch`.

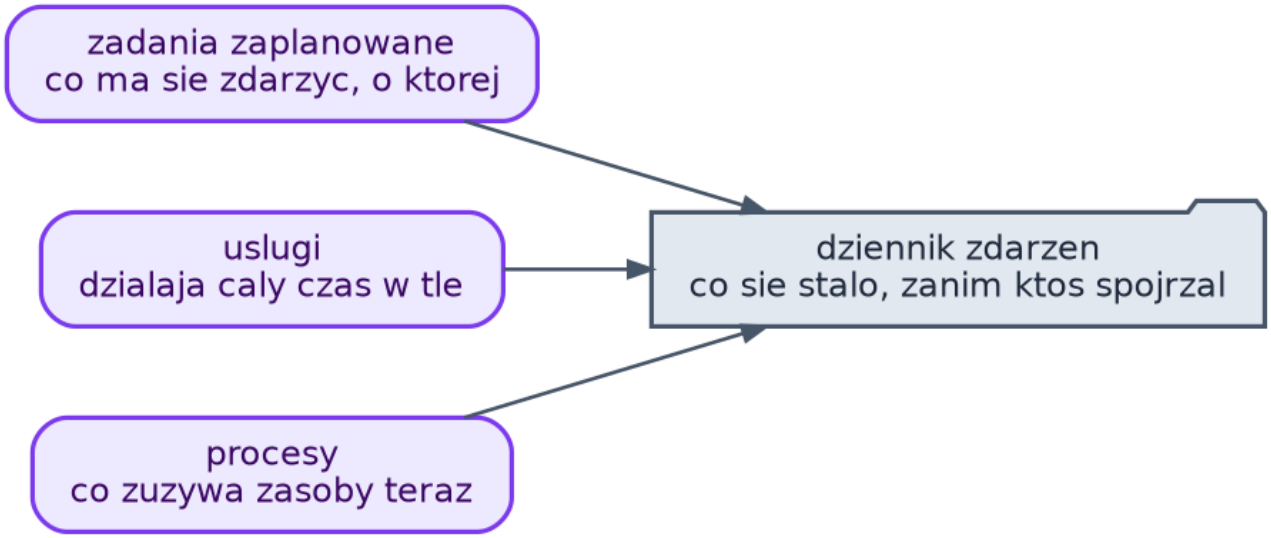
Z życia wzięte

□□ Z ŻYCIA WZIĘTE

Marek zgłasza w poniedziałek o 6:00, że skanery w Swarzędzu nie widzą WMS na ora01, mimo że wszystko wyglądało dobrze w piątek. Sprawdzamy `Get-ScheduledTaskInfo` dla nocnego zadania synchronizacji i `LastTaskResult` pokazuje `1`, nie `0`. Dopiero `Get-WinEvent` na dzienniku operacyjnym harmonogramu pokazuje, że zadanie uruchomiło się o czasie, ale zakończyło błędem zaraz po starcie, bo konto usługowe miało tego dnia wygasłe hasło.

Po co to istnieje

Trzy poziomy tej strony odpowiadają na trzy różne pytania o to, co dzieje się na serwerze bez bieżącego nadzoru administratora. Zadania zaplanowane odpowiadają na „co ma się wydarzyć samo, o określonej porze". Usługi odpowiadają na „co działa non stop w tle, niezależnie od tego, czy ktokolwiek jest zalogowany". Procesy odpowiadają na „co faktycznie zużywa zasoby albo trzyma port w tej właśnie chwili". Dziennik zdarzeń spina wszystkie trzy poziomy wstecz w czasie — jedyne miejsce, gdzie da się sprawdzić, co się stało, zanim ktokolwiek zdążył spojrzeć na ekran.



Trzy warstwy tego, co dzieje sie na serwerze, spiete wspolnym dziennikiem zdarzen (kliknięcie otwiera pełny rozmiar)

Jak to działa

Różnica między `Get-Service` a `Get-CimInstance Win32_Service` wynika z tego, że są to dwa osobne podsystemy zarządzania Windowsem o częściowo pokrywającej się, ale nie identycznej odpowiedzialności. `Get-Service` korzysta z Menedżera kontroli usług (SCM) i zwraca szybko podstawowy stan — działa/nie działa, typ uruchomienia. WMI/CIM ma znacznie szerszy, ale wolniejszy dostęp do metadanych usługi, w tym konta wykonawczego, ścieżki binariów i opisu — sięgamy po CIM dokładnie wtedy, gdy potrzebna informacja nie jest wystawiona przez prostszy, szybszy SCM.

Sekwencja zdarzeń `1074` → `6006` → (restart) → brak `6008` na starcie oznacza planowe, czyste zamknięcie zainicjowane przez kogoś świadomie. Brak `6006` przed `6008` na kolejnym starcie oznacza, że dziennik zdarzeń nie zdążył zapisać swojego własnego, poprawnego zamknięcia — silny sygnał, że system przestał działać nagle, niezależnie od tego, czy `1074` w ogóle wystąpiło wcześniej. `41` z dziennika `Kernel-Power` jest zapisywane przez samo jądro systemu przy starcie po wykryciu, że poprzednie zamknięcie nie przeszło przez normalną sekwencję zamykania — najsilniejszy z czterech sygnałów, bo pochodzi z warstwy niżej niż usługa Dziennika zdarzeń, która przy prawdziwej awarii zasilania też nie zdążyła nic zapisać.

Częste błędy

Wnioskowanie z samego `1074`, że restart był planowy i bezproblemowy. `1074` rejestruje inicjację restartu, nie jego przebieg — system mógł zawiesić się w trakcie samego zamykania, co ujawni dopiero brak `6006` i obecność `6008` na kolejnym starcie.

Użycie `Get-EventLog` na serwerze z dziennikiem `System` liczącym setki tysięcy wpisów. Polecenie działa poprawnie, ale zauważalnie wolniej niż `Get-WinEvent -FilterHashtable` z tym samym filtrem, bo filtruje po stronie klienta zamiast delegować filtrowanie do samego systemu.

Restart usługi bez `-Force` na usłudze z zależnościami i porzucenie próby po pierwszym błędzie. Komunikat `Cannot stop service... because it has dependent services` jest informacją, nie ślepym zaułkiem — `Get-Service -Name NazwaUslugi -DependentServices` pokazuje dokładnie, co jeszcze zostanie zrestartowane razem z usługą główną, zanim doda się `-Force`.

□□ Ćwiczenia

> Wszystko poniżej wykonujemy we własnej sesji PowerShell. Restart usług i zamykanie procesów wymaga ostrożności — testujemy na usługach i procesach, których zatrzymanie nie przeszkodzi w codziennej pracy.

Ćwiczenie 1 — kto trzyma port 3389

```
$pid = (Get-NetTCPConnection -LocalPort 3389 -ErrorAction SilentlyContinue | Select-Object -
First 1).OwningProcess
if ($pid) {
    Get-Process -Id $pid | Select-Object ProcessName, Id, StartTime
} else {
    "Port 3389 nie jest aktualnie zajęty"
```

```
}
```

Sprawdzamy, jaki proces obsługuje port używany przez Pulpit zdalny na naszej maszynie.

Odpowiedź

Na typowej maszynie z Windowsem z włączonym Pulpitem zdalnym port `3389` trzyma proces `svchost` (usługa `TermService` działa wewnątrz współdzielonego procesu hosta usług, nie jako osobny plik wykonywalny) — sama nazwa procesu nie mówi wprost, o którą usługę chodzi, do tego posłużyłoby dodatkowo sprawdzenie `Get-CimInstance Win32_Service -Filter "ProcessId=$pid"`.

Ćwiczenie 2 — sekwencja zdarzeń ostatniego restartu

```
Get-WinEvent -FilterHashtable @{LogName = 'System'; Id = 1074, 6005, 6006, 6008, 41} -
MaxEvents 20 |
    Sort-Object TimeCreated |
    Select-Object TimeCreated, Id, LevelDisplayName
```

Odtwarzamy chronologię ostatniego zamknięcia i startu własnej maszyny na podstawie samych identyfikatorów zdarzeń.

Odpowiedź

Poprawna, planowa sekwencja to `1074` (inicjacja), `6006` (dziennik zatrzymał się poprawnie), a po ponownym starcie `6005` (dziennik wystartował) — bez `6008` ani `41` pomiędzy nimi. Obecność `6008` albo `41` bez poprzedzającego czystego `6006` wskazuje na zamknięcie, które nie przeszło przez normalną sekwencję, niezależnie od tego, czy wcześniej zarejestrowano jakiekolwiek `1074`.

Pytania kontrolne

1. Dlaczego `Get-ScheduledTask` sam z siebie nie pokazuje, czy ostatnie uruchomienie zadania się powiodło?

Odpowiedź

`Get-ScheduledTask` zwraca definicję i bieżący stan zadania (`Ready`, `Running`, `Disabled`), nie historię jego wykonań — informacje o wyniku ostatniego uruchomienia (`LastTaskResult`, `LastRunTime`) przechowuje osobna struktura, dostępna przez `Get-ScheduledTaskInfo`, analogicznie do tego, jak `Get-NetTCPConnection` nie zna nazwy procesu bez osobnego zapytania do `Get-Process`.

2. Serwer zgłasza zdarzenie `41` z dziennika Kernel-Power, ale nie ma żadnego poprzedzającego `1074`. Co to sugeruje?

Odpowiedź

Brak `1074` oznacza, że żaden proces ani użytkownik nie zainicjował zamknięcia z poziomu systemu operacyjnego w sposób, który Windows zdążyłby zarejestrować — w połączeniu z `41` (restart bez czystego zamknięcia) to silny sygnał zewnętrznej przyczyny: zaniku zasilania, twardego resetu sprzętowego albo awarii na poziomie niższym niż system operacyjny, a nie planowego działania administratora czy użytkownika.

Powiązane: Sieć z wiersza poleceń · Skrypty PowerShell: struktura i dobre praktyki · Konfiguracja kont MSA / gMSA · Wysyłanie poczty ze skryptu

Źródła: `Get-Help Get-WinEvent` · `Get-Help Get-ScheduledTask` · dokumentacja Microsoft Learn, „Event ID 41 Kernel-Power" i „Task Scheduler cmdlets"

Revision #41
Created 2025-09-20 06:58:37 UTC by Tomasz Konieczny
Updated 2026-10-26 14:06:53 UTC by Tomasz Konieczny