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).

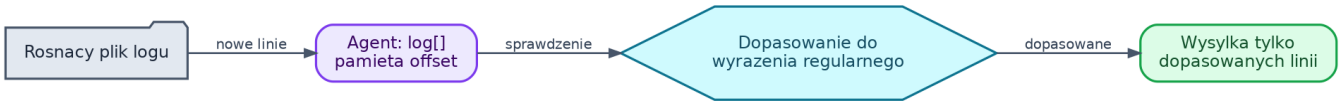
Sprawdzenie ważności certyfikatu TLS

> **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)

Revision #36
Created 2026-09-25 07:40:33 UTC by Tomasz Konieczny
Updated 2026-10-26 14:09:49 UTC by Tomasz Konieczny