Monitorowanie i utrzymanie

Monitorowanie usług oraz utrzymanie serwera.

- Monitorowanie typowych usług
- Utrzymanie serwera Zabbix
- Zabbix: SMART dysków na Windows przez agenta 2

Monitorowanie typowych usług

Poziom: średnio **Dotyczy: Zabbix agent2 6.0 LTS** **Czas: ~25 min**

Standardowe klucze agenta pokrywają obciążenie CPU i miejsce na dysku same z siebie — wszystko, co specyficzne dla konkretnej aplikacji, dopisujemy sami albo bierzemy z gotowego szablonu.

Skróty użyte na tej stronie (12)

Skrót	Co znaczy
CPU	Central Processing Unit: procesor
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
JSONPath	Zapis ścieżki, która wskazuje jedną wartość w dokumencie JSON, np. $.value to pole value na samym wierzchu
LLD	Low-Level Discovery: automatyczne wykrywanie w Zabbixie np. dysków i interfejsów
LTS	Long Term Support: wersja z długim wsparciem poprawkami
MIB	Management Information Base: spis wartości, o które można zapytać urządzenie przez SNMP, np. temperatura albo napięcie
OID	Object Identifier: numer, którym w SNMP wskazujemy jedną wartość urządzenia, na przykład stan baterii UPS
SNMP	Simple Network Management Protocol: protokół odczytu stanu urządzeń sieciowych
SQL	Structured Query Language: język zapytań do relacyjnych baz danych
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
TLS	Transport Layer Security: szyfrowanie połączeń, np. w HTTPS

Fiszka

□□ PLIK KONFIGURACYJNY

Dodanie elementu własnego: UserParameter

```
UserParameter=custom.procesy.count[*],pgrep -c "$1"
```

(gdzie `$1` to pierwszy parametr przekazywany przy wywołaniu klucza, np. `custom.procesy.count[nginx]` policzy procesy nginksa)

Plik z definicją `UserParameter` umieszczamy w katalogu dociąganym przez `Include` z `zabbix_agent2.conf`, nie edytujemy głównego pliku bezpośrednio.

□□ ZMIENIA SYSTEM

Restart agenta po dodaniu UserParameter: systemctl restart

```
sudo systemctl restart zabbix-agent2
```

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

□□ TYLKO ODCZYT

Testowanie elementu lokalnie przed dodaniem we frontendzie: zabbix_agent2 -t

```
zabbix_agent2 -t custom.procesy.count[nginx]
```

Przykład:

```
custom.procesy.count[nginx]                   [t|5]
```

Uważaj: komenda w `UserParameter` wykonuje się na uprawnieniach użytkownika, na którym działa agent — zwykle nieuprzywilejowanego `zabbix`. Odczyt plików wymagających roota (np. logów aplikacji zapisanych z uprawnieniami 600) wymaga albo zmiany uprawnień pliku, albo dopisania wąskiej reguły `sudo` wyłącznie dla tej jednej komendy, nigdy pełnego `sudo` bez ograniczeń dla konta agenta.

□□ **zabbix_agent2** = sam proces/binarka agenta Zabbixa drugiej generacji, wywoływana też ręcznie do testów

□□ PLIK KONFIGURACYJNY

Konfiguracja wtyczki PostgreSQL w agencie2

```
Plugins.PostgreSQL.Default.Uri=tcp://127.0.0.1:5432
Plugins.PostgreSQL.Default.User=zbx_monitor
Plugins.PostgreSQL.Default.Password=HasloMonitora
```

Uważaj: hasło konta monitorującego w pliku konfiguracyjnym agenta jest tekstem jawnym — plik `zabbix_agent2.conf` musi mieć uprawnienia ograniczone wyłącznie do konta `zabbix` (`0640`, właściciel `zabbix:zabbix`), tak samo jak każdy inny plik z poświadczeniami.

□□ TYLKO ODCZYT

Sprawdzenie dostępności usługi TCP: net.tcp.service

```
zabbix_agent2 -t net.tcp.service[https,,443]
```

Przykład:

```
net.tcp.service[https,,443]                    [t|1]
```

□□ **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia · **zabbix_agent2** = sam proces/binarka agenta Zabbixa drugiej generacji, wywoływana też ręcznie do testów

□□ TYLKO ODCZYT

Sprawdzenie ważności certyfikatu TLS: web.certificate.get

```
zabbix_agent2 -t 'web.certificate.get[git.zielona.example,443]'
```

Parametry klucza to kolejno nazwa hosta, port (domyślnie 443) i opcjonalny adres IP, gdy nazwa nie rozwiązuje się z `mon01`. Wynik to JSON: data wygaśnięcia jest w polu `x509.not_after.timestamp` (czas uniksowy), a wynik sprawdzenia łańcucha w `result.value` (`valid` albo `invalid`, powód w `result.message`). Z tych dwóch pól korzysta oficjalny szablon „Website certificate by Zabbix agent 2”.

Przykład (skrót):

```
web.certificate.get[git.zielona.example,443]
[s|{"x509":{"version":3,...,"not_after":{"value":"...","timestamp":1822551720},...},"res
ult":{"value":"valid",...},"sha1_fingerprint":"..."}]
```

Szablon, wyzwalacze i sprawdzanie list unieważnień naszego urzędu opisuje strona Inwentarz certyfikatów i częste problemy: wygasła CRL, łańcuch, SAN, alarm w Zabbixie.

TLS = Transport Layer Security: szyfrowanie połączeń, np. w HTTPS · **IP** = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **JSON** = JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami · **CRL** = Certificate Revocation List: lista certyfikatów unieważnionych przed upływem ich terminu ważności · **SAN** = w certyfikatach: Subject Alternative Name, dodatkowe nazwy hosta w certyfikacie; w pamięciach: Storage Area Network, sieć dyskowa · **zabbix_agent2** = sam proces/binarka agenta Zabbixa drugiej generacji, wywoływana też ręcznie do testów

PLIK KONFIGURACYJNY

Wykrycie wzorca w logu przez UserParameter (w praktyce nieużywane)

```
UserParameter=custom.log.errors,tail -n 0 -F /var/log/aplikacja/app.log
```

TYLKO ODCZYT

Element log[] we frontendzie: Zabbix agent (active)

```
log[/var/log/aplikacja/app.log,"ERROR|FATAL",,,skip]
```

W praktyce wzorzec logu konfigurujemy nie jako `UserParameter`, tylko jako natywny element typu **Zabbix agent (active)** z kluczem `log[]` wprost we frontendzie.

TYLKO ODCZYT

Sprawdzenie, dlaczego element jest nieobsługiwany: zabbix_agent2 -t

```
zabbix_agent2 -t custom.procesy.count[nazwa_procesu]
```

Lokalny test tym samym kluczem, jakiego użył serwer, zwykle odtwarza dokładnie ten sam błąd.

zabbix_agent2 = sam proces/binarka agenta Zabbixa drugiej generacji, wywoływana też ręcznie do testów

Z życia wzięte

> **Z ŻYCIA WZIĘTE**
>
> O 14:30 aplikacja na `app01` zaczyna zgłaszać błędy TLS u części klientów, a nikt jeszcze nie dostał ostrzeżenia z monitoringu. Tomek sprawdza ręcznie `zabbix_agent2 -t web.certificate.get` i widzi, że certyfikat wygasł dwa dni wcześniej, bo element sprawdzający datę ważności nigdy nie został dodany, tylko `net.tcp.service` sprawdzający sam fakt odpowiedzi usługi. Dokłada brakujący element z wyzwalaczem na `not_after` i umawia się z zespołem, że każdy nowy serwis z TLS dostaje oba sprawdzenia od razu.

Element własny — gdy gotowego klucza nie ma

> **Pojęcie: element własny (UserParameter).** Definicja nowego klucza agenta, wywołująca dowolną komendę powłoki na hoście i zwracająca jej wynik jako wartość elementu. To mechanizm, przez który Zabbix monitoruje cokolwiek, do czego da się napisać komendę zwracającą liczbę albo tekst.

Monitorowanie bazy danych

Monitorowanie silnika bazodanowego (liczba połączeń, blokady, opóźnienie replikacji, wolne miejsce w tablespace) opisujemy szczegółowo na stronie Co monitorujemy w bazie danych — z perspektywy Zabbixa istotne jest tylko to, jakim kontem i jakim mechanizmem te dane pobieramy.

Do PostgreSQL i Oracle korzystamy z dedykowanych kont monitorujących o uprawnieniach ograniczonych wyłącznie do odczytu widoków stanu silnika, opisanych na stronach Konta monitorujące — PostgreSQL i Konta monitorujące — Oracle — agent2 z wtyczką `postgresql` łączy się takim kontem cyklicznie i udostępnia zestaw gotowych kluczy (`pgsql.connections`, `pgsql.replication.lag`) bez konieczności pisania własnych zapytań SQL w `UserParameter`.

Monitorowanie portu i certyfikatu

Sprawdzenie dostępności usługi TCP

> **Pojęcie: net.tcp.service.** Klucz sprawdzający, czy dana usługa na wskazanym porcie w ogóle odpowiada na poziomie protokołu (nie tylko czy port jest otwarty) — zwraca `1` albo `0`, bez żadnych dodatkowych informacji o treści odpowiedzi. Warianty obejmują `ssh`, `smtp`, `ftp`, `https`, `tcp` (goły port bez znajomości protokołu).

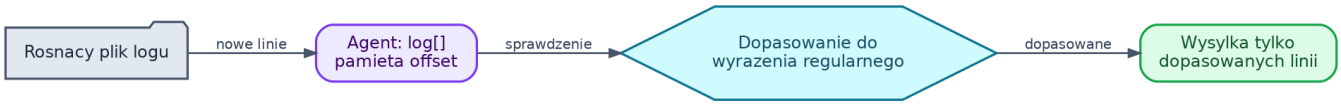
Sprawdzenie ważności certyfikatu TLS

> **Pojęcie: web.certificate.get.** Klucz zwracający w formacie JSON szczegóły certyfikatu TLS wystawionego przez usługę — wystawcę, podmiot, oraz datę ważności `not_after`. Zamiast liczyć dni ręcznie, wyzwalacz odwołuje się do tej wartości funkcją czasową i porównuje ją z bieżącym czasem, dając wcześniejsze ostrzeżenie niż samo „certyfikat wygasł" zwracane przez `net.tcp.service`.

Monitorowanie logu

Wykrycie wzorca w rosnącym pliku logu: log[]

> **Pojęcie: log[].** Element śledzący nowe linie dopisywane do wskazanego pliku i przekazujący do Zabbixa tylko te, które pasują do wyrażenia regularnego — agent pamięta pozycję ostatnio przeczytanego bajtu (offset), więc restart agenta nie powoduje ponownego przetworzenia całego pliku od początku. Element `log[]` działa wyłącznie jako sprawdzenie aktywne — sam agent decyduje, kiedy sprawdzić plik, serwer nie ma jak „zapytać o nową linię" sprawdzeniem pasywnym.



Agent pamięta offset w pliku logu, sprawdza nowe linie wobec wyrażenia regularnego i wysyła do serwera tylko dopasowane (kliknięcie otwiera pełny rozmiar)

Uważaj: rotacja logu (np. przez `logrotate`) bez odpowiedniej flagi `copytruncate` powoduje, że agent traci ślad pliku po jego przeniesieniu — na taki wypadek stosujemy wariant `logrotate2`, świadomy rotacji przez zmianę inode'u, zamiast klasycznego `log[]`.

SNMP w skrócie

> **Pojęcie: SNMP (Simple Network Management Protocol).** Protokół, którym odpytujemy urządzenia sieciowe (przełączniki, routery, UPS-y, drukarki) niemające własnego agenta Zabbixa. Element typu SNMP agent odpytuje wskazany OID (identyfikator obiektu w drzewie MIB danego urządzenia) i zwraca jego wartość — najczęściej wykorzystywany razem z regułą LLD SNMP do automatycznego wykrycia wszystkich interfejsów przełącznika i utworzenia dla każdego elementu ruchu wejściowego/wyjściowego bez ręcznego dodawania portu po porcie.

Sprawdzenie, dlaczego element jest nieobsługiwany

> **Pojęcie: element nieobsługiwany (unsupported item).** Stan elementu, w którym Zabbix nie potrafi już pobrać dla niego wartości — najczęściej z powodu błędu w kluczu, braku uprawnień, albo usunięcia monitorowanego obiektu (np. dysku wykrytego wcześniej przez LLD). Element w tym stanie przestaje zapisywać historię, ale sam w sobie nie generuje powiadomienia, chyba że zbudujemy na to osobny wyzwalacz z funkcją `nodata()`.

Powód błędu sprawdzamy bezpośrednio we frontendzie na liście elementów danego hosta — kolumna informacyjna pokazuje treść błędu zwróconą przez agenta, np. „Unsupported item key" albo „Timeout while executing a shell script".

Częste błędy

Wpisanie hasła konta monitorującego bezpośrednio w komendzie UserParameter. Komenda w `UserParameter` jest widoczna w liście procesów systemu (`ps aux`) w trakcie wykonania — hasła i tokeny przekazujemy przez plik z ograniczonymi uprawnieniami odczytywany przez skrypt, nigdy jako argument wprost.

Zbyt krótki interwał na elemencie wykonującym ciężką komendę. `UserParameter` uruchamiający polecenie trwające dłużej niż ustawiony interwał kumuluje się w kolejce procesów agenta — element odpytywany co 10 sekund, a trwający 15, prędzej czy później zacznie zgłaszać timeouty.

Monitorowanie logu bez uwzględnienia rotacji. Klasyczny `log[]` bez wariantu `copytruncate` po stronie `logrotate` traci ciągłość śledzenia pliku przy każdej rotacji, co w praktyce oznacza zero wykrytych błędów tuż po północy, jeśli rotacja jest ustawiona na ten moment.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — dobór typu elementu

Dla trzech potrzeb: „sprawdzić czy usługa SSH nasłuchuje", „policzyć liczbę procesów danej aplikacji", „wykryć błąd `OutOfMemory` w logu aplikacji w czasie rzeczywistym" — dobieramy odpowiedni mechanizm Zabbixa.

Odpowiedź

Sprawdzenie usługi SSH to gotowy klucz `net.tcp.service[ssh]`, bez potrzeby własnej komendy. Policzenie procesów aplikacji wymaga elementu własnego (`UserParameter` wywołujący `pgrep -c`), bo Zabbix nie ma gotowego klucza dla dowolnej nazwy procesu. Wykrycie wzorca w logu w czasie rzeczywistym to element typu `log[]`, działający wyłącznie jako sprawdzenie aktywne.

Pytania kontrolne

1. Dlaczego `log[]` musi być sprawdzeniem aktywnym, a nie może być pasywnym?

Odpowiedź

Bo śledzenie nowych linii wymaga, żeby agent sam pamiętał pozycję ostatnio przeczytanego bajtu i sam decydował, kiedy sprawdzić plik ponownie — sprawdzenie pasywne odpowiada tylko na pojedyncze zapytanie serwera w danej chwili, bez żadnego stanu utrzymywanego między kolejnymi zapytaniami.

Powiązane: Co monitorujemy w bazie danych · Konta monitorujące — PostgreSQL · Konta monitorujące — Oracle · Host, element danych, wyzwalacz, akcja: jak to się składa

Źródła: dokumentacja Zabbix 6.0, rozdział „Items", „UserParameters" i „SNMP checks" · zabbix.com/integrations/ssl, „Website certificate by Zabbix agent 2” (klucz `web.certificate.get[{#CERT.WEBSITE.HOSTNAME},{#CERT.WEBSITE.PORT},{#CERT.WEBSITE.IP}]`, JSONPath `$.x509.not_after.timestamp` i `$.result.value`, odczyt 26.10.2026)

Utrzymanie serwera Zabbix

Poziom: średnio

Dotyczy: Zabbix server 6.0 LTS, PostgreSQL/MySQL jako baza zapleczowa **Czas: ~25 min**

Instalacja, która działa poprawnie przez pierwszy rok, zaczyna spowalniać w drugim — prawie zawsze dlatego, że nikt nie dopilnował retencji i wielkości tabel historii.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
CPU	Central Processing Unit: procesor
LTS	Long Term Support: wersja z długim wsparciem poprawkami
RAM	Random Access Memory: pamięć operacyjna

Fiszka

TYLKO ODCZYT

Sprawdzenie liczby wierszy w tabelach historii (PostgreSQL): psql

```
sudo -u postgres psql -d zabbix -c "SELECT relname, n_live_tup FROM pg_stat_user_tables
WHERE relname LIKE 'history%' OR relname LIKE 'trends%' ORDER BY n_live_tup DESC;"
```

Przykład:

```
      relname       | n_live_tup
--------------------+------------
 history_uint       |  184201933
 history            |   92841207
 trends_uint        |    4128933
 trends             |    1802441
(4 rows)
```

□□ TYLKO ODCZYT

Podgląd rozmiaru kolejki: zabbix[queue]

```
zabbix[queue]
```

Podgląd kolejki jest dostępny we frontendzie (Raporty → Kolejka), a jej rozmiar liczbowo pobieramy elementem wewnętrznym.

□□ TYLKO ODCZYT

Sprawdzenie liczby procesów roboczych: grep

```
sudo grep -E '^Start(Pollers|Trappers|DBSyncers)' /etc/zabbix/zabbix_server.conf
```

Przykład:

```
StartPollers=10
StartTrappers=5
StartDBSyncers=4
```

□□ **grep** = global regular expression print: szukaj wzorca w tekście

□□ TYLKO ODCZYT

Wypisanie statystyk wewnętrznych serwera: zabbix_server -R diaginfo

```
sudo zabbix_server -R diaginfo
```

Wypisujemy do logu serwera bieżące statystyki wewnętrzne — wykorzystanie cache'u konfiguracji, cache'u wartości, liczbę elementów w kolejce per typ. Komenda nie zwraca wyniku bezpośrednio w terminalu, tylko dopisuje go do logu serwera.

□□ TYLKO ODCZYT

Podgląd wyniku diaginfo w logu: tail

```
sudo tail -n 40 /var/log/zabbix/zabbix_server.log
```

Przykład:

```
18204:20260922:093011.204 ====== history syncer #2 diagnostic information ======
18204:20260922:093011.204 history values processed: 1420318
18204:20260922:093011.204 values per second: 394.53
```

tail = pokaż koniec pliku, z flagą -f śledzi plik na żywo

ZMIENIA SYSTEM

Wykonanie kopii bazy przed aktualizacją: pg_dump

```
sudo -u postgres pg_dump zabbix > /var/backups/zabbix_przed_aktualizacja.sql
```

Zanim rozpoczynamy aktualizację, wykonujemy kopię bazy danych — sama aktualizacja pakietu serwera automatycznie migruje schemat bazy przy pierwszym starcie po aktualizacji, a ten proces jest jednokierunkowy i nie da się go cofnąć bez przywrócenia bazy z backupu.

ZMIENIA SYSTEM

Aktualizacja pakietów serwera i frontendu (RHEL/Rocky): dnf update

```
sudo dnf update zabbix-server-pgsql zabbix-web-pgsql zabbix-agent2
```

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **dnf** = Dandified YUM: menedżer pakietów RHEL/Rocky

ZMIENIA SYSTEM

Restart usługi i podgląd logu migracji schematu: systemctl restart

```
sudo systemctl restart zabbix-server
sudo tail -f /var/log/zabbix/zabbix_server.log
```

Po aktualizacji sprawdzamy log serwera pod kątem migracji schematu bazy przy pierwszym starcie.

Przykład:

```
 21044:20260922:100211.117 current database version (mandatory/optional):
06000000/06000000
 21044:20260922:100211.117 required mandatory version: 06040000
 21044:20260922:100211.117 starting automatic database upgrade
 21044:20260922:100234.809 database upgrade fully completed
```

Uważaj: aktualizacja o więcej niż jedną główną wersję na raz (np. z 5.0 od razu do 6.4) nie jest wspierana wprost — sprawdzamy w dokumentacji wydania, czy wymagane jest przejście przez wersje pośrednie, zanim uruchomimy proces.

systemctl = system control: sterowanie usługami systemd · **tail** = pokaż koniec pliku, z flagą -f śledzi plik na żywo

Z życia wzięte

Z ŻYCIA WZIĘTE

W poniedziałek rano baza pod Zabbixem na `pg01` zapełnia dysk szybciej niż zwykle, a Tomek dostaje o 6:40 alarm o wolnym miejscu. Sprawdza `n_live_tup` na tabelach historii i widzi, że `history_uint` urosło dziesięciokrotnie od ostatniego kwartału, odkąd ktoś podłączył nowy szablon do stu hostów bez skrócenia interwałów. Skraca retencję historii dla mniej krytycznych elementów z 90 do 30 dni, co od razu spowalnia dalszy przyrost, i zapisuje w notatkach zespołu, żeby przy każdym nowym szablonie sprawdzać wpływ na rozmiar bazy.

Housekeeping i retencja

> **Pojęcie: housekeeping.** Wbudowany proces serwera Zabbixa, cyklicznie usuwający z bazy dane starsze niż skonfigurowany okres retencji — bez niego historia rosłaby bez końca, aż do wyczerpania miejsca na dysku pod bazę danych.

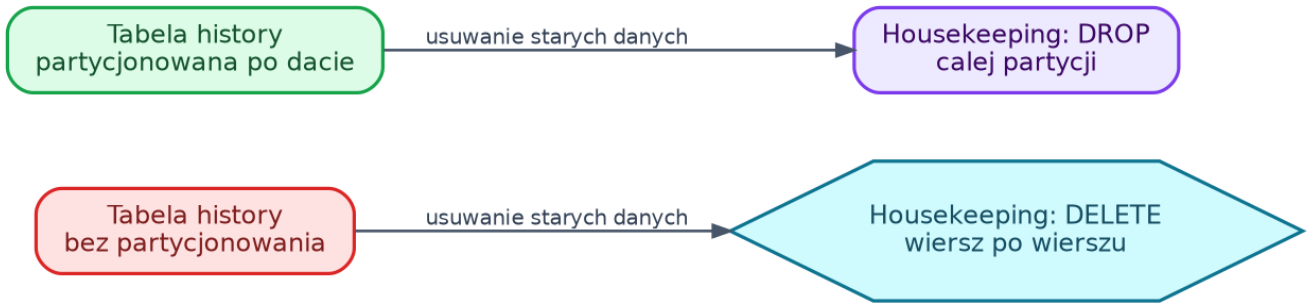
Historia kontra trendy. Retencję konfigurujemy osobno dla dwóch rodzajów danych, opisanych też przy architekturze na stronie Architektura Zabbixa: serwer, proxy, agent, frontend: **historia** to pojedyncze zebrane wartości z pełną rozdzielczością czasową, **trendy** to zagregowane dane godzinowe (min/max/avg), zajmujące ułamek miejsca historii. Typowa praktyka to krótka retencja historii (7–31 dni — wystarczająco długo, żeby zdiagnozować świeży incydent z dokładnością do pojedynczego sprawdzenia) i długa retencja trendów (kilka lat — wystarczająco, żeby porównać obciążenie z tym samym okresem rok wcześniej, już bez dokładności co do sekundy).

Retencję ustawiamy per element danych albo globalnie w konfiguracji housekeepingu — element o krytycznym znaczeniu diagnostycznym (np. log błędów aplikacji) trzymamy dłużej w pełnej historii niż standardowy element obciążenia CPU.

Uważaj: wyłączenie housekeepingu „na chwilę", żeby przyspieszyć bazę pod dużym obciążeniem, jest rozwiązaniem odwrotnym do zamierzonego — bez housekeepingu tabele historii rosną bez ograniczenia, więc obciążenie bazy z czasem tylko się pogłębia.

Partycjonowanie tabel history

> **Pojęcie: partycjonowanie tabeli.** Fizyczny podział jednej dużej tabeli na mniejsze fragmenty (partycje), zwykle po dacie, tak żeby operacja usunięcia starych danych była usunięciem całej partycji, a nie kasowaniem miliardów pojedynczych wierszy poleceniem `DELETE`. Housekeeping działający na niepartycjonowanej tabeli `history` liczącej setki milionów wierszy potrafi trwać godzinami i mocno obciążać bazę operacjami blokowania wierszy — na partycjonowanej tabeli to samo sprowadza się do `DROP` jednej partycji, operacji niemal natychmiastowej.



Housekeeping na tabeli bez partycji usuwa wiersz po wierszu poleceniem DELETE, a na tabeli partycjonowanej po dacie sprowadza się do DROP całej partycji (kliknięcie otwiera pełny rozmiar)

Przy instalacjach powyżej kilku milionów elementów danych wdrożenie natywnego partycjonowania PostgreSQL albo rozszerzenia TimescaleDB (obsługiwanego oficjalnie od Zabbixa 5.0) jest praktycznie obowiązkowe — bez niego housekeeping w pewnym momencie przestaje nadążać z usuwaniem danych szybciej, niż te dane przybywają.

Tryb konserwacji hosta

> **Pojęcie: tryb konserwacji (maintenance).** Okno czasowe zdefiniowane dla wybranych hostów albo grup hostów, w którym Zabbix albo całkowicie zawiesza zbieranie danych, albo (w wariancie „bez zbierania danych") zbiera je nadal, ale tłumi generowanie nowych zdarzeń i powiadomień. Stosujemy tryb konserwacji przed planowaną przerwą serwisową — restartem serwera, aktualizacją, migracją — żeby zaplanowana, świadoma niedostępność nie wygenerowała alarmu i nie obudziła nikogo z dyżuru.

Uważaj: tryb konserwacji ustawiony na zbyt długi okres i zapomniany po zakończeniu prac maskuje realne problemy powstałe już po zaplanowanej przerwie — dobra praktyka to zawsze węższe okno niż szacowany czas prac, z możliwością przedłużenia, zamiast szerokiego okna „na wszelki wypadek".

Kolejka — zabbix[queue]

> **Pojęcie: kolejka (queue).** Lista elementów danych oczekujących na zebranie, których termin sprawdzenia już minął, a serwer albo proxy jeszcze nie zdążył ich obsłużyć. Rosnąca kolejka jest pierwszym sygnałem, że serwer nie nadąża z bieżącym obciążeniem — więcej elementów danych albo krótsze interwały, niż faktycznie jest w stanie obsłużyć liczba skonfigurowanych procesów roboczych (`StartPollers`, `StartTrappers` i pokrewne parametry w `zabbix_server.conf`).

Diagnostyka wydajności serwera

> **Pojęcie: cache wartości (value cache).** Pamięć podręczna w RAM serwera, przechowująca ostatnio odczytane wartości historii, tak żeby ewaluacja wyzwalaczy odwołujących się do funkcji historycznych (`avg()`, `last()` na dłuższym oknie) nie wymagała każdorazowego zapytania do bazy danych. Zbyt mały `ValueCacheSize` w `zabbix_server.conf` objawia się rosnącym obciążeniem samej bazy przy dużej liczbie wyzwalaczy odwołujących się do dłuższych okresów historii.

Częste błędy

Retencja historii ustawiona globalnie i zbyt długo dla wszystkich elementów naraz. Trzymanie 90 dni pełnej historii dla elementu odpytywanego co 10 sekund mnoży rozmiar bazy

dziesięciokrotnie względem retencji 30-dniowej, bez realnej wartości diagnostycznej po pierwszym tygodniu.

Brak partycjonowania przy dużej skali. Instalacja, która wystartowała ze stu hostami i niepartycjonowaną bazą, po dwóch latach i dziesięciu tysiącach hostów ma housekeeping trwający dłużej niż odstęp między jego uruchomieniami — objawia się to permanentnie rosnącą bazą mimo skonfigurowanej krótkiej retencji.

Zapomniany tryb konserwacji. Okno konserwacji ustawione „na tydzień, żeby na pewno starczyło" i nieusunięte po zakończeniu prac maskuje realne alarmy przez cały pozostały czas jego trwania.

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — diagnoza rosnącej kolejki

Kolejka Zabbixa (`zabbix[queue]`) rośnie od kilku dni, mimo że liczba hostów się nie zmieniła. Wskazujemy dwa najbardziej prawdopodobne kierunki dalszej diagnostyki.

Odpowiedź

Pierwszy kierunek to sprawdzenie, czy nie przybyło elementów danych albo nie skrócono interwałów (np. po podłączeniu nowego szablonu do istniejących hostów) bez odpowiedniego zwiększenia liczby procesów roboczych. Drugi kierunek to sprawdzenie obciążenia samej bazy danych — rosnąca kolejka bywa objawem tego, że baza nie nadąża z zapisem historii, a nie tego, że serwerowi brakuje mocy obliczeniowej na samo zbieranie.

Powiązane: Architektura Zabbixa: serwer, proxy, agent, frontend · Host, element danych, wyzwalacz, akcja: jak to się składa · Retencja logów

Źródła: dokumentacja Zabbix 6.0, rozdział „Housekeeper", „Maintenance" i „Upgrade procedure"

Zabbix: SMART dysków na Windows przez agenta 2

Poziom: średnio **Dotyczy: Zabbix agent2 7.0 LTS, smartmontools 7.x, Windows Server** **Czas: ~30 min**

Dokładamy do agenta zainstalowanego według strony Agent, instalacja i konfiguracja monitoring zdrowia dysków SMART, ręcznie i tylko tam, gdzie ma to sens: dyski systemowe tak, dyski w puli Storage Spaces nie.

ZAŁOŻENIA SCENARIUSZA

przykład pokazujemy na hoście Hyper-V `hv01` (Windows Server 2022, węzeł klastra `hvclu01`), bo SMART ma sens tylko na serwerze fizycznym; `fs01` jest maszyną wirtualną na `hvclu01` i fizycznych dysków nie widzi. Zakładamy, że `hv01` ma dwa systemowe dyski NVMe podłączone wprost do płyty, a dyski puli `S2D-Pula` za kontrolerem w trybie HBA, jak na stronie RAID i kontrolery: sprzętowy, programowy i HBA dla S2D. Na `hv02` postępujemy tak samo.

Skróty użyte na tej stronie (9)

Skrót	Co znaczy
DOS	Disk Operating System: bardzo prosty, jednozadaniowy system, na którym stały pierwsze wersje Windows przed Windows NT
HBA	Host Bus Adapter: karta, która tylko wystawia dyski systemowi, bez składania ich w macierz; wymagana w trybie przezroczystym dla S2D
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
JSONPath	Zapis ścieżki, która wskazuje jedną wartość w dokumencie JSON, np. $.value to pole value na samym wierzchu
LLD	Low-Level Discovery: automatyczne wykrywanie w Zabbixie np. dysków i interfejsów
LTS	Long Term Support: wersja z długim wsparciem poprawkami
NVMe	Non-Volatile Memory Express: protokół komunikacji z dyskami SSD podłączonymi przez złącze PCIe, znacznie szybszy niż starsze SATA i SAS, typowy dla szybkich dysków systemowych

Skrót	Co znaczy
RAID	Redundant Array of Independent Disks: sposób łączenia kilku fizycznych dysków w jeden, widziany przez system jako pojedynczy dysk, zwykle z dodatkową odpornością na awarię jednego z nich
SMART	Self-Monitoring, Analysis and Reporting Technology: wbudowany w dysk mechanizm zbierający liczniki o jego kondycji, na przykład temperaturę i liczbę realokowanych sektorów, odczytywany programami takimi jak smartctl

Fiszka

□□ TYLKO ODCZYT

Sprawdzenie wersji smartctl: smartctl --version

```
smartctl --version
```

Wtyczka SMART agenta2 potrzebuje smartmontools w wersji przynajmniej 7.1, starsze wersje nie odczytują poprawnie części atrybutów z nowszych dysków NVMe.

Przykład:

```
smartctl 7.4 2023-08-01 r5530 [x86_64-w64-mingw32-2022]
Copyright (C) 2002-23, Bruce Allen, Christian Franke, www.smartmontools.org
```

□□ **SMART** = Self-Monitoring, Analysis and Reporting Technology: wbudowany w dysk mechanizm zbierający liczniki o jego kondycji, na przykład temperaturę i liczbę realokowanych sektorów, odczytywany programami takimi jak smartctl · **NVMe** = Non-Volatile Memory Express: protokół komunikacji z dyskami SSD podłączonymi przez złącze PCIe, znacznie szybszy niż starsze SATA i SAS, typowy dla szybkich dysków systemowych

□□ TYLKO ODCZYT

Wylistowanie widocznych dysków: smartctl --scan-open

```
smartctl --scan-open
```

Wypisuje ścieżki dysków, które `smartctl` w ogóle potrafi odpytać z tego hosta. Te same ścieżki wpisujemy potem w kluczu ręcznej pozycji.

Przykład:

```
/dev/sda -d nvme # /dev/sda, NVMe device
/dev/sdb -d nvme # /dev/sdb, NVMe device
```

PLIK KONFIGURACYJNY

Wskazanie ścieżki do smartctl: Plugins.Smart.Path

```
Plugins.Smart.Path=C:\smartmontools\bin\smartctl.exe
```

> (gdzie `C:\smartmontools\bin\smartctl.exe` to pełna ścieżka do pliku `smartctl.exe`, u nas celowo w katalogu bez spacji w nazwie)

Pojęcie: zmienna PATH. Lista katalogów, w których system szuka programu, gdy wpisujemy samą jego nazwę bez pełnej ścieżki. Instalator smartmontools dopisuje swój katalog do PATH konta, którym instalowaliśmy program, ale usługa agenta na Windows działa na zupełnie innym koncie, Local System, z własną, osobną listą katalogów. Bez jawnego `Plugins.Smart.Path` agent nie znajdzie `smartctl.exe` wcale, mimo że ręczne uruchomienie tego samego polecenia w naszej własnej sesji działa bez problemu.

Jeśli smartmontools stoi już w domyślnym `C:\Program Files\smartmontools`, spacja w nazwie katalogu potrafi psuć parsowanie ścieżki w pliku konfiguracyjnym. Prościej zainstalować program w katalogu bez spacji, jak w przykładzie wyżej, niż walczyć z cudzysłowami albo skróconą nazwą DOS 8.3.

DOS = Disk Operating System: bardzo prosty, jednozadaniowy system, na którym stały pierwsze wersje Windows przed Windows NT

ZMIENIA SYSTEM

Restart usługi agenta po zmianie konfiguracji: Restart-Service

```
Restart-Service -Name "Zabbix Agent 2"
```

Agent, tak samo jak na Linuksie, nie przeładowuje konfiguracji sam z siebie po edycji pliku.

◫◫ KLIKAMY W INTERFEJSIE

Wyłączenie automatycznego wykrywania dysków SMART na hoście hv01

```
Data collection → Hosts → hv01 → Discovery rules → SMART disk discovery → Status: Disabled
```

Wyłączamy regułę na poziomie samego hosta `hv01`, nie całego szablonu, żeby inne hosty z prostszym magazynem korzystały z automatycznego wykrywania normalnie.

◫◫ **SMART** = Self-Monitoring, Analysis and Reporting Technology: wbudowany w dysk mechanizm zbierający liczniki o jego kondycji, na przykład temperaturę i liczbę realokowanych sektorów, odczytywany programami takimi jak smartctl

◫◫ ZMIENIA SYSTEM

Dodanie ręcznej pozycji dla dysku systemowego: smart.disk.get

```
Data collection → Hosts → hv01 → Items → Create item
Nazwa: SMART, dysk systemowy NVMe0
Typ: Zabbix agent
Klucz: smart.disk.get["/dev/sda", ""]
Typ informacji: Tekst
Interwał aktualizacji: 1h
```

> **Pojęcie: RAID (Redundant Array of Independent Disks).** Sposób łączenia kilku fizycznych dysków tak, żeby system widział je jako jeden, z nadmiarowością albo bez niej, obsługiwany zwykle przez osobny kontroler sprzętowy. Drugi parametr klucza to właśnie typ takiego kontrolera, przez jaki dysk jest widoczny. Pusty łańcuch `""` oznacza zwykły, pojedynczy dysk bez kontrolera RAID pomiędzy nim a systemem, dokładnie nasz przypadek dla dysku NVMe podłączonego wprost do płyty głównej.

Powtarzamy tę samą pozycję dla drugiego dysku systemowego, z kluczem `smart.disk.get["/dev/sdb", ""]`.

◫◫ **RAID** = Redundant Array of Independent Disks: sposób łączenia kilku fizycznych dysków w jeden, widziany przez system jako pojedynczy dysk, zwykle z dodatkową odpornością na awarię jednego z nich · **NVMe** = Non-Volatile Memory Express: protokół komunikacji z dyskami SSD podłączonymi przez złącze PCIe, znacznie szybszy niż starsze SATA i SAS, typowy dla szybkich dysków systemowych

TYLKO ODCZYT

Test z serwera: zabbix_get

```
zabbix_get -s AdresHosta -k 'smart.disk.get["/dev/sda", ""]'
```

Sprawdzamy z poziomu serwera albo proxy, że agent w ogóle potrafi zwrócić dane SMART z konkretnego dysku, zanim zaczniemy budować na tym elementy zależne i wyzwalacze.

Przykład:

```
{"model_name":"Dysk NVMe
systemowy","serial_number":"S3X1NX0K123456","smart_status":{"passed":true},"temperature"
:{"current":38},"power_on_time":{"hours":4102}}
```

SMART = Self-Monitoring, Analysis and Reporting Technology: wbudowany w dysk mechanizm zbierający liczniki o jego kondycji, na przykład temperaturę i liczbę realokowanych sektorów, odczytywany programami takimi jak smartctl · **zabbix_get** = ręczne odpytanie agenta Zabbixa

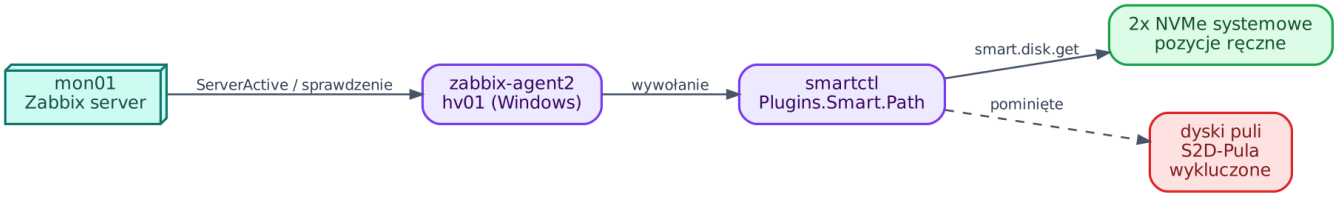
Z życia wzięte

Z ŻYCIA WZIĘTE

O 11:15 wykrywanie SMART na `hv01` zaczyna zgłaszać dziwne, niespójne odczyty dla dysków w puli `S2D-Pula`, a jeden z administratorów dyżurnych chce już budować na tym wyzwalacz. Tomek sprawdza `smartctl --scan-open` i przypomina zespołowi, że dyski za sterownikiem puli nigdy nie były wiarygodnym źródłem dla `smartctl`, bo kontroler nie oddaje ich jako pojedynczych, zwykłych nośników. Wyłącza regułę LLD tylko na `hv01` i zostawia zdrowie puli pod obserwacją `Get-PhysicalDisk`, tak jak ustalono przy pierwszym wdrożeniu tej strony.

Jak to działa

Pojęcie: wtyczka SMART agenta2 (Smart plugin). Rozszerzenie wbudowane w `zabbix-agent2`, które samo nie czyta danych z dysku, tylko za każdym razem uruchamia zewnętrzny program `smartctl` z pakietu smartmontools i parsuje jego wynik do formatu JSON. Bez zainstalowanego `smartctl` na hoście wtyczka nie ma czego wywołać, więc pierwszym krokiem zawsze jest sprawdzenie samego `smartctl`, nie konfiguracja Zabbixa.



Schemat drogi danych SMART: agent 2 wywołuje smartctl, który czyta dyski NVMe wprost, a dyski puli Storage Spaces pomija (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: wykrywanie niskopoziomowe dysków (reguła smart.disk.discovery).** Reguła LLD szablonu „SMART by Zabbix agent 2", która sama odpytuje `smartctl --scan-open` i dla każdego znalezionego dysku tworzy komplet elementów danych (temperatura, liczba godzin pracy, poszczególne atrybuty SMART) razem z wyzwalaczami. Działa dobrze dla dysków podłączonych wprost, bo dla nich `smartctl` widzi jeden fizyczny nośnik i jeden spójny zestaw atrybutów.

Na `hv01` mamy dwa rodzaje dysków: dwa systemowe NVMe podłączone wprost do kontrolera i dyski w puli Storage Spaces Direct `S2D-Pula`, obsługiwane przez sterownik puli, nie wprost przez kontroler dyskowy widziany przez `smartctl`. Zdrowie dysków puli i tak śledzimy przez `Get-PhysicalDisk` z poziomu samego Storage Spaces Direct, łącznie ze stanami takimi jak `Predictive Failure`, opisanymi na stronie S2D: wymiana dysku bez przestoju, gdy pula jest prawie pełna. Zamiast naprawiać albo weryfikować wykrywanie SMART dla przypadku, który i tak monitorujemy innym, bardziej wiarygodnym mechanizmem, wyłączamy regułę LLD na tym hoście i dokładamy tylko dwie ręczne pozycje dla dysków systemowych.

Wyzwalacze

Szablon „SMART by Zabbix agent 2" ma gotowe wyzwalacze reagujące na bity w polu stanu zwracanym przez `smartctl`, między innymi „Some Attributes have been <= threshold", „Device open failed" i „DISK FAILING". Dla naszych dwóch ręcznych pozycji budujemy własne, prostsze wyzwalacze na dwóch danych, które realnie nas interesują.

Wyzwalacz na temperaturę, korzystający z tych samych makr co szablon:

```
avg(/hv01/smart.disk.get["/dev/sda", ""],5m) > {$SMART.TEMPERATURE.MAX.WARN}
```

Domyślne progi w szablonie to 50 stopni dla ostrzeżenia i 65 dla stanu krytycznego. Dopasowujemy je do konkretnego modelu dysku, jeśli producent zaleca inne wartości.

Wyzwalacz na stan zdrowia dysku, wyciągnięty z tego samego surowego JSON-a przez element zależny:

```
last(/hv01/smart.disk.status.nvme0) = 0
```

Element `smart.disk.status.nvme0` jest typu „Element zależny", z elementem nadrzędnym `smart.disk.get["/dev/sda", ""]` i przetwarzaniem wstępnym typu JSONPath, czyli ścieżką wskazującą jedno konkretne pole wewnątrz większego dokumentu JSON, tutaj `$.smart_status.passed`. Dzięki temu nie odpytujemy `smartctl` drugi raz tylko po to, żeby wyciągnąć jedno pole z danych, które i tak już pobraliśmy.

Częste błędy

Objaw	Przyczyna	Co robimy
Element `smart.disk.get` nieobsługiwany, błąd o nieznalezieniu programu	usługa agenta na koncie Local System nie widzi zmiennej PATH ustawionej przy instalacji smartmontools	ustawiamy `Plugins.Smart.Path` na pełną ścieżkę do `smartctl.exe` i restartujemy usługę agenta
`Plugins.Smart.Path` ustawiony, a mimo to błąd parsowania ścieżki	spacja w nazwie katalogu (`C:\Program Files\smartmontools`) źle się parsuje w pliku konfiguracyjnym	instalujemy smartmontools w katalogu bez spacji, na przykład `C:\smartmontools`, albo używamy skróconej nazwy DOS 8.3
Dyski puli Storage Spaces nie pojawiają się w wykrywaniu albo pojawiają się z niewiarygodnymi danymi	dyski za sterownikiem puli nie są dla `smartctl` zwykłym, pojedynczym nośnikiem	nie próbujemy ich monitorować przez SMART, zdrowie puli śledzimy przez `Get-PhysicalDisk` i mechanizmy Storage Spaces
`zabbix_get` z serwera zwraca błąd, mimo że `smartctl` uruchomiony ręcznie na hoście działa	ręczny test w sesji administratora ma inny PATH niż usługa agenta na koncie Local System	testujemy zawsze przez sam element Zabbixa (`zabbix_get` albo podgląd ostatnich danych), nie przez ręczne uruchomienie `smartctl` we własnej sesji

□□ Ćwiczenia

1. Host ma jeden dysk NVMe systemowy i trzy dyski w puli Storage Spaces. Ile ręcznych pozycji SMART dodajemy według zasady z tej strony?

Odpowiedź

Jedną, tylko dla dysku systemowego NVMe. Dyski puli Storage Spaces wykluczamy z monitoringu SMART niezależnie od tego, ile ich jest, bo ich zdrowie śledzimy przez Storage

Spaces, nie przez `smartctl`.

1. `smart.disk.get["/dev/sda", ""]` zwraca dane poprawnie po ręcznym uruchomieniu `smartctl` w sesji administratora, ale ten sam element w Zabbixie jest nieobsługiwany. Co sprawdzamy najpierw?

Odpowiedź

Czy `Plugins.Smart.Path` w konfiguracji agenta wskazuje pełną, poprawną ścieżkę do `smartctl.exe`. Usługa agenta działa na koncie Local System, które ma inny PATH niż nasza sesja administratora, więc samo działające ręczne uruchomienie niczego jeszcze nie gwarantuje.

Powiązane: Agent, instalacja i konfiguracja · Monitorowanie typowych usług · S2D: wymiana dysku bez przestoju, gdy pula jest prawie pełna

Źródła: S.M.A.R.T. monitoring and integration with Zabbix · szablon SMART by Zabbix agent 2, kod źródłowy · ZBX-23750: Plugins.Smart.Path wymagany na Windows · dyskusja na forum Zabbixa o SMART na Windows