WYŻSZA SZKOŁA INFORMATYKI STOSOWANEJ I ZARZĄDZANIA
pod auspicjami Polskiej Akademii Nauk

WYDZIAŁ INFORMATYKI

STUDIA I STOPNIA
(INŻYNIERSKIE)

PRACA DYPLOMOWA

Maciej Korzeń

MONITOROWANIE SIECI, SYSTEMÓW I USŁUG

Praca wykonana pod kierunkiem:

dr inż. Bożena Łopuch

WARSZAWA, 2009 r.

Spis treści

WSTĘP

Wraz ze wzrostem wielkości oraz skomplikowania sieci komputerowych, systemów informatycznych oraz usług, wzrastają wymagania co do ich bezawaryjności oraz dostępności. Dyktowane są one coraz większą ilością i ważnością danych przetwarzanych przez nie. Przekłada się to bezpośrednio na zyski finansowe firm. Instytucji finansowych nie stać na przestoje w funkcjonowaniu systemów produkcyjnych długości kilku godzin. Nawet przestoje kilkuminutowe mogą zaważyć na przyszłości firmy. Dlatego właśnie ważne jest monitorowanie wszystkich ważnych elementów infrastruktury informatycznej, aby móc jak najwcześniej reagować na wszelkie objawy awarii.

Celem przestawionej tutaj pracy jest zaprezentowanie kompletnego i możliwie najbardziej zintegrowanego środowiska monitorującego system informatyczny, które automatyzuje wiele czynności informatycznych, dostarcza narzędzi do wykonywania analiz awarii oraz powiadamia nas o problemach możliwie najwcześniej. Dzięki temu administrator nie musi czekać aż użytkownicy zgłoszą mu awarie. Sam najlepiej wie w jakiej kondycji są poszczególne elementy składowe systemu informatycznego, którym administruje. Opisane rozwiązania przeznaczone są dla średnich i dużych organizacji.

Rozdział 1 prezentuje teoretyczne podstawy i wszystkie niezbędne zagadnienia, które należy opanować aby zrozumieć działania systemów monitorujących.

Rozdział 2 wymienia wszystkie z opisanych w pracy narzędzia. Przy każdym z nich podane są funkcje i możliwości które mogą zostać wykorzystane w wzorcowym środowisku monitorującym.

Rozdział 3 omawia przykładowe narzędzia wraz z ich możliwościami.

Rozdział 4 przedstawia typowe zadania codzienne, tygodniowe i miesięcznie operatora. Omówionych jest także kilka przykładowych sytuacji spotykanych w codziennej pracy wraz ze sposobem postępowania.

Rozdział 5 przedstawia typowe zadania administratora, analogicznie do rozdziału 4. Sytuacje spotykane w pracy administratora różnią się od tych spotykanych na stanowisku operatora. Dotyczą przeważnie funkcjonowania narzędzi.

Do pracy dołączono dwa załączniki. Załącznik A omawia przykładową instalację i konfigurację Nagiosa. Załącznik B to płyta CD zawierająca elektroniczną wersję niniejszej pracy wraz z załącznikiem.

1. PODSTAWOWE POJĘCIA I ZAGADNIENIA

1.1. Definicja monitorowania

Monitorowanie polega na sprawdzaniu obecnego stanu elementów infrastruktury informatycznej pod względem ich prawidłowego funkcjonowania. W razie wykrycia nieprawidłowości automatycznie podjęte mogą zostać przewidziane akcje takie jak wysłanie powiadomień o awarii, próba automatycznej naprawy, itd.

Cele monitorowania to:

- Wykrywanie awarii

  Monitorowanie ma na celu w najlepszej sytuacji ostrzeżenie nas, kiedy wystąpi ryzyko powstania awarii. W większości wypadków nie da się jednak ich przewidzieć z odpowiednim wyprzedzeniem. Dlatego też celem jest jak najszybsze wykrycie nieprawidłowości w systemie informatycznym abyśmy mogli jak najwcześniej zareagować.
- Gromadzenie danych historycznych

  Możemy wyróżnić dwa podstawowe rodzaje danych historycznych:
  - Wartości różnych parametrów mówiących o stanie systemu. Są to na przykład: użycie procesora, użycie pamięci, ilość zajętego miejsca na dyskach. Dzięki temu możemy analizować zachowanie systemu na przestrzeni czasu co dostarcza nam danych porównawczych pomocnych w określeniu obecnego stanu systemu.
  - Komunikaty systemowe. Informują nas one dokładniej w jakim stanie były aplikacje działające w danym systemie. Znajdziemy w nich niejednokrotnie ważne ostrzeżenia oraz komunikaty błędów. Możemy przeglądać wszystkie z nich ręcznie w celu wykrycia nieprawidłowości, ale w wypadku dużych systemów i sieci będzie to praktycznie niemożliwe. Dlatego możemy posłużyć się dedykowanymi narzędziami do analizy dzienników systemowych.

Monitorować możemy prawie wszystkie elementy infrastruktury informatycznej. Począwszy od warunków w serwerowni (zasilanie, temperatura i wilgotność powietrza), poprzez sprawność części składowych serwera (dyski twarde, wersje firmware w poszczególnych podzespołach, stan macierzy dyskowych), funkcjonowanie poszczególnych warstw sieciowych modelu ISO/OSI aż po usługi (poczta, WWW, FTP).

Monitorowanie pozwala nam:

- zmniejszyć do minimum czas niedostępności poszczególnych elementów sieci,
- zmniejszyć przestoje w pracy użytkowników końcowych będące efektem awarii systemów,
- zmniejszyć nakład pracy poprzez automatyzację wielu czynności,
- zapobiegać problemom poprzez wykonywanie czynności prewencyjnych,
- zmniejszyć czas potrzebny do usunięcia awarii,

- zmniejszyć koszt zarządzania systemem informatycznym.

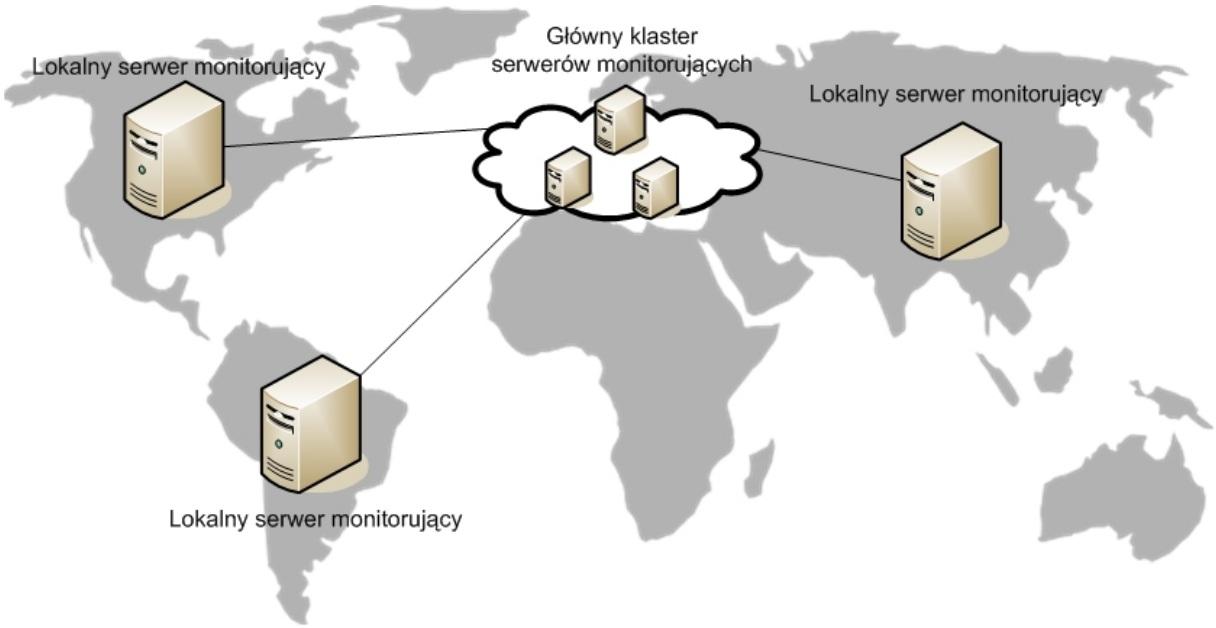
Monitorowanie systemów najlepiej jest wykonywać we wszystkich warstwach, w których mamy możliwość i ma to sens. Warstwa aplikacji może działać bez problemów mimo iż warstwa sieciowa uległa awarii. Prostym przykładem jest redundancja połączeń w warstwie sieci. Jeśli łącze podstawowe ulegnie awarii, to ruch aplikacji będzie przechodził przez łącze zapasowe. Niemniej jednak musimy wiedzieć czy oba łącza (podstawowe i zapasowe) są sprawne oraz podjąć odpowiednie czynności gdy któreś z nich ulegnie awarii.

Przykłady monitorowania w poszczególnych warstwach modelu TCP/IP przedstawiono poniżej:

Warstwa	**Przykład**
Aplikacji	Pobieranie pliku z serwera HTTP.
Transportowa	Nawiązanie połączenia z gniazdem TCP.
Sieciowa	Testowanie osiągalności węzła sieciowego za pomocą pakietów ICMP.
Łącza	Wysyłanie zapytań ARP who-has.

1.2. Architektura systemu monitorującego

Opisana poniżej architektura jest jedną z wielu możliwych. Spotykane w praktyce systemy monitorujące będą różnić się od niej w mniejszym lub większym stopniu. Niemniej jednak oddaje ona ogólne założenia większości spotykanych powszechnie systemów.



Rys. 1.1. Przykład rozproszonej architektury systemu monitorującego.

System monitorujący może być prosty - składający się z jednego serwera - bądź skomplikowany - składający się z wielu serwerów (patrz Rys. 1.1.). Wszystko jest zależne od wielkości i budowy systemu informatycznego, który ma być monitorowany. Jeśli sieć, która jest monitorowana obejmuje

swym zasięgiem lokalizacje na całym świecie, to często w każdej z lokalizacji, a nawet w każdej z podsieci (przy czym jedna lokalizacja może składać się z wielu podsieci) instaluje się osobną stację monitorującą. Wszystkie z tych stacji komunikują się z główną stacją monitorującą, która posiada wiedzę oraz konfigurację dotyczącą całości monitorowanego środowiska. Główna stacja także może składać się z kilku fizycznych elementów w celu zwiększenia wydajności i dostępności.

Aby zwiększyć możliwości systemu monitorującego na każdym z monitorowanych systemów (serwerów) instaluje się tzw. agenta - o ile jest to możliwe. Jest to specjalna aplikacja, która systemowi monitorującemu udostępnia szczegółowe informacje na temat monitorowanego systemu, na którym jest zainstalowana.

Zadania stacji zarządzającej są następujące:

- zbieranie danych,
- zarządzanie komunikatami,
- wykonywanie poleceń na zarządzanych węzłach,
- przekazywanie zebranych danych do innych systemów monitorujących.

Węzłem zarządzanym jest każdy system, który jest monitorowany przez system monitorujący. Może być na nim zainstalowany agent. Jego zadania przestawiają się następująco:

- przechwytywanie komunikatów systemowych,
- monitorowanie wydajności systemu,
- śledzenie komunikatów systemowych i w razie potrzeby podejmowanie odpowiednich czynności,
- buforowanie komunikatów tak aby w razie awarii łączności z główną stacją zarządzającą mogły zostać wysłane przy pierwszej możliwej okazji,
- usuwanie problemów.

System monitorujący gromadzi newralgiczne informacje na temat całości naszego środowiska dlatego należy zadbać o jego bezpieczeństwo. System operacyjny, na którym jest zainstalowany, powinien być zabezpieczony tak, aby osoby niepowołane nie miały do niego dostępu. Trzeba dbać o regularne instalowanie wszelkich aktualizacji. Wszystkie dane wymieniane pomiędzy systemem monitorującym a innymi elementami systemu informatycznego muszą być zabezpieczone przed posłuchaniem przez osoby trzecie. Aby to uzyskać możemy komunikację tego typu przenieść do osobnej, odseparowanej od reszty urządzeń sieci (dedykowane przełączniki lub użycie VLAN-ów) lub zabezpieczyć wymianę komunikatów dodatkową warstwą szyfrującą (TLS).

1.3. Wykrywanie awarii

Jest to najważniejszy cel monitorowania. Poniżej omówione są główne zagadnienia wchodzące w jego skład.

1.3.1. Host

Pod tym pojęciem przeważnie kryje się instancja systemu operacyjnego (potocznie zwana serwerem) podłączona do sieci i posiadająca co najmniej jeden unikatowy adres sieciowy (np. IP). Za pomocą odpowiedniego protokołu (np. ICMP) system monitorujący sprawdza czy dany host działa.

Coraz częściej spotykane są hosty wirtualne. Aby zaoszczędzić zasoby fizyczne (pamięć, procesor, przestrzeń dyskową, miejsce w serwerowni, zużycie prądu) używa się specjalnego oprogramowania wirtualizującego (np. VMware[1], Xen[2], VirtualBox[3]) które pozwala na jednym serwerze fizycznym uruchomić dwie lub więcej instancji systemu operacyjnego.

1.3.2. Usługa

Przeważnie każdy z serwerów udostępnia jedną lub więcej usług. Każda z usług sieciowych komunikuje się z pozostałymi elementami systemu informatycznego za pomocą jednego z wybranych protokołów (np. TCP, UDP). System monitorujący sprawdza czy programy świadczące te usługi działają i funkcjonują poprawnie. Procedura testowania polega przeważnie na nawiązaniu połączenia z programem świadczącym tę usługę, wysłaniu danych testowych, odebraniu odpowiedzi i zweryfikowaniu jej poprawności. Jeżeli w którymś momencie wystąpi błąd (np. nie będziemy mogli połączyć się z programem realizującym usługę bądź otrzymamy błędną odpowiedź) oznaczać to będzie, że usługa nie działa poprawnie i należy podjąć interwencję.

1.3.3. Awaria sieciowa

Każdy dobry system monitorujący powinien umieć wykrywać awarie sieciowe. Awarie takie występują gdy uszkodzeniu ulegnie router, przełącznik albo któryś z kabli sieciowych do niego podłączonych. Ma to wpływ na dostępność wszystkich hostów do których system monitorujący ma dostęp tylko poprzez to urządzenie.

System monitorujący powinien wykryć, że pewna liczba hostów jest niedostępna ze względu na awarię urządzenia znajdującego się po drodze do nich. Powinien rozróżniać niedostępność hosta z powodu awarii sieci od hosta niedziałającego.

1 www.vmware.com
2 www.xen.org
3 www.virtualbox.org

Przykładowo gdy w sieci awarii ulegnie jeden z routerów, do którego podłączone jest kilkadziesiąt hostów, to system monitorujący, który nie potrafi wykryć awarii sieciowej, wyśle wiele powiadomień o awarii - po jednym dla każdego z hostów. Dobry system monitorujący powinien rozpoznać awarię sieciową i wysłać tylko jedno powiadomienie o awarii routera.

1.3.4. Powiadomienie

W razie wystąpienia wszelkiego rodzaju problemów w systemie informatycznym, system monitorujący wysyła o tym powiadomienie. Powiadomienia mogą być realizowane na wiele sposobów. Dwa najpopularniejsze z nich to e-mail oraz SMS. Pierwszy z nich jest najwygodniejszy w czasie godzin pracy kiedy to administrator znajduje się przed komputerem i ma włączonego klienta poczty. Drugi z nich najlepiej sprawdza się gdy administrator jest poza biurem.

Technicznie istnieje wiele możliwości wysyłania SMS-ów. Mogą one być przekazywane za pomocą bramek e-mailowych dostarczanych przez operatorów sieci komórkowych. Możemy także podłączyć bezpośrednio do serwera monitorującego telefon komórkowy z kartą SIM. Należy upewnić się, że wykupimy stały abonament dla tego numeru telefonu oraz, że telefon ma na stałe podłączone zasilanie. Za pomocą dedykowanego oprogramowania będziemy mogli wysyłać SMS-y z powiadomieniami.

Zamiast telefonu komórkowego możemy użyć dedykowanej karty rozszerzeń podłączanej do płyty głównej serwera za pomocą złącza PCI. Z punktu widzenia administratora jest ona wygodniejsza niż leżący obok serwera telefon komórkowy.

Dobrym pomysłem jest także wysyłanie powiadomień za pomocą różnego rodzaju komunikatorów internetowych. Jeżeli system monitorujący nie ma wbudowanej takiej możliwości, to w Internecie znajdziemy wiele gotowych programów i skryptów które pomogą nam dodać obsługę takich powiadomień do systemu.

1.3.5. Eskalacja

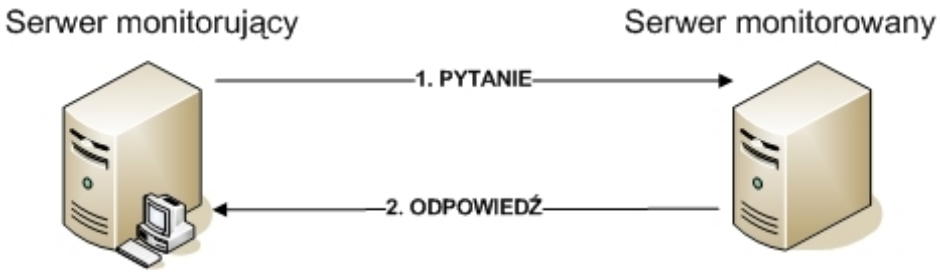
Polega ona na powiadamianiu pracowników wyższego szczebla gdy dana awaria nie jest usunięta przez dłuższy okres czasu.

Przykładowa sytuacja: pracownik techniczny nie poradził sobie w sensownym czasie z usunięciem awarii i zaistniało zagrożenie naruszenie SLA (Service Level Agreement). Wtedy opiekun klienta, którego systemy uległy awarii jest o tym informowany za pomocą SMS-a. Może on wtedy podjąć dodatkowe czynności takie jak przydzielenie dodatkowych osób do usuwania awarii albo poinformowanie klienta o stanie jego systemów.

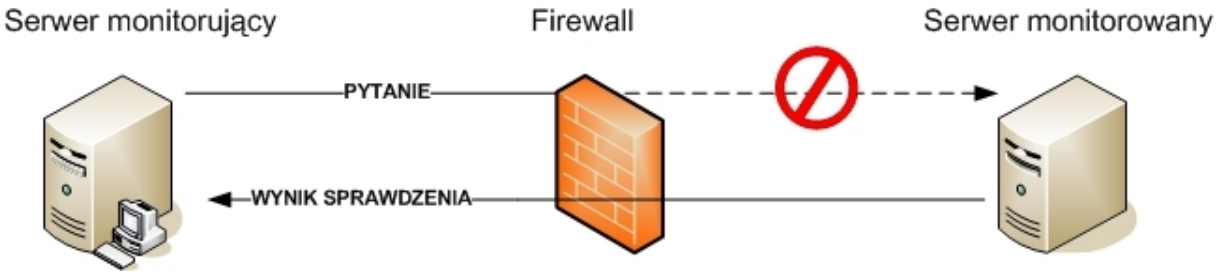
1.3.6. Sprawdzanie aktywne i pasywne

Zwykle to system monitorujący inicjuje czynności sprawdzające czy dany host bądź usługa poprawnie funkcjonuje (Rys. 1.2.). Najprostsze przykłady sprawdzeń aktywnych to wysyłanie do hosta pakietów ICMP *echo-request* bądź nawiązywanie połączenia TCP.

Możliwe jest także wykonywanie sprawdzeń pasywnych (Rys. 1.3.). Wtedy to hosty same wysyłają do systemu monitorującego komunikaty na temat stanu ich samych oraz usług. Takie sprawdzenia znajdują zastosowanie np. gdy monitorujemy system informatyczny należący do klienta, który ze względów bezpieczeństwa nie chce, bądź nie może (jest to zabronione przez procedury bezpieczeństwa, które musi spełniać) dać nam dostępu do własnej sieci. Wtedy klient instaluje we własnym systemie jednego lub więcej agentów systemu monitorującego, które to same dokonują sprawdzeń i ich wyniki wysyłają do naszego systemu monitorującego poprzez bezpieczny kanał komunikacyjny (np. szyfrowane połączenie TCP).



Rys. 1.2. Sprawdzanie aktywne



Rys. 1.3. Sprawdzanie pasywne. Firewall nie zezwala na połączenia przychodzące do serwera monitorowanego.

System monitorujący powinien być wyposażony w mechanizm sprawdzający "świeżość" sprawdzeń pasywnych. Jeśli nie otrzymał od dłuższego czasu informacji o aktualnych stanach monitorowanych obiektów, to znaczy, że najprawdopodobniej agent monitorujący uległ awarii lub występują problemy z komunikacją. Administrator powinien wtedy otrzymać powiadomienie, aby mógł zareagować i naprawić awarię.

1.3.7. Integracja z innymi systemami

Przeważnie w każdej większej firmie używanych jest wiele różnych narzędzi. Każde z nich standardowo ma własną bazę użytkowników i haseł. Jednak zarządzanie wieloma bazami użytkowników jest niewygodne. Dlatego ważną cechą systemu monitorującego (tak samo jak każdego innego oprogramowania używanego wewnątrz organizacji) jest możliwość integracji z pozostałymi systemami, a w szczególności z usługami katalogowymi.

Zamiast tworzyć kolejną osobną bazę użytkowników możemy skonfigurować system monitorujący tak, aby dokonywał autoryzacji na podstawie centralnej bazy. W praktyce bazą tą jest

najczęściej serwer LDAP bądź Active Directory (rozwiązanie Microsoftu bazujące m.in. na standardach takich jak LDAP i Kerberos).

Kolejną ważną cechą systemu monitorującego jest możliwość integracji z systemem obsługującym zgłoszenia (np. Bugzilla[4], Mantis[5], Remedy[6], IBM Rational ClearQuest[7]). W razie wystąpienie awarii mogą być automatycznie zakładane zgłoszenia i przydzielane do odpowiednich osób bądź grup.

1.4. Centralne zarządzanie siecią

Im więcej urządzeń sieciowych wchodzi w skład sieci tym więcej czasu zajmuje zarządzanie nimi. W celu ułatwienia tego zadania powstało wiele aplikacji o mniejszym lub większym zestawie możliwości. Ich zadaniem jest zaoszczędzenie czasu administratora przez automatyzowanie wielu czynności takich jak wsadowa konfiguracja urządzeń, inwentaryzowanie urządzeń, zautomatyzowane tworzenie map połączeń sieciowych, śledzenie anomalii w funkcjonowaniu infrastruktury, tworzenie raportów, zarządzanie konfiguracjami urządzeń, wyszukiwanie urządzeń na podstawie różnych kryteriów, oraz wiele innych. Dzięki dostarczeniu spójnego i wygodnego interfejsu graficznego czynności administracyjne możemy wykonywać szybciej. Nie musimy za każdym razem łączyć się z pojedynczymi urządzeniami.

1.4.1. Urządzenia sieciowe

Są to urządzenia przekazujące dane w sieci komputerowej. Najczęściej spotykane urządzenia sieciowe to router i przełącznik. Rzadziej możemy spotkać bramę, most i hub. Niektóre z urządzeń pełnią kilka funkcji jednocześnie - np. zapora z modułem filtrowania treści w warstwie aplikacji.

1.4.2. Węzły sieci

Węzłem nazywamy każde urządzenie podłączone do sieci. Może to być punkt dystrybucyjny bądź punkt brzegowy. Definicja węzła w dużej mierze zależy od warstwy sieci oraz protokołu w ramach którego go rozpatrujemy. W przypadku sieci Ethernet węzłem będzie każdy element posiadający adres w warstwie drugiej modelu ISO/OSI.

1.4.3. Topologia sieci

Ręczne tworzenie i aktualizowanie map sieci jest czasochłonne - zwłaszcza w przypadku dużych i skomplikowanych systemów informatycznych. Jest to także czynność przydatna i potrzebna jeśli za danym system odpowiada więcej osób. Nie istnieje praktycznie żadne narzędzie które potrafi za nas zrobić kompletną mapę dużej i skomplikowanej sieci, zwłaszcza w kilku warstwach. Niemniej

4 www.bugzilla.org
5 www.mantisbt.org
6 www.bmc.com
7 www-01.ibm.com/software/awdtools/clearquest/

jednak wiele systemów zarządzania siecią potrafi wykrywać połączenia pomiędzy urządzeniami sieciowymi (np. posługując się Cisco Discovery Protocol) i utworzyć na tej podstawie mapę. Stanowi to dobrą podstawę do stworzenia dokładniejszej mapy sieci i oszczędza nakład pracy.

1.5. Wykresy

Do badania stanu części elementów składowych systemów informatycznych bardzo przydatne są wykresy. Dla każdego z mierzonych elementów musimy zgromadzić dane historyczne (tzn. wcześniejsze wyniki pomiaru), które będą nam służyły za punkt odniesienia. Poprzez porównanie wartości z poszczególnych przedziałów czasowych możemy określić jaki był lub jest obecny stan jednego elementu bądź całości infrastruktury. Pozwala to nam wykrywać wcześnie problemy, planować rozbudowę infrastruktury oraz planować okna serwisowe. Poniżej przedstawiono typy wykresów z którymi możemy mieć do czynienia.

1.5.1. Wykresy usług

Mówią nam o wydajności usług i aplikacji w najwyższej warstwie. W pośredni sposób badają ogólną wydajność niższych warstw (sieci/systemu operacyjnego/platformy sprzętowej).

1.5.2. Wykresy części składowych systemów

Są to wykresy związane z parametrami systemu operacyjnego takimi jak użycie pamięci operacyjnej, procesora, obciążenie systemu, itd. Na ich podstawie możemy ocenić czy platforma sprzętowa jest w stanie obsłużyć wszystkie działające na niej aplikacje.

1.5.3. Wykresy parametrów środowiska

Dotyczą one środowiska zewnętrznego w którym znajduje się dany system. Są to między innymi temperatura, parametry zasilania, wilgotność powietrza. Dzięki nim wiemy czy warunki fizyczne nie zagrażają sprzętowi. Zbyt duża temperatura, wysoka wilgotność bądź złe parametry zasilania mogą doprowadzić do awarii serwera i spowodować koszty związane z przestojem świadczenia usługi i wymianą sprzętu.

1.5.4. Wykresy użycia interfejsów sieciowych

Obrazują jaka ilość danych w jednostce czasu jest przesyłana przez interfejsy sieciowe serwerów i urządzeń sieciowych. Dobrze jest oddzielić je od pozostałych wykresów ze względu na to, że jest ich znacznie więcej. Możemy tworzyć wykresy nie tylko dla ilości danych przesłanych przez interfejs, ale także dla ilości błędów, pakietów multicast i broadcast, itd..

1.5.5. Wykresy stanu łącza

Mówią one o sprawności większej części infrastruktury sieciowej w skład której wchodzi co najmniej kilka urządzań sieciowych oraz połączeń między nimi. Dobrze jest badać działanie wszystkich łącz Internetowych którymi dysponujemy. Należy przy tym sprawdzać nie tylko czas powrotu pakietów ICMP ale także straty pakietów.

1.6. Zarządzanie komunikatami

Komunikaty generowane są przez różne części składowe (serwery, aplikacje, itd.) systemów informatycznych. Ze względów bezpieczeństwa powinny one być składowane w innym miejscu niż są generowane. Zapewnia to nam możliwość dokładniejszego prześledzenia wydarzeń w szczególności w takich wypadkach jak awaria bądź włamanie. Jeśli będziemy przechowywać je na każdej maszynie lokalnie, to możemy stracić wszystkie zapisane na niej logi. W razie włamania atakujący z pewnością będzie chciał usunąć ślady kasując lub zmieniając pliki z komunikatami.

Na bezpieczeństwo centralnego serwera komunikatów powinniśmy zwrócić szczególną uwagę. Poza standardowymi czynnościami zabezpieczającymi należy także zadbać o następujące kwestie:

- Fizyczne bezpieczeństwo serwera. Nikt niepowołany nie może się do niego dostać gdyż w ten sposób mógłby uzyskać dostęp do wielu ważnych informacji na temat naszych systemów informatycznych.
- Wyłączenie wszystkich zbędnych usług. Powinniśmy używać tylko bezpiecznych kanałów do łączenia się z serwerem (np. połączenia szyfrowane wykonywane przez dedykowaną sieć).
- Używanie skomplikowanych, silnych haseł.

1.7. Monitorowanie przynależności adresów IP

Polega na śledzeniu powiązań pomiędzy adresami MAC kart sieciowych i adresami IP. Umożliwia nam to wykrywanie błędów w konfiguracji warstwy sieciowej urządzeń podłączonych do sieci, ataków oraz błędów w działaniu serwerów DHCP.

Jeśli nasza sieć nie jest oparta o zarządzalne przełączniki i nie używamy zabezpieczeń przed ARP spoofingiem, to nie powinniśmy umieszczać ważnych serwerów w domenie rozgłoszeniowej do której dostęp mają osoby niezaufane (np. użytkownicy biurowi). Może to doprowadzić do tego, że zwykły użytkownik celowo bądź nieświadomie przypisze sobie adres IP należący do serwera i doprowadzi do przerwy w działaniu usługi.

Jeśli dysponujemy przełącznikami zarządzalnymi to możemy włączyć różne mechanizmy zabezpieczające nas przed taką sytuacją, ale nie zawsze jest to możliwe. W przypadku dużych i dynamicznie rozwijających się sieci wprowadzenie dodatkowych zabezpieczeń może sparaliżować ich funkcjonowanie.

Konflikty adresów IP mogą przypadkowo spowodować także sami administratorzy. Dlatego ważne jest aby monitorować stan sieci na bieżąco i wykrywać wszystkie tego typu nieprawidłowości.

1.8. Wykorzystywane technologie i standardy

Opracowano wiele technologii, protokołów i ogólnie przyjętych standardów służących do monitorowania oraz wykonywania związanych z tym zadań. Poniżej zostaną opisane najważniejsze z nich.

1.8.1. SNMP

Simple Network Management Protocol[8] to protokół służący do nadzorowania i zarządzania urządzeniami sieciowymi. Jego najpopularniejsze zastosowanie to zdalne odczytywanie liczników ilości danych przesłanych przez interfejsy sieciowe, ale możliwości tego protokołu są znacznie większe.

Powstały trzy jego wersje. Przy czym druga z nich posiada trzy odmiany związane z zastosowanymi zabezpieczeniami: party-based, community-based oraz user-based.

SNMP w wersji 1 (SNMPv1) jest pierwszą implementacją protokołu SNMP. Jest najbardziej rozpowszechnioną wersją protokołu SNMP i można uznać ją za standard w zarządzaniu sieciami w Internecie. Korzysta ona z protokołów takich jak User Datagram Protocol (UDP), Internet Protocol (IP), OSI Connectionless Network Service (CLNS), AppleTalk Datagram-Delivery Protocol (DDP), and Novell Internet Packet Exchange (IPX). Wadą tej wersji protokołu jest bardzo niski poziom zabezpieczeń. Autoryzowanie klientów odbywa się wyłącznie na podstawie tzw. *community string*, który jest w praktyce hasłem przesyłanym w nieszyfrowanej formie przez sieć. W czasach gdy go projektowano do bezpieczeństwa nie przykładano dużej wagi. SNMPv1 był krokiem pośrednim do stworzenia protokołów odpowiednich do użycia w dużych, komercyjnych sieciach.

Przy projektowaniu SNMPv2 wprowadzono, w porównaniu do poprzedniej wersji, usprawnienia w wydajności, bezpieczeństwie, poufności i komunikacji między stacjami zarządzającymi. Wprowadzono polecenia GETBULK oraz alternatywne rozwiązanie dla GETNEXT, które pozwala na przesłanie dużej liczby danych w pojedynczym zapytaniu. Nowy model zabezpieczeń nazwany *party-based* został uznany za zbyt skomplikowany i nie spotkał się z powszechnym przyjęciem. Odmiana SNMPv2 o nazwie SNMPv2c porzuca nowe mechanizmy zabezpieczeń i używa *community string* znanego z SNMPv1 do autoryzowania monitorowanych węzłów. Jest to najbardziej rozpowszechniona wersja SNMPv2, pomimo iż nie jest oficjalnym standardem. Kompromisem pomiędzy standardową wersją SNMPv2, a SNMPv2c jest protokół SNMPv2u, który zapewnia większe bezpieczeństwo niż SNMPv1, ale w prostszy sposób niż zostało to zrobione w SNMPv2.

8 SNMPv1 opisane jest w RFC o numerach 1065, 1066 i 1067. SNMPv2p opisane jest w RFC o numerach 1441 i 1452. SNMPv2c opisane jest w RFC o numerach 1901 i 1908. SNMPv2u opisane jest w RFC o numerach 1909 i 1910. SNMPv3 opisane jest w RFC o numerach 3411, 3418 i 3584.

Za obecnie obowiązujący standard protokołu SNMP jest SNMPv3. Wcześniejsze wersje uznane są za historyczne i przestarzałe. SNMPv3 wprowadza trzy ważne usługi: uwierzytelnianie, poufność i kontrolę dostępu.

Każda z operacji wykonywanych w ramach protokołu operuje na określonej zmiennej, tzw. OID (Object IDentifier). Przykładowo zmienna o nazwie sysUpTime (czas, który upłynął od włączenia urządzenia) posiada OID 1.3.6.1.2.1.1.3.0. Struktura OID zbudowana jest na zasadzie drzewa, poszczególne elementy oddzielone są od siebie kropkami. Pierwszy element jest korzeniem, a ostatni liściem.

Protokół działa w architekturze klient-serwer. Na monitorowanych urządzeniach działa tzw. agent SNMP. Stacja zarządzająca SNMP łączy się do agenta SNMP i wysyła polecenia po czym oczekuje na odpowiedź. Każdy agent obsługuje zdefiniowany wcześniej jeden bądź więcej zestawów zmiennych. Zestaw zmiennych określany jest jako MIB (Management Information Base). Stacja zarządzająca, aby potwierdzić że ma prawo do wykonywania odpowiednie zestawu poleceń, musi podać jedno z haseł. Zwykle stosowane są dwa rodzaje haseł (ale może być ich więcej): *public* i *private*. Pierwsze z nich daje dostęp do odczytu tylko ograniczonej liczby zmiennych. Drugie daje pełen dostęp do wszystkich zmiennych. Operacje odczytu służą do sprawdzania aktualnego stanu urządzenia, operacje zapisu pozwalają sterować jego działaniem.

Agent sam może zgłaszać zdarzenia do stacji zarządzającej za pomocą komunikatów Trap.

Domyślnie SNMP działa na porcie 161 protokołów TCP i UDP. W przypadku komunikatów Trap jest to port 162.

Główną zaletą SNMP jest jego prosta budowa - używa UDP i w małym stopniu obciąża łącze. Największą wadą jest to, że dopiero w jego trzeciej wersji zaimplementowano zaawansowaną obsługę szyfrowania, ale wersja ta nie jest jeszcze bardzo rozpowszechniona. Jeśli jesteśmy zmuszeni do używania wersji 1 bądź 2 to pewien stopień prywatności możemy sobie zapewnić używając SNMP tylko w dedykowanej sieci - np. implementując VLAN-y.

W SNMP dostępne są następujące komunikaty: Set, Get, GetNext, Response, Trap, GetBulk, Inform (dwa ostatnie dopiero od wersji 2).

1.8.2. CDP i LLDP

Są to protokoły za pomocą których urządzenia sieciowe informują inne urządzenia o swoim istnieniu. Ułatwia nam to znacznie tworzenie mapy sieci. Informacje uzyskane za pomocą tych protokołów są wykorzystywane przez niektóre programy służące do zarządzania siecią.

Cisco Discovery Protocol[9] jest własnością firmy Cisco. Link Layer Discovery Protocol został zaprojektowany jako niezależny od producenta standard (802.1AB-2009) przez organizację IEEE. LLDP zastępuje CDP oraz podobne, własnościowe protokoły.

Protokoły te działają w warstwie drugiej modelu ISO/OSI. Specjalne pakiety rozgłoszeniowe rozsyłane są w regularnych odstępach czasu.

9 Więcej informacji o CDP znajdziemy w dokumentacji na stronie Cisco: http://www.cisco.com/univercd/home/home.htm.

2. WYBÓR NARZĘDZI

Na rynku oprogramowania oraz w Internecie dostępnych jest wiele dobrych narzędzi. Trudno jest dokonać najlepszego wyboru, gdyż wymaga to wiele czasu poświęconego na testowanie. Pocieszający może być fakt, że w większości wypadków nie będzie potrzebna aplikacja o największych możliwościach, gdyż i tak wykorzystana zostanie zaledwie część z nich. Nawet najlepszy produkt zda się na nic jeśli nie będziemy potrafili go obsłużyć, dlatego ważna jest dobra dokumentacja i wygoda użytkowania. W naszym wyborze ograniczyliśmy się do narzędzi darmowych. Cena oprogramowania jest bardzo ważnym kryterium przy dokonywaniu wyboru. Punktem wyjścia podczas dokonywaniu wyboru było stworzenie kompletnego i zintegrowanego środowiska monitorującego.

Cześć funkcjonalności narzędzi, które zostaną omówione, pokrywa się. Poniżej prezentujemy listę narzędzi, przy każdym z nich wyszczególnione zostały funkcje, które powinny zostać użyte, aby otrzymać kompletne i możliwie najbardziej zintegrowane środowisko monitorujące. Tak stworzone środowisko monitorujące jest w pełni zgodne z założeniami opisanymi w Rozdziale 1.

2.1. Nagios

- wykrywanie awarii,
- powiadomienia,
- automatyczne usuwanie awarii,
- wykresy związane z wydajnością usług i wykorzystaniem części składowych serwera.

2.2. NeDi

- ewidencja wszystkich urządzeń podpiętych do sieci,
- prowadzenie magazynu urządzeń sieciowych,
- śledzenie kontraktów serwisowych urządzeń sieciowych,
- tworzenie mapy sieci,
- wykrywanie błędów w warstwie sieci (zbieranie logów i SNMP Trap z urządzeń sieciowych),
- pomoc w planowaniu rozbudowy infrastruktury sieciowej - raportowanie ilości użytych portów na przełącznikach.

2.3. Cacti

- wykresy obciążenia interfejsów sieciowych w serwerach, routerach i przełącznikach.

2.4. SmokePing

- badanie stanu połączeń z dostawcami Internetu.

2.5. phpLogCon

- przeglądanie dzienników systemowych z serwerów i urządzeń sieciowych zgromadzonych w bazie danych SQL.

2.6. Arpwatch

- wykrywanie konfliktów IP,
- wykrywanie nowych urządzeń sieciowych.

3. OMÓWIENIE NARZĘDZI

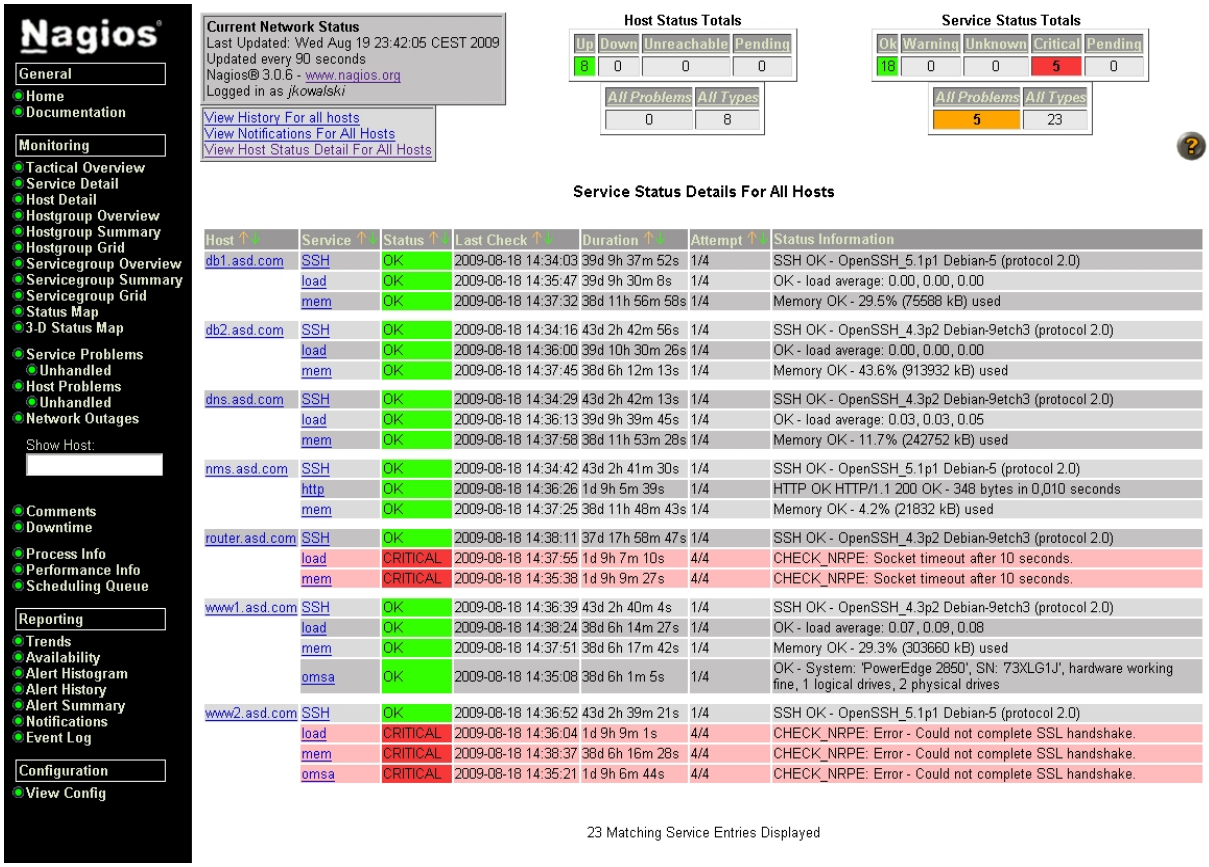
Wspólną cechą łączącą wymienione w tym rozdziale narzędzia jest to, że pracują w systemie operacyjnym Linux. Niektóre z nich mogą być uruchamiane także na innych systemach, ale jest to kwestia indywidualna dla każdego z nich. Dokładną specyfikację wspieranych systemów operacyjnych znajdziemy w dokumentacji każdego narzędzia.

Linux jest popularnym systemem operacyjnym do zastosowań serwerowych. Niemniej jednak jeśli będziemy chcieli użyć innego z popularnych systemów operacyjnych to z pewnością znajdziemy na niego wiele aplikacji o podobnych możliwościach.

3.1. Nagios

Strona domowa: www.nagios.org.

3.1.1. Wstęp



Rys. 3.1. Interfejs WWW Nagiosa – stan usług.

Nagios jest aplikacją, która monitoruje urządzenia oraz sieci. Możemy go uruchomić na prawie każdym systemie uniksowym (w szczególności na Linuksie). Posiada interfejs WWW (Rys. 3.1.) za pomocą którego możemy oglądać stan monitorowanych obiektów oraz wydawać aplikacji niektóre

polecenia. Interfejs WWW jest realizowany za pomocą technologii CGI. Jeśli aplikacja wykryje awarię może nas o tym powiadomić (np. za pomocą poczty elektronicznej). Oczywiście powiadomienia dostaniemy także kiedy sytuacja wróci do normy.

Nagios rozpowszechniany jest na licencji GNU General Public License w wersji 2. Tak więc możemy korzystać z niego zarówno do celów prywatnych jak i służbowych zupełnie za darmo. Mimo to możemy wykupić komercyjne wsparcie w jednej z wielu firm które się tym zajmują. W porównaniu z typowo komercyjnymi systemami monitorującymi Nagios ma te zalety, że możemy go modyfikować w dowolny sposób zależnie od naszych potrzeb. Nie jesteśmy zależni od firm zewnętrznych, nie ograniczają nas kwestie finansowe ani licencje. Jedyne co musimy poświęcić to czas.

3.1.2. Wymagania

Program wymaga aby w systemie zainstalowany był interpreter języka skryptowego Perl. Interfejs WWW potrzebuje serwera WWW z obsługą CGI oraz opcjonalnie autoryzacji HTTP (jest to zalecane ze względów bezpieczeństwa). Apache[10] - najpopularniejszy serwer WWW na systemy Uniksowe - spełnia wszystkie te wymagania.

Funkcja rysowania map wymaga biblioteki GD[11]. Dodatkowo potrzebne są biblioteki do tworzenia obrazów w formatach JPEG i PNG aby biblioteka GD mogła tworzyć obrazy w tych formatach.

Część z pluginów do działania wymaga biblioteki OpenSSL[12] która jest używana przez nie do nawiązywania szyfrowanych połączeń TCP.

3.1.3. Możliwości

W standardowej konfiguracji Nagios potrafi monitorować usługi sieciowe (POP3, SMTP, HTTP, NNTP, PING, itd.) i monitorować zasoby systemowe (obciążenie procesora, użycie dysku, itd.).

Kolejną przydatną funkcją Nagiosa jest czynne reagowanie na awarie. Możemy go tak skonfigurować aby np. zrestartował dany serwer jeśli obciążenie będzie zbyt wysokie.

Jeśli poświęcimy konfiguracji aplikacji odpowiednią ilość czasu to możemy stworzyć niezawodną infrastrukturę monitorującą. Nagios wspiera tworzenie wysoko dostępnych (ang. *high available*) instalacji. Wtedy Nagios jest zainstalowany np. na kilku fizycznych serwerach i awaria jednego z nich nie powoduje awarii całego systemu monitorującego.

W plikach konfiguracyjnych poszczególnych urządzeń możemy określać strukturę połączeń. Dzięki temu Nagios będzie lepiej sobie radził jeśli wystąpią większe awarie sieciowe. Nie ma sensu aby administrator otrzymywał powiadomienie o niedostępności każdego serwera w odległej lokalizacji jeśli uszkodzeniu uległ zdalny router. Jeśli w konfiguracji serwerów określimy do którego routera są one podłączone to Nagios wyśle powiadomienie tylko o niedostępności routera.

10 httpd.apache.org
11 www.boutell.com/gd/
12 www.openssl.org

3.1.4. Opis działania

Wszystkie sprawdzenia Nagios wykonuje za pomocą programów zewnętrznych. Tak więc jeśli potrafimy pisać chociażby skrypty powłoki to możemy stworzyć rozszerzenia (ang. *plugin*), które będą w stanie sprawdzać dowolnie skomplikowane usługi.

Zanim zaczniemy używać aplikacji musimy ją skonfigurować. Należy skopiować przykładowe pliki konfiguracyjne i dostosować je do własnych potrzeb. W Debianie pliki konfiguracyjne znajdują się w katalogu */etc/nagios3.*

Sprawdzanie aktywne

Wyróżnia się tym, że są inicjowane przez Nagiosa i wykonywane są w z góry określonych, regularnych odstępach czasu.

Nagios wywołuje zdefiniowany plugin, przekazuje mu niezbędne informacje na temat tego co ma być sprawdzone (np. adres hosta, wartości progów ostrzegawczego i krytycznego). Plugin dokonuje sprawdzenia hosta bądź usługi i przekazuje wynik do Nagiosa. Ten z kolei na podstawie otrzymanych informacji może w razie potrzeby podjąć niezbędne czynności: wysłać powiadomienia, uruchomić polecenie obsługujące zdarzenie, itd.

Sprawdzenia aktywne wykonywane są w stałych odstępach czasu, zgodnie z tym co zostało zdefiniowane w zmiennych *check_interval* oraz *retry_interval* w definicji hosta bądź usługi. Mogą także być wykonywane na żądanie użytkownika oraz gdy Nagios będzie potrzebował najświeższych informacji o stanie danego obiektu (np. w momencie obliczania dostępności danego fragmentu sieci).

Sprawdzanie pasywne

Charakteryzuje się tym, że jest inicjowane i wykonywane przez zewnętrzne systemy/aplikacje. Wyniki tych sprawdzeń są następnie zgłaszane do Nagiosa. Sprawdzanie pasywne znajduje zastosowanie w sieciach, w których monitorowane zasoby oddzielone są od Nagiosa firewallem. Wtedy w monitorowanej sieci uruchamia się jednego lub wiele specjalnych agentów, które wykonują sprawdzenia i zgłaszają wyniki do Nagiosa.

Sprawdzenia pasywne są także wykorzystywane w dużych, wysoko dostępnych (ang. *high available*) instalacjach Nagiosa.

Schemat działania sprawdzeń pasywnych jest następujący:

- Aplikacja zewnętrzna sprawdza daną usługę bądź hosta.
- Wynik sprawdzenia zapisywany jest do pliku FIFO nazywanego Plikiem poleceń Zewnętrznych. Plik ten jest używany także przez skrypty CGI. Jego domyślna lokalizacja w Debianie to */var/lib/nagios3/rw/nagios.cmd*.
- Nagios regularnie odczytuje zawartość tego pliki i umieszcza odczytane wyniki sprawdzeń w specjalnej kolejce. Kolejka ta używana jest także do przechowywania wyników sprawdzeń aktywnych. W tym momencie Nagios już nie rozróżnia jakiego rodzaju było sprawdzenie.

- Kolejka wyników sprawdzeń jest regularnie sprawdzana przez Nagiosa i w razie potrzeby podejmowane są odpowiednie czynności (powiadomienia, obsługa zdarzeń, itd.).

Do obsługi sprawdzeń pasywnych w Nagiosie istnieje specjalny dodatek o nazwie NSCA. Składa się on z agenta instalowanego w monitorowanej sieci oraz serwera instalowanego na serwerze z Nagiosem, który odbiera wyniki sprawdzeń i zgłasza je do Nagiosa. Więcej o NSCA dowiemy się z dokumentacji Nagiosa.

Pluginy

Pluginy jest to nazwa dla programów zewnętrznych, które wykonują sprawdzenia hostów i usług w Nagiosie. Plugin składa się przeważnie z programu wykonywalnego (może to być dowolny typ programu uruchamianego przez system: skrypt w jakimś języku, skompilowany program, itd.) oraz definicji polecenia.

Zwykle każdy plugin po uruchomieniu go w wierszu poleceń z argumentem *--help* wyświetli dokładną instrukcję jego użycia wraz ze wszystkimi niezbędnymi opisami.

Pluginy możemy tworzyć samodzielnie i w ten sposób rozbudowywać możliwości Nagiosa. Lokalizacja plików wykonywalnych jest dowolna – ważne jest to aby Nagios odczytał napisane przez nas definicje poleceń.

Tworzenie własnych pluginów

Jedną z cech sprawiającą, że Nagios jest bardzo użyteczną aplikacją, jest możliwość pisania własnych pluginów. Dzięki temu mamy możliwość monitorowania niemalże dowolnych i nawet bardzo skomplikowanych usług.

Każdy plugin musi spełniać zaledwie dwa wymagania:

- Zwrócić do procesu macierzystego jedną z możliwych wartości mówiącą o stanie usługi.
- Na standardowym wyjściu wypisać jeden wiersz tekstu.

Oto możliwe do zwrócenia przez plugin wartości oraz ich znaczenie:

Wartość zwracana	**Stan usługi**	**Stan hosta**
0	OK	UP
1	WARNING	UP
2	CRITICAL	DOWN/UNREACHABLE
3	UNKNOWN	DOWN/UNREACHABLE

Zaprezentujemy teraz przykład tworzenia własnego pluginu do Nagiosa. Nie jest to rzecz skomplikowana. Wystarczy przygotować skrypt (bądź plik wykonywalny) z programem, definicję polecenia w Nagiosie i skonfigurować usługę tak jak w przypadku istniejących pluginów.

Nasz Plugin będzie sprawdzał ilość użytej pamięci RAM. Standardowo z Nagiosem plugin realizujący taką funkcję nie jest dostarczany. W Linuksie informacje o użyciu pamięci dostarczane są np. przez program *free*.

Poniżej znajduje się treść skryptu *check_mem*, który został napisany w języku Perl:

```
#!/usr/bin/perl -w
# Copyright (C) 2008, 2009 Maciej Korzeń

use strict;
use Getopt::Std;

use vars qw($arg_c $arg_w $memory_used $memory_total %exit_codes
@output_split $memory_used_percent $memory_used_percent_fmt $command_line);

%exit_codes   = ('OK'      , 0,
                 'WARNING' , 1,
                 'CRITICAL', 2,
                 'UNKNOWN' , 3,);

sub usage()
{
        print "Usage: check_mem -w <warning> -c <critical>\n";
        print "\n";
        print "\t-w PERCENT\tReturn WARNING if used memory is above given
value. It has to be greather than critical level.\n";
        print "\t-c PERCENT\tReturn CRITICAL if used memory is above given
value.\n";
        exit($exit_codes{'UNKNOWN'});
}

$command_line = `free | tail -2 | head -n 1 | awk '{print \$3,\$4}'`;

chomp $command_line;

@output_split = split(/ /, $command_line);
$memory_used  = $output_split[0];
$memory_total = $memory_used + $output_split[1];

if ($#ARGV le 0)
{
        usage();
}

getopts('w:c:');

if ((!$arg_w) or ($arg_w == 0) or (!$arg_c) or ($arg_c == 0) or ($arg_w >=
$arg_c))
{
        usage();
}

$memory_used_percent = $memory_used / $memory_total * 100;
$memory_used_percent_fmt = sprintf "%.1f", $memory_used_percent;
if ($memory_used_percent >= $arg_c)
{
        print "Memory CRITICAL - $memory_used_percent_fmt% ($memory_used kB
) used\n";
        exit $exit_codes{'CRITICAL'};
} elsif ($memory_used_percent >= $arg_w) {
        print "Memory WARNING - $memory_used_percent_fmt% ($memory_used kB)
used\n";
        exit $exit_codes{'WARNING'};
```

```
} else {
        print "Memory OK - $memory_used_percent_fmt% ($memory_used kB)
used\n";
        exit $exit_codes{'OK'};
}
```

NRPE

Pluginy mogą badać stany usług w systemie, na którym są uruchamiane albo korzystając z różnych protokołów sieciowych (przeważnie TCP/IP) stany usług na innych hostach. Jednak nie zawsze wszystko da się sprawdzić poprzez sieć. Aby móc sprawdzać np. parametry fizyczne danego serwera (temperatura, prędkości wentylatorów, itd.) należałoby napisać programy serwera oraz klienta, które takie dane by udostępniały i przesyłały przez sieć.

Dodatkowo ze względów bezpieczeństwa nie wszystkie usługi należy udostępniać poprzez sieć. Np. serwer MySQL można skonfigurować tak, aby był dostępny tylko z lokalnego systemu. W takiej sytuacji nie będzie możliwości sprawdzenia jego dostępności za pomocą tradycyjnego pluginu *check_mysql* działającego na serwerze z Nagiosem.

Tutaj z pomocą przychodzi NRPE – Nagios Remote Plugin Executor. Składa się z serwera NRPE, który jest instalowany na monitorowanym serwerze oraz klienta w postaci pluginu *check_nrpe* używanego przez Nagiosa. Pluginy nie są wywoływane na serwerze z Nagiosem, ale lokalnie na monitorowanym systemie. Nagios za pomocą *check_nrpe* każe serwerowi NRPE wywołać konkretny plugin, a wynik sprawdzenia usługi transportowany jest poprzez sieć i efekt jest taki jakby Nagios wywoływał końcowy plugin lokalnie.

Dzięki temu nie ma konieczności implementowania warstwy sieciowej dodatkowo w każdym pluginie.

Typy stanów

Każdy host i usługa mają przypisany stan (UP, DOWN, OK, WARNING, CRITICAL, itd.). Dodatkowo wyróżniane są typy stanów: miękki (w nomenklaturze Nagiosa - SOFT) oraz twardy (HARD).

Jeśli dany obiekt zmieni swój stan z OK (w przypadku usługi) lub UP (w przypadku hosta) na inny po raz pierwszy to Nagios ustawi jego typ na miękki. Dopiero gdy Nagios stwierdzi kilka razy z rzędu, że dany obiekt nie działa prawidłowo, typ jego stanu zostanie zmieniony na twardy. Mechanizm ten ma za zadanie zapobiegać fałszywym alarmom. Zmienna *max_check_attempts* używana w deklaracji hostów i usług określa ile razy Nagios musi stwierdzić, że dany obiekt jest w stanie nie-OK bądź nie-UP zanim zmieni typ stanu na twardy.

Obsługa zdarzeń

Nagios potrafi reagować na zmiany stanów obiektów poprzez wykonywanie zdefiniowanych poleceń. Możemy wykorzystać to np. do restartowania usług, które przestały funkcjonować albo integracji z systemem obsługi zgłoszeń (możemy automatycznie zakładać zgłoszenia o awarii osobą odpowiedzialnym za dane hosty i usługi).

Niezawodność

Nagios umożliwia tworzenie instalacji wysoko dostępnych i niezawodnych. Najprostszym sposobem na zabezpieczenie się przed awarią całego systemu monitorującego z powodu fizycznej awarii serwera jest zainstalowanie dwóch (lub więcej) instancji Nagiosa na dwóch różnych maszynach fizycznych. Jeden z tych serwerów będzie głównym, a drugi zapasowym. Oba powinny mieć prawie identyczną konfigurację. Jedyna różnica powinna być taka, że Nagios zapasowy nie powinien wysyłać powiadomień. W razie awarii serwera głównego na serwerze zapasowym możemy włączyć wysyłanie powiadomień i w ten sposób zachowamy ciągłość działania. Będziemy mieli czas na usunięcie awarii serwera głównego. Bardzo ważną rzeczą o której musimy pamiętać jest synchronizowanie wszystkich zmian w konfiguracji Nagiosa pomiędzy oboma serwerami.

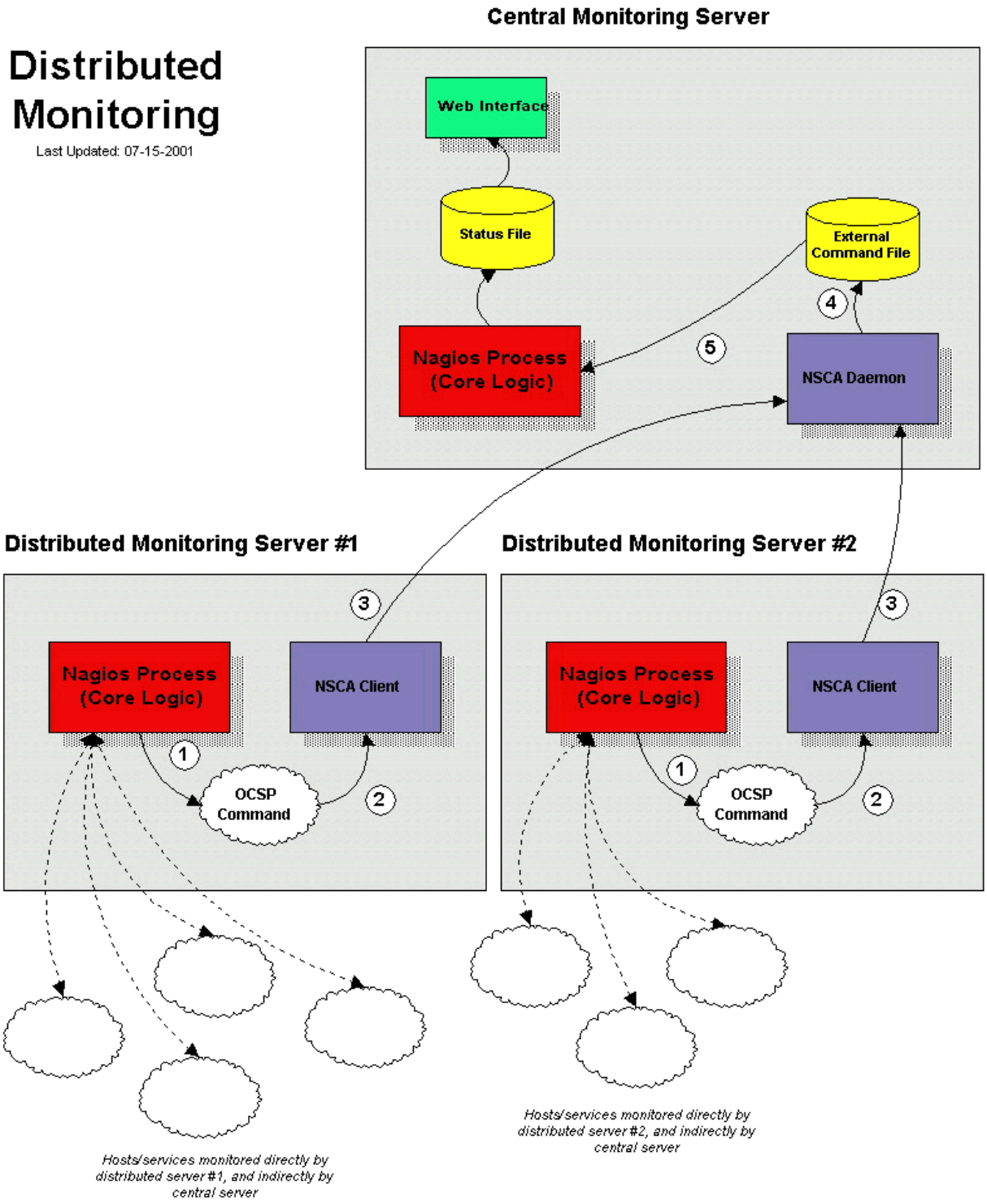
Monitorowanie rozproszone

Jeśli system monitorujący musi monitorować bardzo duże środowisko to w pewnym momencie może się okazać, że Nagios przestanie nadążać ze sprawdzaniem wszystkich hostów i usług. Problem ten można rozwiązać wykonując rozproszoną instalację Nagiosa (Rys. 3.2.). Składa się ona z co najmniej trzech oddzielnych serwerów. Jeden z nich jest serwerem centralnym, a dwa pozostałe to serwery robocze. Wszystkie sprawdzenia wykonywane są aktywnie przez serwery robocze, a ich wyniki za pomocą mechanizmu sprawdzeń pasywnych wysyłane są do serwera centralnego. Serwer centralny dysponuje kompletnym obrazem sieci i udostępnia interfejs WWW. Każdy z serwerów roboczych monitoruje inne hosty i usługi. Dzięki temu obciążenie rozkładane jest aż na trzy serwery fizyczne, a nie tylko na jeden.

Autoryzacja HTTP

Dane udostępniane przez Nagiosa są dość wrażliwe i powinny być chronione. Dlatego domyślnie jest on skonfigurowany tak, aby użytkownik musiał się autoryzować do interfejsu WWW. W rzeczywistości użytkownik nie autoryzuje się do Nagiosa, ale do serwera WWW. Jeśli serwer WWW umożliwia wykorzystanie zewnętrznych usług katalogowych (np. LDAP, Active Directory) to można w efekcie zintegrować Nagiosa z centralną bazą użytkowników w firmie.

Apache – najpopularniejszy Linuksowy serwer WWW – umożliwia wykorzystanie zarówno LDAP-a jaki i Active Directory do autoryzowania użytkowników. Na temat tego drugiego więcej informacji znajduje się na stronie o adresie http://xavier.dusart.free.fr/nagios/en/kerberos.html.



Rys. 3.2. Rozproszona instalacja Nagiosa. Grafika pochodzi z oficjalnej dokumentacji programu.

Domyślna zawartość pliku */etc/nagios3/apache2.conf*:

```
# apache configuration for nagios 3.x
# note to users of nagios 1.x and 2.x:
#    throughout this file are commented out sections which preserve
#    backwards compatibility with bookmarks/config for older nagios
#    versios.
#    simply look for lines following „nagios 1.x:” and „nagios 2.x”
#    comments.

ScriptAlias /cgi-bin/nagios3 /usr/lib/cgi-bin/nagios3
ScriptAlias /nagios3/cgi-bin /usr/lib/cgi-bin/nagios3
# nagios 1.x:
#ScriptAlias /cgi-bin/nagios /usr/lib/cgi-bin/nagios3
#ScriptAlias /nagios/cgi-bin /usr/lib/cgi-bin/nagios3
# nagios 2.x:
#ScriptAlias /cgi-bin/nagios2 /usr/lib/cgi-bin/nagios3
#ScriptAlias /nagios2/cgi-bin /usr/lib/cgi-bin/nagios3

# Where the stylesheets (config files) reside
Alias /nagios3/stylesheets /etc/nagios3/stylesheets
# nagios 1.x:
#Alias /nagios/stylesheets /etc/nagios3/stylesheets
# nagios 2.x:
#Alias /nagios2/stylesheets /etc/nagios3/stylesheets

# Where the HTML pages live
Alias /nagios3 /usr/share/nagios3/htdocs
# nagios 2.x:
#Alias /nagios2 /usr/share/nagios3/htdocs
# nagios 1.x:
#Alias /nagios /usr/share/nagios3/htdocs

<DirectoryMatch (/usr/share/nagios3/htdocs|/usr/lib/cgi-bin/nagios3)>
Options FollowSymLinks

DirectoryIndex index.html

AllowOverride AuthConfig
Order Allow,Deny
Allow From All

AuthName „Nagios Access”
AuthType Basic
AuthUserFile /etc/nagios3/htpasswd.users
# nagios 1.x:
#AuthUserFile /etc/nagios/htpasswd.users
require valid-user
</DirectoryMatch>

# Enable this ScriptAlias if you want to enable the grouplist patch.
# See http://apan.sourceforge.net/download.html for more info
# It allows you to see a clickable list of all hostgroups in the
# left pane of the Nagios web interface
# XXX This is not tested for nagios 2.x use at your own peril
#ScriptAlias /nagios3/side.html /usr/lib/cgi-bin/nagios3/grouplist.cgi
# nagios 1.x:
#ScriptAlias /nagios/side.html /usr/lib/cgi-bin/nagios3/grouplist.cgi
```

Dyrektywy *AuthName*, *AuthType*, *AuthUserFile* oraz *require* dotyczą autoryzacji. Login i hasło podane przez użytkownika sprawdzane są w pliku */etc/nagios3/htpasswd.users*.

Domyślnie plik ten nie istnieje. Aby go utworzyć i dodać do niego nowego użytkownika o loginie *admin1* należy wydać następujące polecenia:

```
# cd /etc/nagios3
# touch htpasswd.users
# chown www-data:www-data htpasswd.users
# chmod 640 htpasswd.users
# htpasswd -c -s htpasswd.users admin1
New password:
Re-type new password:
Adding password for user admin1
```

Argument *-c* polecenia *htpasswd* nakazuje utworzyć nowy plik z hasłami, *-s* każe użyć algorytmu SHA do obliczenia wartości skrótów haseł zamiast domyślnego (i nie zalecanego) algorytmu CRYPT.

Aby utworzonemu właśnie użytkownikowi przyznać pełne prawa administratora w Nagiosie należy wydać polecenie:

```
# sed -i -e 's,nagiosadmin,admin1,' /etc/nagios3/cgi.cfg
```

HTTPS

Powinno się także skonfigurować serwer WWW tak, aby zezwalał tylko na szyfrowany (HTTP over SSL) dostęp do aplikacji.

W poniższym przykładzie użyto prowizorycznego certyfikatu SSL, który jest generowany podczas instalowania Apache. Na systemie działającym produkcyjnie należy użyć certyfikatu wygenerowanego samodzielnie bądź zakupionego u komercyjnego dostawcy.

Należy wydać polecenia:

```
# a2enmod ssl
# a2ensite default-ssl
# rm /etc/apache2/conf.d/nagios3.conf
# sed -i -e 's,^ScriptAlias,#ScriptAlias,' \
/etc/nagios3/apache2.conf
```

Do pliku */etc/apache2/sites-available/default-ssl* na końcu, przed linijką *</VirtualHost>*, należy dopisać:

```
Include /etc/nagio3/apache2.conf
```

Po dokonaniu wymienionych zmian należy zrestartować serwer WWW:

```
# /etc/init.d/apache2 restart
```

Konfiguratory

Domyślnie cała konfiguracja Nagiosa przechowywana jest tylko i wyłącznie w plikach tekstowych. W Internecie znajdziemy jednak co najmniej kilka programów lub rozszerzeń które pozwolą nam na edytowanie konfiguracji programu za pomocą interfejsu graficznego i/lub przechowywanie konfiguracji w bazie danych.

Przechowywanie konfiguracji w bazie danych upraszcza tworzenie bardziej skomplikowanych instalacji systemu monitorującego, które składają się z wielu instancji (serwerów) Nagiosa. Wtedy konfiguracja przechowywana jest w jednym miejscu i nie musimy po każdej zmianie wgrywać jej osobno na każdy z serwerów. Zmiany dokonane w bazie danych widoczne są natychmiast we wszystkich instancjach Nagiosa. Oczywiście takie rozwiązanie ma także wady, gdyż baza danych staje się elementem krytycznym i jej awaria może unieruchomić cały system monitorujący. Ale i z tym możemy sobie poradzić jeśli uruchomimy dwie instancje bazy danych na osobnych serwerach fizycznych i włączymy synchronizację między nimi, bądź użyjemy innego natywnego dla danej bazy danych mechanizmu zapewniającego wysoką dostępność.

Wykresy

Część pluginów oprócz informacji o stanie hosta/usługi dostarcza także ilościowych informacji na temat stanu danej usługi. Dane te mogą zostać użyte przez zewnętrzne programy np. do tworzenia wykresów (użycia pamięci, dysków, procesora, itd.).

Jednym z programów wzbogacającym Nagiosa o możliwość tworzenia wykresów jest NagiosGrapher (http://sourceforge.net/projects/nagiosgrapher/).

Po zainstalowaniu i skonfigurowaniu go, Nagios będzie przekazywał do demona NagiosGraphera dane otrzymane od wszystkich pluginów wraz z nazwą hosta i usługi. NagiosGrapher posiada pliki z definicjami wykresów aby wiedział które z otrzymanych danych są istotne i w jaki sposób nanieść je na wykres. Pliki opisujące wykresy możemy tworzyć sami tak samo jak konfigurujemy Nagiosa.

Jeśli demon NagiosGraphera wykryje, że pojawił się nowy host bądź usługa, to tworzy dla niego nową dyrektywę konfiguracyjną Nagiosa, która w interfejsie WWW umieszcza obok hosta/usługi odnośnik do wykresu. W ten sposób mamy bezpośredni i szybki dostęp do wykresów z poziomu samego Nagiosa.

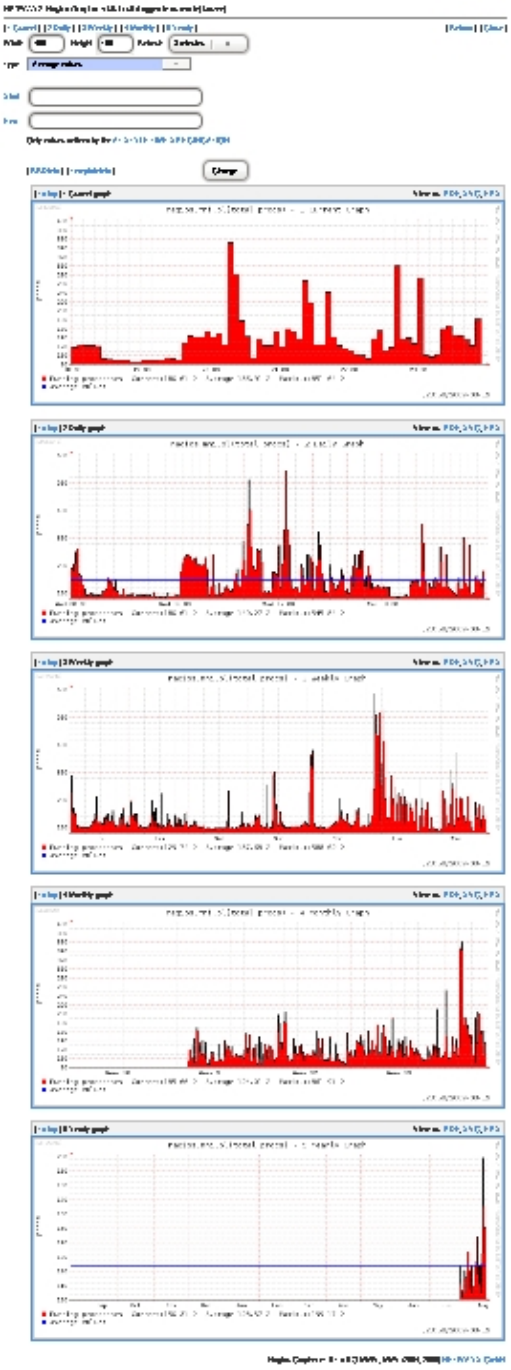
NagiosGrapher do przetrzymywania danych historycznych wykorzystuje narzędzie RRDtool[13]. Wykresy za każdym razem generowane są dynamicznie na żądanie poprzez wykorzystanie skryptu CGI (Rys. 3.3.).

Mapy sieci

Mapy są ważnym elementem dokumentacji sieci. Je także możemy zintegrować z Nagiosem dzięki dodatkowi NagVis[14]. Wymaga on do działania Nagiosa ze skonfigurowanym dodatkiem

13 oss.oetiker.ch/rrdtool/
14 www.nagvis.org

NDOutils (umożliwia on przechowywanie informacji o stanach obiektów w bazie danych MySQL), serwera WWW obsługą PHP, oraz programu Graphviz (jeśli chcemy korzystać z mapy automatycznej).



Rys. 3.3. NagiosGrapher – wykres ilości procesów z kilku przedziałów czasowych (od góry: 6 godzin, dzień, tydzień, miesiąc, rok).

3.2. NeDi

Strona domowa: www.nedi.ch.

3.2.1. Wstęp

Nazwa ta jest skrótem od *Network Discovery Suite*. Narzędzie to ułatwia codzienne czynności związane z administrowaniem siecią, szczególnie w drugiej warstwie modelu ISO/OSI.

3.2.2. Wymagania

Napisany jest w językach Perl (część systemowa - wykrywanie urządzeń, itd.) oraz PHP (interfejs WWW). Korzysta z bazy dany SQL w której przechowuje informacje o urządzeniach działających w sieci. Uruchamiany może być w systemie operacyjnym Linux.

3.2.3. Możliwości

Posiada bardzo wiele możliwości, które możemy podzielić na następujące kategorie:

Urządzenia

Pod hasłem tym rozumiemy urządzenia aktywne sieci: przełączniki oraz routery. NeDi w swojej bazie przechowuje m. in. następujące parametry tych urządzeń:

- adres IP interfejsu zarządzającego,
- rodzaj urządzenia,
- szczegóły systemu operacyjnego,
- lokalizację,
- uptime,
- lista zainstalowanych modułów fizycznych (o ile istnieją),
- połączenia do innych urządzeń,
- listę VLAN-ów,
- listę interfejsów sieciowych wraz z ich podstawowymi parametrami konfiguracyjnymi oraz statystykami,
- listę interfejsów wraz z ich parametrami,
- listę modułów fizycznych (ta funkcja jest przydatna jeśli dysponujemy większymi, składającymi się z modułów urządzeniami).

Na podstawie wymienionych parametrów możemy przeszukiwać bazę urządzeń.

Korzystając z RRDtool program tworzy wykresy użycia procesora, pamięci oraz obciążenia poszczególnych interfejsów (Rys. 3.4.).

Rys. 3.4. NeDi pozwala dostosowywać przedział czasowy wykresów oraz ich rozmiar.

Poprzez interfejs WWW możemy wydawać zdalnie polecenia na urządzeniach.

Jedną z ciekawszych funkcji jest możliwość śledzenia kontraktów serwisowych Cisco. Dzięki temu w razie awarii wiemy które urządzenia są jeszcze wspierane przez producenta.

Urządzenia sieciowe możemy umieszczać w magazynie i przypisywać je do określonych użytkowników. Usprawnia to zarządzanie dużą ilością sprzętu.

Topologia

NeDi dostarcza nam narzędzi m. in. do sprawdzania tras routingu, stanu Spanning Tree.

Jeśli nasze switche lub routery obsługują protokoły takie jak CDP (Cisco Discovery Protocol) lub LLDP (Link Layer Discovery Protocol, to program będzie potrafił samodzielnie zbudować mapę sieci. Istnieje możliwość samodzielnego dodawania i edytowania połączeń między urządzeniami.

Węzły

Węzłem jest każdy adres MAC. Możemy przeszukiwać i wyświetlać wszystkie z nich. Dla każdego z węzłów dostępne są dodatkowe charakterystyki. Oto ważniejsze z nich:

- adres IP,
- nazwa DNS,
- producent karty sieciowej (tzw. OUI),
- VLAN.

Mamy możliwość śledzenia historii danego węzła. Po wpisaniu jego adresu w specjalnym formularzu, NeDi sprawdzi czy działają na nim takie usługi jak NetBios, HTTP, HTTPS, SSH oraz

Telnet, wyświetli wykres użycia interfejsu przełącznika do którego podłączony jest węzeł i wyświetli informacje historyczne mówiące o tym czy dany węzeł zmieniał adres IP lub przełącznik.

Z poziomu interfejsu WWW możemy uruchamiać narzędzia takie jak ping, traceroute, whois i nmap.

NeDi pozwala rejestrować skradzione adresy MAC i IP. W ten sposób możemy szybko namierzyć użytkownika, który ręcznie przypisał sobie nie należący do niego adres IP.

Raporty

NeDi ma możliwość tworzenia raportów na następujące tematy:

- sieci IP (ich lista oraz ilość urządzeń w każdej z nich),
- modele urządzeń sieciowych,
- rodzaje i wersje systemów operacyjnych na urządzeniach sieciowych,
- domeny i tryby VTP,
- rodzaje i liczebność modułów fizycznych w urządzeniach,
- liczba używanych portów na poszczególnych przełącznikach (Rys. 3.5.),
- ilość kart sieciowych podzielona wg. producentów,
- zmiany/porzucenia adresów IP,
- ilość adresów MAC przypadająca na pojedynczy interfejs przełącznika.

Monitoring

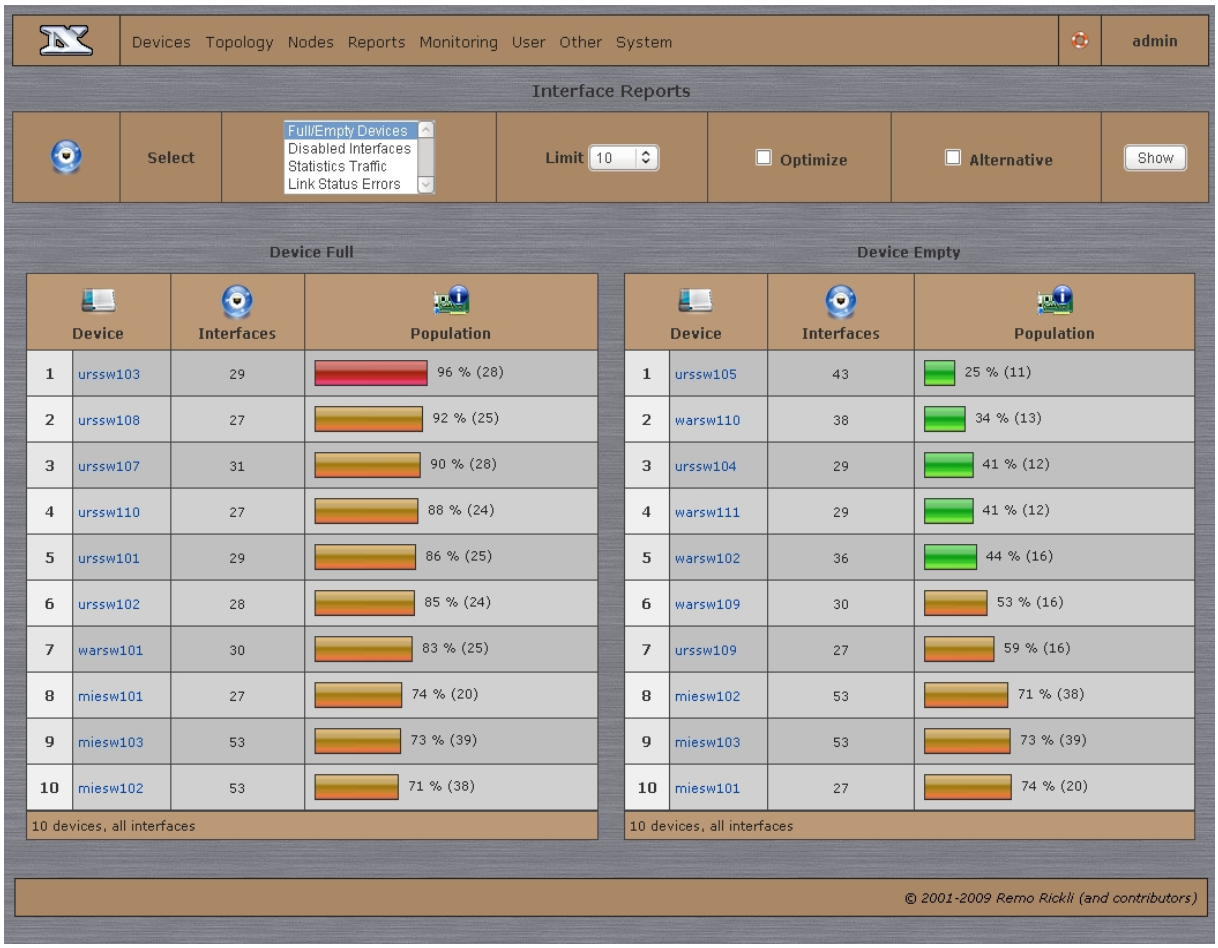
Wszystkie komunikaty z urządzeń sieciowych są przez NeDi rejestrowane i prezentowane w przejrzystej formie. Dzięki temu mamy możliwość szybkiego reagowania na wszelkie nieprawidłowości występujące w sieci.

Komunikaty możemy filtrować i sortować na podstawie następujących kryteriów:

- data,
- źródło,
- treść,
- typ,
- poziom.

Komunikaty od urządzeń NeDi może odbierać na kilka sposobów:

- SNMP Trap,
- Syslog,
- automatyczne logowanie na urządzenia i odczytywanie zawartości dziennika.



Rys. 3.5. Przykładowy raport ilości używanych portów na przełącznikach.

Użytkownicy

Program posiada własną bazę użytkowników. Nie istnieje możliwość wykorzystania zewnętrznej bazy kont.

Każdy z użytkowników posiada m. in. takie właściwości jak:

- nazwa,
- e-mail,
- telefon,
- grupy.

NeDi posiada zestaw pre-definiowanych grup, w których możemy umieszczać użytkowników. Każda grupa posiada inny zestaw uprawnień.

Integracja z Cacti

Aplikację możemy zintegrować z Cacti. Daje to nam możliwość dodawania do tego drugiego programu urządzeń wraz z wykresami za pomocą zaledwie kilku kliknięć myszką. Nie musimy wtedy przechodzić całego procesu dodawania urządzenia do Cacti, wykrywania interfejsów i tworzenia wykresów dla każdego z nich.

3.2.4. Opis działania

Aplikacja automatycznie znajduje w naszej sieci przełączniki i routery i dodaje je do bazy urządzeń. Oprócz tego ewidencjonuje każdy napotkany adres MAC. Skanowanie sieci wykonywane jest regularnie dzięki temu możemy śledzić na bieżąco wszystkie zmiany topologii.

3.3. Cacti

Strona domowa: www.cacti.net.

3.3.1. Wstęp

Jest to bardzo rozbudowane narzędzie służące do rysowania wykresów. Zwykle Cacti służy do prezentowania użycia interfejsów sieciowych (Rys. 3.6.), pamięci i procesora w urządzaniach aktywnych sieci (routery, przełączniki, firewalle, itd.). Dane odczytywane są przeważnie za pomocą protokołu SNMP.

3.3.2. Wymagania

Do działania wymaga RRDtool, bazy danych MySQL oraz serwera WWW (np. Apache) z obsługą języka PHP.

3.3.3. Możliwości

Podobnie jak w przypadku Nagiosa, wszystkie czynności zbierające dane wykonywane są przez zewnętrzne programy. Mamy więc możliwość dowolnego rozszerzenia funkcjonalności i możliwości programu. Prowadzi to do tego, że aplikacja może monitorować niemalże wszystko w dowolny sposób. Program posiada interfejs WWW napisany w PHP za pomocą którego możemy wykonywać wszystkie czynności konfiguracyjne oraz przeglądać zgromadzone dane.

Cacti może autoryzować użytkowników na podstawie własnej bazy kont lub korzystając z zewnętrznego serwera LDAP. Każdemu z użytkowników możemy nadawać uprawnienia do poszczególnych urządzeń i wykresów.

Obsługiwane jest SNMP w wersjach 1, 2, 3. Urządzenia mogą być odpytywane za pomocą zewnętrznego klienta SNMP, wbudowanego w język PHP interfejsu do SNMP lub jeśli używamy pollera Spine, to za pomocą wywołań API dostarczanych przez pakiety `ucd-snmp` oraz `net-snmp`. Przy czym interfejs wbudowany w PHP obsługuje protokół SNMP tylko w wersji 1.

Społeczność zgromadzona wokół Cacti jest duża i rozwinięta. Na stronie domowej oraz forum programu (http://forums.cacti.net) znajdziemy wiele szablonów, skryptów i rozszerzeń napisanych przez użytkowników, które nie wchodzą w skład paczki instalacyjnej. Dzięki temu możemy rozbudowywać Cacti o znajomość nowych urządzeń, nowych rodzajów wykresów oraz uzyskać dodatkową funkcjonalność.

Rys. 3.6. Wykresy obciążenia portów przełącznika.

Poniżej znajduje się lista bardziej użytecznych dodatków:

- **boost**

  Zwiększa wydajność dużych instalacji Cacti między innymi przechowując wygenerowane wykresy w pamięci podręcznej.

- **discovery**
  Umożliwia automatyczne wykrywanie nowych urządzeń w sieci IP.
- **monitor**
  Pokazuje stan wszystkich urządzeń i powiadamia w razie awarii.
- **routerconfigs**
  Automatyzuje robienie kopii zapasowych konfiguracji routerów.
- **weathermap**
  Pozwala tworzyć mapy sieci wraz z naniesionymi informacjami o obciążeniach łącz.
- **nmidcreatepdf**
  Tworzenie raportów w plikach PDF.

3.3.4. Opis działania

Dane zbierane są w określonych odstępach czasu (domyślnie 5 minut) i zapisywane są w plikach RRD. Dane z urządzeń pobierane są przez tzw. poller. Domyślnym pollerem jest cmd.php, który odczytuje dane z każdego urządzenia po kolei. Jeśli sieć składa się z wielu urządzeń to cmd.php może nie zdążyć w ciągu tego przedziału czasu odpytać wszystkie z nich. Dlatego opcjonalnie można użyć pollera o nazwie Spine. Napisany jest w języku C, wykorzystuje wątki POSIX i został zaprojektowany jako szybszy zamiennik cmd.php.

Budowa aplikacji opiera się na trzech głównych elementach:

- Pobieranie Danych
  Dane muszą najpierw zostać odczytane z urządzeń. Ta czynność wykonywana jest przez poller.
- Przechowywanie Danych
  Pobrane dane są zapisywane w plikach RRD aby można było je później przetworzyć.
- Prezentacja Danych
  Cacti wykorzystuje wbudowaną w RRDtool możliwość tworzenia wykresów. Na podstawie zgromadzonych danych historycznych tworzone są wykresy. Dzięki zaawansowanym funkcjom RRD wykresy mogą być tworzone na podstawie danych z prawie dowolnego przedziału czasu. Ponieważ Cacti obsługiwane jest z poziomu przeglądarki WWW, to możliwe jest używanie go praktycznie z każdego systemu operacyjnego.

Zadanie aplikacji napisanej w PHP polega na połączeniu powyższych trzech elementów w spójną całość.

Cała logika tworzenia wykresów zbudowana jest na następujących elementach:

- Data Template
  Definiuje strukturę w jakiej przechowywane będą dane. Po nałożeniu go na konkretnego hosta uzyskamy wyjściowy plik RRD.
- Data Source Item
  Jest to część Data Template. Plik RRD może przechowywać więcej niż jedną zmienną. Data

Source Item jest właśnie taką zmienną.

- Graph Template

  Deklaracja wykresu RRD. Zostanie zastosowana do konkretnego hosta aby utworzyć końcowy wykres.
- Graph Template Item

  Jest częścią Graph Template. Definiuje linie, obszary i napisy w legendzie grafu.
- Graph Końcowa deklaracja wykresu. Uzyskujemy ją poprzez nałożenie Data Template na konkretne urządzenie.

W wielu miejscach używane są szablony, oszczędza to czas przy dodawaniu kolejnych urządzeń. Gdy zmieniony zostanie szablon wykresu, to zmiany zostaną zastosowane do wszystkich istniejących wykresów utworzonych na jego podstawie. Zmiany dokonane w szablonie danych nie są propagowane do utworzonych na jego podstawie plików RRD. Zawartość tych plików można zmienić, ale tylko samodzielnie za pomocą programu *rrdtool* z poziomu wiersza poleceń.

Na trochę innej zasadzie działają szablony hostów. Służą one do łączenia ze sobą źródeł danych i szablonów wykresów z danym typem hosta. Wtedy po nałożeniu szablonu hosta na konkretne urządzenie wszystkie zdefiniowane wykresy utworzą się automatycznie. Zmiany dokonane w szablonie hosta nie są propagowane do istniejących już urządzeń korzystających z tego szablonu.

3.4. SmokePing

Strona domowa: oss.oetiker.ch/smokeping/.

3.4.1. Wstęp

Narzędzie to bada ogólny stan infrastruktury sieciowej poprzez mierzenie opóźnień oraz strat pakietów na łączach. Podobnie jak w przypadku poprzednich narzędzi wyniki pomiarów udostępniane są przez WWW i prezentowane w formie wykresów.

Program napisany jest w języku skryptowym Perl, jego interfejs WWW zrealizowany jest w technologii CGI.

3.4.2. Wymagania

Do działania programu wymagane są następujące aplikacje i biblioteki:

- RRDtool[15] - gromadzenie danych historycznych i rysowanie wykresów.
- FPing[16] - podstawowa próbka sprawdzająca działanie hostów za pomocą protokołu ICMP.
- libwww-perl[17] - moduł do Perla. Potrzebny m. in. do komunikacji pomiędzy główną i podrzędną instancją SmokePing.
- Serwer WWW z obsługą CGI.

15 oss.oetiker.ch/rrdtool/
16 fping.sourceforge.net
17 Ten oraz wszystkie pozostałe moduły do Perla, znajdziemy na stronie www.cpan.org.

- Perl - interpreter języka.

Opcjonalnie możemy zainstalować także poniższe programy aby zwiększyć funkcjonalność aplikacji:

- EchoPing[18] - wymagany do działania wszystkich próbek EchoPing. Obejmuje różne protokoły.
- Curl[19] - sprawdzanie działania serwera HTTP i FTP.
- dig[20] - sprawdzanie działania serwera DNS.
- SSH[21] - sprawdzanie działania serwera SSH.
- Socket6 - moduł do Perla. Dodaje możliwość sprawdzania hostów, które mają tylko adresy IPv6.
- Net::Telnet - moduł do Perla. Dodaje możliwość sprawdzania działania usługi Telnet.
- Net::DNS - moduł do Perla. Dodaje alternatywny sposób na sprawdzanie działania serwerów DNS.
- Net::LDAP - moduł do Perla. Dodaje możliwość sprawdzania usługi LDAP.
- IO::Socket::SSL - ta biblioteka Perla jest wymagana jeśli chcemy używać protokołu LDAP z TLS (dodatkowa warstwa szyfrowania).
- Authen::Radius - moduł do Perla. Dodaje możliwość sprawdzania działania usługi Radius.
- SpeedyCGI[22] - przyśpiesza działanie programów CGI napisanych w Perlu.

3.4.3. Możliwości

Aplikacja, podobnie jak Nagios, posiada możliwość wysyłania powiadomień. Tak samo jak w Cacti możemy interaktywnie przybliżać wykresy zaznaczając kursorem obszar, który nas interesuje. Razem z programem dostarczany jest duży zbiór rozszerzeń. Możemy mierzyć nie tylko czas odpowiedzi na pakiety ICMP, ale także czas odpowiedzi różnych aplikacji.

Rodzaje wykresów

Każdy wykres zawiera informacje o jednym lub większej ilości celów. Celem zwykle jest adres IP, ale może to być praktycznie dowolny obiekt - np. aplikacja.

W programie możemy wyróżnić trzy główne rodzaje wykresów:

- Ogólny

  Zawiera bieżące wartości pomiarów dla danego celu. Jeżeli cel jest badany zarówno z serwera głównego jak i zapasowego to są pokazane wyniki pomiarów z obu z nich. W trybie wielohostowym widoczne są wszystkie z hostów. Na wykresie zobaczymy dodatkowo takie statystyki jak *av md* (średnia mediana), *av ls* (średnia strata), *av sd* (średnie odchylenie

18 echoping.sourceforge.net
19 curl.haxx.se
20 dig to polecenie wchodzące w skład oprogramowania BIND. Strona domowa: www.isc.org/software/bind.
21 www.openssh.com
22 daemoninc.com/SpeedyCGI

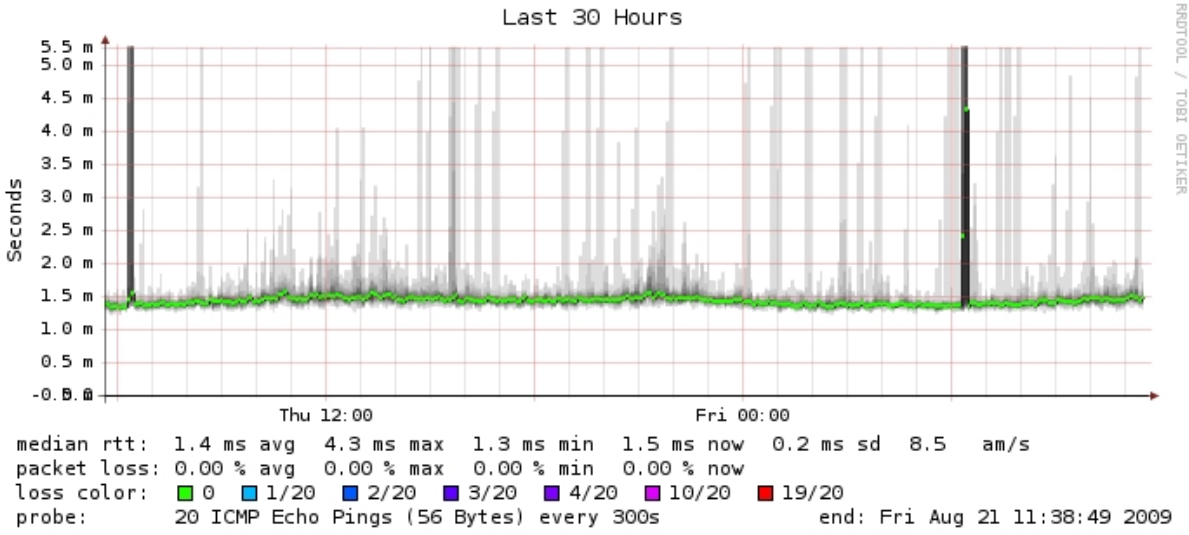
standardowe wielu pomiarów w każdej z rund), *am/as* (stosunek średniej mediany do średniego standardowego odchylenia).

- Szczegółowy

  Obrazuje kilka parametrów dla pojedynczego celu. Kolor pionowych kresek oznacza wielkość strat pakietów. Szare odcienie oznaczają rozpiętość poszczególnych próbek w ramach jednego pomiaru. Zaznaczając obszar na wykresie możemy zawęzić przedział czasowy. Zawiera następujące dodatkowe statystyki: *avg* (średnia mediana), *max* (maksymalna mediana), *min* (minimalna mediana), *now* (bieżąca mediana), *sd* (odchylenie standardowe mediany), *am/s* (stosunek średniej mediany do standardowego odchylenia. Przykładowy wykres znajduje się na Rys. 3.7.

- Wielohostowy

  Większa wersja wykresu ogólnego. Zawiera informacje na temat dwóch lub więcej celów. Dodatkowo zawiera następujące statystyki: *av md* (średnia mediana), *av ls* (średnia strata), *av sd* (średnie standardowe odchylenie), *am/as* (stosunek średniej mediany do średniego standardowego odchylenia).



Rys. 3.7. Wykres badania stanu łącza za pomocą pakietów ICMP.

Architektura rozproszona

Zwykle SmokePing działa na jednym serwerze, z którego sprawdza wszystkie zdefiniowane cele. Jeśli mamy potrzebę zainstalowania kilku instancji programu, aby np. badać wydajność sieci w kilku miejscach bądź dostępność usługi z kilku lokalizacji, to możemy do tego wykorzystać architekturę rozproszoną SmokePinga. Nie musimy wtedy konfigurować osobno każdej instancji programu. Jedna z instancji staje się główną (ang. *master*) i zawiera kompletną konfigurację. Kontroluje ona na tej podstawie zachowanie instancji podrzędnych (ang. *slave*).

Wszystkie dane historyczne przechowywane są na serwerze głównym. Serwer ten służy także

do wyświetlania wykresów. Instancje podrzędne pobierają swoją konfigurację z serwera głównego – dzięki temu cała konfiguracja przechowywana jest w jednym miejscu i automatycznie propagowana do wszystkich instancji programu.

Każdy z serwerów podrzędnych komunikuje się z serwerem głównym poprzez interfejs WWW programu. Zaraz po uruchomieniu łączy się z nim aby uzyskać listę czynności do wykonania. Po sprawdzeniu wszystkich celów wyniki zgłasza do serwera głównego. Jeśli w międzyczasie konfiguracja uległa zmianie to zostanie to zakomunikowane serwerowi podrzędnemu zaraz po zgłoszeniu przez niego wyników.

Jeśli serwer podrzędny nie będzie mógł połączyć się z serwerem głównym, to zapisze wyniki na dysku i spróbuje zgłosić je podczas ponownego połączenia. Ponieważ wyniki będą zapisane na dysku, to zostaną zapamiętane pomiędzy restartami aplikacji.

3.4.4. Opis działania

SmokePing mierzy jaki czas jest potrzebny aby pakiet dotarł do celu i wrócił. Podczas każdego pomiaru wysyła kilka pakietów, mierzy ich czas powrotu, wyniki sortuje i wybiera medianę (środkowy pomiar). Wartości pozostałych czasów na wykresie widoczne są jako delikatny cień na tle głównej linii. Zdarza się, że po wysłaniu pakietu ICMP program nie otrzymuje odpowiedzi. Oznacza to, że nastąpiła strata pakietu. Powoduje to zmianę koloru linii mediany. Łącząc ze sobą te wszystkie parametry możemy uzyskać obraz ogólnego stanu sieci. Duże straty pakietów mogą świadczyć o wadliwych urządzeniach sieciowych, a długie czasy powrotu pakietu o mało wydajnej infrastrukturze.

Możemy mierzyć funkcjonowanie różnych warstw modelu ISO/OSI. Nie tylko warstwy sieci bądź transportu. Pomiary dokonywane w warstwie aplikacji sprawdzają zarówno sprawność aplikacji jaki i wszystkich warstw znajdujących się poniżej - między innymi warstwy sieci.

3.5. phpLogCon

Strona domowa: www.phplogcon.org.

3.5.1. Wstęp

Program służy do przeglądania logów systemowych zgromadzonych w bazie danych SQL bądź plikach tekstowych.

3.5.2. Wymagania

Aplikacja jest napisana w języku PHP w związku z tym wymaga serwera WWW z jego obsługą. Obsługujemy ją za pomocą przeglądarki WWW. Konfiguracja może być przechowywana w pliku bądź w bazie danych.

3.5.3. Możliwości

Aplikacja rozpoznaje kilka formatów zapisu logów w plikach. Obsługuje także standardowy format znany z systemów Uniksowych. Mamy zupełną dowolność w wyborze sposobu dodawania logów do bazy. Autorzy programu sugerują użycie serwera rsyslog lub WinSyslog, które posiadają wbudowaną obsługę bazy danych, ale nie jest to koniecznością. Najważniejsze jest aby schemat bazy danych był zgodny z tym, który jest używany przez phpLogCon.

W wielu miejscach generowane są kontekstowe odnośniki oraz menu. Pomagają one szybko odnaleźć co się kryje np. pod danym adresem IP. Można także skonfigurować zewnętrzną bazę wiedzy (ang. *knwoledge base*) do której będą generowane linki.

Źródłem komunikatów może być praktycznie wszystko. Np. urządzenia sieciowe, Linux, Windows, itd.

Jeśli użyty zostanie system userDB (przechowywanie konfiguracji programu w bazie danych), który domyślnie nie jest włączony, to będzie można tworzyć konta użytkowników. phpLogCon rozróżnia dwa typy kont: administratorzy i zwykli użytkownicy. Istnieje możliwość tworzenia grup użytkowników i przypisywania uprawnień do nich. Można np. utworzyć osobne grupy z osobnymi uprawnieniami do przeglądania dzienników serwerów pocztowych i firewalli.

3.5.4. Opis działania

Bezpieczeństwo

Serwer na którym działa phpLogCon nie powinien być dostępny z Internetu. Najlepiej umieścić go w zaufanej sieci lokalnej, do której dostęp mają tylko odpowiednie osoby (np. administratorzy i operatorzy). Dodatkowe bezpieczeństwo zapewni łączenie się do aplikacji wyłącznie za pomocą protokołu HTTPS. Użycie mechanizmu userDB umożliwi tworzenie dedykowanych kont dla użytkowników zabezpieczonych hasłami, ale nie powinien on być używany jako jedyny środek zabezpieczający przed niepowołanym dostępem. Zabezpieczenia najlepiej jest wdrożyć w kilku warstwach.

Wydajność

Jeśli systemy generują bardzo dużą ilość komunikatów, to bardziej wydajne jest przechowywanie ich w plikach tekstowych niż w bazie danych.

phpLogCon dostarcza gotowe narzędzia do usuwania starych rekordów z tabel w bazach danych. Pierwsze z nich jest interaktywne i działa z poziomu przeglądarki, a drugie jest nieinteraktywne i może być uruchamiane z poziomu konsoli, co pozwala wywoływać je także automatycznie z CRON-a. Należy dbać o regularne kasowanie starych wpisów gdyż to przyśpiesza działanie programu.

Wyszukiwanie

Tworząc frazy do wyszukania w komunikatach można posługiwać się składnią, która pozwala na tworzenie nawet skomplikowanych warunków. Dostępny jest także formularz, który pomaga w tworzeniu filtrów wyszukiwania. Przeszukiwać można wszystkie dostępne pola komunikatów (data, priorytet, host, program, PID, typ, treść, itd.). Obsługiwane są wyrażenia regularne. W razie potrzeby dostępne są operatory logiczne (negacja, alternatywa, koniunkcja).

3.6. Arpwatch

Strona domowa: http://www-nrg.ee.lbl.gov/.

3.6.1. Wstęp

Aplikacja została stworzona przez Network Research Group w Lawrence Berkeley National Laboratory. Rozpowszechniana jest na licencji BSD. Umożliwia śledzenie zmian w powiązaniach adresów MAC z adresami IP w sieci Ethernet.

3.6.2. Wymagania

Do działania aplikacja wymaga biblioteki libpcap[23] która została także napisana przez Network Research Group w Lawrence Berkeley National Laboratory.

3.6.3. Możliwości

Program ten monitoruje sieć w poszukiwaniu nowych powiązań pomiędzy adresami MAC i IP oraz zmian w tych powiązaniach. W efekcie dzięki temu możemy wykryć nowe urządzenia podłączane do naszej sieci oraz konflikty IP (gdy więcej niż jedna karta sieciowa posługuje się tym samym adresem IP). Jest to bardzo ważne z punktu widzenia bezpieczeństwa oraz stabilności sieci. Każdy konflikt adresów IP w przypadku serwerów doprowadza w praktyce do zaprzestania świadczenia usługi.

3.6.4. Opis działania

O wszystkich tych sytuacjach administrator powiadamiany jest za pomocą e-maili, wszystkie komunikaty zapisywane są też w dziennikach systemowych. Dodatkowym ułatwieniem jest to, że program rozpoznaje producentów kart sieciowych na podstawie adresów MAC co czasami może pomóc w szybkim znalezieniu przyczyny problemu. Mamy możliwość ignorowania ruchu z całego interfejsu sieciowego lub tylko z wybranego zakresu adresów. W ten sposób możemy np. ignorować sieci w których konflikty adresów IP są częste i niegroźne w skutkach. Wszystkie znalezione powiązania MAC-IP zapisywane są w pliku na dysku wraz z datą i godziną ostatniego pojawienia się w sieci. Ruch sieciowy może zostać odczytany z pliku w formacie *pcap*.

23 www.tcpdump.org

4. ZADANIA OPERATORA

W tym rozdziale przedstawimy w jaki sposób operator będzie wykorzystywał opisane narzędzia. Część firm posiada specjalne działy bądź zespoły operatorów, którzy są pracownikami technicznymi i pośredniczą pomiędzy użytkownikami systemów informatycznych, a administratorami. Zajmują się odbieraniem telefonów, poczty elektronicznej oraz przyjmowaniem zgłoszeń na dowolne inne sposoby. Taki podział obowiązków jest często praktykowany gdyż praca administratora jest w dużej mierze koncepcyjna i przyjmowanie zgłoszeń oraz śledzenie systemów monitorujących może znacznie ją utrudnić.

4.1. Bieżące

Operatorzy powinni na bieżąco:

- sprawdzać stan hostów i usług w Nagiosie. Mogą to robić albo poprzez śledzenie awarii za pomocą interfejsu WWW bądź czytając przychodzące powiadomienia,
- sprawdzać komunikaty o stanie sieci pojawiające się w NeDi,
- przeglądać logi w phpLogCon,
- czytać powiadomienia od Arpwatcha bądź sprawdzać dziennik systemowy, w którym jego komunikaty są zapisywane.

4.2. Cykliczne

Cyklicznie (np. raz na miesiąc) operator powinien:

- przygotowywać za pomocą NeDi raporty na temat ilości użytych portów sieciowych na routerach i przełącznikach w celu zaplanowania przyszłych zakupów urządzeń sieciowych.
- dbać o to aby wszystkie urządzenia miały wykupioną opiekę serwisową oraz wszystkie niezbędne licencje. W tym celu można wykorzystać w NeDi funkcję inwentaryzacji urządzeń.
- sprawdzać za pomocą SmokePing stan wszystkich łącz do Internetu z których korzysta firma. Należy sprawdzać czy łącza zapewniają potrzebną przepustowość i niezawodność.

4.3. W razie potrzeby

Gdy zaistnieje taka potrzeba (np. zgłoszenie awarii):

- za pomocą Cacti i SmokePing operator może sprawdzić obciążenia portów na przełącznikach oraz sprawność łącz WAN.
- zaplanować okno serwisowe w Nagiosie, konfigurując planową przerwę w działaniu hosta/usługi.
- przygotować w Nagiosie raport na temat dostępności hosta bądź usługi.
- odnaleźć określone adresy MAC za pomocą NeDi, dodać je do listy ukradzionych węzłów,

- reagować na wszelkiego rodzaju zgłoszenia.

W razie wykrycia awarii lub problemu z którego rozwiązaniem operator nie jest w stanie sobie poradzić, przekazuje odpowiednią informację administratorowi.

4.4. Przykłady

Poniżej znajdują się przykłady spotykanych w życiu sytuacji.

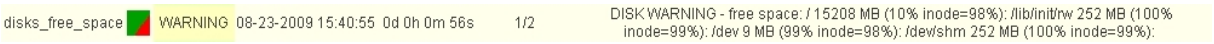
4.4.1. Przykład 1: Sprawdzanie wolnego miejsca na dysku - Nagios

Zakładamy, że do sprawdzania ilości wolnego miejsca na dysku Nagios używa następującego polecenia:

```
/usr/lib/nagios/plugins/check_disk -w 10% -c 5%
```

Polecenie to sprawdza wszystkie lokalne dyski. W przytoczonym przykładzie jeśli wolnego miejsca jest mniej niż 5%, to zgłaszana jest sytuacja krytyczna. Jeśli wolnego miejsca jest więcej niż 5%, ale mniej niż 10%, to zgłaszane jest ostrzeżenie.

Można monitorować wolną przestrzeń dla każdego dysku z osobna, ale niesie to za sobą konieczność definiowania w Nagiosie osobnej usługi dla każdego z dysków. Sprawdzanie za jednym razem wszystkich dysków oszczędza czas poświęcony na konfigurację programu (oraz ewentualną rekonfigurację w razie zmiany ilości dysków) oraz sprawia, że Nagios musi sprawdzać mniej usług (serwer jest mniej obciążony).



Rys. 4.1. Ostrzeżenie w Nagiosie o małej ilości wolnej przestrzeni na jednym z dysków.

Gdy operator ujrzy w interfejsie WWW Nagiosa komunikat podobny do widocznego na Rys. 4.1., najpierw musi zalogować się na serwer, którego dotyczy ostrzeżenie i sprawdzić czy można zwiększyć ilość wolnego miejsca usuwając niepotrzebne pliki. Najodpowiedniejszą osobą do stwierdzenia, które dane są niepotrzebne jest administrator serwera, który ewentualnie skieruje operatora do osoby mogącej wskazać pliki do skasowania. Jeśli usunięcie jakichkolwiek danych nie jest możliwe, to należy rozważyć dołożenie dodatkowej przestrzeni dyskowej co wiąże się przeważnie z zaplanowaniem okna serwisowego i wyłączeniem serwera.

W przypadku systemu Uniksowego, aby znaleźć katalog zajmujący najwięcej miejsca możemy posłużyć się następującą metodą:

Lokalizujemy dysk na którym kończy się wolne miejsce. Wydajemy polecenie `df -h`:

```
# df -h
System plików         rozm. użyte dost. %uż. zamont. na
/dev/sda2             7,3G  4,7G  2,3G  68% /
tmpfs                 507M     0  507M   0% /lib/init/rw
udev                   10M   40K   10M   1% /dev
tmpfs                 507M     0  507M   0% /dev/shm
/dev/sda4              26G  26G   0   100% /usr/local/pgsql/data
```

Jak widzimy w katalogu `/usr/local/pgsql/data` nie ma już wolnego miejsca. Wchodzimy więc do niego i za pomocą programu `du` sprawdzamy co zajmuje najwięcej miejsca:

```
# cd /usr/local/pgsql/data
# du -s * | sort -n
4       lost+found
4       pg_hba.conf
4       pg_ident.conf
4       pg_tblspc
4       pg_twophase
4       PG_VERSION
4       postmaster.opts
4       postmaster.pid
16      postgresql.conf
28      pg_multixact
136     pg_subtrans
15760   pg_clog
85516   pg_log
352444  global
557744  pg_xlog
9227100 base
```

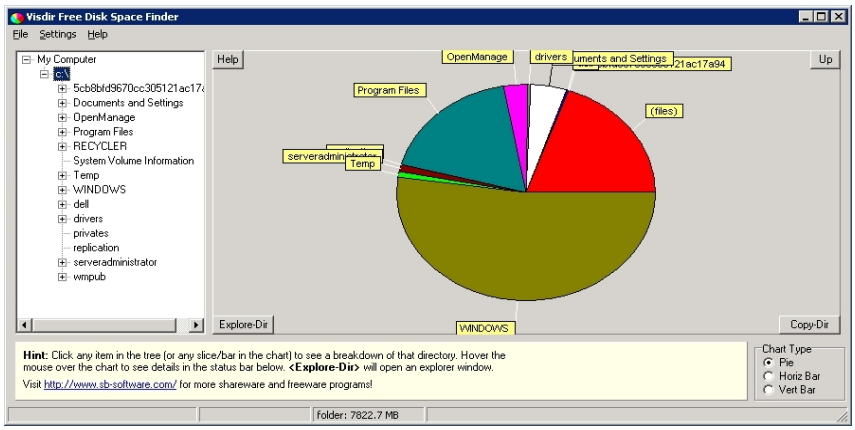
Katalog o nazwie `base` zajmuje najwięcej miejsca. Wchodzimy do niego i znów za pomocą `du` sprawdzamy które pliki/katalogi zajmują najwięcej miejsca.

```
# cd base
# du -s * |sort -n
3736    10818
3816    1
3832    10819
3912    55887634
3984    55901760
5088    55861014
5136    64590570
5668    55846475
6052    64659926
6420    37636178
9500    55815029
208000  64610334
250908  55825090
337488  64581978
525500  53859937
1425628 20793641
1783200 63730890
4639228 40140800
```

Jak możemy się domyślić są to katalogi wewnętrzne serwera PostgreSQL i jeśli operator nie dysponuje szczegółową wiedzą na temat tego serwera bazodanowego, to nie powinien nic kasować. W tym przypadku musi skontaktować się z administratorem serwera bądź administratorem bazy danych i przekazać mu informacje o tym, które katalogi zajmują najwięcej miejsca.

Często wiele miejsca na dysku serwera zajmują dzienniki systemowe (katalog */var/log*) lub dzienniki aplikacji. Lokalizacja tych drugich zależna jest od aplikacji – należy skonsultować się w tej sprawie z osobami odpowiedzialnymi za działające na serwerze aplikacje.

W przypadku serwera z systemem operacyjnym Windows w podobny sposób możemy użyć programu VisDir[24]. Pokazuje on takie same informacje, co `du`, ale w sposób graficzny (Rys. 4.2.). Aby go użyć musimy zalogować się poprzez konsolę graficzną do serwera – tzn. bezpośrednio bądź np. za pomocą zdalnego pulpitu lub VNC.



Rys. 4.2. VisDir pokazujący rozmiary katalogów na dysku C:.

Być może z dysku serwera można skasować inne pliki, które nie zajmują najwięcej przestrzeni, ale ich usunięcie zwolni wystarczającą ilość miejsca. Są to jednak przypadki bardzo indywidualne i wymają dokładniejszej wiedzy na temat serwera. W takiej sytuacji należy sprawdzić nie tylko te katalogi, które zajmują najwięcej miejsca.

4.4.2. Przykład 2: Sprawdzanie błędu połączenia TCP - Nagios

Gdy w Nagiosie pojawi się komunikat o tym, że nie można nawiązać połączenia z portem TCP (Rys. 4.3.), to operator powinien to w pierwszej kolejności zweryfikować ręcznie.

http	CRITICAL	08-23-2009 14:53:34	5d 9h 12m 44s	1/2	Brak trasy do hosta
mem	CRITICAL	08-23-2009 14:53:34	5d 9h 12m 34s	1/2	Connection refused or timed out
resolv_conf	CRITICAL	08-23-2009 14:52:41	5d 9h 12m 33s	1/2	Connection refused or timed out
ssh	CRITICAL	08-23-2009 14:52:41	5d 9h 12m 26s	1/2	Brak trasy do hosta
time	CRITICAL	08-23-2009 14:52:41	5d 11h 44m 39s	1/2	Connection refused or timed out

Rys. 4.3. Nagios – przykładowe komunikaty których przyczyną może być błąd w nawiązywaniu połączenia TCP.

24 www.sb-software.com/visdir/

Sprawdzenia najlepiej jest dokonać z tego samego serwera na którym działa Nagios. Można to zrobić z innego komputera o ile firewalle po drodze nie blokują takich połączeń (wymagana tu jest dokładniejsza znajomość danego środowiska aby to stwierdzić) lub komputer z którego dokonujemy sprawdzenia znajduje się w tej samej domenie rozgłoszeniowej co komputer, który sprawdzamy.

Zakładając, że Nagios stwierdził niemożność połączenia się z portem 1234 na serwerze o adresie IP 10.0.0.10, należy wydać następujące polecenie:

```
# telnet 10.0.0.10 1234
```

Prawidłowo nawiązane połączenie TCP powinno spowodować pojawienie się takich komunikatów:

```
# telnet 10.0.0.10 1234
Trying 10.0.0.10...
Connected to 10.0.0.10.
Escape character is '^]'.
```

Aby wyjść z programu telnet naciskamy *[Ctrl]+[]]* co spowoduje przejście do wiersza poleceń tego programu. Następnie wydajemy polecenie `quit`:

```
telnet> quit
```

Spowoduje to powrót do wiersza zachęty powłoki.

Jeśli pakiety będą ignorowane przez firewall to wydanie polecenia `telnet 10.0.0.10 1234` spowoduje tylko pojawienie się napisu `Trying 10.0.0.10...` i program będzie oczekiwał aż do przekroczenia czasu oczekiwania na odpowiedź, co objawia się komunikatem `telnet: Unable to connect to remote host: Connection timed out`.

Jeśli pakiety będą jawnie odrzucane przez serwer docelowy objawi się to komunikatem `telnet: Unable to connect to remote host: Connection refused`. Może to być spowodowane również przez firewall, który nie zezwala na takie połączenia bądź przez to, że aplikacja nasłuchująca na danym porcie przestała działać.

W takiej sytuacji należy sprawdzić czy pakiety docierają do serwera docelowego. Jeśli operator ma dostęp do niego to może zrobić to samodzielnie używając programu tcpdump.

Procedura przedstawia się następująco:

- należy zalogować się na serwer docelowy i wydać polecenie:

```
# tcpdump -i any -n port 1234
```

- na serwerze z którego testujemy połączenie wydajemy polecenie:

```
# telnet 10.0.0.10 1234
```

- sprawdzamy czy na ekranie serwera docelowego pojawiła się informacja o przychodzącym pakiecie z serwera źródłowego.

Jeśli pakiet dotarł do serwera docelowego, to należy zgłosić to do osoby odpowiedzialnej za daną aplikację, aby sprawdziła czy działa ona prawidłowo i ewentualnie usunęła awarię.

Jeśli pakiet nie dotarł do serwera docelowego, to operator musi sprawdzić w podobny sposób (za pomocą programu *tcpdump* lub innego sniffera[25]) wszystkie routery przez które przechodzi połączenie lub zgłosić to do osoby odpowiedzialnej za warstwę sieciową.

4.4.3. Przykład 3: Błąd połączenia IP

Gdy Nagios zgłosi nam, że nie może połączyć się z serwerem, operator powinien najpierw sprawdzić czy serwer odpowiada na pakiety ICMP *echo-request* (polecenie *ping*). Przykład poprawnego pingowania serwera o adresie 10.0.0.1:

```
$ ping -c 10 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.379 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.361 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.257 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.279 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.308 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=0.332 ms
64 bytes from 10.0.0.1: icmp_seq=7 ttl=64 time=0.358 ms
64 bytes from 10.0.0.1: icmp_seq=8 ttl=64 time=0.256 ms
64 bytes from 10.0.0.1: icmp_seq=9 ttl=64 time=0.281 ms
64 bytes from 10.0.0.1: icmp_seq=10 ttl=64 time=0.303 ms

--- 10.0.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9001ms
rtt min/avg/max/mdev = 0.256/0.311/0.379/0.044 ms
```

Jeśli serwer będzie nieosiągalny to wynik polecenia będzie podobny do tego:

```
$ ping -c 10 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.50.1 icmp_seq=2 Destination Host Unreachable
From 10.0.50.1 icmp_seq=3 Destination Host Unreachable
From 10.0.50.1 icmp_seq=4 Destination Host Unreachable
From 10.0.50.1 icmp_seq=6 Destination Host Unreachable
From 10.0.50.1 icmp_seq=7 Destination Host Unreachable
From 10.0.50.1 icmp_seq=8 Destination Host Unreachable
From 10.0.50.1 icmp_seq=9 Destination Host Unreachable
From 10.0.50.1 icmp_seq=10 Destination Host Unreachable

--- 10.0.0.1 ping statistics ---
10 packets transmitted, 0 received, +8 errors, 100% packet loss,
time 9004ms, pipe 3
```

W takim przypadku należy upewnić się czy wina leży w warstwie sieciowej czy po stronie serwera, który może być wyłączony. W tym celu operator powinien sprawdzić za pomocą programu *traceroute* (lub podobnego) w którym miejscu w sieci połączenie jest przerywane. Jeśli kończy się na ostatnim routerze przed serwerem, to należy wtedy fizycznie sprawdzić serwer - może jest wyłączony lub jego system operacyjny odmówił posłuszeństwa. Jeśli natomiast połączenie kończy się na jednym z pośrednich routerów, to należy zgłosić taką awarię do administratora routera, który jako ostatni

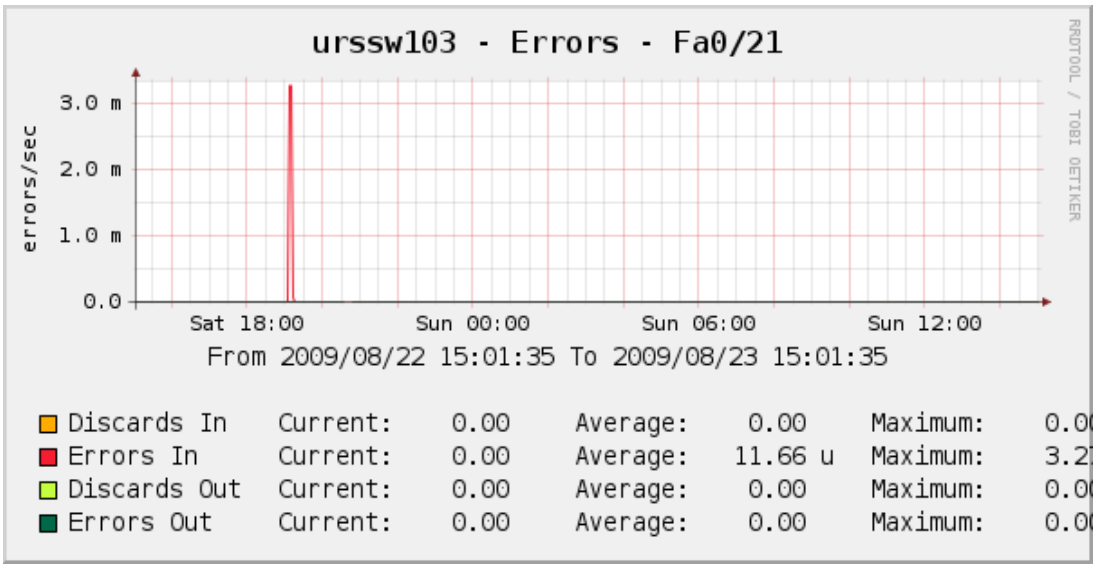
25 Program do podsłuchiwania ruchu sieciowego.

odbiera pakiet. Być może pakiet zatrzymywany jest przez jeden z firewalli znajdujących się po drodze - wtedy znalezienie przyczyny awarii wymaga współpracy z jego administratorem.

Częstą przyczyną awarii tego typu jest źle skonfigurowany routing na urządzeniach sieciowych. Nie ma jednak uniwersalnego sposobu na rozwiązanie tego typu problemu. Wymaga to konsultacji i współpracy z administratorem sieci, który dokładnie zna całe środowisko sieciowe, wie jakie zmiany były ostatnio dokonywane i którędy pakiety danego połączenia powinny być kierowane.

4.4.4. Przykład 4: Błędy w warstwie drugiej

Kiedy operator otrzyma zgłoszenie o występowaniu problemów pomiędzy przełącznikiem, a serwerem podłączonym do niego musi sprawdzić statystyki interfejsu tego przełącznika (zakładamy że przełącznik jest zarządzalny). Numer portu na przełączniku może podać osoba zgłaszająca (jeśli posiada taką informację) lub powinien znajdować się w dokumentacji sieci. Jeśli osoba zgłaszająca problem zna adres MAC karty sieciowej która jest podłączona do switcha, to można także na tej podstawie odnaleźć numer portu. Wszystkie potrzebne informacje statystyczne możemy uzyskać logując się na urządzenie i sprawdzając odpowiednie liczniki. Wymaga to jednak znajomości systemu operacyjnego urządzenia.



Rys. 4.4. Cacti – przykładowy wykres ilości błędów na porcie przełącznika.
Widzimy, że ok. godziny 18:00 wystąpiły błędy.

Wszystkie z wymienionych czynności możemy wykonać dużo szybciej za pomocą NeDi. Operator powinien zalogować się do aplikacji, na liście urządzeń znaleźć dany przełącznik i zlokalizować poszukiwany interfejs. Z informacji zawartych w tabeli interfejsów można szybko odczytać jaka prędkość połączenia została wynegocjowana na porcie, jaki jest tryb dupleksu, stany liczników danych odebranych i wysłanych, liczników błędów odbierania i wysyłania. Jeśli liczniki błędów mają wartości inne niż 0 oraz port pracuje w trybie *half-duplex* to należy spróbować wymusić połączenie w trybie *full-duplex*, który pozwala uniknąć kolizji w trakcie transmisji. Jeśli to nie pomoże

należy sprawdzić czy kabel łączący urządzenia jest w pełni sprawny np. wymieniając go na nowy. Jeśli błędy będą występowały dalej prawdopodobnie uszkodzony jest port w przełączniku bądź karta sieciowa w serwerze. Możemy to sprawdzić poprzez podłączenie serwera do innego portu w przełączniku, a w razie niepowodzenia wymianę karty sieciowej w serwerze.

Do sprawdzenia czy na interfejsie switcha występują błędy równie dobrze możemy użyć Cacti (Rys. 4.4.).

4.4.5. Przykład 5: Błąd NRPE - Nagios

SSH	OK	08-23-2009 15:05:10	5d 11h 6m 45s	1/2	SSH OK - OpenSSH_4.3p2 Debian-9etch3 (protocol 2.0)
load	CRITICAL	08-23-2009 15:05:11	0d 0h 0m 10s	1/2	Connection refused by host
mem	CRITICAL	08-23-2009 15:05:12	0d 0h 0m 9s	1/2	Connection refused by host
omsa	CRITICAL	08-23-2009 15:04:56	0d 0h 0m 25s	1/2	Connection refused by host
resolv_conf	CRITICAL	08-23-2009 15:04:49	0d 0h 0m 32s	1/2	Connection refused by host
time	CRITICAL	08-23-2009 15:04:51	0d 0h 0m 30s	1/2	Connection refused by host

Rys. 4.5. Nagios – pięć z sześciu widocznych na rysunku usług zgłasza ten sam błąd. Ponieważ do ich sprawdzania używamy NRPE, to przyczyną może być awaria demona NRPE.

Jeśli do sprawdzania jednej bądź więcej usług na danym serwerze używamy NRPE i pojawi się w Nagiosie błąd związany z nią (Rys. 4.5.), to przyczyną może być wyłączony demon NRPE. Tego typu błąd bardzo prosto możemy rozpoznać wtedy kiedy co najmniej kilka usług sprawdzanych jest za pomocą NRPE i przy wszystkich z nich pojawią się takie same opisy błędów.

Aby to zweryfikować operator musi zalogować się na sprawdzany serwer.

W przypadku serwera Uniksowego należy wydać polecenie:

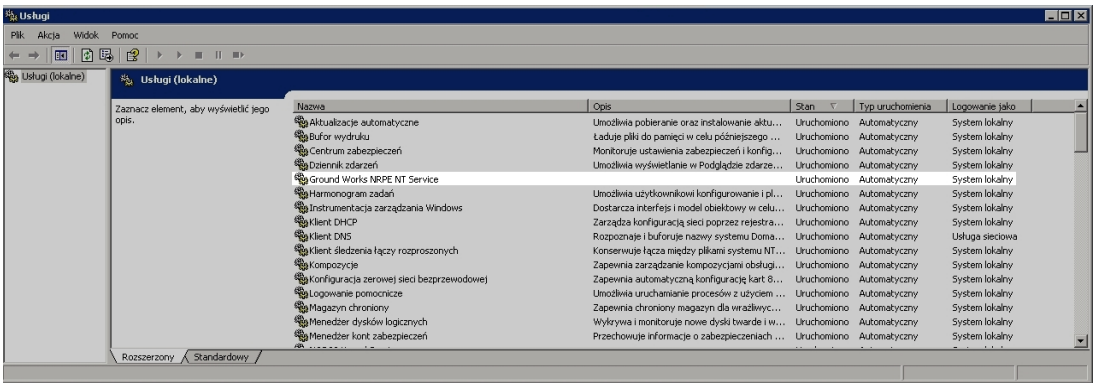
```
# ps aux | fgrep -i nrpe
nagios   30220  0.0  0.1   7996  2992 ?
S<s  22:39   0:00 /usr/sbin/nrpe -c /etc/nagios/nrpe.cfg -d
```

Jeśli wynik będzie podobny do tego powyżej – tzn. będzie uruchomiony program NRPE, to wszystko jest w porządku i problem może leżeć w warstwie sieciowej. Gdy nie będzie ani jednego działającego procesu o nazwie `nrpe`, to należy zrestartować demona. Polecenie restartujące jest zależne od systemu i jego dystrybucji. Np. w systemie Debian GNU/Linux robi się to poleceniem:

```
# /etc/init.d/nagios-nrpe-server restart
Stopping nagios-nrpe: nagios-nrpe.
Starting nagios-nrpe: nagios-nrpe.
```

W przypadku systemu operacyjnego Windows należy sprawdzić czy usługa demona NRPE jest uruchomiona (Rys. 4.6.). Jeśli nie jest, to należy ją uruchomić.

Po wykonaniu powyższych czynności operator powinien odczekać kilka minut i ponownie sprawdzić stan usług serwera w Nagiosie. Jeśli demon NRPE uruchomił się prawidłowo, to stan usług w Nagiosie także powinien wrócić do normy.

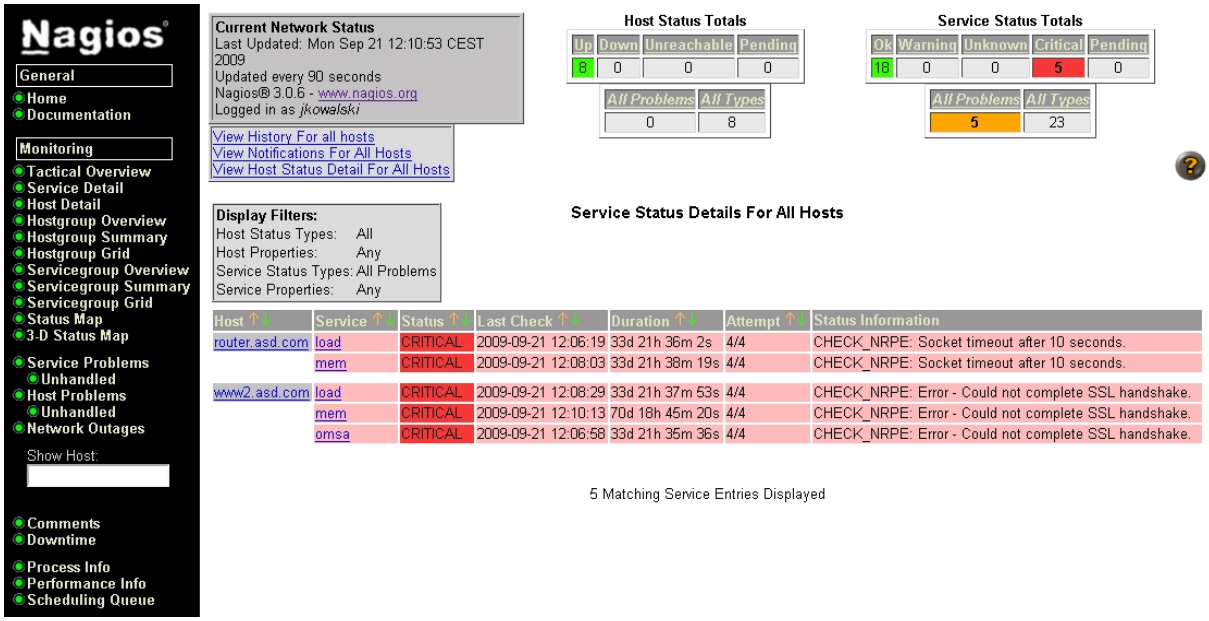


Rys. 4.6. Lista usług w systemie Windows.
Na zrzucie ekranu widzimy, że usługa NRPE jest uruchomiona.

4.4.6. Przykład 6: Śledzenie stanu obiektów - Nagios

Operator powinien na bieżąco śledzić w Nagiosie stan usług i hostów. Bardzo pomocne są w tym dwa widoki udostępniana przez interfejs WWW programu. Pierwszy to *Service Problems* (lista usług, które mają w danej chwili problemy) oraz *Host Problems* (lista hostów, które mają w danej chwili problemy). Na tych listach znajdują się obiekty, które są w innym stanie niż *OK* (Rys. 4.7.). Każdy z tych widoków posiada dodatkowo wersję *Unhandled*. Jest to taki widok na którym pokazywane są tylko te problematyczne obiekty, którymi nikt do tej pory się nie zajął. Każdą z awarii hosta bądź usługi możemy "zatwierdzić". Polega to na dodaniu komentarza do danego obiektu i ustawieniu go jako "zatwierdzonego" (ang. *handled*). W ten sposób osoby używające Nagiosa mogą poinformować współużytkowników o tym, że zajmują się daną awarią. W komentarzu można umieścić np. informację o powodzie awarii oraz przewidywanym czasie jej usunięcia.

Każda ze stron wyświetlających listę problemów odświeża się w przeglądarce WWW domyślnie co 90 sekund. Dzięki temu operator może mieć otwarte okno przeglądarki z odpowiednim widokiem i zawsze będzie widział bieżący stan infrastruktury.



Rys. 4.7. Nagios - widok *Service Problems*.

Widoki zawierające wyłącznie listę problemów są dużo wygodniejsze od widoków z listą wszystkich elementów, gdyż wyświetlają tylko ważne dla operatora informacje. Jeśli Nagios monitoruje bardzo dużo obiektów to wyświetlenie widoku ze wszystkimi obiektami obciąża serwer monitorujący, sieć oraz komputer operatora. Należy pamiętać, że widoki są automatycznie odświeżane co 90 sekund - generuje to więc regularne obciążenie serwera, sieci i komputera.

Każdy z wymienionych widoków można wybrać z lewego panelu w interfejsie WWW Nagiosa (Rys. 4.8.).

Rys. 4.8. Interfejs WWW Nagiosa - lewy panel z listą widoków.

Operator zamiast śledzić stan infrastruktury za pomocą przeglądarki może wyłącznie czytać powiadomienia, które wysyła do niego Nagios. Poniżej znajduje się przykładowe powiadomienie

mówiące o tym, że system operacyjny używa większość z dostępnej pamięci RAM.

```
To: admin@asd.com
Subject: ** PROBLEM alert 1 - www1.asd.com/mem is WARNING **
From: nagios@asd.com
Date: Mon, 21 Sep 2009 03:36:52 +0200

***** Nagios  *****

Notification Type: PROBLEM

Service: mem
Host: www1.asd.com
State: WARNING for 0d 0h 3m 2s
Address: 10.0.50.66

Info:

Memory WARNING - 7.1% (3984 kB) free

Date/Time: Mon Sept 21 03:36:52 CEST 2009

ACK by:
Comment:
```

4.4.7. Przykład 7: Monitorowanie stanu sieci - NeDi

NeDi dostarcza operatorowi dwóch użytecznych widoków. Oba można wybrać z menu *Monitoring* na górnym pasku interfejsu WWW programu.

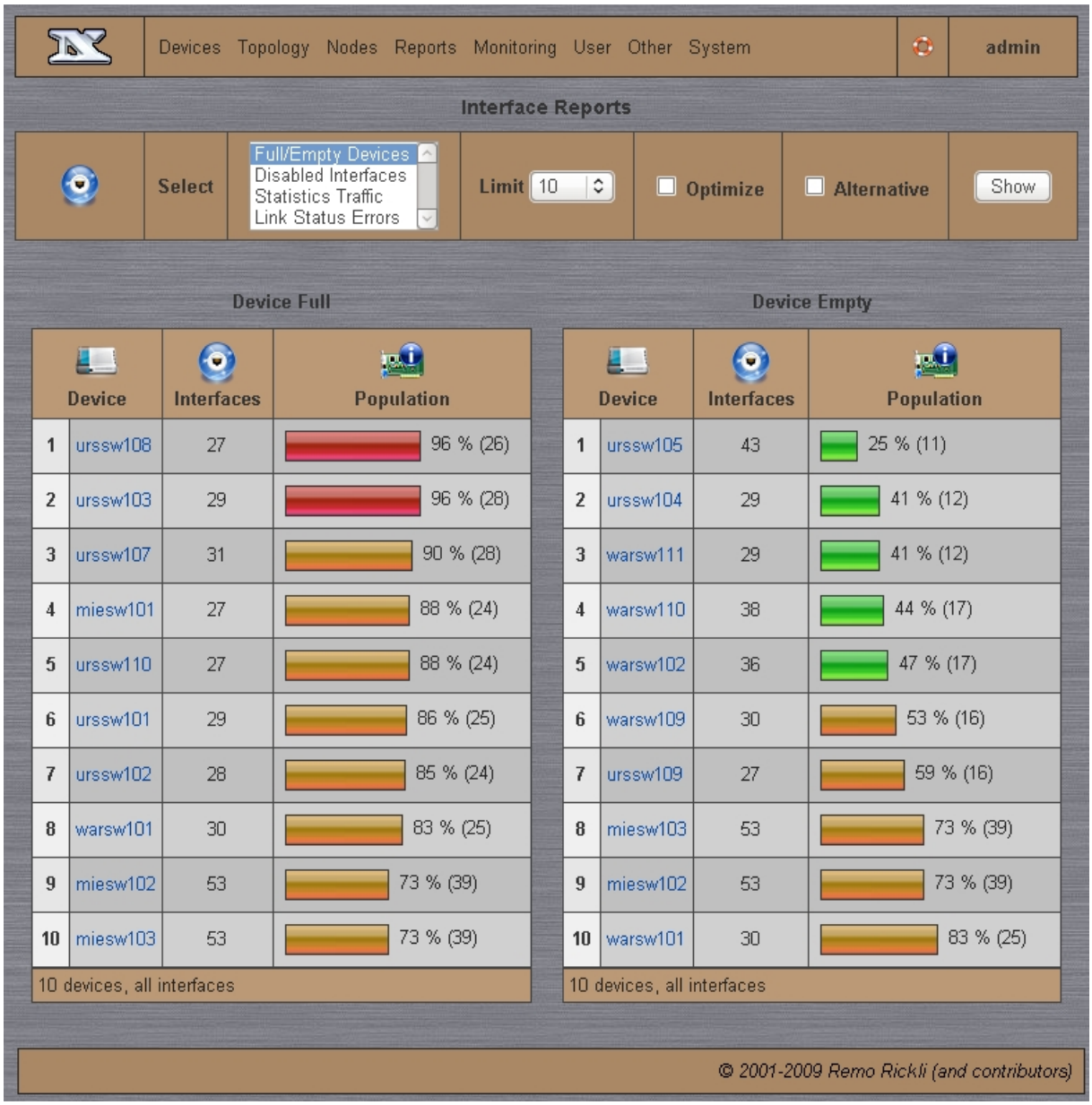
Pierwszy z widoków to *Health* (zdrowie) oraz *Messages* (wiadomości). Pierwszym z nich sześć wykresów umieszczonych w trzech kategoriach: monitoring, ruch wejściowy/wyjściowy oraz błędy (Rys. 4.9.). Poniżej wykresów umieszczone są najnowsze komunikaty ostrzeżeń i błędów w warstwie sieci naszej infrastruktury informatycznej. Podobnie jak Nagios śledzi stan infrastruktury w warstwie aplikacji, tak NeDi w widoku *Health* pozwala operatorowi na bieżąco śledzić stan w warstwie sieci.

Widok *Messages* zawiera komunikaty przydatne w znajdowaniu błędów w konfiguracji urządzeń. Operator powinien śledzić je na bieżąco i wszystkie swoje spostrzeżenia zgłaszać do odpowiednich osób.

Zawartość obu wymienionych widoków odświeża się automatycznie co 1 minutę.

Rys. 4.9. NeDi - widok *Monitoring Health*. Wykresy stanu urządzeń oraz ważniejsze komunikaty.

4.4.8. Przykład 8: Raport zajętości interfejsów - NeDi



Rys. 4.10. NeDi - raport użycia interfejsów przełączników sieciowych.

Należy regularnie śledzić stan wykorzystania infrastruktury sieciowej i dostosowywać ją do pozostałych zmian oraz wykorzystania całości systemu informatycznego. Jednym z elementów, który należy śledzić jest ilość wolnych portów w przełącznikach. NeDi posiada gotowy raport dzięki któremu operator niemalże natychmiast jest w stanie sprawdzić które urządzenie posiada najmniej, a które najwięcej wolnych portów. Gdy switch ma zero bądź bardzo mało nieużywanych portów, to należy rozważyć zamienienie go na urządzenie z większą ilością interfejsów bądź dołożenie kolejnego przełącznika.

Jeśli któreś urządzenie jest wykorzystane w małym stopniu, to można rozważyć zamienienie go na urządzenie o mniejszej ilości portów - o ile istnieje taka możliwość.

Aby wyświetlić raport użycia interfejsów, należy w NeDi z górnego menu z działu *Reports*

wybrać pozycję *Interfaces*. Następnie w formularzu wybrać co najmniej raport *Full/Empty Devices* i kliknąć na przycisk *Show*. Przykładowy raport znajduje się na Rys 4.10.

4.4.9. Przykład 9: Analizowanie stanu łącza - SmokePing

Operator powinien w SmokePing regularnie przeglądać wykresy testowania wszystkich łącz Internetowych, którymi dysponuje firma. Internet jest obecnie niezbędnym medium za pomocą którego prowadzony jest biznes. Brak Internetu oznacza praktycznie przestój w pracy firmy. Dlatego należy sprawdzać czy jakość usług świadczonych przez ISP jest wystarczająca. Zbyt duże opóźnienia oraz straty pakietów świadczą o złej jakości usługi i powinny być sygnałem alarmowym. Gdy łącze przez długi czas funkcjonuje niezadowalająco należy rozważyć zmianę ISP.

Osobna kwestia to przepływność łącza. Powinna być ona dostatecznie duża aby bez problemu obsłużyć ruch generowany przez pracowników w godzinach szczytu. W razie potrzeby należy wykupić u ISP łącze o większej pojemności.

5. ZADANIA ADMINISTRATORA

Praca administratora jest w dużej mierze koncepcyjna. To on instaluje i konfiguruje narzędzia monitorujące, które są używane przez operatorów. Posiada przy tym większą wiedzę, ale i większą odpowiedzialność. O wszystkich awariach i nieprawidłowościach powiadamiany jest przez operatora. Czasami współpracuje bezpośrednio z użytkownikami systemów informatycznych, którymi administruje, aby usunąć awarie bądź dokonać zmian w systemach.

5.1. Cyklicznie

Administrator powinien cyklicznie:

- analizować wydajność sieci na podstawie raportów dostarczonych przez operatora. Musi dbać o to aby sieć była w stanie sprostać obciążeniu generowanemu przez usługi, aplikacje i użytkowników.

5.2. W razie potrzeby

Czynności, które administrator wykonuje w razie potrzeby (tzn. po zgłoszeniu tego przez operatora bądź inną osobę):

- dodawania, usuwanie i zmienianie hostów i usług w Nagiosie,
- usuwanie awarii z którymi nie poradził sobie operator,
- dodawanie, usuwanie i zmienianie urządzeń i wykresów w Cacti,
- przygotowywanie szablonów dla nowych rodzajów urządzeń i wykresów w Cacti,
- dodawanie, usuwanie i zmienianie wykresów w SmokePing,
- aktualizowanie mapy sieci (ręczne konfigurowanie połączeń między urządzeniami jeśli nie są wykrywane automatycznie) w NeDi,
- zarządzanie magazynem urządzeń, usuwanie starych urządzeń, dodawanie nowych, dbanie o aktualność informacji w NeDi,
- konfigurowanie Arpwatcha.

5.3. Przykłady

5.3.1. Przykład 1: Nie działa interfejs WWW Nagiosa

Rys. 4.11. Interfejs WWW Nagiosa nie działa.

Jeśli administrator otrzyma zgłoszenie, że interfejs WWW Nagiosa przestał działać (Rys. 4.11.), to musi tę informację zweryfikować. Gdy w miejscu standardowego widoku Nagios zaprezentuje ekran z komunikatem *Error: Could not read host and service status information!* (jak na ilustracji), to może to być spowodowane niedziałającym demonem aplikacji. W dalszej kolejności administrator powinien zalogować się na serwer monitorujący i za pomocą polecenia

```
# ps aux | fgrep /usr/bin/nagios3
```

sprawdzić czy demon jest uruchomiony. Jeśli polecenie to nie zwróci to ani jednego wiersza tekstu, to Nagios najprawdopodobniej jest wyłączony. Należy wtedy uruchomić go poleceniem:

```
# /etc/init.d/nagios3 restart
Restarting nagios3 monitoring daemon: nagios3Waiting for nagios3
daemon to die...
     .
```

Jeśli wynik polecenia będzie taki jak w przytoczonym tutaj przykładzie, będzie to oznaczać, że demon aplikacji uruchomił się poprawnie i interfejs WWW powinien być już sprawny.

Przyczyn awarii demona może być wiele. Począwszy od wadliwego sprzętu, który może objawiać się losowymi błędami działania serwera aż do przyczyn najbardziej błahych – błędów w plikach konfiguracyjnych. Tych ostatnich jest wiele i każdy przypadek należy potraktować indywidualnie, nie ma uniwersalnego rozwiązania takiego problemu. Błędy konfiguracyjne objawiają się tego typu komunikatami podczas próby restartu demona:

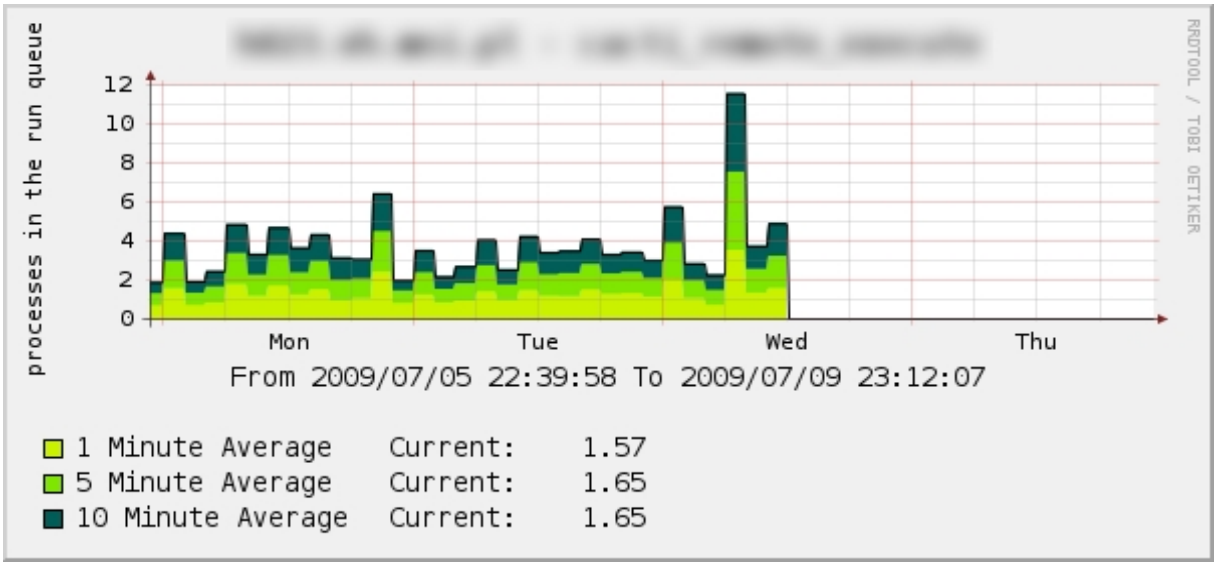
```
# /etc/init.d/nagios3 restart
Restarting nagios3 monitoring daemon: nagios3

Nagios 3.0.6
Copyright (c) 1999-2008 Ethan Galstad (http://www.nagios.org)
Last Modified: 12-01-2008
License: GPL

Reading configuration data...

Error: Invalid object definition type 'hostgroupa' in file
'/etc/nagios3/conf.d/hostgroup_nrpe_mem.cfg' on line 1.

***> One or more problems was encountered while processing the
config files...

Check your configuration file(s) to ensure that they contain valid
directives and data defintions.  If you are upgrading from a previous
version of Nagios, you should be aware that some variables/definitions
may have been removed or modified in this version.  Make sure to read
the HTML documentation regarding the config files, as well as the
'Whats New' section to find out what has changed.

Errors in config! Failed!
Failed!
```

W takim przypadku administrator musi dokładnie prześledzić komunikaty wyświetlone na ekranie i usunąć wszystkie błędy zgłoszone przez Nagiosa. Dopiero potem może ponownie spróbować uruchomić demona. Gdy okaże się, że w plikach konfiguracyjnych nadal występują błędy to procedurę trzeba powtórzyć od początku.

Należy jak najbardziej ograniczyć grono osób, które mają możliwość zmiany konfiguracji Nagiosa. W przeciwnym wypadku mogą czekać nas częste i niespodziewane awarie wywołane zmianami w konfiguracji dokonanymi przez osoby nie posiadające dostatecznej wiedzy na temat Nagiosa.

5.3.2. Przykład 2: Cacti przestało rysować wykresy urządzenia

Administrator otrzymuje od operatora informację, że wykresy jednego z urządzeń w Cacti przestały się rysować (Rys. 4.12.). W celu zdiagnozowania problemu administrator powinien wykonać opisane poniżej czynności.



Rys. 4.12. Cacti – z nieznanych powodów wykres urywa się w środę.
Prawdopodobnie jest to błąd w warstwie aplikacji lub sieci.

Należy sprawdzić czy na urządzeniu włączona jest usługa SNMP. Na każdym urządzeniu dedykowanym oraz w każdym systemie operacyjnym robi się to inaczej. Polecenia które przytoczymy poniżej są charakterystyczne dla systemu operacyjnego Debian GNU/Linux. Należy wydać polecenie `ps aux | fgrep /usr/sbin/snmpd | fgrep -v grep` i sprawdzić czy jego wynik jest podobny do tego:

```
# ps aux | fgrep /usr/sbin/snmpd | fgrep -v grep
root     19970  0.0  0.2   9084  4464 ?        S    18:58   0:00
/usr/sbin/snmpd -Lsd -Lf /dev/null -p /var/run/snmpd.pid 10.0.0.15
```

Jeśli tak, to znaczy, że demon SNMP działa. W przeciwnym wypadku należy go zrestartować:

```
# /etc/init.d/snmpd restart
Restarting network management services: snmpd.
```

Następnie sprawdzamy czy demon SNMP nasłuchuje na gnieździe UDP:

```
# netstat -lpun | egrep ':161.*snmpd'
udp        0      0 10.0.0.15:161           0.0.0.0:* 19970/snmpd
```

Jeśli wynik polecenia jest podobny do tego powyżej, to znaczy, że demon działa prawidłowo. W przeciwnym wypadku należy sprawdzić dzienniki systemowe i poszukać przyczyny niedziałania programu.

Jeśli lokalnie na serwerze jest skonfigurowany firewall to musimy sprawdzić czy przepuszcza on pakiety przychodzące na port 161 UDP. Sposobów i filozofii konfiguracji firewalla jest wiele i nie ma jedynej słusznej metody na sprawdzenie czy pakiety są przepuszczane. Nie możemy więc podać tutaj prostego sposobu na sprawdzenie tego.

Kolejny krok to sprawdzenie czy pakiety docierają z serwera monitorującego do urządzenia docelowego. Najlepiej zrobić to uruchamiając program tcpdump z odpowiednimi parametrami na

serwerze monitorowanym i spróbować połączyć się z demonem SNMP z serwera monitorującego. Na serwerze monitorowanym administrator musi więc wydać polecenie takie jak poniżej lub podobne:

```
# tcpdump -i any -n proto \\udp and port 161
```

W tym samym czasie na serwerze monitorującym za pomocą polecenia *snmpwalk* należy sprawdzić czy komunikacja między serwerami już działa. Zakładając, że używamy protokołu SNMP w wersji 2c, community to *public*, a serwer monitorowany ma adres 10.0.0.20, wydajemy polecenie:

```
# snmpwalk -v 2c -c public 10.0.0.20 system
SNMPv2-MIB::sysDescr.0 = STRING: Linux www1 2.6.18-6-686 #1 SMP Tue May 5
00:40:20 UTC 2009 i686
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-TC::linux
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (73994) 0:12:19.94
SNMPv2-MIB::sysContact.0 = STRING: admin@asdzxc.com
SNMPv2-MIB::sysName.0 = STRING: www1
SNMPv2-MIB::sysLocation.0 = STRING: Serwerownia1
SNMPv2-MIB::sysORLastChange.0 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORID.1 = OID: IF-MIB::ifMIB
SNMPv2-MIB::sysORID.2 = OID: SNMPv2-MIB::snmpMIB
SNMPv2-MIB::sysORID.3 = OID: TCP-MIB::tcpMIB
SNMPv2-MIB::sysORID.4 = OID: IP-MIB::ip
SNMPv2-MIB::sysORID.5 = OID: UDP-MIB::udpMIB
SNMPv2-MIB::sysORID.6 = OID: SNMP-VIEW-BASED-ACM-MIB::vacmBasicGroup
SNMPv2-MIB::sysORID.7 = OID: SNMP-FRAMEWORK-MIB::snmpFrameworkMIBCompliance
SNMPv2-MIB::sysORID.8 = OID: SNMP-MPD-MIB::snmpMPDCompliance
SNMPv2-MIB::sysORID.9 = OID: SNMP-USER-BASED-SM-MIB::usmMIBCompliance
SNMPv2-MIB::sysORDescr.1 = STRING: The MIB module to describe generic
objects for network interface sub-layers
SNMPv2-MIB::sysORDescr.2 = STRING: The MIB module for SNMPv2 entities
SNMPv2-MIB::sysORDescr.3 = STRING: The MIB module for managing TCP
implementations
SNMPv2-MIB::sysORDescr.4 = STRING: The MIB module for managing IP and ICMP
implementations
SNMPv2-MIB::sysORDescr.5 = STRING: The MIB module for managing UDP
implementations
SNMPv2-MIB::sysORDescr.6 = STRING: View-based Access Control Model for
SNMP.
SNMPv2-MIB::sysORDescr.7 = STRING: The SNMP Management Architecture MIB.
SNMPv2-MIB::sysORDescr.8 = STRING: The MIB for Message Processing and
Dispatching.
SNMPv2-MIB::sysORDescr.9 = STRING: The management information definitions
for the SNMP User-based Security Model.
SNMPv2-MIB::sysORUpTime.1 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORUpTime.2 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORUpTime.3 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORUpTime.4 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORUpTime.5 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORUpTime.6 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORUpTime.7 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORUpTime.8 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORUpTime.9 = Timeticks: (1) 0:00:00.01
```

Jeśli ujrzymy wynik podobny do tego powyżej, to znaczy, że komunikacja już działa i Cacti może odpytywać urządzenie. W tym samym czasie tcpdump na serwerze monitorowanym powinien pokazać, że miała miejsce komunikacja:

```
# tcpdump -i any -n proto \\udp and port 161
tcpdump: WARNING: Promiscuous mode not supported on the „any” device
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 96 bytes
19:14:09.023073 IP 10.0.1.46.47509 > 10.0.0.20.161: GetNextRequest(26)  .
1.3.6.1.2.1.1
19:14:09.023242 IP 10.0.0.20.161 > 10.0.1.46.47509: GetResponse(90)  .
1.3.6.1.2.1.1.1.0=[|snmp]
19:14:09.023245 IP 10.0.0.20.161 > 10.0.1.46.47509: GetResponse(90)  .
1.3.6.1.2.1.1.1.0=[|snmp]
19:14:09.023693 IP 10.0.1.46.47509 > 10.0.0.20.161: GetNextRequest(28)  .
1.3.6.1.2.1.1.1.0
19:14:09.023820 IP 10.0.0.20.161 > 10.0.1.46.47509: GetResponse(38)  .
1.3.6.1.2.1.1.2.0=.1.3.6.1.4.1.8072.3.2[|snmp]
19:14:09.023825 IP 10.0.0.20.161 > 10.0.1.46.47509: GetResponse(38)  .
1.3.6.1.2.1.1.2.0=.1.3.6.1.4.1.8072.3.2[|snmp]
19:14:09.024317 IP 10.0.1.46.47509 > 10.0.0.20.161: GetNextRequest(28)  .
1.3.6.1.2.1.1.2.0
19:14:09.024441 IP 10.0.0.20.161 > 10.0.1.46.47509: GetResponse(31)  .
1.3.6.1.2.1.1.3.0=96300
19:14:09.024446 IP 10.0.0.20.161 > 10.0.1.46.47509: GetResponse(31)  .
1.3.6.1.2.1.1.3.0=96300
19:14:09.024834 IP 10.0.1.46.47509 > 10.0.0.20.161: GetNextRequest(28)  .
1.3.6.1.2.1.1.3.0
19:14:09.024956 IP 10.0.0.20.161 > 10.0.1.46.47509: GetResponse(40)  .
1.3.6.1.2.1.1.4.0=[|snmp]
[...]
```

Jeśli *tcpdump* pokazał pakiety, ale *snmpwalk* nie wyświetlił prawidłowego wyniku oznaczać to może, że problem leży po stronie systemu operacyjnego serwera monitorowanego i trzeba podjąć dodatkowe czynności w celu jego usunięcia.

Jeśli tcpdump nie zarejestrował żadnych pakietów, to prawdopodobnie zostały one zatrzymane przez jeden z routerów bądź firewalli znajdujących się pomiędzy serwerem monitorującym i monitorowanym. Ponieważ każde środowisko sieciowe jest inaczej zbudowane nie da się w prosty sposób stwierdzić które urządzenie zatrzymuje pakiety. Administrator musi za pomocą programu `traceroute` bądź podobnego zbadać na którym urządzeniu pakiety się zatrzymują i zbadać czy urządzenie to otrzymuje je i czy przekazuje je dalej, a jeśli tak to dokąd. Po raz kolejny przydatny będzie tutaj sniffer pakietów (tcpdump bądź podobny program).

5.3.3. Przykład 3: Dodawanie statystyk interfejsów serwera Linux do Cacti

W przypadku urządzeń sieciowych takich jak przełączniki, obsługa protokołu SNMP w większości przypadków włączona jest domyślnie bądź włączenie jej jest bardzo proste. Odrobinę więcej kłopotów sprawić może włączenie obsługi SNMP na serwerach z zainstalowanym systemem Linux. Poniżej przedstawione są czynności jakie administrator musi wykonać aby zainstalować, skonfigurować i uruchomić demona SNMP w systemie Debian GNU/Linux oraz aby następnie dodać taki serwer do Cacti.

- W pierwszej kolejności należy zainstalować pakiet *snmpd*, który zawiera demona SNMP:

```
# aptitude install snmpd
```

- Następnie należy otworzyć w edytorze plik */etc/default/snmpd* i dokonać następującej zmiany:
  - Wiersz

```
SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p
/var/run/snmpd.pid 127.0.0.1'
```

    należy zamienić na

```
SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -p /var/run/snmpd.pid'
```

- W pliku */etc/snmp/snmpd.conf*:
  - Należy dodać znak komentarza na początku wiersza:

```
com2sec paranoid  default         public
```

    oraz usunąć znak komentarza z początku wiersza:

```
#com2sec readonly  default         public
```

  - Można zmienić nazwę community z *public* na dowolną inną co zwiększy bezpieczeństwo.
- Zrestartować serwer SNMP poleceniem:

```
# /etc/init.d/snmpd restart
```

- Jeśli serwer posiada włączony lokalny firewall, to należy przepuścić na nim datagramy UDP przychodzące na port 161.

W tym momencie serwer powinien już odpowiadać na zapytania SNMP. Aby dodać urządzenie do Cacti należy:

- dodać nowe urządzenie,
- utworzyć wykresy dla tego urządzenia,
- dodać je do drzewa wykresów.

Gdy po kilkunastu minutach Cacti zbierze już pierwsze dane będziemy mogli zobaczyć nowe wykresy w drzewie wykresów.

5.3.4. Przykład 4: Logowanie komunikatów do centralnego serwera

Przekazywanie komunikatów do centralnego serwera logów w każdej aplikację i systemie operacyjnym konfiguruje się w inny sposób. Przedstawione zostaną tutaj przykład konfiguracji przełącznika CISCO (IOS), Linuksa oraz serwera WWW Apache.

Zakładamy, że na serwerze logów zainstalowany jest już demon syslog, który nasłuchuje na porcie 514 UDP.

CISCO IOS

W trybie konfiguracji urządzenia należy wydać polecenie:

```
logging 1.2.3.4
```

gdzie *1.2.3.4* to adres IP serwera syslog. Od tego momentu wszystkie komunikaty przesyłane będą także do serwera o podanym adresie.

Linux

Poniższy przykład przedstawia instalację oraz konfigurację serwera Syslog-ng w systemie operacyjnym Debian GNU/Linux. Aby zainstalować Syslog-ng należy wydać polecenie:

```
# aptitude install syslog-ng
```

Następnie do pliku */etc/syslog-ng/syslog-ng.conf* należy dopisać:

```
destination dr_syslog { tcp("1.2.3.4"); };
log { source(s_all); destination(dr_syslog); };
```

oraz zrestartować Syslog-ng poleceniem:

```
# /etc/init.d/syslog-ng restart
```

Od tego momentu wszystkie komunikaty będą wysyłane do centralnego serwera.

Apache

Serwer WWW został wybrany jako jeden z możliwych przykładów. Wiele aplikacji może wysyłać komunikaty do sysloga. Apache nie potrafi samodzielnie przesyłać komunikatów przez sieć, ale jeśli lokalny serwer syslog zostanie skonfigurowany tak, aby przesyłał komunikaty do serwera centralnego, to efekt – Apache wysyłający logi do centralnego serwera – zostanie osiągnięty.

Sposób logowania komunikatów błędów określamy za pomocą słowa kluczowego *ErrorLog*. Wystarczy dopisać poniższą linijkę do konfiguracji Apache:

```
ErrorLog syslog:local7
```

Następnie należy zrestartować serwer WWW i logi nie będą zapisywane w pliku, tylko wysyłane bezpośrednio do sysloga.

PODSUMOWANIE I WNIOSKI

Jednym z celi pracy było zaprojektowanie pełnego środowiska monitorującego w oparciu o powszechnie dostępne narzędzia. Przedstawione programy nadają się do użycia w większości firm i organizacji. Wszystkie z nich można uruchomić na systemie operacyjnym Linux, który mimo iż jest darmowy, to zyskuje sobie coraz większe uznanie nawet w dużych i poważnych firmach.

Za pomocą opisanych technik i narzędzi można monitorować najważniejsze części składowe systemu informatycznego. Dzięki temu organizacja jest w stanie zmniejszyć ilość i długość awarii, co wymiernie przekłada się na wyniki finansowe. Pośrednio ma to wpływ na postrzeganie organizacji w środowisku, zwłaszcza jeśli świadczy ona usługi w branży IT takie jak kolokacja bądź zapewnianie dostępu do Internetu.

Do powiadamiania o awariach używany jest Nagios. Niektóre z pozostałych narzędzi posiadają podobną funkcjonalność. Z racji, że to on monitoruje infrastrukturę pod kątem dostępności, jest naturalnym wyborem aby powiadomienia były przez niego wysyłane.

Wykresy obciążenia kart i portów sieciowych rysowane są przez Cacti. Właśnie do tego celu ten program został stworzony i realizuje to najlepiej ze wszystkich omówionych narzędzi. Posiada tak bogate możliwości, że często większość z nich nie będzie nawet przez nas wykorzystywana.

Do zarządzania urządzeniami sieciowymi (mosty, routery, przełączniki) posłuży NeDi. Program umie komunikować się z urządzeniami wielu producentów. Potrafi to robić zarówno za pomocą powszechnie używanego protokołu SNMP jak i za pomocą CLI. Jest to ważne jeśli w sieci znajdują się urządzenia wielu producentów. Producenci urządzeń sieciowych dostarczają dedykowane programy do zarządzania ich urządzeniami, ale używanie jednocześnie kilku takich programów jest męczące i mija się z celem. NeDi eliminuje ten problem.

Monitorowanie urządzeń podłączanych do sieci nie wymaga użycia skomplikowanych programów i systemów. Do tego celu wystarczy w zupełności Arpwatch. Pomoże nam w wykrywaniu włamań do sieci oraz konfliktów adresów IP.

Największym wyzwaniem jest wzrost systemów monitorujących wraz ze wzrostem wielkości i skomplikowania systemów i sieci, które są monitorowane. W parze z tym idzie integrowanie ze sobą różnych narzędzi gdyż to oszczędza czas administratora i usprawnia pracę. Oba te zagadnienia mam zamiar zgłębić w najbliższym czasie.

BIBLIOGRAFIA

Strony internetowe (stan na dzień 28 września 2009)

1. http://nagios.sourceforge.net/docs/3_0/ - Oficjalna dokumentacja Nagiosa.
2. http://www.monitoringexchange.org/ - Pluginy, dodatki i rozszerzenia do Nagiosa.
3. http://www.nedi.ch/ - Strona domowa NeDi oraz oficjalna dokumentacja.
4. http://docs.cacti.net/ - Oficjalna dokumentacja Cacti.
5. http://forums.cacti.net/ - Forum użytkowników Cacti.
6. http://www.phplogcon.org/doc - Oficjalna dokumentacja phpLogCon.
7. http://oss.oetiker.ch/smokeping/doc/index.en.html - Oficjalna dokumentacja SmokePing.
8. http://www.cisco.com/web/PL/techsupport/index.html - Internetowa baza wiedzy Cisco.

Książki

1. **Wolfgang Barth**: *Nagios, 2nd Edition. System and Network Monitoring*. No Starch Press. 2008.
2. **Karol Krysiak**: *Sieci komputerowe. Kompendium. Wydanie II.* Helion. 2005
3. **Dinangkur Kundu**, **S. M. Ibrahim Lavlu**: *Cacti 0.8 Network Monitoring.* Packt Publishing. 2009.
4. **David Josephsen**: *Building a Monitoring Infrastructure with Nagios.* Prentice Hall. 2007.
5. **Douglas Mauro**, **Kevin Schmidt**: *Essential SNMP, Second Edition.* O'Reilly Media. 2005.
6. **Max Schubert**, **Derrick Bennett**, **Jonathan Gines**, **Andrew Hay**, **John Strand**: *Nagios 3 Enterprise Network Monitoring: Including Plug-Ins and Hardware Devices.* Syngress. 2008.
7. **Thomas A. Limoncelli**, **Christina J. Hogan**, **Strata R. Chalup**: *The Practice of System and Network Administration.* Addison-Wesley Professional. 2007.
8. **Kevin Dooley**, **Ian J. Brown**: *Cisco. Receptury.* Helion. 2004.
9. **William Stallings**: *Protokoły SNMP i RMON. Vademecum profesjonalisty.* Helion. 2003.
10. **E. Schetina, K. Green, J. Carlson**: *Bezpieczeństwo w sieci*. Helion. 2002.
11. **Mark A. Dye**, **Rick McDonald**, **Antoon „Tony" W. Rufi**: *Akademia sieci Cisco. CCNA Exploration. Semestr 1.* Mikom. 2008.
12. **Rick Graziani**, **Allan Johnson**: *Akademia sieci Cisco. CCNA Exploration. Semestr 2.* Mikom. 2008.

SPIS RZECZY

ZAŁĄCZNIK A

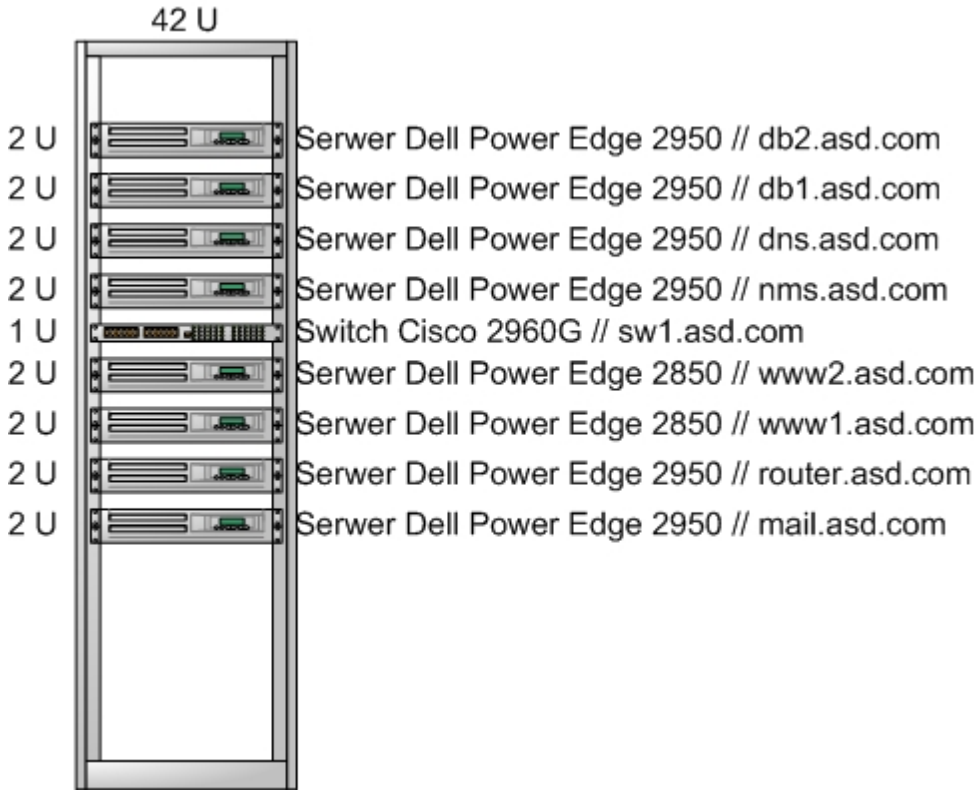
Monitorowanie serwerów i usług na przykładzie systemu Nagios

A. WSTĘP

W niniejszym załączniku przedstawiony zostanie opis konfiguracji programu Nagiosa wraz z gotową konfiguracją do monitorowania przykładowego środowiska.

A.1. ŚRODOWISKO TESTOWE

Poniżej znajdują się dane środowiska testowego na przykładzie którego stworzona zostanie konfiguracja Nagiosa. Zostało ono zobrazowane na rysunkach A.1.1. i A.1.2..



Rys. A.1.1. Schemat środowiska testowego - szafa rack.

Nazwa firmy: ASD S.A.

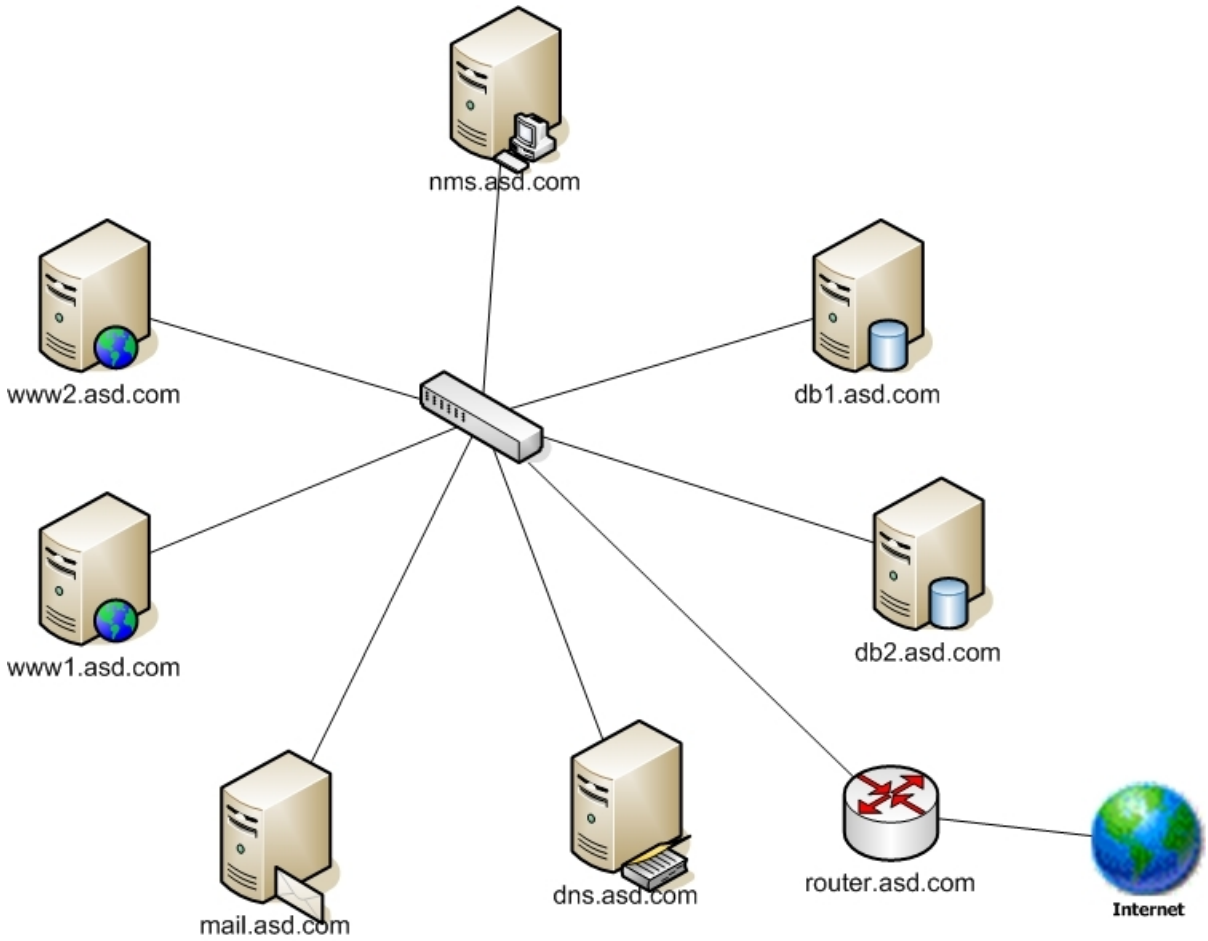
Domena DNS: asd.com

Każdy z serwerów podłączony jest do sieci serwerowej.

Adres sieci: 10.0.50.0/24

Router: 10.0.50.1/24

Serwer DNS: 10.0.50.51



Rys. A.1.2. Schemat środowiska testowego - warstwa sieci.

A.1.1. Lista serwerów

Nazwa	Adres IP	Funkcja serwera	Osoby odpowiedzialne.
nms.asd.com	10.0.50.79	Monitorowanie sieci, serwerów i usług	Główny administrator
dns.asd.com	10.0.50.51	Lokalny serwer DNS	Główny administrator
www1.asd.com	10.0.50.66	Serwer WWW	Administratorzy WWW
www2.asd.com	10.0.50.67	Serwer WWW	Administratorzy WWW
db1.asd.com	10.0.50.40	Serwer MySQL	Administratorzy baz danych
db2.asd.com	10.0.50.50	Serwer MySQL	Administratorzy baz danych
router.asd.com	10.0.50.1	Router	Główny administrator
mail.asd.com	10.0.50.90	Serwer pocztowy	Główny administrator

A.1.2. Kontakty

Imię i nazwisko	Grupa	E-mail
Jan Kowalski	Główny Administrator	jkowalski@asd.com
Piotr Wiśniewski	Administrator baz danych	pwisniewski@asd.com
Krzysztof Kowalczyk	Administrator baz danych	kkowalczyk@asd.com
Witold Witkowski	Administrator WWW	wwitkowski@asd.com
Zbigniew Malinowski	Administrator WWW	zmalinowski@asd.com

Wszystkie serwery podłączone są do switcha sw1.asd.com.

Dostęp do Internetu zapewnia router.asd.com. Posiada on skonfigurowane publiczne adresy IP na publicznym interfejsie sieciowym. Przychodzące połączenia na wybrane publiczne adresy IP NAT-owane są na odpowiadające im wewnętrzne (10.0.50.0/24) adresy IP. Dodatkowo router pełni rolę firewalla.

Oczywiście prawdziwe środowisko produkcyjne powinno być zaprojektowane w dużo bardziej bezpieczny i niezawodny sposób, ale nie jest to przedmiotem tej pracy.

Na każdym z serwerów zainstalowany jest system operacyjny Debian GNU/Linux 5.0 w wersji na architekturę sprzętową i386.

A.2. DEBIAN

Podstawowym systemem operacyjnym użytym przez nas będzie Debian GNU/Linux. Wybrany został ze względu na stabilność, wygodę użycia oraz "wolną" licencję. Pozwala ona na używanie tego systemu za darmo do praktycznie dowolnych celów. Dodatkowo mamy pełen dostęp do kodów źródłowych i możemy modyfikować je według własnych potrzeb. Nie jesteśmy także zależni od firm trzecich oraz ich procedur. Nie musimy czekać na wykupienie licencji oraz ich dostarczenie. Wystarczy np. pobrać z Internetu obrazy płyt instalacyjnych Debiana i możemy już w pełni korzystać z jego możliwości.

Oczywiście na rynku dostępnych jest wiele innych równie dobrych systemów operacyjnych. Nie ma systemu najlepszego i uniwersalnego. Najlepszy jest ten, który znamy. Nic nam po nawet najwspanialszym narzędziu jeśli nie będziemy w stanie go używać.

Debian posiada trzy główne wersje: stabilną, testową oraz niestabilną. Na serwerach zalecane jest używanie wersji stabilnej. I tak też zrobimy. Użyjemy wersji o nazwie kodowej Lenny, czyli obecnej stabilnej.

A.3. NAGIOS

W naszym przykładowym środowisku będziemy używać Nagiosa w wersji 3.0.6 gdyż w momencie pisania niniejszej pracy taka właśnie wersja była dostępna w stabilnej gałęzi (Lenny) systemu Debian. Aplikację zainstalujemy z paczki systemowej - nie będziemy jej ręcznie kompilować.

A.3.1. Struktura katalogu konfiguracyjnego

Wszystkie pliki konfiguracyjne Nagiosa muszą mieć sufiks *.cfg*.

Plik apache2.conf

Jest to plik konfiguracyjny używany przez serwer WWW. Konfiguruje Apache'a tak aby udostępniał interfejs WWW do Nagiosa.

Plik cgi.cfg

Zawiera konfigurację programów CGI (interfejsu WWW). Odwołuje się także do głównego pliku konfiguracyjnego Nagiosa aby CGI wiedziały jak cały Nagios jest skonfigurowany.

Plik commands.cfg

Zawiera definicje poleceń.

Plik nagios.cfg

Główny plik konfiguracyjny. W nim określamy jak ma się zachowywać i działać główny demon aplikacji oraz CGI. Od tego pliku należy zacząć modyfikowanie konfiguracji.

Plik resource.cfg

Tutaj umieszczamy makra konfiguracyjne. W makrach mogą być zapisane np. hasła. Dzięki temu programu CGI nie będą mogły odczytać tych haseł.

Katalog conf.d

Katalog w którym będziemy umieszczać definicje naszych obiektów. Za pomocą obiektów definiujemy hosty, usługi, kontakty, grupy (hostów, usług, kontaktów) oraz polecenia. Innymi słowy opisujemy co i jak chcemy monitorować. Plików z definicjami obiektów może być dowolnie wiele.

Katalog stylesheets

Katalog ten zawiera pliki CSS używane przez interfejs WWW.

W plikach konfiguracyjnych linie zaczynające się od **#** (ang. *hash*) są traktowane jako komentarze i zawartość takiej linii jest ignorowane. Znak ***;*** (średnik) oraz wszystko co znajduje się w danej linii po nim także jest taktowane jako komentarz i ignorowane. Nazwy zmiennych muszą zaczynać się na początku wiersza. Nie mogą być poprzedzane jakimikolwiek białymi znakami.

Dodatkowo wielkość liter w ich nazwach ma znaczenie.

Szczegółowy opis zawartości poszczególnych plików konfiguracyjnych znajdziemy w dokumentacji Nagiosa.

Obiekty wchodzą w skład całej logiki systemu odpowiedzialnej za monitorowanie i powiadamianie. W Nagiosie wyróżniamy następujące typy obiektów:

Nazwa	**Opis**
Host	Jeden z podstawowych typów obiektów. Są to zwykle fizyczne urządzenia znajdujące się w naszej sieci (serwery, komputery stacjonarne, drukarki, routery, przełączniki, firewalle, itd.), posiadają adres w którejś z warstw modelu ISO/OSI, udostępniają co najmniej jedną usługę (HTTP, SMTP, SSH, itd.). Hosty mogą być połączone ze sobą za pomocą relacji rodzic-dziecko odpowiadających np. połączeniom sieciowym w warstwie 2 bądź 3 (modelu ISO/OSI). Relacje te są używane przez Nagiosa do prawidłowego reagowania na awarie sieciowe. Oczywiście nie musimy kurczowo trzymać się powyżej opisanych schematów przy projektowaniu naszego systemu monitorującego. Nagios pozwala nam monitorować dowolnie skomplikowane i abstrakcyjne obiekty. Nie muszą to być wcale sieci oparte o protokół IP.
Grupa hostów	Pozwala grupować hosty w jeden obiekt. Później do takiej grupy hostów możemy np. przypisać usługę, w efekcie czego działanie tej usługi będzie sprawdzane na każdym hoście, który należy do grupy. Oszczędza to nam czas i upraszcza konfigurację. Za pomocą tego mechanizmu możemy także tworzyć grupy będące funkcjonalnym odwzorowaniem przeznaczenia poszczególnych serwerów. Możemy stworzyć np. grupy *Serwery WWW*, *Serwery bazodanowe*, itd.
Usługa	Obok obiektów typu host jest to jeden z podstawowych obiektów w Nagiosie. Każdy z hostów może posiadać wiele usług. Przykładowe usługi to: • parametry hosta (obciążenie procesora, użycie pamięci, obciążenie dysków, temperatura, itd.) • serwery aplikacji (HTTP, SMTP, IMAP, POP3, SSH, FTP, itd.)
Grupa usług	Grupa usług grupuje usługi w taki sposób jak grupa hostów grupuje hosty. Możemy np. usługi takie jak SMTP, SMTPS (SMTP over SSL), IMAP, IMAPS, POP3, POP3 zgrupować pod nazwą *Poczta*.
Kontakt	Reprezentuje pojedynczą osobę (ale nie koniecznie), która odpowiedzialna jest za jakieś hosty bądź usługi i będzie otrzymywała powiadomienia w razie wystąpienia bądź zakończenia awarii. Zwykle posiada zdefiniowaną jedną lub więcej metod powiadamiania. Przykładowe metody powiadamiania to SMS, e-mail, pager, IM (Instant Messaging). Tutaj także mamy pełną dowolność w konfiguracji obiektów. Kontaktem może być np. cała jednostka organizacyjna w firmie, a powiadomienia będą wysyłane na alias e-mailowy do którego należy wiele osób.
Grupa Kontaktów	Nadaje jednemu bądź większej ilości Kontaktów wspólną nazwę. Później ta Grupa może być np. przypisana do poszczególnych hostów (lub grupy hostów). Np. do grupy hostów *Serwery WWW* możemy przypisać grupę kontaktów *Administratorzy WWW* i w razie awarii jednego z serwerów wszyscy administratorzy WWW otrzymają powiadomienia.
Przedział czasowy	Jest on wykorzystywany w definicjach kontaktów oraz hostów i usług. W pierwszym przypadku mówi o tym kiedy mogą być wysyłane powiadomienia do danego kontaktu (np. tylko w godzinach pracy albo we wszystkie dni oprócz

	soboty i niedzieli). W pozostałych przypadkach definiuje kiedy dane hosty i usługi mają być sprawdzane. Możemy np. uznać, że nie ma sensu sprawdzania w weekend czy działają drukarki biurowe gdyż są one używane przez pracowników tylko w czasie pracy biura.
Polecenie	Definicje te są wykorzystywane przez hosty, usługi, Powiadomienia, Obsługę Zdarzeń, itd. Mówią o tym jakie polecenia systemowe należy wykonać (aby np. sprawdzić czy dana usługa działa).
Eskalacja powiadomienia	Eskalacje używane są po to aby powiadamiać szerszą grupę osób jeśli dana awaria nie zostanie usunięta w określonym czasie. Możemy np. zażyczyć sobie aby kierownik zespołu został powiadomiony o awarii jeśli dyżurujący po godzinach pracy inżynier nie usunie awarii w ciągu pół godziny. Dodatkowo jeśli po godzinie awaria nadal nie będzie usunięta i SLA (Service Level Agreement) zostanie naruszone to powiadomienie może zostać wysłane do dyrektora działu.

Oto krótki przykład do czego możemy użyć różnych grup w konfiguracji Nagiosa.

Załóżmy, że mamy kilka hostów, które świadczą usługi związane z pocztą e-mail. Usługi te to SMTP, SMTPS, POP3, POP3S, IMAP oraz IMAPS. Tworzymy więc grupę hostów o nazwie *Serwery Poczty* i umieszczamy w niej wszystkie z tych serwerów. Następnie tworzymy grupę usług o nazwie *Poczta* i umieszczamy w niej wszystkie wymienione usługi. Potem te dwie grupy łączymy ze sobą. Uzyskujemy dzięki temu logiczne odwzorowanie funkcji poszczególnych hostów i usług, upraszczamy konfigurację i oszczędzamy czas.

Możemy połączyć to także z grupą kontaktów. W firmie mamy kilku administratorów poczty. Tworzymy więc grupę kontaktów o nazwie *Administratorzy Poczty* i przypisujemy ich do grupy usług *Poczta* oraz grupy hostów *Serwery Poczty*. W razie awarii jednego z hostów bądź jednej z usług, odpowiednia grupa administratorów otrzyma powiadomienie.

Definicje obiektów mogą być umieszczone w jednym bądź wielu plikach. Lokalizacje plików (bądź katalogów) konfiguracyjnych deklarujemy za pomocą dyrektyw *cfg_file* i/lub *cfg_dir* w głównym pliku konfiguracyjnym.

Aby ułatwić tworzenie skomplikowanych definicji obiektów, Nagios udostępnia nam takie mechanizmy jak szablony oraz dziedziczenie obiektów. Tak więc zamiast definiować te same pary klucz - wartość w wielu obiektach możemy utworzyć jeden obiekt będący szablonem i kazać innym obiektom dziedziczyć po nim.

Logi aplikacji zapisywane są do pliku */var/log/nagios3/nagios.log*. Archiwalne logi umieszczane są w katalogu */var/log/nagios3/archive*. O ich rotowanie program dba sam, nie ma potrzeby używania do tego aplikacji zewnętrznych takich jak logrotate.

Wiele dodatkowych plików przechowywanych jest w katalogach */var/cache/nagios3* oraz */var/lib/nagios3*. Są to np. pamięć podręczna aplikacji, dane które mają zostać zapamiętane pomiędzy restartami, stany hostów i usług, itd..

Konfiguracje wszystkich pluginów przechowywane są w katalogu */etc/nagios-plugins/config*. Główny plik konfiguracyjny odwołuje się do niego za pomocą dyrektywy *cfg_dir*.

Poniżej znajduje się opis ważniejszych elementów użytych w konfiguracji poszczególnych typów obiektów. Pełna specyfikacja znajduje się w dokumentacji Nagiosa.

A.3.2. Wprowadzenie do konfiguracji

A.3.2.1. Host

Wzór:

```
define host {
        host_name nazwa_hosta
        alias alias
        address adres
        max_check_attempts liczba
        check_period nazwa_przedziału_czasowego
        contacts nazwa_kontaktu
        notification_interval liczba
        notification_period nazwa_przedziału_czasowego
        contact_groups nazwa_grupy_kontaktów
        check_command nazwa_polecenia
        notification_options [d,u,r,f,s,n]
        parents nazwa_hosta
        hostgroups nazwa_grupy
        active_checks_enabled [0/1]
        passive_checks_enabled [0/1]
        obsess_over_host [0/1]
        check_freshness [0/1]
        event_handler_enabled [0/1]
        event_handler nazwa_polecenia
        flap_detection_enabled [0/1]
        process_perf_data [0/1]
        retain_status_information [0/1]
        retain_nonstatus_information [0/1]
        notifications_enabled [0/1]
        notes notatka
        notes_url url
        action_url url
        icon_image plik_obrazu
        icon_image_alt napis
}
```

Fragmenty wyróżnione tekstem **pogrubionym** są wymagane, pozostałe są opcjonalne.

Opis poszczególnych słów kluczowych:

Słowo kluczowe	Opis
host_name nazwa_hosta	Zwięzła nazwa dla hosta. Później będzie ona używana przez inne obiekty do odwoływania się do tego konkretnego hosta.
alias alias	Dłuższa nazwa/opis hosta.
address adres	Przeważnie jest to adres IP hosta. Ale może to być praktycznie dowolna wartość. Będzie później ona przekazywana np. do pluginów, które sprawdzają czy host działa prawidłowo. Można tu wpisać nazwę DNS, ale w razie awarii serwerów DNS Nagios nie będzie w stanie rozwiązać tej nazwy.

	Używane adresów IP jest pewniejsze. Jeśli nie podamy adresu hosta, to Nagios potraktuje nazwę jako adres.
max_check_attempts liczba	Ile razy Nagios ma sprawdzić hosta zanim stwierdzi, że nie działa. Polecenie sprawdzające musi stwierdzić właśnie tyle razy pod rząd, że jest w innym stanie niż *OK* zanim stwierdzi, że nastąpiła awaria.
check_period nazwa_przedziału_czasowego	Nazwa przedziału czasowego który mówi kiedy mają być dokonywane sprawdzenia.
contacts nazwa_kontaktu	Nazwa kontaktu, który ma otrzymywać powiadomienia. Jeśli chcemy podać więcej niż jeden kontakt to musimy oddzielić je przecinkami.
notification_interval liczba	Co ile jednostek czasu (*interval_length)* mają być wysyłanie powiadomienia o tym, że występuje awaria. Jeśli wartość zmiennej ustawimy na *0*, to Nagios wyśle tylko jedno powiadomienie.
notification_period nazwa_przedziału_czasowego	Nazwa przedziału czasowego który określa kiedy mają być wysyłane powiadomienia.
contact_groups nazwa_grupy_kontaktów	Nazwa grupy kontaktów, która ma otrzymywać powiadomienia o danej usłudze. Jeśli chcemy podać więcej niż jedną grupę to musimy oddzielić je przecinkami. W definicji usługi musimy określić przynajmniej jeden kontakt (*contacts*) bądź jedną grupę kontaktów (*contact_groups*).
check_command nazwa_polecenia	Nazwa polecenia, które będzie użyte do sprawdzenia czy dany host działa.
notification_options [d,u,r,f,s,n]	Określa jakie rodzaje powiadomień mają być wysyłane. Możliwe wartości: **d** - Host wejdzie w stan DOWN **u** - Host wejdzie w stan UNREACHABLE **r** - Host wejdzie w stan OK **f** - Host będzie zbyt szybko zmieniać stany (ang. *flapping*) co może doprowadzić do zalewu powiadomień **s** - rozpocznie się bądź zakończy zaplanowane wyłączenie usługi **n** - powiadomienia nie będą wysyłane Jeśli nie podamy żadnej z wymienionych powyżej wartości, to Nagios uzna, że chcemy otrzymać wszystkie możliwe powiadomienia. Przykład: jeśli temu słowu kluczowemu przypiszemy wartość **d,r** to będziemy otrzymywać powiadomienia gdy host nie będzie działał oraz gdy zakończy się awaria.
parents nazwa_hosta	Oddzielona przecinkami lista hostów, które są "rodzicami" definiowanego hosta. Zwykle używa się tego słowa kluczowego do zdefiniowania połączeń sieciowych pomiędzy urządzeniami.
hostgroups nazwa_grupy	Definiuje do jakich grup hostów należy dany host. Kolejne nazwy grup należy oddzielić przecinkami. Przynależność hosta do grupy można także definiować za pomocą słowa kluczowego *members* w definicji grupy hostów.

`active_checks_enabled [0/1]`	Włącza lub wyłącza aktywne sprawdzanie. Możliwe wartości: • **0** - wyłącz • **1** - włącz
`passive_checks_enabled [0/1]`	Włącza lub wyłącza pasywne sprawdzanie. Możliwe wartości: • **0** - wyłącz • **1** - włącz
`obsess_over_host [0/1]`	Ten parametr używany jest w przypadku rozproszonych instalacji systemu monitorującego. Ustawiamy jego wartość na **1** tylko na zdalnych serwerach aby zgłaszały stany hostów do głównego serwera monitorującego.
`check_freshness [0/1]`	Parametr używany jest gdy sprawdzanie dokonywane jest pasywnie. Jeśli ostatni raz stan został zgłoszony zbyt dawno temu, to Nagios uzna go za nieaktualny i wymusi aktywne sprawdzenie. Możliwe wartości: • **0** - wyłącz • **1** - włącz
`event_handler_enabled [0/1]`	Określa czy obsługa zdarzeń ma być włączona. Możemy skonfigurować Nagiosa tak aby sam podejmował określone akcje w razie wystąpienia awarii. Np. aby zrestartował serwer za pomocą interfejsu zdalnego zarządzania (Dell DRAC, HP iLO, IBM RSA, IPMI, itd.) gdy maszyna przestanie odpowiadać. Możliwe wartości: • **0** - wyłącz • **1** - włącz
`event_handler nazwa_polecenia`	Nazwa zdefiniowanego polecenia, które ma być wykonane jeśli zmieni się stan usługi i parametr *event_handler_enabled* ma wartość *1*.
`flap_detection_enabled [0/1]`	Włącza lub wyłącza wykrywanie "flapowania". Flapowanie to zbyt częste zmiany stanu, które powodują sztorm powiadomień. Może to być spowodowane np. źle skonfigurowanymi przedziałami stanów ostrzegawczego i krytycznego bądź niestabilną pracą sieci. Możliwe wartości: • **0** - wyłącz • **1** - włącz
`process_perf_data [0/1]`	Włącza lub wyłącza przetwarzanie danych wydajnościowych zwracanych przez dany plugin (program sprawdzający stan usługi). Pluginy mogą zwracać dane wydajnościowe, które później mogą być przekazywane do innych programów. W ten właśnie sposób działają różne aplikacje będące dodatkami do Nagiosa, które na podstawie danych wydajnościowych rysują wykresy (np. obciążenia procesora, użycia pamięci, itd.). Możliwe wartości: • **0** - wyłącz • **1** - włącz
`retain_status_information [0/1]`	Zapamiętywanie informacji o stanie hosta pomiędzy restartami Nagisa.

	Możliwe wartości: • **0** - wyłącz • **1** - włącz
retain_nonstatus_information [0/1]	Zapamiętywanie informacji nie dotyczących stanu hosta pomiędzy restartami Nagisa. Możliwe wartości: • **0** - wyłącz • **1** - włącz
notifications_enabled [0/1]	Włącza lub wyłącza powiadomienia dla definiowanego właśnie hosta. Możliwe wartości: • 0 - wyłącz • 1 - włącz Powiadomienia możemy także włączać i wyłączać za pomocą interfejsu WWW.
notes notatka	Notatka która będzie widoczna w rozszerzonym widoku hosta w interfejsie WWW.
notes_url url	Adres pod którym dostępne są dodatkowe informacje o hoście. W interfejsie WWW będzie on dostępny w postaci ikony czerwonego folderu.
action_url url	Adres pod którym dostępne są dodatkowe czynności dla danego hosta. W interfejsie WWW będzie on dostępny w postaci ikony czerwonej plamki (ang. *splat*).
icon_image plik_obrazu	Nazwa pliku z obrazem który będzie wyświetlany w interfejsie WWW. Obraz powinien mieć wymiary 40x40 pikseli oraz znajdować się w katalogu */usr/share/nagios/htdocs/images/logos*.
icon_image_alt napis	Opis obrazu określonego w *icon_image*. Będzie użyty w parametrze ALT HTML-owego tagu *IMG*.

A.3.2.2. Grupa hostów

Wzór:

```
define hostgroup {
      hostgroup_name nazwa
      alias alias
      members hosty
      hostgroup_members grupy_hostów
}
```

Fragmenty wyróżnione tekstem **pogrubionym** są wymagane, pozostałe są opcjonalne.

Opis poszczególnych słów kluczowych:

Nazwa	**Opis**
hostgroup_name nazwa	Krótka nazwa definiowanej grupy hostów.
alias alias	Dłuższa, opisowa nazwa grupy.
members hosty	Lista krótkich nazw hostów oddzielonych przecinkami, które maja wchodzić w skład grupy. Można użyć tej deklaracji jako alternatywy dla *hostgroups* w definicji hosta.
hostgroup_members grupy_hostów	Lista grup hostów oddzielonych przecinkami. Hosty

	wchodzące w skład tych grup będą przypisane także do definiowanej grupy.

A.3.2.3. Usługa

Wzór:

```
define service {
        host_name nazwa_hosta
        service_description nazwa_usługi
        check_command nazwa_polecenia
        max_check_attempts liczba
        check_interval liczba
        retry_interval  liczba
        check_period nazwa_przedziału_czasowego
        notification_interval liczba
        notification_period nazwa_przedziału_czasowego
        contacts nazwy_kontaktów
        hostgroup_name nazwa_grupy_hostów
        contact_groups nazwy_grup_kontaktów
        notification_options [w,u,c,r,f,s,n]
        active_checks_enabled [0/1]
        passive_checks_enabled [0/1]
        obsess_over_service [0/1]
        check_freshness [0/1]
        notifications_enabled [0/1]
        event_handler_enabled [0/1]
        event_handler nazwa_polecenia
        flap_detection_enabled [0/1]
        process_perf_data [0/1]
        retain_status_information [0/1]
        retain_nonstatus_information [0/1]
        is_volatile [0/1]
        servicegroups nazwa_grupy_usług
}
```

Fragmenty wyróżnione tekstem **pogrubionym** są wymagane. Reszta słów kluczowych jest opcjonalna.

Opis poszczególnych słów kluczowych:

Słowo kluczowe	Opis
host_name nazwa_hosta	Nazwa hosta do którego przypisana będzie ta usługa. Jeśli usługa ma być przypisana do więcej niż jednego hosta, to ich nazwy należy rozdzielić przecinkami.
service_description nazwa_usługi	Nazwa usługi, którą właśnie definiujemy. Może zawierać spacje, myślniki i przecinki. Średniki i cudzysłowy nie powinny być używane. Nazwy wszystkich usług przypisanych do jednego hosta myszą być różne.
check_command nazwa_polecenia	Nazwa polecenia, które będzie użyte do sprawdzenia czy dana usługa działa.
max_check_attempts liczba	Ile razy Nagios ma sprawdzić usługę zanim stwierdzi, że nie działa. Polecenie sprawdzające musi stwierdzić właśnie tyle razy pod rząd, że usługa jest w innym stanie niż *OK* zanim stwierdzi, że nastąpiła awaria.
check_interval liczba	Co ile jednostek czasu (długość jednostki czasu określa

(lub: normal_check_interval liczba)	słowo kluczowe *interval_length*) ma być wykonywane sprawdzanie usługi. Dotyczy ono tylko sytuacji jeśli usługa jest w jednym ze stanów: - *OK,* - nie-*OK* i była sprawdzona **max_check_attempts** razy.
retry_interval liczba (lub: retry_check_interval liczba)	Odpowiednik **check_interval** ale w sytuacji gdy usługa jest w stnie nie-*OK* i nie została jeszcze sprawdzona **max_check_attempts** razy.
check_period nazwa_przedziału_czasowego	Nazwa przedziału czasowego który mówi kiedy mają być dokonywane sprawdzenia usługi.
notification_interval liczba	Co ile jednostek czasu (*interval_length)* mają być wysyłanie powiadomienia o tym, że usługa nie działa. Jeśli wartość zmiennej ustawimy na *0*, to Nagios wyśle tylko jedno powiadomienie o awarii.
notification_period nazwa_przedziału_czasowego	Nazwa przedziału czasowego który mówi kiedy mają być wysyłane powiadomienia dotyczące danej usługi.
contacts nazwy_kontaktów	Nazwa kontaktu, który ma otrzymywać powiadomienia o danej usłudze. Jeśli chcemy podać więcej niż jeden kontakt to musimy oddzielić je przecinkami.
hostgroup_name nazwa_grupy_hostów	Lista grup hostów do których ma być przypisana usługa. Jeśli podajemy nazwy więcej niż jednej grupy to należy oddzielić je przecinkami. W definicji usługi można użyć tej dyrektywy zamiast *host_name* albo wraz z *host_name*.
contact_groups nazwy_grup_kontaktów	Nazwa grupy kontaktów, która ma otrzymywać powiadomienia o danej usłudze. Jeśli chcemy podać więcej niż jedną grupę to musimy oddzielić je przecinkami. W definicji usługi musimy określić przynajmniej jeden kontakt (*contacts*) bądź jedną grupę kontaktów (*contact_groups*).
notification_options [w,u,c,r,f,s,n]	Określa jakie rodzaje powiadomień mają być wysyłane. Możliwe wartości: **w** - usługa wejdzie w stan WARNING **u** - usługa wejdzie w stan UNREACHABLE **c** - usługa wejdzie w stan CRITICAL r - usługa wejdzie w stan OK **f** - usługa będzie zbyt szybko zmieniać stany (ang. *flapping*) co może doprowadzić do zalewu powiadomień **s** - rozpocznie się bądź zakończy zaplanowane wyłączenie usługi **n** - powiadomienia nie będą wysyłane Jeśli nie podamy żadnej z wymienionych powyżej wartości, to Nagios uzna, że chcemy otrzymać wszystkie możliwe powiadomienia. Przykład: jeśli temu słowu kluczowemu przypiszemy wartość **w,u,r** to będziemy otrzymywać powiadomienia gdy usługa będzie w stanie ostrzegawczym lub jest nieosiągalna bądź wróciła do prawidłowego stanu.
active_checks_enabled [0/1]	Włącza lub wyłącza aktywne sprawdzanie danej usługi. Możliwe wartości: • 0 - wyłącz • 1 - włącz

`passive_checks_enabled [0/1]`	Włącza lub wyłącza pasywne sprawdzanie danej usługi. Możliwe wartości: • 0 - wyłącz • 1 - włącz
`obsess_over_service [0/1]`	Ten parametr używany jest w przypadku rozproszonych instalacji systemu monitorującego. Ustawiamy jego wartość na **1** tylko na zdalnych serwerach aby zgłaszały stany usług do głównego serwera monitorującego.
`check_freshness [0/1]`	Parametr używany jest gdy usługa sprawdzana jest pasywnie. Jeśli ostatni raz stan usługi został zgłoszony zbyt dawno temu, to Nagios uzna go za nieaktualny i wymusi aktywne sprawdzenie. Możliwe wartości: • **0** - wyłącz • **1** - włącz
`notifications_enabled [0/1]`	Włącza lub wyłącza powiadomienia dla definiowanej właśnie usługi. Możliwe wartości: • 0 - wyłącz • 1 - włącz Powiadomienia możemy także włączać i wyłączać za pomocą interfejsu WWW.
`event_handler_enabled [0/1]`	Określa czy obsługa zdarzeń ma być włączona. Możemy skonfigurować Nagiosa tak aby sam podejmował określone akcje w razie wystąpienia awarii. Np. aby zrestartował serwer gdy obciążenie na nim będzie zbyt wysokie. Możliwe wartości: • **0** - wyłącz • **1** - włącz
`event_handler nazwa_polecenia`	Nazwa zdefiniowanego polecenia, które ma być wykonane jeśli zmieni się stan usługi i parametr *event_handler_enabled* ma wartość *1*.
`flap_detection_enabled [0/1]`	Włącza lub wyłącza wykrywanie "flapowania" usługi. Flapowanie to zbyt częste zmiany stanu, które powodują sztorm powiadomień. Może to być spowodowane np. źle skonfigurowanymi przedziałami stanów ostrzegawczego i krytycznego bądź niestabilną pracą sieci. Możliwe wartości: • **0** - wyłącz • **1** - włącz
`process_perf_data [0/1]`	Włącza lub wyłącza przetwarzanie danych wydajnościowych zwracanych przez dany plugin Możliwe wartości: • **0** - wyłącz • **1** - włącz
`retain_status_information [0/1]`	Zapamiętywanie informacji o stanie usługi pomiędzy restartami Nagisa. Możliwe wartości: • **0** - wyłącz • **1** - włącz
`retain_nonstatus_information`	Zapamiętywanie informacji nie dotyczących stanu usługi

`[0/1]`	pomiędzy restartami Nagisa. Możliwe wartości: • **0** - wyłącz • **1** - włącz
`is_volatile [0/1]`	Jeśli wartość tej zmiennej ustawimy na **1**, to Nagios założy, że po każdym sprawdzeniu stanu usługi sama zresetuje się ona do stanu *OK*. Tak więc za każdym sprawdzeniem jeśli usługa nie będzie w stanie *OK*, Nagios potraktuje ją jak gdyby właśnie nastąpiła awaria - wyśle powiadomienia, itd..
`servicegroups nazwa_grupy_usług`	Definiuje do jakich grup usług należy dana usługa. Wiele nazw grup należy oddzielić przecinkami. Przynależność usług do grup można także definiować za pomocą słowa kluczowego *members* w definicji grupy usług.

A.3.2.4. Grupa usług

Wzór:

```
define servicegroup {
      servicegroup_name nazwa
      alias alias
      members usługi
      servicegroup_members grupy_usług
}
```

Fragmenty wyróżnione tekstem **pogrubionym** są wymagane. Reszta słów kluczowych jest opcjonalna.

Opis poszczególnych słów kluczowych:

Nazwa	**Opis**
`servicegroup_name nazwa`	Krótka nazwa dla definiowanej grupy.
`alias alias`	Dłuższa, opisowa nazwa grupy.
`members usługi`	Lista krótkich nazw grup oddzielonych przecinkami, które maja wchodzić w skład grupy. Można użyć tej deklaracji jako alternatywy dla *servicegroups* w definicji usługi.
`servicegroup_members grupy_usług`	Lista grup usług oddzielonych przecinkami. Usługi wchodzące w skład tych grup będą przypisane także do definiowanej grupy.

A.3.2.5. Kontakt

Wzór:

```
define contact {
       contact_name nazwa_kontaktu
       host_notifications_enabled [0/1]
       service_notifications_enabled [0/1]
       host_notification_period nazwa_przedziału_czasowego
       service_notification_period nazwa_przedziału_czasowego
       host_notification_options [d,u,r,f,s,n]
       service_notification_options [w,u,c,r,f,s,n]
       host_notification_commands nazwa_polecenia
       service_notification_commands nazwa_polecenia
       email adres_email
       pager numer
}
```

Fragmenty wyróżnione tekstem **pogrubionym** są wymagane. Reszta słów kluczowych jest opcjonalna.

Opis poszczególnych słów kluczowych:

Zmienna	**Opis**
contact_name nazwa_kontaktu	Krótka nazwa kontaktu. Będzie używana w innych obiektach w celu odwołania się do tego kontaktu.
host_notifications_enabled [0/1]	Czy kontakt ma otrzymywać powiadomienia dotyczące hostów. Możliwe wartości: • **0** - nie • **1** - tak
service_notifications_enabled [0/1]	Czy kontakt ma otrzymywać powiadomienia dotyczące usług. Możliwe wartości: • **0** - nie • **1** - tak
host_notification_period nazwa_przedziału_czasowego	Kontakt będzie otrzymywał powiadomienia dotyczące hostów tylko w określonym przedziale czasowym.
service_notification_period nazwa_przedziału_czasowego	Kontakt będzie otrzymywał powiadomienia dotyczące usług tylko w określonym przedziale czasowym.
host_notification_options [d,u,r,f,s,n]	Pozwala jakiego rodzaju powiadomienia dotyczące hostów mają być wysyłane do definiowanego kontaktu. Możliwe argumenty: **d** - gdy host zmieni stan na DOWN **u** - gdy host zmieni stan na UNREACHABLE r - gdy host powróci do stanu normalnego funkcjonowania f - gdy host zacznie "flapować". Flapowanie to zbyt częste zmiany stanu, które powodują sztorm powiadomień. s - gdy rozpocznie lub zakończy się zaplanowane wyłączenie hosta n - nie wysyłaj powiadomień
service_notification_options [w,u,c,r,f,s,n]	Pozwala jakiego rodzaju powiadomienia dotyczące usług mają być wysyłane do definiowanego kontaktu. Możliwe

	argumenty: **w** - gdy usługa zmieni stan na ostrzegawczy **u** - gdy usługa zmieni stan na nieznany c - gdy usługa zmieni stan na ostrzegawczy r - gdy usługa zacznie pracować poprawnie f - gdy usługa zacznie "flapować". Flapowanie to zbyt częste zmiany stanu, które powodują sztorm powiadomień. n - nie wysyłaj powiadomień
host_notification_commands nazwa_polecenia	Lista zdefiniowanych poleceń, które mają być wywołane gdy trzeba powiadomić kontakt o problemie z hostem. Możemy wpisać tu wiele nazw poleceń, ale musimy oddzielić ich nazwy przecinkami. Wszystkie podane polecenia zostaną wywołane gdy Nagios będzie chciał powiadomić dany kontakt.
service_notification_commands nazwa_polecenia	Lista zdefiniowanych poleceń, które mają być wywołane gdy trzeba powiadomić kontakt o problemie z usługą. Możemy wpisać tu wiele nazw poleceń, ale musimy oddzielić ich nazwy przecinkami. Wszystkie podane polecenia zostaną wywołane gdy Nagios będzie chciał powiadomić dany kontakt.
email adres_email	Adres e-mail kontaktu na który mają być wysyłane powiadomienia.
pager numer	Zwykle wpisujemy tu numer pagera bądź telefonu komórkowego. Ale może to być praktycznie dowolna wartość - wszystko zależy od tego jak zdefiniujemy polecenia wysyłające powiadomienia. Możemy wpisać np. adres e-mail bramki przekazującej wiadomości na pager należący do kontaktu.

A.3.2.6. Grupa kontaktów

Wzór:

```
define contactgroup {
      contactgroup_name nazwa_grupy_kontaktów
      alias alias
      members lista_kontaktów
      contactgroup_members lista_grup_kontaktów
}
```

Fragmenty wyróżnione tekstem **pogrubionym** są wymagane. Reszta słów kluczowych jest opcjonalna.

Opis poszczególnych słów kluczowych:

Zmienna	**Opis**
contactgroup_name nazwa_grupy_kontaktów	Krótka nazwa dla definiowanej grupy kontaktów
alias alias	Dłuższa nazwa dla grupy kontaktów.
members lista_kontaktów	Lista krótkich nazw kontaktów, które mają być częścią definiowanej grupy. Poszczególne nazwy należy oddzielić przecinkami. Dyrektywa ta może być użyta jako alternatywa do *contactgroups* w definicji kontaktu.

`contactgroup_members lista_grup_kontaktów`	Członkami grupy mogą być inne grupy. W tej zmiennej możemy podać krótkie nazwy innych grup kontaktów. Nazwy należy oddzielić od siebie przecinkami.

A.3.2.7. Przedział czasu

Wzór:

```
define timeperiod {
      timeperiod_name nazwa
      alias alias
      [weekday] przedziały
}
```

Fragmenty wyróżnione tekstem **pogrubionym** są wymagane. Reszta słów kluczowych jest opcjonalna.

Opis poszczególnych słów kluczowych:

Zmienna	Opis
`timeperiod_name nazwa`	Krótka nazwa dla przedziału czasowego.
`alias alias`	Dłuższa nazwa dla definiowanego przedziału.
`[weekday] przedziały`	[weekday] to nazwa dnia tygodnia w języku angielskim. Deklarację tą należy powtórzyć dla każdego dnia tygodnia który ma wchodzić w skład przedziału. przedziały to oddzielona przecinkami lista przedziałów w formacie HH:MM-HH:MM, gdzie część przed myślnikiem to godzina rozpoczęcia przedziały, a po myślniku to godzina zakończenia przedziału. Jeśli któryś z dni tygodnia chcemy pominąć w przedziale to po prostu go nie wymieniamy w definicji.

Definicje przedziałów mogą być znacznie bardziej skomplikowane. Ich szczegółowy opis znajdziemy w dokumentacji Nagiosa. Do naszych zastosować wystarczające będą proste definicje ograniczające się do dyrektyw opisanych powyżej.

A.3.2.8. Polecenie

Wzór:

```
define command {
      command_name nazwa
      command_line polecenie
}
```

Fragmenty wyróżnione tekstem **pogrubionym** są wymagane. Reszta słów kluczowych jest opcjonalna.

Opis poszczególnych słów kluczowych:

Zmienna	Opis
`command_name nazwa`	Krótka nazwa dla polecenia. Będzie używana przez inne obiekty do odwoływania się do tego konkretnego polecenia.
`command_line polecenie`	Polecenie systemowe które ma zostać wykonane. Nie musi być ujęte

	w cudzysłowy. Znak dolara ma specjalne znaczenie i jeśli chcemy umieścić go w poleceniu dosłownie, to musimy umieścić go dwa razy z rzędu.

Polecenie systemowe to przeważnie bezwzględna ścieżka do programu binarnego wraz z argumentami. Argumenty są oddzielone od siebie spacjami - zupełnie tak, jak polecenia wywoływane z poziomu powłoki systemowej. Polecenie systemowe może zawierać makra. Makra są traktowane podobnie jak zmienne w powłoce - bezpośrednio przed wykonaniem polecenia zamieniane są na konkretne wartości.

Odwołując się do polecenia z poziomu innego obiektu możemy przekazać dodatkowe argumenty. Argumenty te są zapamiętywane w makrach o nazwach typu $ARGx$, gdzie x to kolejne liczby od 1 do 32.

To jak dane makro będzie przechowywało wartość i czy w ogóle będzie miało jakąś wartość jest zależne od kontekstu w jakim jest wywoływane dane polecenie. Zestaw dostępnych makr jest różny dla każdego typu obiektu.

Szczegółowa lista makr wraz z uwzględnieniem kontekstu w jakim są dostępne znajduje się w dokumentacji programu.

Lista ważniejszych makr:

Makro	**Opis**
`$NOTIFICATIONTYPE$`	Typ powiadomienia.
`$HOSTNAME$`	Krótka nazwa hosta. Jest to wartość zmiennej *host_name* z definicji hosta.
`$HOSTSTATE$`	Obecny stan hosta.
`$HOSTADDRESS$`	Zawartość zmiennej *address* z definicji hosta.
`$HOSTOUTPUT$`	Napis zwrócony przez program sprawdzający hosta.
`$LONGDATETIME$`	Data i czas zdarzenia zapisana w długim formacie. Np. *Tue Jul 7 19:30:18 CEST 2009*.
`$CONTACTEMAIL$`	Adres e-mail kontaktu. Jest to wartość zmiennej *email* z definicji kontaktu.
`$SERVICEDESC$`	Długa nazwa usługi. Jest to zawartość zmiennej *service_description* z definicji usługi.
`$HOSTALIAS$`	Zawartość zmiennej *alias* z definicji hosta.
`$SERVICESTATE$`	Aktualny stan usługi.
`$SERVICEOUTPUT$`	Napis zwrócony przez program sprawdzający stan usługi.
`$SERVICESTATETYPE$`	Typ stanu usługi. *SOFT* lub *HARD*.
`$SERVICEATTEMPT$`	Określa które sprawdzenie z rzędu usługa jest w danym stanie.
`$HOSTSTATETYPE$`	Typ stanu hosta. *SOFT* lub *HARD*.
`$HOSTATTEMPT$`	Określa które sprawdzenie z rzędu host jest w danym stanie.

A.3.2.9. cgi.cfg

Opis znaczenia niektórych zmiennych z pliku *cgi.cfg*:

Zmienna	Opis
`authorized_for_system_information`	Lista nazw użytkowników oddzielonych przecinkami. Użytkownicy Ci będą mieli dostęp do informacji dostarczanych przez Extended Information CGI (extinfo.cgi). Podanie jako wartości tej zmiennej * spowoduje, że wszyscy autoryzowani użytkownicy będą mieli dostęp do tych informacji.
`authorized_for_configuration_information`	Lista nazw użytkowników oddzielonych przecinkami. Użytkownicy Ci będą mieli dostęp do wszystkich informacji konfiguracyjnych (hosty, usługi, itd.). Domyślnie użytkownicy mogą przeglądać tylko konfiguracje hostów i usług dla których są kontaktami. Podanie jako wartości tej zmiennej * spowoduje, że wszyscy autoryzowani użytkownicy będą mieli dostęp do tych informacji.
`authorized_for_system_commands`	Lista nazw użytkowników oddzielonych przecinkami. Użytkownicy Ci będą mogli wyłączać i restartować program oraz przełączać jego tryb pomiędzy *active* i *standby*. Podanie jako wartości tej zmiennej * spowoduje, że wszyscy autoryzowani użytkownicy będą mogli wydawać polecenia.
`authorized_for_all_services`	Lista nazw użytkowników oddzielonych przecinkami. Użytkownicy Ci będą mieli dostęp do wszystkich informacji konfiguracyjnych usług. Domyślnie użytkownicy mogą przeglądać tylko konfiguracje usług dla których są kontaktami. Podanie jako wartości tej zmiennej * spowoduje, że wszyscy autoryzowani użytkownicy będą mieli dostęp do tych informacji.
`authorized_for_all_hosts`	Lista nazw użytkowników oddzielonych przecinkami. Użytkownicy Ci będą mieli dostęp do wszystkich informacji konfiguracyjnych hostów. Domyślnie użytkownicy mogą przeglądać tylko konfiguracje hostów dla których są kontaktami. Podanie jako wartości tej zmiennej * spowoduje, że wszyscy autoryzowani

	użytkownicy będą mieli dostęp do tych informacji.
authorized_for_all_service_commands	Lista nazw użytkowników oddzielonych przecinkami. Użytkownicy Ci będą mogli używać CGI poleceń (cmd.cgi) do wydawania poleceń związanych ze wszystkimi usługami. Domyślnie użytkownik może wydawać tylko polecenia związane z usługą dla której jest on kontaktem. Podanie jako wartości tej zmiennej * spowoduje, że wszyscy autoryzowani użytkownicy będą mogli wydawać polecenia.
authorized_for_all_host_commands	Lista nazw użytkowników oddzielonych przecinkami. Użytkownicy Ci będą mogli używać CGI poleceń (cmd.cgi) do wydawania poleceń związanych ze wszystkimi hostami. Domyślnie użytkownik może wydawać tylko polecenia związane z hostem dla którego jest on kontaktem. Podanie jako wartości tej zmiennej * spowoduje, że wszyscy autoryzowani użytkownicy będą mogli wydawać polecenia.

A.3.2.10. Dziedziczenie

Przy dziedziczeniu obiektów najważniejsze są słowa kluczowe *name*, *use*, oraz *register*. *name* to nazwa obiektu.

use jest używane w obiekcie, który dziedziczy cechu innego obiektu. W obiekcie-dziecku jako argument dla tego słowa kluczowego podajemy nazwę (*name*) obiektu-rodzica.

register używane jest w obiekcie-rodzicu. W Nagiosie domyślnie wszystkie obiekty są rejestrowane (ang. *register*). Jeśli tworzymy obiekt-szablon to nie chcemy aby był on rejestrowany. Więc w obiekcie-rodzicu używamy słowa kluczowego *register* z wartością 0 aby nie był on rejestrowany.

Oto przykład dziedziczenia:

```
# obiekt-rodzic
define service {
        name                            generic-service
        max_check_attempts              2
        check_period                    24x7
        check_interval                  1
        retry_interval                  1
        notification_interval           120
        notification_period             24x7
        notification_options            w,u,c,r,f
        contact_groups                  admins
```

```
        active_checks_enabled           1
        passive_checks_enabled          1
        obsess_over_service             0
        check_freshness                 0
        notifications_enabled           1
        event_handler_enabled           1
        flap_detection_enabled          1
        process_perf_data               1
        retain_status_information       1
        retain_nonstatus_information    1
        is_volatile                     0
        register                        0
}

# obiekt-dziecko
define service {
        use                     generic-service
        name                    ssh-service
        servicegroups           SSH_Servers
        service_description     SSH
        check_command           check_ssh
}
```

Najpierw definiujemy obiekt o nazwie *generic-service*, który będzie w naszym przypadku szablonem. Następnie definiujemy obiekt o nazwie *ssh-service*, który dziedziczy (słowo kluczowe *use*) po obiekcie generic-service. Tak więc w *ssh-service* nie musimy konfigurować np. powiadomień, sprawdzeń, itd. Oczywiście takich obiektów jak *ssh-service* może być więcej.

Zamienne zdefiniowane w obiekcie-dziecku mają pierwszeństwo nad zmiennymi ustawionymi w obiekcie-rodzicu. Przyjrzyjmy się definicji:

```
define host {
                host_name               www1
                check_command           check-host-alive
                notification_options    d,u,r
                max_check_attempts      5
                name                    szablon-glowny
}
define host {
                host_name               www2
                max_check_attempts      3
                use                     szablon-glowny
}
```

Jak widzimy oba hosty mają zdefiniowaną zmienną *max_check_attempts*. Wartość przypisana tej zmiennej w obiekcie-dziecku (*3*) nadpisuje wartość odziedziczoną po obiekcie-rodzicu (*5*).

Dziedziczenie nie musi być jednopoziomowe. Obiekt-dziecko może być obiektem-rodzicem dla innego obiektu. Nie ma limitu ilości poziomów dziedziczenia.

Definicje szablonów nie muszą być kompletne. Może w nich brakować wymaganych zmiennych. Warunek jest taki, że niekompletny obiekt nie może być zarejestrowany.

Obiekty mogą dziedziczyć po wielu szablonach. Wystarczy jako wartość zmiennej *use* podać oddzieloną przecinkami listę szablonów.

A.3.2.11. Pluginy

W Debianie pluginy dostarczane są przez dwa pakiety: *nagios-plugins-basic* oraz *nagios-plugins-standard*. Programy wykonywalne umieszczone są w katalogu */usr/lib/nagios/plugins*, a pliki z definicjami poleceń w katalogu */etc/nagios-plugins/config*.

Nagios odczytuje definicje poleceń pluginów z katalogu */etc/nagios-plugins/config*, gdy plik */etc/nagios3/nagios.cfg* zawiera poniższą linijkę:

```
cfg_dir=/etc/nagios-plugins/config
```

Przykład:

Domyślnie stan hostów sprawdzany jest za pomocą polecenia *check-host-alive*. Jego definicja znajduje się w pliku */etc/nagios-plugins/config/ping.cfg* i ma postać:

```
# 'check-host-alive' command definition
define command{
        command_name    check-host-alive
       command_line    /usr/lib/nagios/plugins/check_ping -H
'$HOSTADDRESS$' -w 5000,100% -c 5000,100% -p 1
        }
```

Jak widzimy polecenie to wywołuje program */usr/lib/nagios/plugins/check_ping* i poprzez makro *$HOSTADDRESS$* przekazuje mu adres IP do sprawdzenia.

A.3.3. Obsługa zdarzeń

Obsługa zdarzeń włączana jest za pomocą zmiennej *enable_event_handlers* w głównym pliku konfiguracyjnym poprzez przypisanie jej wartości *1*. Dodatkowo musimy włączyć obsługę zdarzeń także dla każdego z hostów i każdej z usług osobno nadając zmiennej *event_handler_enabled* wartość *1* w deklaracji obiektu.

Programy obsługujące zdarzenia (może to być dowolny rodzaj pliku wykonywalnego obsługiwanego przez system operacyjny) muszą przyjmować co najmniej następujące makra jako argumenty:

Dla usługi: $SERVICESTATE$, $SERVICESTATETYPE$, $SERVICEATTEMPT$.

Dla hosta: $HOSTSTATE$, $HOSTSTATETYPE$, $HOSTATTEMPT$.

Należy pamiętać, że użytkownik systemowy z uprawnieniami którego działa Nagios musi mieć prawo wykonywania programów obsługujących zdarzenia. Domyślnie w Debianie użytkownikiem tym jest *nagios*.

A.3.4. Wymagania

Interfejs WWW do działania wymaga oczywiście serwera WWW. Najbardziej rozpowszechnionym serwerem WWW na systemy Linuksowe jest Apache. W Debianie Nagios posiada już gotowe pliki konfiguracyjne do Apache.

A.3.5. Instalacja

Instalacja Nagiosa nie jest skomplikowana. Więcej czasu należy poświęcić jego konfiguracji. Wszystkie czynności instalacyjne wykonujemy jako użytkownik *root*. Najpierw instalujemy niezbędne pakiety:

```
nms:~# aptitude install nagios3 nagios-plugins nagios-nrpe-plugin apache2
Czytanie list pakietów... Gotowe
Budowanie drzewa zależności
Odczyt informacji o stanie... Gotowe
Odczyt dodatkowych informacji o stanie
Inicjalizacja stanów pakietów... Gotowe
Odczyt opisów zadań... Gotowe
Następujące NOWE pakiety zostaną zainstalowane:
  apache2 apache2-mpm-worker{a} apache2-utils{a} apache2.2-common{a}
  fping{a} libapr1{a} libaprutil1{a} libexpat1{a} libfreetype6{a}
  libgd2-noxpm{a} libjpeg62{a} libmysqlclient15off{a} libnet-snmp-perl{a}
  libperl5.10{a} libpng12-0{a} libpq5{a} libradius1{a}
  libradiusclient-ng2{a} libsensors3{a} libsnmp-base{a} libsnmp15{a}
  libsysfs2{a} libtalloc1{a} libwbclient0{a} mysql-common{a}
  nagios-images{a} nagios-nrpe-plugin nagios-plugins
  nagios-plugins-basic{a} nagios-plugins-standard{a} nagios3
  nagios3-common{a} nagios3-doc{a} openssl{a} openssl-blacklist{a} qstat{a}
  radiusclient1{a} samba-common{a} smbclient{a} snmp{a} ssl-cert{a}
0 pakietów aktualizowanych, 41 instalowanych, 0 do usunięcia i 0 nie
aktualizowanych.
Do pobrania 33,5MB archiwów. Zajęte po rozpakowaniu: 82,5MB.
Kontynuować? [T/n/?] T
[...]
```

Na wszystkie pytania konfiguracyjne odpowiadamy zgodnie z domyślnymi odpowiedziami.

Po zainstalowaniu Nagiosa należy wstępnie go skonfigurować. Na początku usuwamy niepotrzebne sufiksy typu *_nagiosX* z nazw plików w katalogu *conf.d*:

```
# ls /etc/nagios3/conf.d
# rename 's,_nagios.,,' /etc/nagios3/conf.d/*; ls /etc/nagios3/conf.d
./  ../  contacts_nagios2.cfg  extinfo_nagios2.cfg  generic-
host_nagios2.cfg  generic-service_nagios2.cfg  host-gateway_nagios3.cfg
hostgroups_nagios2.cfg  localhost_nagios2.cfg  services_nagios2.cfg
timeperiods_nagios2.cfg
./  ../  contacts.cfg  extinfo.cfg  generic-host.cfg  generic-service.cfg
host-gateway.cfg  hostgroups.cfg  localhost.cfg  services.cfg
timeperiods.cfg
```

A.3.6. Monitorowanie sieci testowej

Skonfigurujemy teraz Nagiosa tak, aby monitorował naszą testową sieć.

Należy pamiętać, że proces Nagiosa działa z uprawnieniami dedykowanego użytkownika i musi mieć uprawnienia do czytania wszystkich plików konfiguracyjnych. Aby za każdym razem nie poprawiać uprawnień ręcznie stworzymy skrypt */etc/nagios3/restart* o następującej treści:

```
#!/bin/sh

chown -R nagios /etc/nagios3
nagios3 -v /etc/nagios3/nagios.cfg
if [ "$?" = "0" ]; then
        /etc/init.d/nagios3 restart
fi
```

Należy pamiętać aby nadać plikowi zawierającymi skrypt możliwość wykonywania:

```
# chmod 700 /etc/nagios3/restart
```

Skrypt ten zmienia właściciela wszystkich plików i katalogów w */etc/nagios3* na użytkownika *nagios*. Następnie sprawdza poprawność plików konfiguracyjnych. Jeśli nie znajdzie żadnych błędów, to zrestartuje program. Zabezpieczy to nas jednocześnie przed restartowaniem aplikacji jeśli pliki konfiguracyjne zawierają błędy.

Pliki znajdujące się w katalogu */etc/nagios3/conf.d* nie będą nam potrzebne, konfigurację stworzymy sami od podstaw. Usuwamy pliki poleceniem:

```
# rm /etc/nagios3/conf.d/*
```

Najpierw stworzymy przedziały czasowe. Poniżej znajduje się zawartość pliku */etc/nagios3/conf.d/timeperiods.cfg*:

```
define timeperiod {
        timeperiod_name 24x7
        alias           24 godziny na dobę, 7 dni w tygodniu
        sunday          00:00-24:00
        monday          00:00-24:00
        tuesday         00:00-24:00
        wednesday       00:00-24:00
        thursday        00:00-24:00
        friday          00:00-24:00
        saturday        00:00-24:00
}

define timeperiod {
        timeperiod_name godziny_pracy
        alias           Standardowe godziny pracy
        monday          09:00-17:00
        tuesday         09:00-17:00
        wednesday       09:00-17:00
        thursday        09:00-17:00
        friday          09:00-17:00
}
```

Jak widać zdefiniowane zostały dwa przedziały o nazwach *24x7* oraz *godziny_pracy*. Pierwszy z nich obejmuje cały tydzień. Drugi obejmuje tylko standardowe godziny pracy biura: od poniedziałku do piątku w godzinach 9:00 - 17:00.

Następnie zdefiniujemy kontakty. Nowa zawartość pliku */etc/nagios3/conf.d/contacts.cfg* prezentuje się następująco:

```
define contact {
        name                            contact-template
        service_notification_period     24x7
        host_notification_period        24x7
        service_notification_options    w,u,c,r
        host_notification_options       d,r
        service_notification_commands   notify-service-by-email
        host_notification_commands      notify-host-by-email
        register                        0
}
define contact {
        use                             contact-template
        contact_name                    jkowalski
        alias                           Jan Kowalski
        email                           jkowalski@asd.com
}
define contactgroup {
        contactgroup_name               admins
        alias                           Nagios Administrators
        members                         jkowalski
}
define contact {
        use                             contact-template
        contact_name                    pwisniewski
        alias                           Piotr Wiśniewski
        email                           pwisniewski@asd.com
}
define contact {
        use                             contact-template
        contact_name                    kkowalczyk
        alias                           Krzysztof Kowalczyk
        email                           kkowalczyk@asd.com
}
define contactgroup {
        contactgroup_name               dbadmins
        alias                           Administratorzy baz danych
        members                         pwisniewski, kkowalczyk
}
define contact {
        use                             contact-template
        contact_name                    wwitkowski
        alias                           Witold Witkowski
        email                           wwitkowski@asd.com
}
define contact {
        use                             contact-template
        contact_name                    zmalinowski
        alias                           Zbigniew Malinowski
        email                           zmalinowski@asd.com
}

define contactgroup {
        contactgroup_name               wwwadmins
        alias                           Administratorzy WWW
        members                         wwitkowski, zmalinowski
}
```

Kontakty oraz grupy kontaktów zdefiniowaliśmy tak, aby odzwierciedlały stan rzeczywisty.

Jak widzimy użyliśmy w definicjach szablonu. Utworzyliśmy szablon kontaktu o nazwie *contact-template* i nie musieliśmy wielokrotnie powtarzać części słów kluczowych.

Powiadomienia e-mailowe domyślnie wysyłane są za pomocą systemowego polecenia *mail*, które korzysta z lokalnie zainstalowanego serwera MTA. Konfiguracja MTA to zupełnie osobne zagadnienie i nie będziemy go tutaj poruszać. W Internecie można znaleźć wiele źródeł informacji na ten temat. W Debianie domyślnym MTA jest Exim. Jego wstępnej konfiguracji (która w większości prostych serwerów w zupełności wystarcza) można dokonać wydając polecenie:

```
# dpkg-reconfigure exim4-config
```

Teraz utworzymy obiekty hostów dla wszystkich naszych serwerów. Oto lista plików konfiguracyjnych, które należy utworzyć w katalogu */etc/nagios3/conf.d* wraz z ich zawartością:

Plik *host_template.cfg*:

```
define host {
        name                            host-template
        notifications_enabled           1
        event_handler_enabled           1
        flap_detection_enabled          1
        process_perf_data               1
        retain_status_information       1
        retain_nonstatus_information    1
        check_command                   check-host-alive
        max_check_attempts              10
        notification_interval           0
        notification_period             24x7
        notification_options            d,u,r
        contact_groups                  admins
        register                        0
}
```

Plik *host_db1.asd.com.cfg*:

```
define host {
        use             host-template
        host_name       db1.asd.com
        alias           Serwer DB 1
        address         10.0.50.40
        parents         nms.asd.com
        contact_groups  admins,dbadmins
}
```

Plik *host_db2.asd.com.cfg*:

```
define host {
        use             host-template
        host_name       db2.asd.com
        alias           Serwer DB 2
        address         10.0.50.50
        parents         nms.asd.com
        contact_groups  admins,dbadmins
}
```

Plik host_dns.asd.com.cfg:

```
define host {
        use             host-template
        host_name       dns.asd.com
        alias           Serwer DNS
        address         10.0.50.51
        parents         nms.asd.com
}
```

Plik *host_nms.asd.com.cfg*:

```
define host {
        use             host-template
        host_name       nms.asd.com
        alias           Serwer monitoringowy
        address         127.0.0.1
}
```

Plik *host_router.asd.com.cfg*:

```
define host {
        use             host-template
        host_name       router.asd.com
        alias           Router
        address         10.0.50.1
        parents         nms.asd.com
}
```

Plik *host_www1.asd.com.cfg*:

```
define host {
        use             host-template
        host_name       www1.asd.com
        alias           Serwer WWW 2
        address         10.0.50.66
        parents         nms.asd.com
        contact_groups  admins,wwwadmins
}
```

Plik *host_www2.asd.com.cfg*:

```
define host {
        use             host-template
        host_name       www2.asd.com
        alias           Serwer WWW 1
        address         10.0.50.67
        parents         nms.asd.com
        contact_groups  admins,wwwadmins
}
```

Stworzyliśmy szablon hosta o nazwie *host-template*, który jest używany przez wszystkie zdefiniowane hosty. Dzięki temu definicje hostów są dość krótkie i zwięzłe.

Za pomocą dyrektywy *parents* określiliśmy serwer nadrzędny (w warstwie 3 modelu ISO/ISO) dla każdego z serwerów. Wyjątkiem jest serwer nms.asd.com, który różni się od pozostałych dwiema rzeczami:

- jego adres IP to 127.0.0.1, a nie adres z puli 10.0.50.0/24
- nie posiada określonego hosta-rodzica

Jest tak, ponieważ w Nagiosie mapę sieci definiuje się względem hosta na którym działa Nagios. To on jest centrum monitorowanej sieci.

Ponieważ mamy już zdefiniowane hosty, to możemy przypisać każdemu z nich chociaż po jednej usłudze do sprawdzenia. Stworzymy również i wykorzystamy grupę hostów. Poniżej przedstawiono pliki wraz z zawartością, które należy utworzyć:

Plik */etc/nagios3/conf.d/hostgroup_ssh.cfg*:

```
define hostgroup {
        hostgroup_name  ssh-servers
        alias           SSH servers
        members         db1.asd.com, db2.asd.com, dns.asd.com, nms.asd.com,
router.asd.com, www1.asd.com, www2.asd.com
}
```

Plik */etc/nagios3/conf.d/service_template.cfg*:

```
define service {
        name                            service-template
        active_checks_enabled           1
        passive_checks_enabled          1
        obsess_over_service             1
        check_freshness                 0
        notifications_enabled           1
        event_handler_enabled           1
        flap_detection_enabled          1
        process_perf_data               1
        retain_status_information       1
        retain_nonstatus_information    1
        notification_interval           0
        is_volatile                     0
        check_period                    24x7
        check_interval                  5
        retry_interval                  1
        max_check_attempts              4
        notification_period             24x7
        notification_options            w,u,c,r
        contact_groups                  admins
        register                        0
}
```

Plik */etc/nagios3/conf.d/service_ssh.cfg*:

```
define service {
        hostgroup_name                  ssh-servers
        service_description             SSH
        check_command                   check_ssh
        use                             service-template
}
```

W tym momencie wstępna wersja konfiguracji Nagiosa jest już gotowa. Możemy zrestartować program (aby odczytał nowe pliki konfiguracyjne):

```
# /etc/nagios3/restart
```

Aby móc zalogować się do interfejsu WWW musimy jeszcze każdemu z administratorów

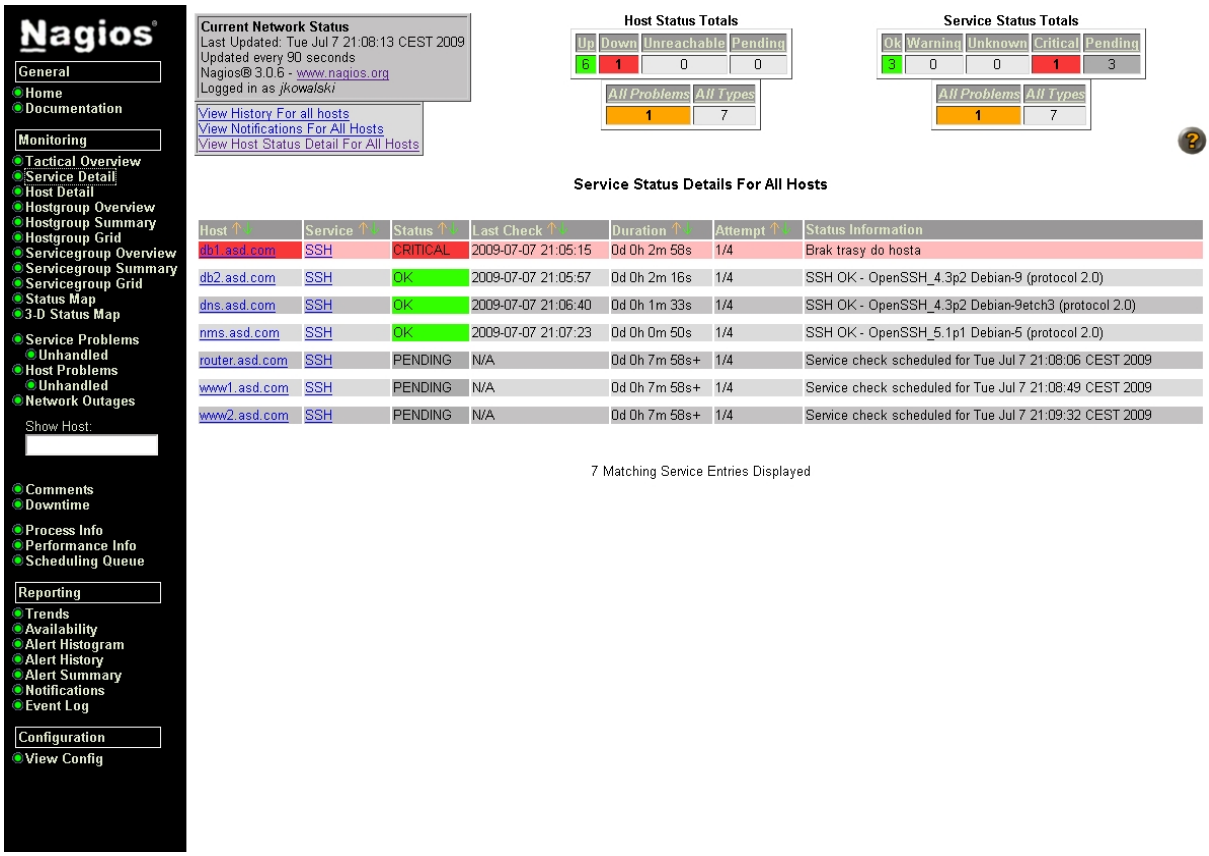
utworzyć hasło. Procedura ta została opisana w głównej części pracy w rozdziale 3.1.4 w części *Autoryzacja HTTP*.

Interfejs WWW ma adres *https://nms.asd.com/nagios3/*. Jeśli przeglądarka WWW wyświetli nam ostrzeżenie na temat certyfikatu SSL, to należy je zignorować.

Po wejściu na podaną stronę zostaniemy w pierwszej kolejności poproszeni o podanie loginu i hasła. Logujemy się na użytkownika *jkowalski*. Strona powitalna została zaprezentowana na Rys. A.3.1.

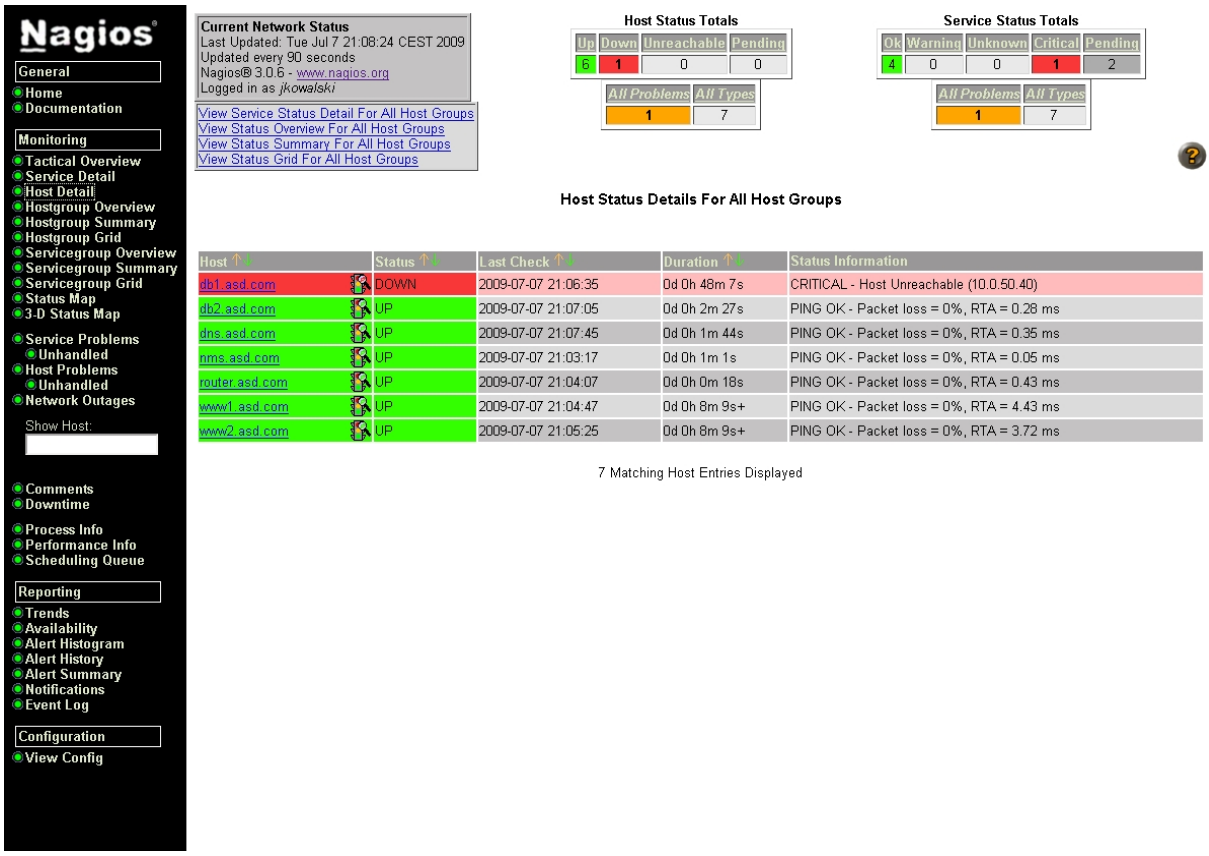
Rys. A.3.1. Nagios - strona powitalna.

Po kliknięciu w lewej kolumnie na *Service Detail* zobaczymy stronę z listą usług (Rys. A.3.2.). Na tym etapie wygląda ona jeszcze dość ubogo.



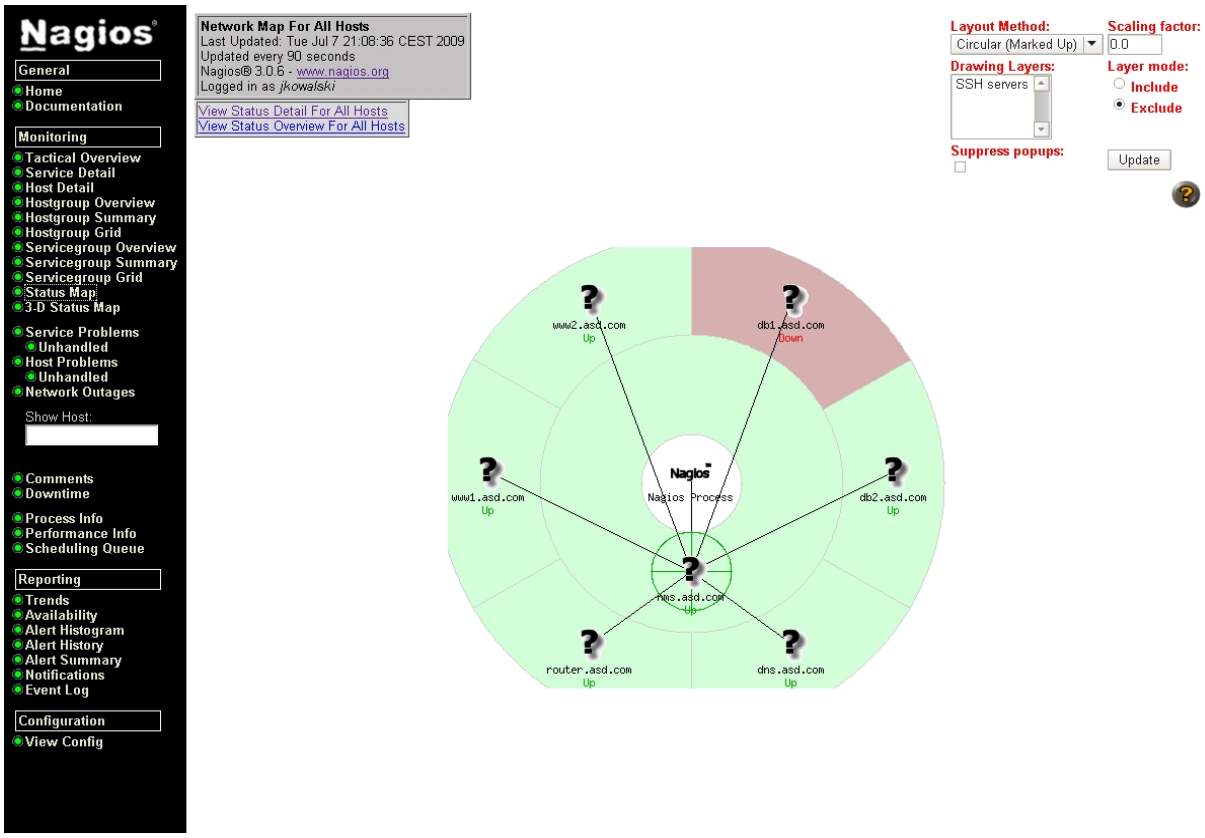
Rys. A.3.2. Nagios - lista usług.

Podobnie jest na stronie z listą hostów (*Host Detail*) (Rys. A.3.3.).



Rys. A.3.3. Nagios - lista hostów.

Widok mapy (*Status Map*) utworzonej na podstawie zawartości zmiennej *parents* w definicjach hostów zobaczymy na Rys. A.3.4.



Rys. A.3.4. Nagios - mapa hostów.

A.3.7. Pluginy

Poniżej znajduje się lista wartych uwagi pluginów dostarczanych i instalowanych standardowo w Debianie. Definicje wszystkich z nich znajdują się w katalogu */etc/nagios-plugins/conf*. Wszystkie z opisanych poniżej pluginów zbierają dane z systemu na którym zostały uruchomione.

A.3.7.1. check_disk

Argumenty:

1: poziom ostrzegawczy

2: poziom krytyczny

3: punkt montowania.

Sprawdza ilość wolnego miejsca na dysku zamontowanym w podanym katalogu. Ilośc wolnego miejsca podajemy w procentach.

Przykład użycia:

```
define service {
        use                             service-template
        host_name                       nms.asd.com
        service_description             Free space on /var
        check_command                   check_disk!20%!10%!/var
}
```

A.3.7.2. check_all_disks

Argumenty:

1: poziom ostrzegawczy

2: poziom krytyczny.

Sprawdza ilość wolnego miejsca na każdym z używanych dysków. Jeśli wolnego miejsca jest mniej niż określiliśmy w argumentach, to zwraca odpowiedni stan błędu.

Plugin ten jest o tyle wygodny, że nie musimy definiować osobnej usługi dla każdego z dysków w serwerze. Wszystkie dyski będą sprawdzane przez jedną usługę.

Plugin ten jest zdefiniowany w pliku */etc/nagios-plugins/config/disk.cfg*.

Przykład użycia:

Aby dodać do hosta nms.asd.com usługę sprawdzającą ilość wolnego miejsca na dyskach należy utworzyć plik o podanej poniżej zawartości:

```
define service {
        use                             service-template
        host_name                       nms.asd.com
        service_description             Disk Space
        check_command                   check_all_disks!20%!10%
}
```

A.3.7.3. check_users

Argumenty:

1: poziom ostrzegawczy

2: poziom krytyczny.

Sprawdza ilość zalogowanych do systemu użytkowników.

Przykład użycia:

```
define service {
        use                             service-template
        host_name                       nms.asd.com
        service_description             Current Users
        check_command                   check_users!20!50
}
```

A.3.7.4. check_procs

Argumenty:

1: poziom ostrzegawczy

2: poziom krytyczny.

Sprawdza ilość uruchomionych w systemie procesów.

Przykład użycia:

```
define service {
        use                             service-template
        host_name                       nms.asd.com
        service_description             Total Processes
        check_command                   check_procs!250!400
}
```

A.3.7.5. check_procs_zombie

Argumenty:

1: poziom ostrzegawczy

2: poziom krytyczny.

Sprawdza ilość uruchomionych w systemie procesów będących w stanie Zombie. W prawidłowo działającym systemie nie powinno być wiele takich procesów.

Przykład użycia:

```
define service {
        use                             service-template
        host_name                       nms.asd.com
        service_description             Zombie Processes
        check_command                   check_procs_zombie!10!20
}
```

A.3.7.6. check_procs_httpd

Argumenty:

1: poziom ostrzegawczy

2: poziom krytyczny.

Sprawdza ilość uruchomionych w lokalnym systemie procesów o nazwie pliku wykonywalnego *httpd*.

Przykład użycia:

```
define service {
        use                             service-template
        host_name                       nms.asd.com
        service_description             HTTPd Processes
        check_command                   check_procs_httpd!100!120
}
```

A.3.7.7. check_load

Argumenty:

1: poziom ostrzegawczy obciążenia badanego w okresie jednej minuty

2: poziom ostrzegawczy obciążenia badanego w okresie pięciu minut

3: poziom ostrzegawczy obciążenia badanego w okresie piętnastu minut

4: poziom krytyczny obciążenia badanego w okresie jednej minuty

5: poziom krytyczny obciążenia badanego w okresie pięciu minut

6: poziom krytyczny obciążenia badanego w okresie piętnastu minut.

Sprawdza obciążenie (load) systemu. Jeśli chociaż jeden wskaźnik obciążenia zostanie przekroczony, to plugin zwróci stan ostrzegawczy lub krytyczny (zależnie od tego jak wysokie będzie obciążenie. Np. Jeśli w poniższym przykładzie obciążenie systemu będzie wynosiło *2.00, 3.00, 4.00*, to plugin zwróci stan ostrzegawczy - obciążenie badane w okresie 15 minut przekroczyło ustalony przez nas stan ostrzeżenia.

Przykład użycia:

```
define service {
        use                             service-template
        host_name                       nms.asd.com
        service_description             Current Load
        check_command                   check_load!5.0!4.0!3.0!10.0!6.0!4.0
}
```

A.3.7.8. check_http

Nie wymaga podawania argumentu.

Sprawdza czy na hoście o nazwie podanej jako wartość zmiennej *host_name* działa serwer WWW.

Przykład użycia:

```
define service {
        use                             service-template
        host_name                       www1.asd.com
        service_description             HTTP
        check_command                   check_http
}
```

A.3.7.9. check_https

Nie wymaga podawania argumentu.

Sprawdza czy na hoście o nazwie podanej jako wartość zmiennej *host_name* działa serwer HTTPS (HTTP over SSL).

Przykład użycia:

```
define service {
        use                             service-template
        host_name                       www2.asd.com
        service_description             HTTPS
        check_command                   check_https
}
```

A.3.7.10. check_ping

Sprawdza dostępność danego hosta PING-ując go (pakiety ICMP ECHO/REPLY).

Argumenty:

1: poziom ostrzegawczy

2: poziom krytyczny.

Każdy z argumentów ma postać *OPÓŹNIENIE,STRATA*. *OPÓŹNIENIE* to maksymalne opóźnienie wyrażane w milisekundach. *STRATA* to maksymalna ilość pakietów, które mogą ulec zaginięciu wyrażona w procentach.

Przykład użycia:

```
define service {
        use                             service-template
        host_name                       router.asd.com
        service_description             PING
        check_command                   check_ping!100.0,20%!500.0,60%
}
```

A.3.7.11. check_dhcp

Sprawdza czy na podanym hoście działa serwer DHCP.

Nie przyjmuje żadnych argumentów.

Przykład użycia:

```
define service {
        use                     service-template
        host_name               dhcp.asd.com
        service_description     DHCP
        check_command           check_dhcp
}
```

A.3.7.12. check_dns

Sprawdza czy na podanym hoście działa serwer DNS odpytując go o domenę *www.google.com*.

Nie przyjmuje żadnych argumentów.

Przykład użycia:

```
define service {
        use                     service-template
        host_name               dns.asd.com
        service_description     DNS
        check_command           check_dns
}
```

A.3.7.13. check_ftp

Sprawdza czy na podanym hoście działa serwer FTP.

Nie przyjmuje żadnych argumentów.

Przykład użycia:

```
define service {
        use                     service-template
        host_name               www1.asd.com
        service_description     FTP
        check_command           check_ftp
}
```

A.3.7.14. check_ldap

Sprawdza czy na podanym hoście działa serwer LDAP.

Argumenty:

1: Podstawowy DN.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               dns.asd.com
       service_description     LDAP
       check_command           check_ldap!dc=asd,dc=com
}
```

A.3.7.15. check_ldaps

Sprawdza czy na podanym hoście działa serwer LDAP z szyfrowaniem poprzez TLS.

Argumenty:

1: Podstawowy DN.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               dns.asd.com
       service_description     LDAPS
       check_command           check_ldaps!dc=asd,dc=com
}
```

A.3.7.16. check_pop

Sprawdza czy na podanym hoście działa serwer POP3.

Nie przyjmuje argumentów.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               mail.asd.com
       service_description     POP3
       check_command           check_pop
}
```

A.3.7.17. check_spop

Sprawdza czy na podanym hoście działa serwer SPOP3 (POP3 over SSL) na porcie TCP 995.

Nie przyjmuje argumentów.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               mail.asd.com
       service_description     SPOP3
       check_command           check_spop
}
```

A.3.7.18. check_smtp

Sprawdza czy na podanym hoście działa serwer SMTP.

Nie przyjmuje argumentów.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               mail.asd.com
       service_description     SMTP
       check_command           check_smtp
}
```

A.3.7.19. check_ssmtp

Sprawdza czy na podanym hoście działa serwer SSMTP (SMTP over SSL).

Nie przyjmuje argumentów.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               mail.asd.com
       service_description     SMTP
       check_command           check_ssmtp
}
```

A.3.7.20. check_imap

Sprawdza czy na podanym hoście działa serwer IMAP.

Nie przyjmuje argumentów.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               mail.asd.com
       service_description     IMAP
       check_command           check_imap
}
```

A.3.7.21. check_simap

Sprawdza czy na podanym hoście działa serwer SIMAP (IMAP over SSL) na porcie TCP 993.

Nie przyjmuje argumentów.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               mail.asd.com
       service_description     SIMAP
       check_command           check_simap
}
```

A.3.7.22. check_mailq_exim

Sprawdza ilość listów w kolejce MTA Exim. Zdefiniowane są także odpowiedniki tego pluginu dla następujących MTA:

- Sendmail
- Qmail
- Postfix.

Argumenty:

1: ilość listów, która ma generować poziom ostrzegawczy

2: ilość listów, która ma generować poziom krytyczny.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               nms.asd.com
       service_description     mailq
       check_command           check_mail_exim
}
```

A.3.7.23. check_mysql

Sprawdza czy na podanym hoście/hostach działa serwer MySQL.

Nie przyjmuje argumentów.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               db1.asd.com
       service_description     MySQL
       check_command           check_mysql
}
```

A.3.7.24. check_ntp

Sprawdza czy na podanym hoście/hostach działa serwer NTP.

Nie przyjmuje argumentów.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               ntp.asd.com
       service_description     NTP
       check_command           check_ntp
}
```

A.3.7.25. check_pgsql

Sprawdza czy na podanym hoście/hostach działa serwer PostgreSQL.

Nie przyjmuje argumentów.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               db2.asd.com
       service_description     PostgreSQL
       check_command           check_pgsql
}
```

A.3.7.26. check_tcp

Sprawdza czy na podanym hoście/hostach na określonym porcie TCP coś nasłuchuje.

Argumenty:

1: Numer portu TCP do sprawdzenia.

Przykład użycia:

```
define service {
       use                     service-template
       host_name               nms.asd.com
       service_description     Syslog
       check_command           check_tcp!514
}
```

A.3.8. NRPE

NRPE należy skonfigurować po stronie serwera monitorującego oraz monitorowanego.

Najpierw przedstawimy konfigurację serwera monitorowanego.

Serwer NRPE dostarczany jest przez pakiet *nagios-nrpe-server*. Oprócz serwera będziemy potrzebowali jeszcze pluginów, które będą sprawdzały działanie usług. Wydajemy więc polecenie:

```
# aptitude install nagios-nrpe-server nagios-plugins
```

Następnie należy skonfigurować serwer NRPE. Jego pliki konfiguracyjne znajdują się w katalogu */etc/nagios* i są to:

- *nrpe.cfg* - główny plik konfiguracyjny
- *nrpe_local.cfg* - tutaj możemy umieszczać lokalne dla danego serwera zmiany w konfiguracji albo definicje usług.

W plikach konfiguracyjnych umieszczamy ogólne dyrektywy oraz definicje usług. Składnia tych ostatnich jest dużo prostsza niż w plikach konfiguracyjnych Nagiosa. Zwykle jest to jedna linijka, która łączy wymyśloną przez nas nazwę usługi z konkretnym poleceniem systemowym. Polecenie to jest ścieżką do programu sprawdzającego działanie jakiejś usługi wraz z argumentami. W pliku *nrpe.cfg* umieszczone są już standardowe dyrektywy konfiguracyjne. Powinniśmy dostosować wartość zmiennej *allowed_hosts*, która zawiera listę adresów IP oddzielonych przecinkami. Adresy te mają możliwość odpytywać demona NRPE. Aby Nagios mógł działać prawidłowo musimy dopisać tam jego adres IP (**10.0.50.79**).

Wartości domyślne pozostałych zmiennych będą dla nas wystarczające.

Domyślnie skonfigurowane są już usługi o nazwach *check_users* (sprawdza liczbę zalogowanych do systemu użytkowników), *check_load* (sprawdza obciążenie systemu), *check_hda1* (sprawdza ilość wolnego miejsca na dysku /dev/hda1), *check_zombie_procs* (sprawdza liczbę procesów w stanie Zombie), *check_total_procs* (sprawdza liczbę wszystkich procesów uruchomionych w systemie).

Plugin *check_hda1* raczej nie będzie nam potrzebny i możemy (ale nie musimy) możemy jego deklarację usunąć z pliku konfiguracyjnego bądź poprzedzić znakiem komentarza (#).

Przykład deklaracji wymienionych powyżej usług:

```
command[check_users]=/usr/lib/nagios/plugins/check_users -w 5 -c 10
command[check_load]=/usr/lib/nagios/plugins/check_load -w 15,10,5 -c
30,25,20
command[check_hda1]=/usr/lib/nagios/plugins/check_disk -w 20% -c 10% -p
/dev/hda1
command[check_zombie_procs]=/usr/lib/nagios/plugins/check_procs -w 5 -c 10
-s Z
command[check_total_procs]=/usr/lib/nagios/plugins/check_procs -w 150 -c
200
```

Jak widzimy każda deklaracja ma postać:

```
command[NAZWA]=/ŚCIEŻKA/DO/PROGRAMU ARGUMENT1 ARGUMENT2 ...ARGUMENTx
```

Nazwy usług zwykle zaczynają się od prefiksu *check_*. Natomiast programy sprawdzające usługi znajdują się w katalogu */usr/lib/nagios/plugins*.

Dyrektywy konfiguracyjne NRPE możemy też zapisywać w plikach umieszczonych w katalogu */etc/nagaios/nrpe.d*. Domyślnie katalog ten nie istnieje, ale demon NRPE próbuje odczytać jego zawartość gdyż w pliku *nrpe.cfg* jest następujące polecenie konfiguracyjne:

```
include_dir=/etc/nagios/nrpe.d/
```

Po każdej zmianie zawartości plików konfiguracyjnych należy zrestartować demona poleceniem:

```
# /etc/init.d/nagios-nrpe-server restart
```

Demon NRPE nasłuchuje na porcie TCP 5666.

Konfiguracja NRPE po stronie serwera monitorującego wygląda następująco:

Do odpytywania serwera NRPE używany jest plugin check_nrpe. Jego plik wykonywalny to */usr/lib/nagios/plugins/check_nrpe*. Aby z poziomu powłoki systemowej przetestować działanie serwera NRPE możemy wydać polecenie:

```
# /usr/lib/nagios/plugins/check_nrpe -H 10.0.50.50 -c check_load
```

10.0.50.50 to adres IP serwera NRPE, a *check_load* to nazwa usługi. Przykładowy wynik takiego polecenia:

```
# /usr/lib/nagios/plugins/check_nrpe -H 10.0.50.50 -c check_load
OK - load average: 8.01, 8.01, 8.00|load1=8.010;15.000;30.000;0;
load5=8.010;12.000;25.000;0; load15=8.000;10.000;20.000;0;
```

Zmodyfikujemy konfigurację naszego Nagiosa tak, aby za pomocą NRPE monitorował obciążenie następujących serwerów:

- dns.asd.com
- www1.asd.com
- www2.asd.com
- db1.asd.com
- db2.asd.com

W pierwszej kolejności musimy zainstalować oraz skonfigurować NRPE na wymienionych serwerach. Na każdym z nich wydajemy więc polecenia:

```
# aptitude install nagios-nrpe-server nagios-plugins
# sed -i -e 's,allowed_hosts=127.0.0.1, allowed_hosts=10.0.50.79,'
# /etc/nagios/nrpe.cfg
# /etc/init.d/nagios-nrpe-server restart
```

Kolejne czynności wykonujemy na serwerze nms.asd.com. Tworzymy grupę hostów, do której później przypiszemy nowa usługę:

Plik */etc/nagios3/conf.d/hostgroup_nrpe_load.cfg*:

```
define hostgroup {
        hostgroup_name  nrpe-load-servers
        alias           NRPE load servers
        members         db1.asd.com, db2.asd.com, dns.asd.com,
router.asd.com, www1.asd.com, www2.asd.com
}
```

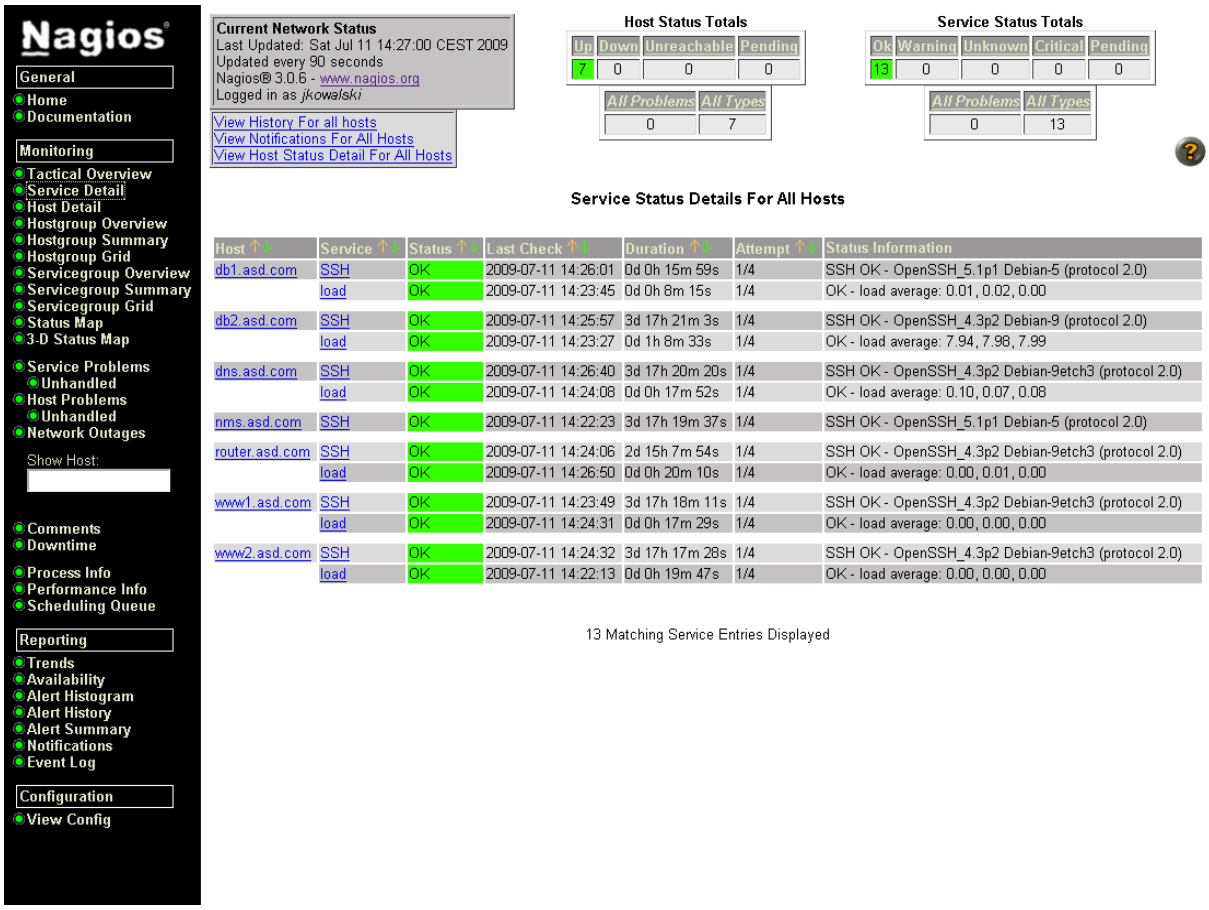
Plik */etc/nagios3/conf.d/service_nrpe_load.cfg*:

```
define service {
        hostgroup_name                  nrpe-load-servers
        service_description             load
        check_command                   check_nrpe_1arg!check_load
        use                             service-template
}
```

Po zmodyfikowaniu konfiguracji należy zrestartować Nagiosa. Po kilkunastu minutach, kiedy nowe usługi zostaną sprawdzone, widok listy usług w interfejsie WWW będzie wyglądał podobnie do tego na Rys. A.3.5.

A.3.9. Tworzenie własnych pluginów

Skrypt *check_mem* opisany w zasadniczej części pracy należy umieścić na każdym z hostów, na którym ma być sprawdzana ilość zajętej pamięci. Skrypt zapisać możemy np. w pliku */usr/local/bin/check_mem*. Oczywiście należy nadać plikowi ze skryptem atrybut wykonywalności.



Rys. A.3.5. Przykładowy widok po zmodyfikowaniu usług.

Na serwerze na którym działa Nagios należy utworzyć definicję polecenia:

Plik */etc/nagios-plugins/config/mem.cfg*:

```
define command {
        command_name    check_mem
        command_line    /usr/local/bin/check_mem -w 90 -c 95
}
```

W przypadku hosta nms.asd.com należy utworzyć następującą definicję usługi:

Plik */etc/nagios3/conf.d/service_mem_for_nms.asd.com.cfg*:

```
define service {
        host_name                       nms.asd.com
        service_description             mem
        check_command                   check_mem
        use                             service-template
}
```

Na hostach db1.asd.com, db2.asd.com, dns.asd.com, router.asd.com, www1.asd.com i www2.asd.com zużycie pamięci będzie monitorowane poprzez NRPE. W tym celu na każdym z tych serwerów należy umieścić skrypt w podanej wcześniej lokalizacji. Dodatkowo do pliku */etc/nagios/nrpe_local.cfg* należy dopisać następującą definicję usługi:

```
command[check_mem]=/usr/local/bin/check_mem -w 90 -c 95
```

Po takiej zmianie konfiguracji należy zrestartować na każdym z tych serwerów demona NRPE.

Na serwerze nms.asd.com utworzymy grupę hostów na których będziemy monitorować zużycie pamięci oraz przypiszemy im odpowiednią usługę. Oto pliki jakie należy utworzyć wraz z ich zawartością:

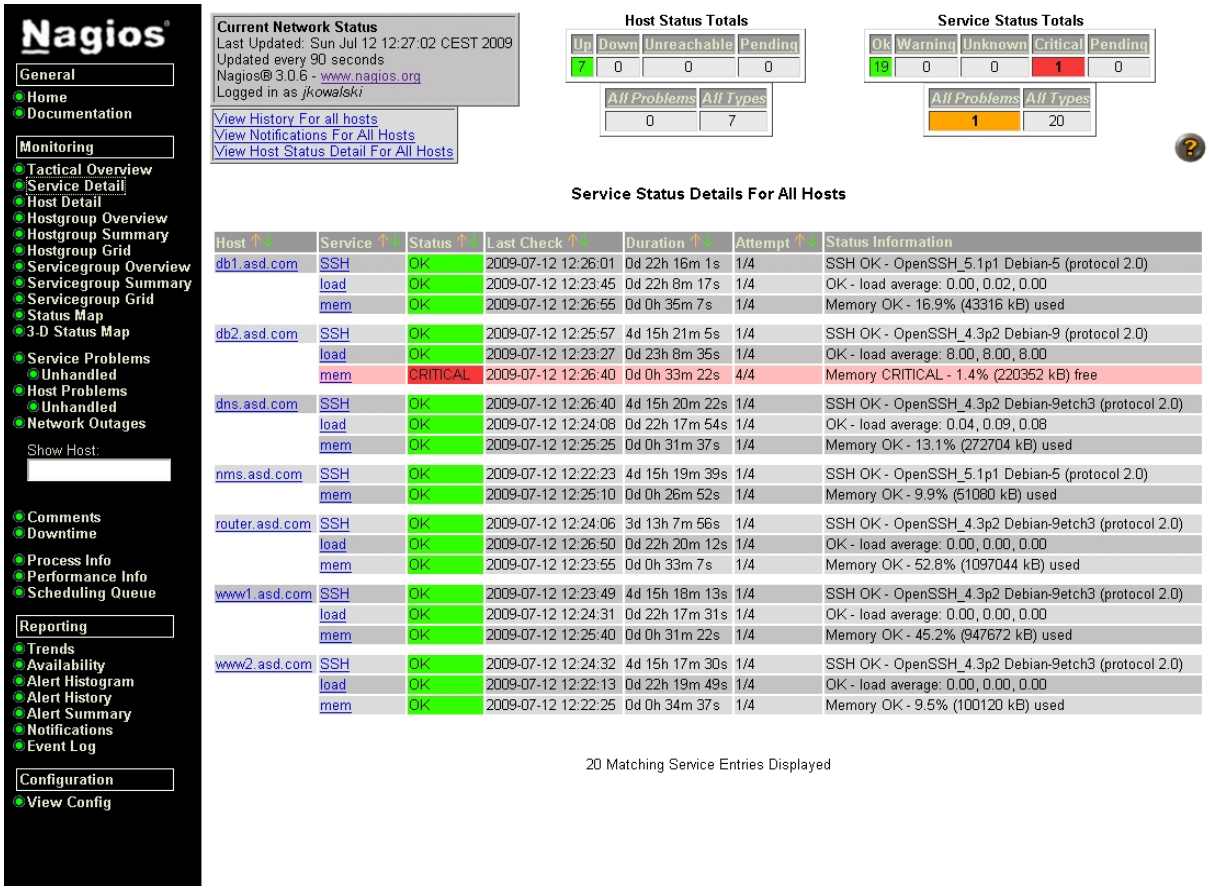
Plik */etc/nagios3/conf.d/hostgroup_nrpe_mem.cfg*:

```
define hostgroup {
        hostgroup_name  nrpe-mem-servers
        alias           NRPE mem servers
        members         db1.asd.com, db2.asd.com, dns.asd.com,
router.asd.com, www1.asd.com, www2.asd.com
}
```

Plik */etc/nagios3/conf.d/service_nrpe_mem.cfg*:

```
define service {
        hostgroup_name                  nrpe-mem-servers
        service_description             mem
        check_command                   check_nrpe_1arg!check_mem
        use                             service-template
}
```

Po zrestartowaniu Nagiosa i odczekaniu kilkunastu minut, widok listy usług w interfejsie WWW powinien być podobny do tego na Rys. A.3.6.



Rys. A.3.6. Widok listy usług po dodaniu usługi sprawdzającej ilość używanej pamięci operacyjnej.

Jak widzimy na zrzucie ekranu, na serwerze db2.asd.com stan użycia pamięci jest krytyczny.

A.3.10. Zewnętrzne pluginy na przykładzie Dell OMSA

W Internecie możemy znaleźć wiele pluginów napisanych przez osoby trzecie. Istnieje także wiele portali, które zbierają w jednym miejscu i katalogują. Przykładem takiego portalu jest *http://www.monitoringexchange.org/*. Posiada rozbudowaną bazę pluginów, którą możemy przeszukiwać. Jeden z takich pluginów wykorzystamy w naszym środowisku testowym.

Każdy z większych producentów serwerów (IBM, HP, Dell, Intel) dostarcza rozwiązania umożliwiające monitorowanie stanu sprzętu z poziomu systemu operacyjnego zainstalowanego na nim. W przypadku serwerów używanych w naszym środowisku testowym jest to technologia Dell OpenManage. Po zainstalowaniu w systemie operacyjnym (wspierany jest Windows i Linux) agenta OpenManage, mamy dostęp do informacji o stanie różnych urządzeń wchodzących w skład serwera. Możemy sprawdzać temperaturę, prędkości wentylatorów, stan poszczególnych dysków wchodzących w skład macierzy, wersje oprogramowania zainstalowanego na różnych podzespołach, itd.

Do tego celu wykorzystamy plugin *check_openmanage*, który możemy znaleźć na stronie *http://folk.uio.no/trondham/software/check_openmanage.html*. Wcześniej jednak musimy na każdym z serwerów które chcemy monitorować zainstalować agenta Dell OpenManage. Procedurę tę omówimy tylko skrótowo gdyż nie jest ona głównym zagadnieniem omawianym w tej pracy.

Stan sprzętu będziemy monitorować na serwerach www1.asd.com oraz www2.asd.com. Procedura instalacji agenta OpenManage na każdym z nich została opisana poniżej. Jest ona charakterystyczna dla systemu operacyjnego Debian GNU/Linux i na innych systemach może znacznie się różnić. Więcej informacji na temat Dell OpenManage znajdziemy na stronie *www.dell.com*.

Repozytorium z pakietami instalacyjnymi dla Debiana znajdziemy pod adresem *ftp://ftp.sara.nl/pub/sara-omsa/*. Procedura instalacji jest następująca:

Pobieramy klucz GPG którym podpisane są pakiety:

```
# wget -O - http://ftp.sara.nl/debian_sara.asc | apt-key add  -
```

Dodajemy repozytorium z pakietami do /etc/apt/sources.list:

```
# echo "deb ftp://ftp.sara.nl/pub/sara-omsa dell sara" >>
/etc/apt/sources.list
```

Aktualizujemy bazę pakietów:

```
# apt-get update
```

Instalujemy agenta OpenManage:

```
# apt-get install dellomsa
```

Teraz możemy przystąpić do instalacji pluginu sprawdzającego stan serwera:

```
# wget -O /usr/local/bin/check_openmanage
http://folk.uio.no/trondham/software/check_openmanage-
3.4.6/check_openmanage
# chmod a+rx /usr/local/bin/check_openmanage
```

Przykład uruchomienia plugina z poziomu powłoki systemowej:

```
# /usr/local/bin/check_openmanage
OK - System: 'PowerEdge 2950', SN: '1234567', hardware working fine, 0
logical drives, 0 physical drives
```

Następnie tworzymy plik */etc/nagios/nrpe.d/omsa.cfg* o nazstępującej treści:

```
command[check_omsa]=/usr/local/bin/check_openmanage
```

Po czym restartujemy serwer NRPE:

```
# /etc/init.d/nagios-nrpe-server restart
```

Opisana wyżej procedura przygotuje hosty do monitorowania. Czynności konfiguracyjne po stronie Nagiosa prezentują się następująco:

Tworzymy definicję grupy hostów w pliku */etc/nagios3/conf.d/hostgroup_nrpe_omsa.cfg*:

```
define hostgroup {
        hostgroup_name  nrpe-omsa-servers
        alias           NRPE omsa servers
        members         www1.asd.com, www2.asd.com
}
```

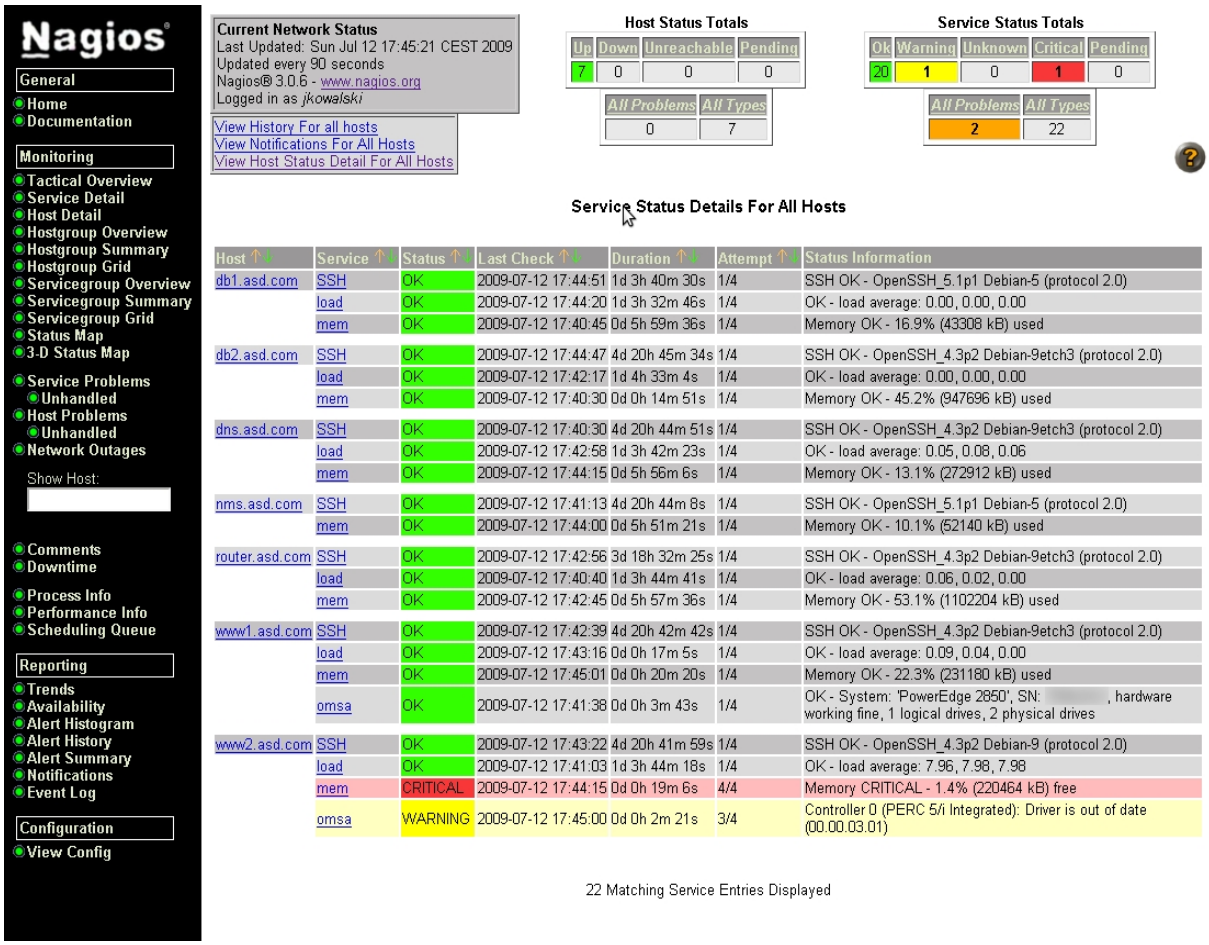
Definiujemy usługę *omsa* w pliku */etc/nagios3/conf.d/service_nrpe_omsa.cfg*:

```
define service {
        hostgroup_name                  nrpe-omsa-servers
        service_description             omsa
        check_command                   check_nrpe_1arg!check_omsa
        use                             service-template
}
```

Restartujemy Nagiosa i odczekujemy kilkanaście minut aż zostaną wykonane sprawdzenia dodanych przez nas usług.

Przykładowy zrzut ekranu z dodanymi nowymi usługami zaprezentowany został na Rys. A.3.7.

Jak widzimy na liście usług hostów www1.asd.com oraz www2.asd.com pojawiły się usługi, które skonfigurowaliśmy. Na pierwszym z nich OpenManage pokazuje, że wszystko działa poprawnie. Natomiast na drugim z nich kontroler RAID (PERC to skrót od PowerEdge RAID Controller) ma nieaktualny firmware co powoduje, że Nagios wyświetla ostrzeżenie.



Rys. A.3.7. Widok listy usług po dodaniu usługi *OMSA*.

A.3.11. Obsługa zdarzeń

Omówimy teraz implementację obsługi zdarzeń na przykładzie serwera WWW. Serwerowi www2.asd.com dodamy usługę *HTTP* i skonfigurujemy jej skrypt obsługujący zdarzenia.

Tworzymy plik */etc/nagios3/conf.d/service_http_for_www2.asd.com.cfg* o następującej treści:

```
define service {
        host_name                       nms.asd.com
        service_description             http
        check_command                   check_http
        use                             service-template
        event_handler                   restart-httpd
}
```

Do pliku */etc/nagios3/commands.cfg* dopisujemy poniższą deklarację:

```
define command {
        command_name    restart-httpd
        command_line    /usr/local/bin/nagios-restart-httpd $SERVICESTATE$
$SERVICESTATETYPE$ $SERVICEATTEMPT$ $HOSTADDRESS$
}
```

Zawartość skryptu */usr/local/bin/nagios-restart-httpd*:

```
#!/bin/sh

servicestate=$1
servicestatetype=$2
serviceattempt=$3
server=$4

if [ "$servicestate" = "CRITICAL" ]; then
        if [ "$servicestatetype" = "SOFT" ]; then
                if [ "$serviceattempt" = "3" ]; then
                        ssh "root@${server}" "/etc/init.d/apache2 restart"
                fi
        else # HARD
                ssh "root@${server}" "/etc/init.d/apache2 restart"
        fi
fi
exit 0
```

Nie należy zapominać o nadaniu odpowiednich praw plikowi ze skryptem oraz zmianie właściciela:

```
# chown nagios:nagios /usr/local/bin/nagios-restart-httpd
# chmod 750 /usr/local/bin/nagios-restart-httpd
```

Utworzyliśmy usługę o nazwie *http* i przypisaliśmy ją do hosta www2.asd.com. Usłudze przypisaliśmy także polecenie, które ma być wywołane w momencie jej awarii. Polecenie to wywołuje skrypt */usr/local/bin/nagios-restart-httpd* z odpowiednimi argumentami. Skrypt do poprawnego działania wymaga, aby użytkownik *nagios* mógł zalogować się za pomocą SSH na serwer www2.asd.com na użytkownika root. Należy użyć do tego autoryzacji kluczem SSH niezabezpieczonym hasłem. Opis konfiguracji SSH wykracza jednak poza zakres tego dokumentu.

Skrypt *nagios-restart-httpd* zrestartuje serwer WWW jednak tylko w dwóch sytuacjach:

- Jeśli po raz trzeci z rzędu usługa będzie w stanie krytycznym i będzie to stan typu miękkiego.
- Jeśli usługa pierwszy raz będzie w stanie krytycznym i będzie to stan typu twardego.

Inaczej moglibyśmy doprowadzić do tego, że serwer WWW byłby restartowany raz za razem w nieskończoność.

Jeśli dwa restarty nie przywrócą poprawnego funkcjonowania usługi to z pewnością niezbędna będzie interwencja administratora.

ZAŁĄCZNIK B

Do pracy dołączona jest płyta CD zawierająca elektroniczne wersje dokumentu wraz z Załącznikiem A w formatach PDF i ODT.