Architektura i agent

Serwer, proxy, agent oraz pojęcia modelu danych.

- Architektura Zabbixa: serwer, proxy, agent, frontend
- Agent Zabbixa: instalacja i konfiguracja
- Host, element danych, wyzwalacz, akcja: jak to się składa

Architektura Zabbixa: serwer, proxy, agent, frontend

Poziom: podstawy **Dotyczy: Zabbix 6.0 LTS / 6.4** **Czas: ~20 min**

Zanim zaczniemy dodawać hosty i szablony, sprawdzamy z jakich elementów Zabbix w ogóle się składa i którędy faktycznie płyną dane od monitorowanej maszyny do wykresu na pulpicie.

Skróty użyte na tej stronie (7)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
CPU	Central Processing Unit: procesor
DMZ	Demilitarized Zone: osobna sieć za zaporą dla usług dostępnych z internetu, odgrodzona od sieci wewnętrznej; w FortiGate 100F tak jest opisany jeden z portów
LTS	Long Term Support: wersja z długim wsparciem poprawkami
PHP	PHP: Hypertext Preprocessor, język programowania, w którym napisany jest frontend Zabbixa i wiele innych aplikacji webowych
SNMP	Simple Network Management Protocol: protokół odczytu stanu urządzeń sieciowych
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia

Z życia wzięte

Z ŻYCIA WZIĘTE

O 7:50 Tomek dostaje powiadomienie, że host `app01` jest niedostępny w Zabbixie, chociaż aplikacja na nim działa bez żadnych skarg od użytkowników. Sprawdza kierunek połączenia w `zabbix_agentd.conf` i widzi, że ktoś dzień wcześniej dopisał `ServerActive`, ale zapomniał

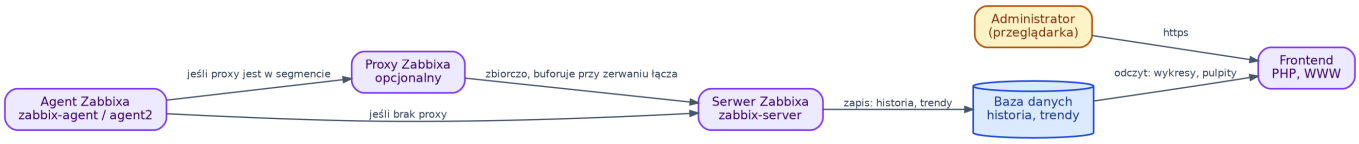
otworzyć na firewallu portu 10051 w stronę serwera. Po dodaniu reguły agent od razu zaczyna wysyłać zaległe dane, a status hosta wraca do zielonego bez żadnej zmiany po stronie samej aplikacji.

Czym jest Zabbix

Pojęcie: Zabbix. System monitoringu typu open source, zbierający metryki (dostępność usługi, zużycie zasobów, liczniki aplikacyjne) z monitorowanych hostów, oceniający je względem zdefiniowanych progów i wyzwalający akcje — powiadomienie, zgłoszenie, komendę zdalną — gdy próg zostanie przekroczony. W odróżnieniu od narzędzi typu „sprawdź czy port odpowiada" Zabbix trzyma pełną historię zebranych wartości, więc odpowiada nie tylko na pytanie „czy działa teraz", ale też „jak to wyglądało tydzień temu".

Cztery elementy i jak przepływają dane

Zanim wejdziemy w konfigurację, rozkładamy Zabbixa na cztery role — każda odpowiada za inny fragment drogi, jaką pokonuje pojedyncza zebrana wartość.



Wartość płynie od agenta albo proxy do serwera, dalej do bazy danych, a frontend czyta ją dopiero z bazy (kliknięcie otwiera pełny rozmiar)

Pojęcie: serwer Zabbixa (zabbix-server). Centralny proces, który przechowuje konfigurację (hosty, szablony, wyzwalacze), planuje zbieranie danych, ocenia wyzwalacze i uruchamia akcje. Sam serwer niczego nie wystawia użytkownikowi — to proces działający w tle, komunikujący się z bazą danych i z agentami albo proxy.

> **Pojęcie: baza danych Zabbixa.** Miejsce, w którym serwer trzyma zarówno konfigurację, jak i zebrane dane — historię pojedynczych wartości i trendy, czyli zagregowane godzinowe minimum/maksimum/średnią. To ta baza rośnie najszybciej ze wszystkich elementów instalacji, temat rozwijamy na stronie Utrzymanie serwera Zabbix.

> **Pojęcie: frontend Zabbixa.** Aplikacja webowa (PHP), przez którą konfigurujemy hosty i szablony oraz przeglądamy wykresy i pulpity. Frontend czyta i zapisuje bezpośrednio do tej samej bazy danych co serwer — nie rozmawia z serwerem przez żadne własne API w trakcie zwykłej pracy, poza kilkoma wyjątkami jak ręczne uruchomienie akcji.

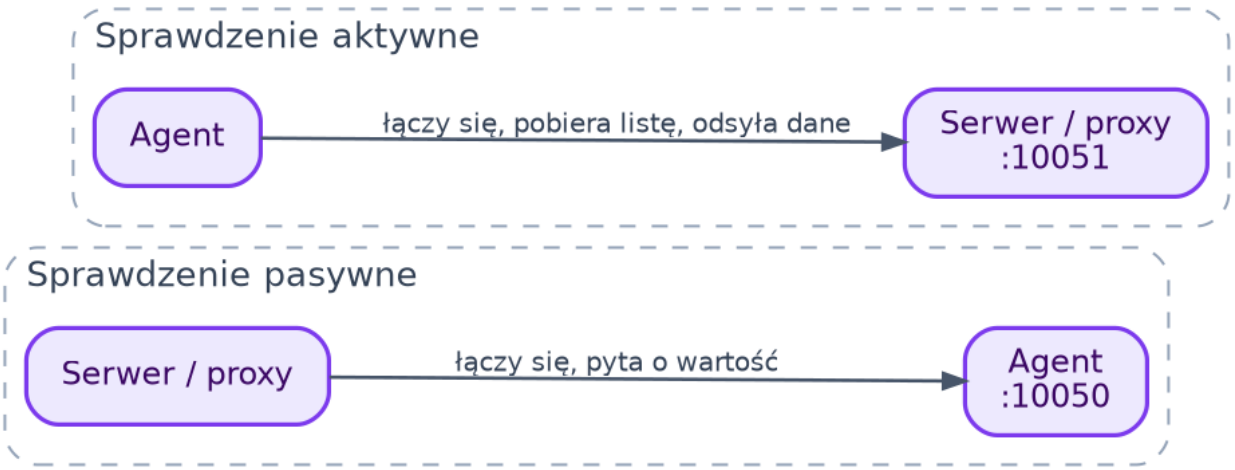
> **Pojęcie: agent Zabbixa (zabbix-agent / zabbix-agent2).** Lekki proces instalowany na monitorowanym hoście, który udostępnia lokalne metryki (obciążenie CPU, zajętość dysku, status usługi systemd) na żądanie serwera albo proxy. Instalację i konfigurację agenta opisujemy osobno na stronie Agent — instalacja i konfiguracja.

> **Pojęcie: proxy Zabbixa (zabbix-proxy).** Opcjonalny pośrednik między agentami a serwerem głównym, zbierający dane lokalnie i przesyłający je do serwera zbiorczo. Proxy nie ma własnego frontendu ani osobnej konfiguracji hostów — cała konfiguracja proxy jest kopią fragmentu konfiguracji serwera głównego, pobieraną automatycznie.

Droga pojedynczej wartości wygląda więc tak: agent (albo urządzenie SNMP, albo sam serwer wykonujący sprawdzenie zewnętrzne) → proxy, jeśli jest w danym segmencie zdefiniowane → serwer → baza danych → frontend, który dopiero z tej bazy rysuje wykres. Żaden z tych elementów nie omija bazy danych poza samym momentem zbierania — nawet frontend nie pyta serwera „jaka jest aktualna wartość", tylko czyta to, co serwer już zapisał.

Agent pasywny kontra agent aktywny

To jest rozróżnienie, które trzeba mieć jasne, zanim zabierzemy się za konfigurację `zabbix_agentd.conf` — bo od niego zależy, kto się do kogo łączy, i to determinuje wymagania na firewallu w obie strony.



Przy sprawdzeniu pasywnym serwer albo proxy łączy się do agenta, przy aktywnym to agent łączy się do serwera albo proxy (kliknięcie otwiera pełny rozmiar)

Pojęcie: sprawdzenie pasywne (passive check). Serwer albo proxy sam, w zaplanowanym momencie, łączy się do agenta i pyta o wartość konkretnego klucza. To agent nasłuchuje, a serwer inicjuje połączenie — model „serwer pyta, agent odpowiada".

Pojęcie: sprawdzenie aktywne (active check). Agent sam, cyklicznie, łączy się do serwera albo proxy, pobiera od niego listę elementów, jakie ma dla tego hosta zbierać, i sam z własnej inicjatywy odsyła zebrane wartości. To odwraca kierunek połączenia — serwer nie musi mieć żadnej trasy sieciowej do agenta, wystarczy że agent ma trasę do serwera.

Cecha	Pasywne	Aktywne
Kto inicjuje połączenie	serwer/proxy → agent	agent → serwer/proxy
Kierunek ruchu na firewallu	do hosta monitorowanego	z hosta monitorowanego
Parametr w `zabbix_agentd.conf`	`Server`	`ServerActive`
Typowe zastosowanie	hosty w tej samej sieci co serwer	hosty za NAT, laptopy, segmenty bez dostępu przychodzącego
Obciążenie agenta	niskie, odpowiada na żądanie	agent sam utrzymuje harmonogram i kolejkę

W praktyce w jednej instalacji obie metody współistnieją na tym samym hoście — nic nie stoi na przeszkodzie, żeby część elementów szablonu była pasywna, a inne (typowo `zabbix_agent.hostname`, `log[]`, elementy niskopoziomowego wykrywania) aktywne, bo agent i tak musi się zgłosić do serwera po listę, żeby wiedzieć co monitorować aktywnie.

Porty 10050 i 10051

Port 10050 to port, na którym nasłuchuje **agent** — zarówno przy sprawdzeniach pasywnych (serwer łączy się na ten port, żeby zapytać o wartość), jak i jako port, spod którego agent sam wychodzi przy sprawdzeniach aktywnych.

Port 10051 to port, na którym nasłuchuje **serwer i proxy** — agent przy aktywnym sprawdzeniu łączy się na ten port, żeby odebrać listę elementów i odesłać zebrane dane. Na ten sam port przychodzi też ruch z proxy do serwera nadrzędnego.

Uważaj: pomylenie tych dwóch portów jest najczęstszą przyczyną „host niedostępny" przy pierwszym podłączaniu nowego serwera — reguła firewalla otwarta w złym kierunku (np. przepuszczony ruch wychodzący 10050 zamiast przychodzącego, albo odwrotnie) daje dokładnie taki sam objaw jak brak reguły w ogóle.

Po co stawiamy proxy

Proxy nie jest wymagane w małej, płaskiej sieci — dodajemy je świadomie, z jednego z dwóch powodów.

Odciążenie serwera głównego. Odpytywanie tysięcy elementów danych z setek hostów generuje ruch sieciowy i obciążenie procesora na samym akcie zbierania (parsowanie odpowiedzi SNMP, nawiązywanie połączeń TCP). Proxy przejmuje tę pracę lokalnie w swoim segmencie i do serwera głównego wysyła już tylko zebrane, gotowe wartości w mniejszej liczbie większych paczek.

Monitorowanie segmentu bez bezpośredniej trasy do serwera. Oddział, sieć DMZ albo środowisko chmurowe z ograniczonym ruchem wychodzącym — zamiast otwierać firewalla dla każdego hosta osobno do centralnego serwera, otwieramy jedną regułę dla proxy, a reszta hostów rozmawia wyłącznie z lokalnym proxy.

Dodatkowa własność proxy, która bywa równie ważna jak odciążenie: proxy buforuje dane lokalnie, jeśli traci łączność z serwerem głównym. Krótka awaria łącza między lokalizacjami nie oznacza dziury w historii — proxy dosyła zaległe dane, gdy połączenie wraca.

Baza danych Zabbixa i jej przyrost

Baza danych trzyma dwa rodzaje danych pomiarowych o zupełnie różnym tempie wzrostu, co rozwijamy szerzej przy retencji na stronie Utrzymanie serwera Zabbix: **historię** (`history*`) — każdą pojedynczą zebraną wartość z osobnym znacznikiem czasu, oraz **trendy** (`trends*`) — zagregowane dane godzinowe (min/max/avg), trzymane znacznie dłużej, bo zajmują ułamek

miejsca historii.

Rozmiar tabel historii rośnie wprost proporcjonalnie do trzech rzeczy naraz: liczby elementów danych, częstości ich odpytywania (interwału) i długości okresu retencji. Podwojenie liczby monitorowanych hostów albo skrócenie interwału o połowę podwaja przyrost bazy — to jest pierwsza rzecz, którą sprawdzamy, gdy dysk pod bazę Zabbixa zaczyna się kończyć szybciej niż zakładano.

Zabbix a monitoring aplikacyjny wyższego poziomu

Zabbix odpowiada dobrze na pytania o dostępność, zasoby i proste metryki aplikacyjne wystawiane przez `UserParameter` (temat rozwijamy na stronie Monitorowanie typowych usług), ale nie jest narzędziem do śledzenia pojedynczego żądania przechodzącego przez wiele mikroserwisów naraz — do tego służą dedykowane systemy śledzenia rozproszonego (distributed tracing), poza zakresem tej strony. Rozróżnienie to jest praktyczne przy planowaniu monitoringu nowej aplikacji: pytanie „czy usługa odpowiada i ile zużywa zasobów” trafia do Zabbixa, pytanie „dlaczego akurat to jedno żądanie trwało trzy sekundy, przechodząc przez pięć usług” wymaga innego narzędzia.

Częste błędy

Traktowanie proxy jako serwera zapasowego. Proxy nie ma własnej konfiguracji hostów ani szablonów niezależnej od serwera głównego — awaria serwera głównego zatrzymuje ewaluację wyzwalaczy i wysyłkę powiadomień w całej instalacji, proxy tylko buforuje surowe dane do czasu, aż serwer wróci.

Otwarcie firewalla tylko w jedną stronę dla sprawdzeń aktywnych. Sprawdzenie aktywne wymaga ruchu wychodzącego z hosta monitorowanego na port 10051 serwera/proxy — samo otwarcie portu 10050 na hoście monitorowanym (jak przy pasywnym) nic tu nie da.

Mylenie „agent nie odpowiada" z „host niedostępny w Zabbiksie". To drugie jest stanem hosta ustalanym przez serwer na podstawie nieudanych prób połączenia — objaw bywa identyczny zarówno przy złej regule firewalla, złym adresie w konfiguracji hosta, jak i przy zatrzymanym procesie agenta; diagnostykę rozkładamy na czynniki pierwsze na stronie Agent — instalacja i konfiguracja.

□□ Ćwiczenia

> Poniższe ćwiczenia wykonujemy na dowolnym hoście testowym z zainstalowanym agentem Zabbixa.

Ćwiczenie 1 — rozpoznanie kierunku połączenia

Mając plik `zabbix_agentd.conf` z wypełnionym tylko parametrem `Server=10.20.10.51` (bez `ServerActive`), określamy: czy to sprawdzenie pasywne czy aktywne, i w którą stronę musi być otwarty firewall.

Odpowiedź

To sprawdzenie pasywne — parametr `Server` ogranicza, z jakich adresów agent przyjmie połączenie przychodzące na port 10050. Firewall na hoście monitorowanym musi przepuszczać ruch przychodzący z adresu serwera na port 10050; sam host monitorowany nie musi mieć żadnej reguły wychodzącej dla Zabbixa.

Pytania kontrolne

1. Dlaczego host za NAT-em bez publicznego adresu prawie zawsze wymaga sprawdzeń aktywnych, a nie pasywnych?

Odpowiedź

Bo sprawdzenie pasywne wymaga, żeby serwer mógł zainicjować połączenie do agenta — bez publicznego adresu i przekierowania portu serwer fizycznie nie ma jak się połączyć. Sprawdzenie aktywne odwraca kierunek: to agent łączy się na zewnątrz do serwera, co działa zza NAT-u bez żadnej dodatkowej konfiguracji sieciowej po stronie hosta monitorowanego.

2. Co się stanie z danymi zbieranymi przez proxy, jeśli łączność między proxy a serwerem głównym zerwie się na godzinę?

Odpowiedź

Proxy nie przestaje zbierać danych od swoich hostów — buforuje je lokalnie we własnej bazie i dosyła zaległość do serwera głównego, gdy połączenie wróci. Wyzwalacze zależne od tych danych zostają jednak ocenione z opóźnieniem, dopiero po dotarciu zaległych wartości do serwera.

3. Dlaczego frontend Zabbixa przestaje pokazywać nowe dane, jeśli padnie sam proces serwera, mimo że baza danych i frontend nadal działają?

Odpowiedź

Frontend czyta wyłącznie to, co już leży w bazie danych — to serwer jest jedynym elementem odpowiedzialnym za zbieranie nowych wartości z agentów i zapisywanie ich do tej bazy. Bez działającego serwera baza danych po prostu przestaje otrzymywać nowe wpisy, więc frontend wciąż poprawnie wyświetla historię sprzed awarii, ale nic nowszego.

Powiązane: Agent — instalacja i konfiguracja · Host, element danych, wyzwalacz, akcja: jak to się składa · Utrzymanie serwera Zabbix

Źródła: dokumentacja Zabbix 6.0, rozdział „Concepts" i „Proxies" · `man zabbix_agentd`

Agent Zabbixa: instalacja i konfiguracja

□□ Poziom: podstawy **□□ Dotyczy: Zabbix agent2 6.0 LTS, RHEL/Rocky 9, Debian/Ubuntu**

□ Czas: ~20 min

Zanim dodajemy nowy host do monitoringu, instalujemy i konfigurujemy na nim agenta, a dopiero potem sprawdzamy z serwera, czy odpowiada.

□□ Skróty użyte na tej stronie (4)

Skrót	Co znaczy
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
LTS	Long Term Support: wersja z długim wsparciem poprawkami
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SELinux	Security-Enhanced Linux: dodatkowa warstwa uprawnień w jądrze RHEL

Fiszka

□□ TYLKO ODCZYT

Dodanie repozytorium Zabbixa: RHEL/Rocky

```
sudo rpm -Uvh https://repo.zabbix.com/zabbix/6.0/rhel/9/x86_64/zabbix-release-6.0-4.el9.noarch.rpm
```

Dodajemy oficjalne repozytorium Zabbixa, bo wersja w standardowych repozytoriach dystrybucji jest zwykle znacznie starsza niż aktualna gałąź LTS.

□□ **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **LTS** = Long Term Support: wersja z długim wsparciem poprawkami · **rpm** = RPM Package Manager: format i narzędzie paczek RHEL

□□ ZMIENIA SYSTEM

Instalacja agenta z repozytorium: RHEL/Rocky

```
sudo dnf install -y zabbix-agent2 zabbix-agent2-plugin-postgresql
```

Instalujemy sam agent oraz — opcjonalnie — pakiet wtyczki, jeśli host ma monitorowaną bazę PostgreSQL kontem monitorującym, temat rozwijamy na stronie Monitorowanie typowych usług.

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

□□ ZMIENIA SYSTEM

Dodanie repozytorium i instalacja agenta: Debian/Ubuntu

```
wget https://repo.zabbix.com/zabbix/6.0/ubuntu/pool/main/z/zabbix-release/zabbix-release_6.0-4+ubuntu22.04_all.deb
sudo dpkg -i zabbix-release_6.0-4+ubuntu22.04_all.deb
sudo apt update
sudo apt install -y zabbix-agent2
```

Sekwencja ma sens tylko wklejona w całości — pobranie pakietu z definicją repozytorium, jego instalacja, odświeżenie listy pakietów, a dopiero potem instalacja właściwego agenta.

□□ **wget** = World Wide Web get: pobieranie plików · **dpkg** = Debian package: niskopoziomowe narzędzie instalujące paczki .deb w Debianie i Ubuntu · **apt** = Advanced Package Tool: menedżer pakietów Debiana i Ubuntu

□□ PLIK KONFIGURACYJNY

Podstawowa konfiguracja: zabbix_agentd.conf

```
Server=10.20.10.51
ServerActive=10.20.10.51
Hostname=NazwaHosta
HostMetadata=linux
Include=/etc/zabbix/zabbix_agent2.d/*.conf
```

(gdzie `10.20.10.51` to adres serwera Zabbixa albo lokalnego proxy, a `NazwaHosta` musi dokładnie odpowiadać nazwie hosta wpisanej w konfiguracji Zabbixa — wielkość liter ma znaczenie!)

Pojęcie: Server. Lista adresów, z których agent przyjmuje połączenia przychodzące przy sprawdzeniach pasywnych — jeśli połączenie przyjdzie z adresu spoza tej listy, agent je odrzuci niezależnie od tego, czy klucz istnieje.

Pojęcie: ServerActive. Adres (albo lista adresów, rozdzielona przecinkami), do którego agent sam się łączy przy sprawdzeniach aktywnych, żeby pobrać listę elementów do zbierania i odesłać wyniki.

Pojęcie: Hostname. Nazwa, pod jaką agent przedstawia się serwerowi przy sprawdzeniach aktywnych — musi być identyczna z polem „Nazwa hosta widoczna" (visible name jest inną, kosmetyczną wartością) w konfiguracji hosta we frontendzie, inaczej serwer nie skojarzy przychodzących danych z żadnym hostem.

Pojęcie: HostMetadata. Dowolny tekst opisujący hosta, wykorzystywany przy rejestracji automatycznej do dopasowania reguły — najczęściej wpisujemy tu rolę hosta albo system operacyjny, żeby reguła rejestracji mogła przypisać go do właściwej grupy i szablonu bez ręcznej interwencji.

Pojęcie: Include. Katalog albo pojedynczy plik dociągany do głównej konfiguracji — trzymamy tu osobne pliki na wtyczki i `UserParameter`, żeby nie edytować głównego `zabbix_agent2.conf` przy każdej drobnej zmianie.

□□ TYLKO ODCZYT

Rejestracja automatyczna hosta: reguła po stronie serwera

Rejestracja automatyczna (autoregistration) nie jest ustawieniem agenta — to reguła zdefiniowana we frontendzie serwera, dopasowująca nowe, nieznane jeszcze hosty po wartości `HostMetadata` albo po adresie IP i automatycznie tworząca dla nich wpis hosta z przypisaną grupą i szablonem. Agent musi mieć jedynie poprawnie ustawiony `ServerActive` i unikalny `Hostname` — resztą zajmuje się reguła po stronie serwera, więc nowy host trafia do monitoringu bez ręcznego klikania we frontendzie.

□□ **IP** = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)

□□ ZMIENIA SYSTEM

Uruchomienie i włączenie w autostarcie: systemd

```
sudo systemctl enable --now zabbix-agent2
```

Włączamy usługę na start systemu i uruchamiamy ją od razu, w jednej komendzie.

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

□□ TYLKO ODCZYT

Test z serwera: zabbix_get

```
zabbix_get -s AdresHosta -k agent.ping
```

Wykonujemy z serwera albo proxy zapytanie identyczne z tym, jakie realnie wysyła Zabbix przy sprawdzeniu pasywnym — to jest pierwsza komenda diagnostyczna, gdy host we frontendzie pokazuje status niedostępny.

Przykład:

```
1
```

> **Pojęcie: agent.ping.** Najprostszy klucz agenta, zwracający zawsze wartość `1`, jeśli agent w ogóle odpowiada — nie sprawdza niczego poza samym faktem, że proces żyje i odpowiada na porcie 10050, więc jest pierwszym, a nie jedynym testem.

□□ **zabbix_get** = ręczne odpytanie agenta Zabbixa

□□ TYLKO ODCZYT

Test lokalny na hoście: zabbix_agent2 -t

```
zabbix_agent2 -t agent.ping
```

Wykonujemy sprawdzenie lokalnie, z poziomu samego hosta, z pominięciem sieci — jeśli ten test przechodzi, a `zabbix_get` z serwera nie, problem leży w sieci albo w firewallu, a nie w samym agencie.

Przykład:

```
agent.ping                                [t|1]
```

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

TYLKO ODCZYT

Sprawdzenie logu agenta

```
sudo tail -n 50 /var/log/zabbix/zabbix_agent2.log
```

Log agenta pokazuje błędy startowe (np. zły format pliku konfiguracyjnego), nieudane próby połączenia do serwera przy sprawdzeniach aktywnych, oraz odrzucone połączenia przychodzące spoza listy `Server`.

Przykład:

```
24601:20260922:081512.442 cannot send list of active checks: [111] Connection refused
```

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

Z życia wzięte

Z ŻYCIA WZIĘTE

O 10:05 nowy host `docker02` uparcie pokazuje status niedostępny, mimo że Tomek widzi proces agenta działający w `systemctl status`. Testuje lokalnie `zabbix_agent2 -t agent.ping` i dostaje `1`, więc problem musi leżeć między siecią a serwerem. Sprawdza `zabbix_get` z serwera i dostaje timeout, a po chwili w logu firewalla widzi, że reguła na porcie 10050 została dodana dla złego adresu źródłowego, bo skopiowano ją z innego hosta bez zmiany IP.

Agent 1 kontra agent 2

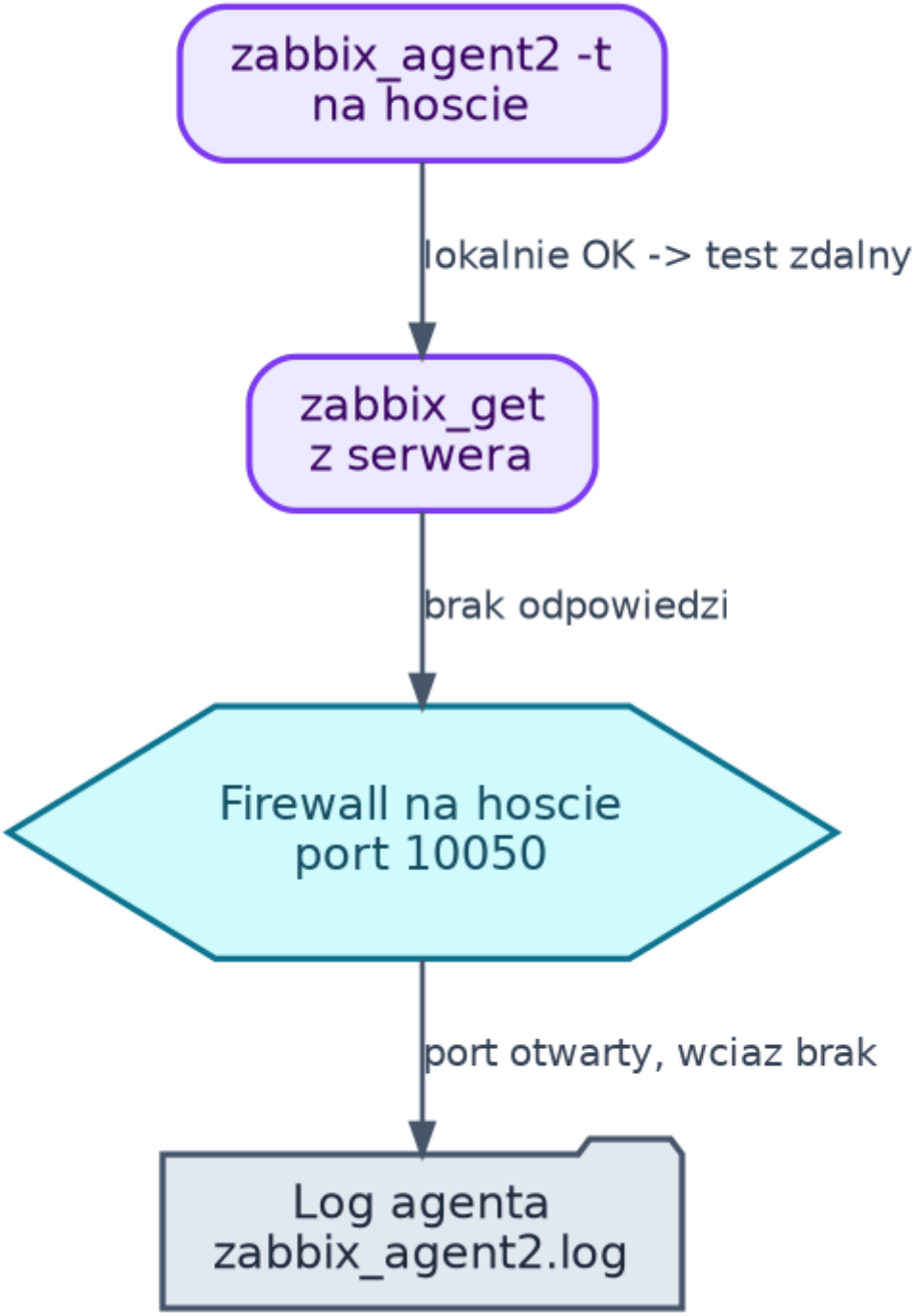
> **Pojęcie: zabbix-agent (agent 1).** Klasyczny agent napisany w C, obsługujący wszystkie standardowe klucze i moduły ładowane (`LoadModule`). Jest jednowątkowy na sprawdzenie — każde sprawdzenie typu `log[]` czy `vfs.file.cksum` na dużym pliku potrafi na chwilę zablokować kolejne sprawdzenia tego samego procesu roboczego.

> **Pojęcie: zabbix-agent2 (agent 2).** Nowsza implementacja w Go, kompatybilna wstecz z kluczami agenta 1, ale działająca w modelu asynchronicznym i rozszerzona o wtyczki (pluginy) do popularnych usług — m.in. gotowe monitorowanie MySQL, PostgreSQL, Docker bez pisania własnych `UserParameter`. Od Zabbixa 6.0 agent2 jest wariantem domyślnie zalecanym do nowych instalacji, a plik konfiguracyjny i większość parametrów pokrywają się z agentem 1 jeden do jednego.

W dalszej części strony posługujemy się nazwą `zabbix-agent2`, ale każda z poniższych fiszek działa analogicznie dla klasycznego `zabbix-agent`, poza sekcją o wtyczkach.

Firewall i SELinux — dwa najczęstsze powody niedostępności hosta

Zanim zaczniemy szukać błędu w samej konfiguracji Zabbixa, sprawdzamy te dwa elementy systemu operacyjnego — w praktyce odpowiadają za większość zgłoszeń „host niedostępny" przy pierwszym podłączaniu nowej maszyny.



Diagnostykę zaczynamy od testu lokalnego, dopiero potem sprawdzamy zdalnie, firewall i log agenta (kliknięcie otwiera pełny rozmiar)

Firewalld — otwarcie portu 10050 dla sprawdzeń pasywnych:

```
sudo firewall-cmd --permanent --add-port=10050/tcp
sudo firewall-cmd --reload
```

Wykonujemy to na hoście monitorowanym, jeśli w ogóle korzystamy ze sprawdzeń pasywnych — przy wyłącznie aktywnych sprawdzeniach ten port nie musi być otwarty, bo połączenie idzie w drugą stronę.

SELinux — sprawdzenie, czy blokuje agenta:

```
sudo ausearch -m avc -c zabbix_agentd --start recent
```

Wyszukujemy w logu audytu odmowy SELinuksa dotyczące procesu agenta z ostatniego okresu — typowym objawem jest agent działający lokalnie bez problemu, ale niepotrafiący odczytać pliku spoza domyślnie dozwolonych ścieżek (np. własnego `UserParameter` odwołującego się do logu aplikacji w niestandardowej lokalizacji).

Przykład:

```
type=AVC msg=audit(1758525312.221:481): avc:  denied  { read } for  pid=24601
comm="zabbix_agentd" name="app.log" dev="dm-2" scontext=system_u:system_r:zabbix_agent_t:s0
tcontext=unconfined_u:object_r:var_log_t:s0 tclass=file
```

Zamiast wyłączać SELinux globalnie, nadajemy dedykowany kontekst plikowi albo katalogowi:

```
sudo semanage fcontext -a -t zabbix_agent_t "SciezkaDoPliku(/.*)?"
sudo restorecon -Rv SciezkaDoPliku
```

Częste błędy

Wielkość liter w Hostname. `Hostname=DOCKER02` w pliku konfiguracyjnym i `docker02` we frontendzie to dla Zabbixa dwa różne hosty — agent zgłasza się poprawnie, ale serwer nie kojarzy przychodzących danych z żadnym zdefiniowanym hostem, więc historia pozostaje pusta mimo działającego agenta.

Zmiana konfiguracji bez restartu usługi. Agent, w odróżnieniu od niektórych demonów, nie przeładowuje `zabbix_agentd.conf` sam z siebie po edycji — po każdej zmianie pliku wykonujemy `sudo systemctl restart zabbix-agent2`.

Testowanie wyłącznie z serwera, nigdy lokalnie. Wynik `zabbix_get` mówi tylko, że coś nie działa gdzieś między serwerem a agentem — bez testu lokalnego (`zabbix_agent2 -t`) nie da się jednym krokiem odróżnić problemu sieciowego od problemu samego agenta.

□□ Ćwiczenia

>

> Wykonujemy na maszynie testowej z zainstalowanym agentem — najlepiej takiej, na której możemy bezkarnie popsuć plik konfiguracyjny.

Ćwiczenie 1 — diagnostyka niedostępnego hosta

Host pokazuje we frontendzie status niedostępny. `zabbix_agent2 -t agent.ping` lokalnie zwraca `1`. Wskazujemy kolejność sprawdzeń, jaką wykonujemy dalej.

Odpowiedź

Skoro lokalnie agent odpowiada, problem leży między siecią a serwerem: sprawdzamy `zabbix_get -s AdresHosta -k agent.ping` z poziomu serwera, następnie firewall na hoście monitorowanym (`firewall-cmd --list-all`), następnie zgodność `Hostname` w pliku konfiguracyjnym z nazwą hosta we frontendzie co do wielkości liter, na końcu log agenta pod kątem odrzuconych połączeń.

Pytania kontrolne

1. Dlaczego test `zabbix_agent2 -t` lokalnie i test `zabbix_get` z serwera mogą dać różny wynik dla tego samego klucza?

Odpowiedź

Bo omijają różne warstwy — test lokalny sprawdza wyłącznie logikę samego klucza wewnątrz agenta, pomijając sieć i listę `Server`. Test z serwera przechodzi przez sieć, firewall po obu stronach i listę dozwolonych adresów w `Server`, więc może się nie udać z powodów niemających nic wspólnego z samym kluczem.

Powiązane: Architektura Zabbixa: serwer, proxy, agent, frontend · Host, element danych, wyzwalacz, akcja: jak to się składa · Monitorowanie typowych usług

Źródła: dokumentacja Zabbix 6.0, rozdział „Installation" i „Zabbix agent 2" · `man zabbix_agent2` · `man firewall-cmd`

Host, element danych, wyzwalacz, akcja: jak to się składa

Poziom: podstawy **Dotyczy: Zabbix 6.0 LTS / 6.4** **Czas: ~25 min**

Bez tego rozdziału reszta konfiguracji Zabbixa jest zestawem magicznych zaklęć — tu układamy pojęcia, z których zbudowana jest każda strona monitoringu w tym systemie.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
LLD	Low-Level Discovery: automatyczne wykrywanie w Zabbixie np. dysków i interfejsów
LTS	Long Term Support: wersja z długim wsparciem poprawkami
SMS	Short Message Service: krótka wiadomość tekstowa na telefon, która dociera także bez internetu
SNMP	Simple Network Management Protocol: protokół odczytu stanu urządzeń sieciowych
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Z życia wzięte

Z ŻYCIA WZIĘTE

Nowy administrator w zespole pyta Tomka o 13:00, dlaczego usunięcie jednego elementu danych z hosta `app01` nagle zostawiło na pulpicie czerwony alarm o „elemencie nieobsługiwanym", chociaż nic już go nie zbiera. Tomek tłumaczy, że element i wyzwalacz to dwie osobne rzeczy, więc usunięcie elementu bez usunięcia powiązanego wyzwalacza zostawia ten wyzwalacz żywy, tylko bez danych do oceny. Razem usuwają oba, po czym alarm znika z listy problemów.

Host — punkt odniesienia dla wszystkiego innego

Pojęcie: host. Pojedynczy monitorowany obiekt — zwykle serwer albo urządzenie sieciowe, ale równie dobrze abstrakcyjna jednostka bez fizycznego adresu (np. „monitoring zewnętrznej strony WWW"). Host sam w sobie nie zbiera niczego — jest kontenerem, do którego przypisujemy elementy danych, wyzwalacze i szablony. Host należy do co najmniej jednej grupy hostów, a grupy służą wyłącznie do organizacji uprawnień i filtrowania widoków, nie mają wpływu na to, co i jak jest zbierane.

Element danych — pojedyncza metryka

Pojęcie: element danych (item). Definicja pojedynczej metryki zbieranej z hosta — jedna linia mówiąca „co zbieramy, jak często i skąd". Element sam w sobie tylko gromadzi wartości w historii, nie ocenia niczego i nie wysyła żadnych powiadomień — to zadanie wyzwalaczy, opisanych niżej.

Pojęcie: klucz elementu (item key). Identyfikator mówiący agentowi (albo innemu mechanizmowi zbierania), jaką konkretnie wartość zwrócić — np. `vfs.fs.size[/,pfree]` (procent wolnego miejsca w systemie plików zamontowanym w `/`) albo `system.cpu.load[percpu,avg1]` (obciążenie procesora, wartość znormalizowana per rdzeń, średnia z minuty). Parametry w nawiasach kwadratowych doprecyzowują, o którą konkretnie instancję metryki chodzi.

Pojęcie: interwał aktualizacji. Jak często element jest odpytywany — ustawiany osobno dla każdego elementu, typowo od kilkudziesięciu sekund (metryki krytyczne, np. dostępność usługi) do kilkunastu minut (metryki wolno zmieniające się, np. wersja zainstalowanego oprogramowania). Krótszy interwał oznacza dokładniejszą historię, ale też szybszy przyrost bazy danych — kompromis, który świadomie dobieramy per element, a nie ustawiamy jednakowo wszędzie.

Pojęcie: typ elementu. Skąd faktycznie pochodzi wartość — najczęstsze typy to Zabbix agent (pasywny, serwer pyta agenta), Zabbix agent (active) (aktywny, agent sam wysyła), SNMP agent, Zabbix trapper (wartość wysyłana z zewnątrz komendą `zabbix_sender`, bez cyklicznego odpytywania) oraz Zabbix internal (metryki samego procesu Zabbixa, np. długość kolejki).

Wyzwalacz — logika oceny

Pojęcie: wyzwalacz (trigger). Wyrażenie logiczne oceniające wartości jednego lub kilku elementów danych i przyjmujące jeden z dwóch stanów: PROBLEM (warunek spełniony) albo OK (warunek niespełniony). Sam wyzwalacz niczego nie wysyła — jedynie zmienia stan, który następnie mogą odczytać akcje.

Pojęcie: wyrażenie wyzwalacza. Formuła odwołująca się do funkcji historycznych na elemencie danych, np.:

```
last(/NazwaHosta/vfs.fs.size[/,pfree])<15
```

Wyrażenie to nie porównuje bieżącej wartości bezpośrednio, tylko wynik funkcji `last()` (ostatnia zebrana wartość) — inne popularne funkcje to `avg()` (średnia z okna czasowego, np. `avg(...,5m)`, przydatna do wygładzania krótkich skoków) oraz `nodata()` (brak jakiejkolwiek wartości przez zadany czas, kluczowe do wykrycia sytuacji, w której element w ogóle przestał być zbierany, a nie tylko przekroczył próg).

Pojęcie: poziom istotności (severity). Etykieta wyzwalacza od Not classified przez Warning, Average, High, aż po Disaster — sama w sobie nic nie robi, ale jest głównym kryterium filtrowania w akcjach: politykę „wysyłaj SMS tylko dla Disaster, e-mail dla wszystkiego od Average w górę" budujemy właśnie na tym polu.

Uważaj: wyzwalacz w stanie PROBLEM, dla którego żadna akcja nie ma dopasowanego warunku, nigdy nie wygeneruje powiadomienia — stan będzie widoczny na pulpicie, ale cicho, co jest częstą przyczyną przeoczonych awarii przy źle poukładanych warunkach akcji.

Akcja — most między stanem a reakcją

> **Pojęcie: akcja (action).** Reguła łącząca zdarzenie (najczęściej zmianę stanu wyzwalacza na PROBLEM albo z powrotem na OK) z konkretną reakcją — wysłaniem powiadomienia, wykonaniem zdalnej komendy albo utworzeniem zgłoszenia w zewnętrznym systemie.



Wartość zebrana przez element danych trafia do wyzwalacza, który przy zmianie stanu tworzy zdarzenie odczytywane przez akcję (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: warunek akcji (condition).** Filtr decydujący, czy dana akcja w ogóle się uruchomi dla konkretnego zdarzenia — np. „grupa hostów = Serwery produkcyjne" AND „poziom istotności ≥ Average". Wiele warunków łączonych jest logiką AND/OR, konfigurowaną osobno dla typu warunku.

> **Pojęcie: operacja akcji (operation).** Konkretna reakcja wykonywana, gdy warunki są spełnione — wysłanie wiadomości do zdefiniowanego użytkownika albo grupy użytkowników wybranym medium (e-mail, SMS, webhook), z opcjonalnym opóźnieniem eskalacji (np. „jeśli po 30 minutach nikt nie potwierdził, powiadom też drugą osobę").

Szablon i dziedziczenie

> **Pojęcie: szablon (template).** Gotowy zestaw elementów danych, wyzwalaczy, reguł wykrywania i pulpitów, przypinany do hosta zamiast konfigurowania każdej z tych rzeczy ręcznie dla każdej maszyny osobno. Podłączenie szablonu MySQL do stu serwerów baz danych oznacza sto identycznie skonfigurowanych zestawów monitoringu, utrzymywanych z jednego miejsca.

> **Pojęcie: dziedziczenie szablonu.** Zmiana wprowadzona w szablonie propaguje się automatycznie na wszystkie podłączone do niego hosty — dodanie nowego elementu do szablonu MySQL od razu pojawia się na każdym hoście z tym szablonem, bez ręcznego powielania zmiany. Element odziedziczony z szablonu można na poziomie konkretnego hosta nadpisać (np. inny próg wyzwalacza dla wyjątkowo obciążonego serwera), ale wtedy przestaje on podążać za dalszymi zmianami tego konkretnego pola w szablonie.

Reguła wykrywania niskopoziomowego i prototypy

> **Pojęcie: reguła wykrywania niskopoziomowego (LLD, low-level discovery).** Mechanizm automatycznego wykrywania zmiennej liczby podobnych obiektów na hoście — systemów plików, interfejsów sieciowych, procesorów, kontenerów Dockera — i tworzenia dla każdego z nich elementów danych i wyzwalaczy bez ręcznego dodawania po jednym na sztukę.

> **Pojęcie: prototyp elementu / wyzwalacza.** Szablon elementu albo wyzwalacza powiązany z regułą LLD, zawierający makro (np. `{#FSNAME}`), które reguła podstawia konkretną wartością dla każdego wykrytego obiektu. Dodanie nowego dysku do serwera z aktywną regułą LLD systemów plików skutkuje automatycznym pojawieniem się nowego elementu monitorującego zajętość tego dysku przy najbliższym cyklu wykrywania — bez żadnej ręcznej interwencji.

Makra i kolejność rozstrzygania

> **Pojęcie: makro.** Zmienna podstawiana w miejscu użycia — we frameworku szablonów pozwala napisać jedną definicję wyzwalacza z progiem `{$FS.PFREE.MIN}` zamiast osobnej definicji z wpisaną na sztywno liczbą dla każdego hosta z osobna.

Wartość makra jest rozstrzygana w ściśle określonej kolejności, od najbardziej szczegółowej do najbardziej ogólnej:

1. wartość ustawiona bezpośrednio na hoście,

2. wartość dziedziczona z szablonu podłączonego do hosta,
3. wartość globalna, ustawiona raz dla całej instalacji Zabbixa,
4. wartość domyślna zdefiniowana przy tworzeniu samego makra.

Dzięki tej kolejności ustawiamy jeden globalny próg `{$FS.PFREE.MIN} = 15` dla wszystkich hostów naraz, a dla pojedynczego serwera z wyjątkowo ciasnym dyskiem nadpisujemy go lokalnie na hoście na `5`, bez dotykania definicji globalnej ani szablonu.

Zdarzenie i problem

> **Pojęcie: zdarzenie (event).** Pojedynczy zapis powstający w chwili zmiany stanu wyzwalacza — jedno zdarzenie przy przejściu w PROBLEM, drugie przy powrocie do OK. To zdarzenia, a nie same wyzwalacze, są tym, co faktycznie odczytują akcje przy podejmowaniu decyzji o powiadomieniu.

> **Pojęcie: problem (problem).** Zdarzenie typu PROBLEM, które nie zostało jeszcze rozwiązane (wyzwalacz nie wrócił do stanu OK) — lista aktywnych problemów jest głównym, pierwszym widokiem otwieranym po zalogowaniu do frontendu, pokazującym wszystko, co w tej chwili wymaga uwagi w całej monitorowanej infrastrukturze.

> **Pojęcie: potwierdzenie problemu (acknowledge).** Ręczne oznaczenie przez administratora, że dany problem jest już zauważony i ktoś się nim zajmuje — nie zmienia to stanu samego wyzwalacza (wciąż jest PROBLEM, dopóki metryka faktycznie nie wróci do normy), ale pozwala odróżnić na liście problemów te, którymi już ktoś się zajął, od tych wciąż nietkniętych.

Częste błędy

Traktowanie elementu i wyzwalacza jako jednej rzeczy. Element tylko zbiera i zapisuje wartość — usunięcie elementu bez usunięcia powiązanych z nim wyzwalaczy zostawia wyzwalacze w stanie „element nieobsługiwany", generujące własny, osobny alarm.

Nadpisanie wartości na hoście i zapomnienie o tym. Wyjątek ustawiony bezpośrednio na hoście przestaje reagować na zmiany w szablonie — pół roku później nikt nie pamięta, dlaczego jeden serwer ma inny próg niż cała reszta grupy, a dokumentacja tego wyjątku nie istnieje.

Zbyt czuła reguła LLD. Reguła wykrywania odpytywana zbyt często na hoście z dużą liczbą dynamicznie tworzonych obiektów (np. kontenery uruchamiane i usuwane co kilka minut) potrafi generować i kasować setki elementów dziennie, co samo w sobie obciąża serwer i zaśmieca historię krótkotrwałymi seriami danych.

□□ Ćwiczenia

> Zadania sprawdzające, bez potrzeby dostępu do żywej instalacji Zabbixa.

Ćwiczenie 1 — złożenie łańcucha pojęć

Dla zdania „gdy wolne miejsce na `/var` spadnie poniżej 10%, wyślij e-mail administratorom" wskazujemy, który fragment odpowiada elementowi danych, który wyzwalaczowi, a który akcji.

Odpowiedź

Element danych to cykliczne zbieranie wartości wolnego miejsca na `/var` (np. `vfs.fs.size[/var,pfree]`). Wyzwalacz to wyrażenie logiczne `last(...)<10`, zmieniające stan na PROBLEM po przekroczeniu progu. Akcja to reguła łącząca zmianę stanu tego wyzwalacza (warunek) z wysłaniem e-maila do grupy administratorów (operacja).

Pytania kontrolne

1. Dlaczego zmiana progu w makrze globalnym nie wpływa na host, który ma ten sam próg nadpisany lokalnie?

Odpowiedź

Bo kolejność rozstrzygania makr sprawdza najpierw wartość ustawioną bezpośrednio na hoście i na niej się zatrzymuje — wartość globalna jest brana pod uwagę tylko wtedy, gdy ani host, ani żaden podłączony do niego szablon nie definiuje własnej wartości tego makra.

2. Po co w ogóle rozdzielać element danych od wyzwalacza, skoro oba dotyczą tej samej metryki?

Odpowiedź

Bo jeden element danych bywa źródłem dla wielu niezależnych wyzwalaczy o różnych progach i różnej istotności (np. Warning przy 20% wolnego miejsca i Disaster przy 5%), a rozdzielenie zbierania od oceny pozwala też zmieniać próg alarmowy bez dotykania definicji samego zbierania danych i bez utraty już zgromadzonej historii.

Powiązane: Architektura Zabbixa: serwer, proxy, agent, frontend · Monitorowanie typowych usług · Utrzymanie serwera Zabbix

Źródła: dokumentacja Zabbix 6.0, rozdział „Data collection", „Triggers" i „Discovery"