Nagios Version 3.x Dokumentation

http://www.nagios.org

Last Updated: 01-15-2009
[Inhaltsverzeichnis]

Deutsche Übersetzung: Wolfgang Nieder
(A.d.Ü.: die Übersetzung ist so genau wie möglich am Ursprungstext orientiert. Leider gibt es bei Begriffen immer wieder mehrere Möglichkeiten, so dass ich mich für eine Version entscheiden musste. Oft habe ich den englischen Ausdruck in Klammern hinzugefügt, um dem Leser eine eigene Auswahl zu ermöglichen. Die Berücksichtigung der neuen deutschen Rechtschreibung ist mir nicht immer gelungen, teilweise habe ich auch bewusst dagegen verstoßen, gerade wenn dabei Wortungetüme mit zig Silben entstanden wären.
Falls im Text von "ich" die Rede ist, bezieht sich das auf den Autor der Originaldokumentation, Ethan Galstad [soweit nicht anders angegeben]).

Nagios 3.x Dokumentation

Inhaltsverzeichnis

Über Nagios

Hoch zu: Inhalt

Siehe auch: Schnellstart-Installationsanleitungen

Was ist das?

Nagios® ist eine System- und Netzwerküberwachungsapplikation. Sie überwacht Hosts und Services, die Sie angeben, und alarmiert Sie, wenn sich die Dinge verschlechtern und wenn sie wieder besser werden.

Nagios wurde ursprünglich entwickelt, um unter Linux zu laufen, allerdings sollte es unter den meisten anderen Unix-Derivaten ebenfalls funktionieren.

Einige der vielen Features von Nagios umfassen:

- Überwachen von Netzwerkdiensten (SMTP, POP3, HTTP, NNTP, PING, etc.)
- Überwachen von Host-Ressourcen (Prozessorauslastung, Diskbelegung, usw.)
- Einfaches Plugin-Design, das es Benutzern erlaubt, schnell eigene Service-Prüfungen zu entwickeln
- Parallel laufende Service-Prüfungen
- Die Möglichkeit, Netzwerk-Host-Hierarchien mit Hilfe von "Eltern"-Hosts zu definieren, um die Erkennung von Hosts zu erlauben, die "down" sind und die Unterscheidung zwischen Hosts, die "down" bzw. unerreichbar sind.
- Benachrichtigung von Kontakten, wenn Service- oder Host-Probleme auftreten bzw. gelöst werden (über e-Mail, Pager oder benutzerdefinierte Methoden)
- Die Möglichkeit, Routinen zur Ereignisbehandlung (event handler) zu definieren, die bei Host- oder Service-Ereignissen ablaufen, um proaktive Problemlösungen zu erlauben
- Automatische Rotation von Protokolldateien
- Unterstützung, um redundante Überwachungs-Hosts zu implementieren
- Optionales Web-Interface, um den aktuellen Netzwerkstatus, Benachrichtigungs- und Problemverläufe, Protokolldateien usw. anzusehen

Systemvoraussetzungen

Die einzige Voraussetzung für den Betrieb von Nagios ist eine Maschine, auf der Linux (oder eine UNIX-Variante) läuft und ein C-Compiler. Voraussichtlich werden Sie auch noch TCP/IP konfigurieren wollen, weil die meisten Service-Prüfungen über das Netzwerk durchgeführt werden.

Sie *müssen* nicht die CGIs benutzen, die in Nagios enthalten sind. Wenn Sie sich allerdings entscheiden, sie zu benutzen, muss zusätzlich die folgende Software installiert sein...

1. Ein Web-Server (vorzugsweise Apache)
2. Thomas Boutells gd library Version 1.6.3 oder höher (wird benötigt von den statusmap- und trends-CGIs)

Lizenzierung

Nagios ist unter den Bedingungen der GNU General Public License Version 2 lizenziert, wie sie von der Free Software Foundation veröffentlicht wird. (A.d.Ü.: eine deutsche Übersetzung finden Sie hier.) Die Lizenz gibt Ihnen das Recht, Nagios unter bestimmten Bedingungen zu kopieren, zu verteilen und/oder zu modifizieren. Lesen Sie die 'LICENSE'-Datei in der Nagios-Distribution oder lesen Sie die (englische) Online-Version der Lizenz für nähere Details.

Nagios wird geliefert WIE ES IST OHNE GARANTIE IRGENDEINER ART, EINSCHLIESZLICH DER GARANTIE FÜR DESIGN, VERMARKTBARKEIT ODER DER TAUGLICHKEIT FÜR EINEN BESTIMMTEN ZWECK.

Danksagungen

Verschiedene Leute haben zu Nagios beigetragen, z.B. durch Meldung von Fehlern, vorschlagen von Verbesserungen, schreiben von Plugins usw. Eine Liste von einigen der vielen Mitwirkenden an der Entwicklung von Nagios finden Sie auf http://www.nagios.org.

Beschaffen der neuesten Version

Sie können auf http://www.nagios.org nach neuen Versionen von Nagios suchen.

Was gibt es Neues in Nagios 3

Hoch zu: Inhalt

Wichtig: Bitte lesen Sie die Dokumentation und die FAQs unter nagios.org, bevor Sie eine Frage an die Mailing-Listen stellen.

Änderungsprotokoll

Das (englischsprachige) Änderungsprotokoll für Nagios finden Sie online unter http://www.nagios.org/development/changelog.php oder in der **Changelog**-Datei im root-Verzeichnis der Source-Code-Distribution.

Änderungen und neue Features

1. **Dokumentation**:
   - **Dokumentationsaktualisierungen** - ich (A.d.Ü.: der Autor, nicht der Übersetzer ;-)) kämpfe mich langsam durch das Umschreiben aller Teile der Dokumentation. Dies wird eine Weile dauern, da es (1) eine Menge Dokumentation ist und (2) das Schreiben der Dokumentation nicht zu meinen Lieblingsbeschäftigungen zählt. Es kann also sein, dass vorübergehend einige Teile nicht ganz zu anderen passen. Ich hoffe, dass die Änderungen die Dinge einfacher/klarer für neue und erfahrene Benutzer gleichermaßen machen.
2. **Makros**:
   - **Neue Makros** - Neue Makros wurden hinzugefügt, dazu gehören: $TEMPPATH$, $LONGHOSTOUTPUT$, $LONGSERVICEOUTPUT$, $HOSTNOTIFICATIONID$, $SERVICENOTIFICATIONID$, $HOSTEVENTID$, $SERVICEEVENTID$, $SERVICEISVOLATILE$, $LASTHOSTEVENTID$, $LASTSERVICEEVENTID$, $HOSTDISPLAYNAME$, $SERVICEDISPLAYNAME$, $MAXHOSTATTEMPTS$, $MAXSERVICEATTEMPTS$, $TOTALHOSTSERVICES$, $TOTALHOSTSERVICESOK$, $TOTALHOSTSERVICESWARNING$, $TOTALHOSTSERVICESUNKNOWN$, $TOTALHOSTSERVICESCRITICAL$, $CONTACTGROUPNAME$, $CONTACTGROUPNAMES$, $CONTACTGROUPALIAS$, $CONTACTGROUPMEMBERS$, $NOTIFICATIONRECIPIENTS$, $NOTIFICATIONISESCALATED$, $NOTIFICATIONAUTHOR$, $NOTIFICATIONAUTHORNAME$, $NOTIFICATIONAUTHORALIAS$, $NOTIFICATIONCOMMENT$, $EVENTSTARTTIME$, $HOSTPROBLEMID$, $LASTHOSTPROBLEMID$, $SERVICEPROBLEMID$, $LASTSERVICEPROBLEMID$, $LASTHOSTSTATE$, $LASTHOSTSTATEID$, $LASTSERVICESTATE$, $LASTSERVICESTATEID$.
     Es gibt auch zwei neue spezielle Zeit-Makros bei Prüfungen nach Bedarf: $ISVALIDTIME:$ und $NEXTVALIDTIME:$.
   - **Entfernte Makros** - Das alte $NOTIFICATIONNUMBER$-Makro wurde zu Gunsten der neuen $HOSTNOTIFICATIONNUMBER$- und $SERVICENOTIFICATIONNUMBER$-Makros verworfen.
   - **Änderungen** - Die $HOSTNOTES$- und $SERVICENOTES$-Makros können nun selbst Makros enthalten, wie z.B. die $HOSTNOTESURL$-, $HOSTACTIONURL$-, $SERVICENOTESURL$- und $SERVICEACTIONURL$-Makros.
   - Makros sind normalerweise als Umgebungsvariablen verfügbar, wenn Prüfungen,

Eventhandler, Benachrichtigungen und andere Befehle ausgeführt werden. Das kann in großen Nagios-Installationen ziemlich CPU-intensiv sein, so dass Sie dieses Verhalten mit der enable_environment_macros-Option deaktivieren können.
   - Informationen zu Makros finden Sie hier.
3. **Geplante Ausfallzeiten** (Scheduled Downtime):
   - Einträge zu geplanten Ausfallzeiten werden nicht länger in einer eigenen Datei (bisher angegeben in der *downtime_file*-Direktive in der Hauptkonfigurationsdatei) gespeichert. Aktuelle und zurückgehaltene (retained) Einträge werden nun in der Statusdatei bzw. in der Statusaufbewahrungsdatei (retention file) gespeichert.
4. **Kommentare**:
   - Host- und Service-Kommentare werden nicht länger in einer eigenen Datei (bisher angegeben in der *comment_file*-Direktive in der Hauptkonfigurationsdatei) gespeichert. Aktuelle und zurückgehaltene (retained) Kommentare werden nun in der Statusdatei bzw. in der Statusaufbewahrungsdatei (retention file) gespeichert.
   - Bestätigungskommentare, die nicht als "persistent" gekennzeichnet sind, werden nur dann gelöscht, wenn die Bestätigung gelöscht wird. Sie wurden bisher automatisch beim (erneuten) Start von Nagios gelöscht, was nicht ideal war.
5. **aufbewahrte Zustandsdaten** (State Retention Data):
   - Statusinformationen für einzelne Kontakte werden nun über erneute Programmstarts hinweg aufbewahrt.
   - Kommentare und Ausfallzeit-IDs werden nun über erneute Programmstarts hinweg aufbewahrt und sollten eindeutig sein, solange diese aufbewahrten Daten nicht gelöscht oder ignoriert werden.
   - Die Variablen retained_host_attribute_mask und retained_service_attribute_mask wurden hinzugefügt, um zu kontrollieren, welche Host-/Service-Attribute global über erneute Programmstarts hinweg aufbewahrt werden
   - Die Variablen retained_process_host_attribute_mask und retained_process_service_attribute_mask wurden hinzugefügt, um zu kontrollieren, welche Host-/Service-Attribute global über erneute Programmstarts hinweg aufbewahrt werden
   - Die Variablen retained_contact_host_attribute_mask und retained_contact_service_attribute_mask wurden hinzugefügt, um zu kontrollieren, welche Host-/Service-Attribute global über erneute Programmstarts hinweg aufbewahrt werden
6. **Flattererkennung**:
   - Die *flap_detection_options*-Direktive wurde den Host- und Service-Definitionen hinzugefügt, um festzulegen, welche Host-/Service-Zustände bei der Flattererkennungslogik benutzt werden sollen (als Standard werden alle Zustände benutzt).
   - Prozentuale Zustandsänderungen und Statusverläufe werden nun aufbewahrt und aufgezeichnet, selbst wenn die Flattererkennung deaktiviert ist.
   - Hosts und Services werden sofort auf Flattern geprüft, wenn die Flattererkennung auf programmweiter Basis aktiviert ist.
   - Hosts und Services, die flattern, werden protokolliert, wenn die Flattererkennung auf programmweiter Basis deaktiviert ist.
   - Mehr Informationen zur Flattererkennung finden Sie hier.
7. **Externe Befehle**:
   - ein neuer externer Befehl PROCESS_FILE wurde hinzugefügt, um externe Befehle zu verarbeiten, die in einer externen (regulären) Datei stehen. Das ist nützlich, um große Mengen von passiven Prüfungen mit langen Ausgaben zu verarbeiten oder für das Scripting von "normalen" Befehlen. Mehr Informationen finden Sie hier.
   - eigene Befehle können nun an Nagios erteilt werden. Eigene Befehlsnamen beginnen mit einem Unterstrich (_) und werden nicht intern vom Nagios-Daemon verarbeitet. Sie können allerdings von einem geladenen NEB-Modul verarbeitet werden. (A.d.Ü.: NEB=Nagios Event Broker)
   - Die check_external_commands-Option ist nun per Default aktiviert, was bedeutet, dass Nagios

bereits im "Auslieferzustand" externe Befehle verarbeitet. Bei allen vorigen Nagios-Version (2.x und früher) ist diese Version deaktiviert.

8. **Statusdaten**:
   - Kontaktstatusinformationen (letzte Benachrichtigungszeit, Benachrichtigungen aktiviert/deaktiviert, usw.) werden nun in den Status- und Aufbewahrungs-Dateien gespeichert, obwohl sie nicht von den CGIs verarbeitet werden.
9. **eingebettetes Perl**:
   - Die Variablen enable_embedded_perl und use_embedded_perl_implicitly zur Kontrolle der Nutzung des eingebetteten Perl-Interpreters wurden hinzugefügt.
   - Perl-Scripts/Plugins können Nagios nun mitteilen, ob sie mit dem eingebetteten Perl-Interpreter ausgeführt werden sollen oder nicht. Das ist sinnvoll, wenn Sie einige schwierige Scripte haben, die nicht sauber mit ePN laufen.
   - Mehr Informationen zu diesen neuen Optionen finden Sie hier.
10. **Anpassungsfähige Überwachung**:
    - Die Zeitfenster für Host- und Service-Prüfungen können nun mit den entsprechenden externen Befehlen (CHANGE_HOST_CHECK_TIMEPERIOD bzw. CHANGE_SVC_CHECK_TIMEPERIOD) "im Fluge" geändert werden. Schauen Sie hier nach verfügbaren anpassungsfähigen Überwachungsbefehlen.
11. **Benachrichtigungen**:
    - Eine *first_notification_delay*-Option wurde den Host- und Service-Definitionen hinzugefügt, um (was sonst) eine Verzögerung einzuführen zwischen dem ersten Auftreten eines Host-Service-Problems und der ersten Benachrichtigung, die versandt wird. In früheren Versionen mussten Sie mächtig mit Eskalationen herumbasteln, um dies zu erreichen. Nun ist dieses Feature auch für Normalsterbliche verfügbar.
    - Benachrichtigungen werden nun für flatternde Host/Services versandt, wenn die Flattererkennung auf einer Host- oder Service-spezifischen oder programmweiten Basis deaktiviert ist. Das $NOTIFICATIONTYPE$-Makro hat in dieser Situation den Wert "FLAPPINGDISABLED".
    - Benachrichtigungen können versandt werden, wenn geplanten Ausfallzeiten (Downtimes) für Hosts oder Services beginnen, enden oder abgebrochen werden. Das $NOTIFICATIONTYPE$-Makro hat in diesem Fall den Wert "DOWNTIMESTART", "DOWNTIMEEND" oder "DOWNTIMECANCELLED". Um Benachrichtigungen zu Ereignissen bei geplanten Ausfallzeiten zu bekommen, geben Sie "s" oder "downtime" in Ihren Kontakt-, Host- und/oder Service-Benachrichtigungsoptionen an.
    - Mehr Informationen zu Benachrichtigungen finden Sie hier.
12. **Objektdefinitionen**:
    - Service-Abhängigkeiten können nun erstellt werden, um einfach "gleicher Host"-Abhängigkeiten für verschiedene Services auf einem oder mehreren Hosts zu definieren. (Lesen Sie mehr)
    - Erweiterte Host- und Service-Definitionen (hostextinfo bzw. serviceextinfo) sind "veraltet". Alle Werte der erweiterten Definitionen sind nun in Host- und Service-Definitionen eingeflossen. Nagios 3 wird alte erweiterte Informationsdefinitionen einlesen, aber eine Warnung ausgeben. Zukünftige Versionen von Nagios (4.x und später) werden keine separaten erweiterten Informationsdefinitionen mehr unterstützen.
    - Neue *hostgroup_members*-, *servicegroup_members*- und *contactgroup_members*-Direktiven wurden den hostgroup-, servicegroup- bzw. contactgroup-Definition hinzugefügt. Dies erlaubt es Ihnen, Hosts, Services oder Kontakte von Untergruppen in Ihre Gruppendefinitionen aufzunehmen.
    - Neue *notes*, *notes_url* und *action_url*-Direktiven wurden den Hostgroup- und Servicegroup-Definitionen hinzugefügt.
    - Kontaktdefinitionen enthalten die neuen *host_notifications_enabled*-, *service_notifications_enabled*- und *can_submit_commands*-Direktiven, um Benachrichtigungen besser zu kontrollieren und

festzulegen, ob sie Befehle über das Web-Interface erteilen dürfen.
- Host- und Service-Abhängigkeiten unterstützen nun eine optionale *dependency_period*-Direktive. Dies erlaubt Ihnen, die Zeiten zu begrenzen, in denen Abhängigkeiten gültig sind.
- Die *parallelize*-Direktive in Service-Definitionen ist nun veraltet und wird nicht länger benutzt. Alle Service-Prüfungen laufen in Nagios 3 parallel.
- Es gibt keine (inhärenten) Längenbegrenzungen bei Hostnamen oder Service-Beschreibungen mehr.
- Erweiterte reguläre Ausdrücke werden nun benutzt, wenn Sie die use_regexp_matching Konfigurationsoption aktivieren. "Matching" bei regulären Ausdrücken wird nur in bestimmten Objektdefinitionsdirektiven benutzt, die *****, **?**, **+** oder \ enthalten.
- Eine neue *initial_state*-Direktive wurde den Host- und Service-Definitionen hinzugefügt, so dass Sie Nagios mitteilen können, dass ein Host/Service einen bestimmten Status haben soll, wenn Nagios startet (statt UP bzw. OK, was immer noch der Default ist).

13. **Objektvererbung**:
    - Sie können nun Objektvariablen/-werte von mehreren Vorlagen (Templates) erben, indem Sie mehr als einen Vorlagennamen in der *use*-Direktive von Objektdefinitionen angeben. Dies kann einige sehr mächtige (und komplexe) Vererbungs-Setups erlauben. (Lesen Sie mehr)
    - Services können nun Kontaktgruppen, Benachrichtigungsintervalle und Benachrichtigungsperioden von ihrem verbundenen Host erben, wenn nichts anderes angegeben wurde. (Lesen Sie mehr)
    - Host- und Service-Eskalationen können nun Kontaktgruppen, Benachrichtigungsintervalle und Eskalationszeitfenster von ihrem verbundenen Host bzw. Service erben, wenn nichts anderes angegeben wurde. (Lesen Sie mehr)
    - Zeichenkettenvariablen in Host-, Service- und Kontaktdefinitionen können nun eine Vererbung verhindern, indem Sie einen Wert von "null" (ohne Anführungszeichen) als Wert der Variable angeben. (Lesen Sie mehr)
    - Die meisten Zeichenkettenvariablen in lokalen Objektdefinitionen können nun an die Zeichenkettenwerte angehängt werden, die vererbt werden. Dies ist ziemlich hilfreich in großen Konfigurationen. (Lesen Sie mehr)
14. **Leistungsverbesserungen**:
    - Es wurden Möglichkeiten hinzugefügt, Objektkonfigurationsdateien vorher zwischenzuspeichern und die Prüfung auf zirkuläre Pfaderkennung vom Überprüfungsprozess auszuschließen. Das kann die Anlaufzeit von Nagios in großen Umgebungen immens beschleunigen. Lesen Sie mehr hier.
    - Eine neue use_large_installation_tweaks-Option wurde hinzugefügt, die die Leistung in großen Umgebungen verbessern sollte. Lesen Sie mehr dazu hier.
    - Eine Reihe von internen Verbesserungen wurden gemacht mit Blick darauf, wie Nagios interne Datenstrukturen und Objekt- (z.B. Host- und Service-) Beziehungen behandelt. Diese Verbesserungen sollten in einer Beschleunigung von größeren Umgebungen resultieren.
    - Die neue external_command_buffer_slots-Option wurde hinzugefügt, um Ihnen die Skalierung in großen Umgebungen zu erleichtern. Um optimale Ergebnisse zu erzielen, sollten Sie überlegen, die Nutzung der Pufferbereiche über einen Zeitraum hinweg mit Hilfe von MRTG grafisch darzustellen.
15. **Plugin-Ausgaben**:
    - Mehrzeilige Ausgaben von Plugins wird nun für Host- und Service-Prüfungen unterstützt. Hurra! Die Plugin-API wurde aktualisiert, um mehrzeilige Ausgaben in einer Weise zu unterstützen, die Rückwärtskompatibilität mit älteren Plugins gewährleistet. Zusätzliche Ausgabezeilen (außer der ersten) werden in den neuen $LONGHOSTOUTPUT$- und $LONGSERVICEOUTPUT$-Makros gespeichert.
    - Die maximale Länge von Plugin-Ausgaben wurde auf 4K erhöht (von etwa 350 Bytes in früheren Versionen). Diese 4K-Grenze wurde willkürlich festgelegt, damit "durchdrehende"

Plugins nicht zuviel Daten an Nagios zurückliefern können.
- Mehr Informationen zu den Plugins, mehrzeiligen Ausgaben und maximaler Plugin-Ausgabelänge finden Sie hier.

16. **Service-Prüfungen**:
    - Nagios prüft nun per Default nach verwaisten Service-Prüfungen.
    - Eine neue enable_predictive_service_dependency_checks-Option wurde hinzugefügt, um zu kontrollieren, ob Nagios vorausschauende Prüfungen für Services initiiert, von denen andere abhängig sind (in Abhängigkeitsdefinitionen). Vorausschauende Prüfungen sollen helfen, dass die Abhängigkeitslogik so präzise wie möglich ist. (Lesen Sie mehr)
    - Ein neues "zwischengespeicherte Service-Prüfung"-Feature wurde implementiert, das die Leistung für viele Leute signifikant verbessern kann. Statt mit einem Plugin den Zustand eines Service zu prüfen, kann Nagios oft ein zwischengespeichertes Service-Prüfergebnis nutzen. Mehr Informationen darüber finden Sie hier.
17. **Host-Prüfungen**:
    - Host-Prüfungen laufen nun parallel! Host-Prüfungen liefen bisher nach einander, was bedeutete, dass sie für einen großen Stau im Sinne der Leistung sorgten. Das ist vorbei! (Lesen Sie hier)
    - Wiederholungsprüfungen für Hosts werden nun wie Wiederholungsprüfungen für Services durchgeführt. Das heißt, dass Host-Definitionen nun eine neue *retry_interval*-Direktive haben, die angibt, wieviel Zeit gewartet werden soll, bevor der Host erneut geprüft wird. :-)
    - Regelmäßig geplante Host-Prüfungen beeinträchtigen nun nicht länger die Leistung. Vielmehr können sie nun zusammen mit der neuen Zwischenspeicher-Prüflogik die Leistung steigern (siehe unten).
    - Die neue check_for_orphaned_hosts-Option erlaubt die Prüfung von verwaisten Host-Prüfungen. Das wird nun gebraucht, da Host-Prüfungen parallel laufen.
    - Die neue enable_predictive_host_dependency_checks-Option wurde hinzugefügt, um zu kontrollieren, ob Nagios vorausschauende Prüfungen für Hosts initiiert, von denen andere abhängig sind (in Abhängigkeitsdefinitionen). Vorausschauende Prüfungen sollen helfen, dass die Abhängigkeitslogik so präzise wie möglich ist. (Lesen Sie mehr)
    - Ein neues "zwischengespeicherte Host-Prüfung"-Feature wurde implementiert, das die Leistung für viele Leute signifikant verbessern kann. Statt mit einem Plugin den Zustand eines Hosts zu prüfen, kann Nagios oft ein zwischengespeichertes Host-Prüfergebnis nutzen. Mehr Informationen darüber finden Sie hier.
    - Passive Host-Prüfungen mit einem DOWN- oder UNREACHABLE-Ergebnis können nun automatisch in ihren korrekten Zustand aus Sicht der Nagios-Instanz umgesetzt werden, die sie empfängt. Das ist sehr nützlich in Failover- und verteilten Überwachungsumgebungen. Mehr Informationen zur Übersetzung passiver Host-Prüfzustände finden Sie hier.
    - Passive Host-Prüfungen setzen einen Host normalerweise in einen HARD-Zustand. Das kann nun durch die Aktivierung der passive_host_checks_are_soft-Option geändert werden.
18. **Frische-Prüfungen** (Freshness checks):
    - Eine neue freshness_threshold_latency-Option wurde hinzugefügt, damit Sie die Zahl von Sekunden angeben können, die zu jedem Host- oder Service-Frischeschwellwert hinzugerechnet werden soll, den Nagios automatisch ermittelt.
19. **IPC**:
    - Der IPC-Mechanismus, der benutzt wird, um Host-/Service-Prüfergebnisse von den (Enkel-)Kind-Prozessen zurück an den Nagios-Daemon zu liefern, wurde geändert! Das sollte helfen, Last-/Verzögerungsprobleme in Verbindung mit der Verarbeitung einer hohen Zahl von passiven Prüfungen in verteilten Umgebungen zu reduzieren.
    - Prüfergebnisse werden nun übertragen, indem sie in Dateien in das Verzeichnis geschrieben werden, das durch die check_result_path-Option angegeben wurde. Dateien, die älter sind als der in der max_check_result_file_age-Option angegebene Wert werden gnadenlos ohne weitere Verarbeitung gelöscht.

20. **Zeitfenster**:
    - Zeitfenster waren reif für eine gründliche Überarbeitung und wurden schließlich erweitert, um Datumsausnahmen, Datumssprünge (alle drei Tage) usw. zu erlauben! Das sollte Ihnen dabei helfen, wenn Sie Benachrichtungszeitfenster für Pager-Bereitschaften definieren.
    - Mehr Informationen zu den neuen Zeitfensterdirektiven finden Sie hier und hier.
21. **Ereignisvermittlung** (Event Broker):
    - Aktualisierte NEB-API-Version
    - modifizierter Rückruf (callback) für anpassungsfähige Programmstatusdaten
    - Rückruf (callback) für anpassungsfähige Kontaktstatusdaten hinzugefügt
    - Rückrufe vor der eigentlichen Prüfung von Hosts- und Services hinzugefügt, um es Modulen zu erlauben, interne Host-/Service-Prüfungen abzubrechen/zu übersteuern.
22. **Web-Interface**:
    - Hostgruppen- und Servicegruppen-Zusammenfassungen zeigen nun Aufschlüsselungen nach wichtigen/unwichtigen Problemen wie das TAC-CGI.
    - Kleinere Layout-Änderungen bei Host- und Service-Detailansichten im extinfo-CGI.
    - Neue Prüfstatistiken in der "Performance Info"-Anzeige hinzugefügt.
    - In verschiedenen CGIs Splunk-Integrationsoptionen hinzugefügt. Die Integration wird durch die enable_splunk_integration- und splunk_url-Optionen in der CGI-Konfigurationsdatei gesteuert.
    - Neue notes_url_target- und action_url_target-Optionen hinzugefügt, um zu kontrollieren, in welchem Frame notes- und action-URLs geöffnet werden.
    - Neue lock_author_names-Option hinzugefügt, um Änderungen des Autorennamens zu verhindern, wenn Benutzer Kommentare, Bestätigungen und geplante Ausfallzeiten eingeben.
23. **Debugging-Info**:
    - Die im Konfigurations-Script verfügbaren DEBUGx-Compile-Optionen wurden entfernt.
    - Debugging-Informationen können nun in eine separate Debug-Datei geschrieben werden, die automatisch rotiert wird, wenn sie eine bestimmte benutzerdefinierte Größe erreicht. Dies sollte das Debugging von Problemen viel einfacher machen, weil Sie Nagios nicht erneut kompilieren müssen. Die volle Unterstützung zum Schreiben von Debugging-Informationen in eine Datei wird während der Alpha-Entwicklungsphase eingebaut, so dass sie vielleicht noch nicht komplett ist, wenn Sie es ausprobieren.
    - Variablen, die das Debug-Protokoll im debug_file, den debug_level, die debug_verbosity und die max_debug_file_size beeinflussen.
24. **Update-Prüfungen**:
    - Nagios wird nun etwa einmal am Tag prüfen, ob eine neue Version verfügbar ist. Dies ist nützlich, um über Sicherheits-Updates oder neue Versionen informiert zu sein. Hinweise werden im Web-Interface zu sehen sein.
    - Variablen, die die Update-Prüfung beeinflussen, sind check_for_updates und bare_update_check.
25. **Verschiedenes**:
    - **Variable temporärer Pfad** - Eine neue temp_path-Variable wurde hinzugefügt, um ein Verzeichnis anzugeben, das Nagios für temporäre Dateien nutzen kann.
    - **eindeutige Benachrichtigungs- und Ereignisnummern** - Jeder Host- und Service-Benachrichtigung wird nun eine eindeutige ID-Nummer zugewiesen. Eine weitere eindeutige ID wird nun allen Host- und Service-Zustandsänderungen zugewiesen. Die eindeutige ID kann jeweils über die folgenden Makros abgefragt werden: $HOSTNOTIFICATIONID$, $SERVICENOTIFICATIONID$, $HOSTEVENTID$, $SERVICEEVENTID$, $LASTHOSTEVENTID$, $LASTSERVICEEVENTID$.
    - **Neue Makros** - Es gibt einige neue Makros (außer denen, die bereits an anderer Stelle erwähnt wurden). Es sind: $HOSTGROUPNAMES$, $SERVICEGROUPNAMES$, $HOSTACKAUTHORNAME$, $HOSTACKAUTHORALIAS$, $SERVICEACKAUTHORNAME$ und $SERVICEACKAUTHORALIAS$.

- **Erntefrequenz** (reaper_frequency) - Die alte *service_reaper_frequency*-Variable wurde umbenannt zu check_result_reaper_frequency, da sie nun auch benutzt wird, um Host-Prüfergebnisse zu verarbeiten.
- **Max. Erntezeit** (max reaper time) - Eine neue max_check_result_reaper_time-Variable wurde hinzugefügt, um die zulässige Laufeit eines einzelnen Ernteereignisses zu begrenzen.
- **Nichtganzzahlige Intervalle** - Nichtganzzahlige Benachrichtungs- und Prüfintervalle (z.B. "3.5" Minuten) werden nun in Host-, Service-, Host-Eskalations- und Service-Eskalationsdefinitionen unterstützt.
- **Maskierte Befehlsargumente** (Escaped command arguments) - Sie können nun Ausrufezeichen (!) in Ihren Befehlsargumenten benutzen, indem Sie diesen mit einem Backslash (\) maskieren. Wenn Sie einen Backslash in Ihren Befehlsargumenten verwenden müssen, dann muss auch dieser mit einem Backslash maskiert werden.
- **Mehrzeilige Systembefehlsausgabe** - Nagios wird nun mehrzeilige Ausgaben von Systembefehlen (Benachrichtigungs-Scripte usw.) bis zu einer Länge von 4K einlesen. Das passt zu der Grenze von Plugin-Ausgaben, die bereits genannt wurde. Ausgaben von Systembefehlen werden nicht direkt von Nagios verarbeitet, aber die Unterstützung gibt es trotzdem.
- **Bessere Planungsinformationen** - Detailliertere Informationen werden ausgegeben, wenn Nagios mit der -s Kommandozeilenoption ausgeführt wird. Diese Informationen können genutzt werden, um die Zeit des Starts/Neustarts von Nagios zu reduzieren.
- **Zusammengefasste Statusdateiaktualisierungen** - Die alte *aggregate_status_updates*-Option wurde entfernt. Alle Statusdateiaktualisierungen werden nun mit einem Intervall von mindestens einer Sekunde zusammengefasst.
- **Neuer Performance-Daten-Dateimodus** - Eine neue "p"-Option wurde den host_perfdata_file_mode- und service_perfdata_file_mode-Optionen hinzugefügt. Dieser neue Modus wird die Datei in einem nicht-blockierenden Schreib-/Lesemodus öffnen, der nützlich für Pipes ist.
- **Zeitzonen-Offset** - Eine neue use_timezone-Option wurde hinzugefügt, um verschiedene Nagios-Instanzen zu betreiben, die in anderen als der lokalen Zeitzone laufen.

Hinweise für Neulinge

Hoch zu: Inhalt

Siehe auch: Schnellstart-Installationsanleitung

Herzlichen Glückwunsch zur Wahl von Nagios! Nagios ist ziemlich mächtig und flexibel, aber es kann viel Arbeit bedeuten, es so zu konfigurieren, wie Sie es haben wollen. Sobald Sie damit vertraut sind, wie es funktioniert und was es für Sie tun kann, dann werden Sie nicht mehr ohne leben wollen. :-) Hier sind einige wichtige Dinge, die zu beachten sind, wenn Sie zum ersten Mal Nagios benutzen:

1. **Entspannen Sie sich - es wird einige Zeit dauern.** Erwarten Sie nicht, dass alles gleich so funktioniert, wie Sie sich das vorstellen. Das Aufsetzen von Nagios kann ein bisschen an Arbeit erfordern - teilweise wegen der Optionen, die Nagios bietet, teilweise weil Sie wissen müssen, was Sie in Ihrem Netzwerk überwachen wollen (und wie das am Besten zu tun ist).
2. **Nutzen Sie die Schnellstartanleitungen.** Die Schnellstart-Installationsanleitung ist so ausgelegt, dass die meisten neuen Benutzer ziemlich schnell ein einfaches Nagios zum Laufen bekommen. Innerhalb von 20 Minuten ist Nagios installiert und überwacht Ihr lokales System. Sobald das erledigt ist, können Sie lernen, wie Nagios konfiguriert wird, um mehr zu tun.
3. **Lesen Sie die Dokumentation.** Nagios kann schwierig zu konfigurieren sein, wenn Sie ein Gespür dafür haben, was passiert, und ziemlich unmöglich, wenn Sie keins haben. Stellen Sie sicher, dass Sie die Dokumentation lesen (besonders die Abschnitte "Nagios konfigurieren" und "Die Grundlagen"). Heben Sie sich die fortgeschrittenen Themen auf, bis Sie ein gutes Verständnis der Grundlagen haben.
4. **Suchen Sie die Hilfe von anderen.** Wenn Sie die Dokumentation gelesen haben, sich die Beispiel-Konfigurationsdateien angesehen und immer noch Probleme haben, dann senden Sie eine e-Mail mit der Beschreibung Ihrer Probleme an die *nagios-users*-Mailing-Liste. Aufgrund der Arbeit für dieses Projekt kann ich die meisten der direkt an mich gesandten Fragen nicht beantworten, so dass die beste Quelle die Mailing-Liste sein dürfte. Wenn Sie bereits einiges gelesen haben und eine gute Problembeschreibung liefern, dann stehen die Chancen gut, dass jemand Ihnen Hinweise geben kann, um die Dinge zum Laufen zu bringen. Mehr (englischsprachige) Informationen, wie Sie sich den Mailing-Listen anschließen oder die Archive durchsuchen können, finden Sie unter http://www.nagios.org/support/. Das deutsche Nagios-Portal finden Sie unter http://www.nagios-portal.de

Schnellstart-Installationsanleitungen

Hoch zu: Inhalt

Siehe auch: Nagios aktualisieren, Konfigurationsüberblick

Einführung

Diese Schnellstartanleitungen sind dazu gedacht, Ihnen einfache Anweisungen zu liefern, wie Sie Nagios innerhalb von 20 Minuten aus dem Quellcode installieren und Ihren lokalen Rechner damit überwachen. Hier werden keine fortgeschrittenen Installationsoptionen vorgestellt - lediglich die Grundlagen, die für 95% aller Benutzer funktionieren, die anfangen wollen.

Anleitungen

Schnellstart-Anleitungen sind momentan für die folgenden Linux-Distributionen verfügbar:

- Schnellstart Fedora
- Schnellstart openSUSE
- Schnellstart Ubuntu

Sie finden zusätzliche (englischsprachige) Schnellstartanleitungen im NagiosCommunity.org wiki. Sie finden keine Anleitung für Ihr Betriebssystem? Schreiben Sie eine Anleitung und stellen Sie diese ins Wiki!

Wenn Sie Nagios auf einem Betriebssystem oder einer Linux-Distribution installieren, die oben nicht aufgeführt ist, lesen Sie die Schnellstart Fedora-Anleitung, um einen Überblick zu bekommen, was zu tun ist. Befehlsnamen, Pfade und anderes variiert stark zwischen verschiedenen Betriebssystemen/Distributionen, so dass Sie wahrscheinlich die Installationsanleitung ein wenig anpassen müssen, damit sie für Ihren besonderen Fall funktioniert.

Anpassungen nach der Installation

Sobald Nagios installiert ist und funktioniert, wollen Sie ohne Zweifel mehr als nur Ihre lokale Maschine überwachen. Prüfen Sie die folgenden Dokumentationen, wie andere Dinge überwacht werden können...

- Windows-Maschinen überwachen
- Linux/Unix-Maschinen überwachen
- Netware-Server überwachen
- Router/Switches überwachen
- Netzwerkdrucker überwachen
- Öffentlich zugängliche Dienste überwachen (HTTP, FTP, SSH, etc.)

Nagios aktualisieren

Hoch zu: Inhalt

Siehe auch: Schnellstart-Installationsanleitung

Inhalt

aktualisieren von früheren Nagios 3.x-Versionen

Sobald neuere Alpha-, Beta- oder stabile Versionen von Nagios 3.x herauskommen, sollten Sie dringend über eine Aktualisierung nachdenken. Neuere Ausgaben enthalten Behebungen kritischer Fehler, so dass es wichtig ist, aktuell zu sein. Wenn Sie bereits Nagios wie in den Schnellstartanleitungen beschrieben aus dem Quellcode installiert haben, dann können Sie einfach neuere Versionen installieren. Sie müssen dazu noch nicht einmal root-Berechtigungen haben, weil bereits alles passiert ist, was als root-Benutzer getan werden muss. Hier der Aktualisierungsprozess...

Stellen Sie sicher, dass Sie eine gute Datensicherung Ihrer bestehenden Nagios-Installation und der Konfigurationsdateien haben. Wenn irgendetwas schief geht oder nicht funktioniert, dann können Sie auf diese Weise schnell Ihre alte Nagios-Version wiederherstellen.

Werden Sie der nagios-Benutzer. Debian/Ubuntu-Benutzer sollten *sudo -s nagios* benutzen.

```
su -l nagios
```

Holen Sie sich das Quellcode-Archiv der letzten Nagios-Version (besuchen Sie http://www.nagios.org/download/ für den Verweis auf die letzte Version).

```
wget http://osdn.dl.sourceforge.net/sourceforge/nagios/nagios-3.x.tar.gz
```

Entpacken Sie das Quellcode-Archiv.

```
tar xzf nagios-3.x.tar.gz
cd nagios-3.x
```

Starten Sie das Nagios-configure-Script mit dem Namen der Gruppe mit Berechtigungen für das "external command file", z.B. so:

```
./configure --with-command-group=nagcmd
```

Kompilieren Sie den Nagios-Quellcode.

```
make all
```

Installieren Sie aktualisierte Programme, Dokumentation und Web-Interface. Ihre vorhandenen Konfigurationsdaten werden in diesem Schritt nicht überschrieben.

```
make install
```

Überprüfen Sie Ihre Konfigurationsdateien und starten Sie Nagios erneut.

```
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
/sbin/service nagios restart
```

Das war's - Sie sind fertig!

Aktualisierung von Nagios 2.x

Es sollte nicht allzu schwierig sein, von Nagios 2.x auf Nagios 3.x zu aktualisieren. Die Aktualisierung ist im Wesentlichen die gleiche wie die von bestehenden 3.x-Versionen. Allerdings müssen Sie Ihre Konfigurationsdateien ein wenig ändern, damit sie mit Nagios 3 funktionieren:

- Die alte *service_reaper_frequency*-Variable in der Hauptkonfigurationsdatei wurde umbenannt in check_result_reaper_frequency.
- Das alte *$NOTIFICATIONNUMBER$*-Makro entfällt zugunsten der $HOSTNOTIFICATIONNUMBER$- und $SERVICENOTIFICATIONNUMBER$-Makros.
- Die alte *parallelize*-Direktive in Service-Definitionen ist veraltet und wird nicht länger benutzt, weil alle Service-Prüfungen parallel ablaufen.
- Die alte *aggregate_status_updates*-Option wurde entfernt. Alle Statusdatei-Aktualisierungen werden nun mit einem minimalen Intervall von einer Sekunde zusammengefasst.
- Erweiterte Host- und erweiterte Service-Definitionen sind veraltet. Sie werden noch von Nagios gelesen und verarbeitet, aber es wird empfohlen, dass Sie diese Direktiven in die entsprechenden Host- und Service-Definitionen verschieben.
- Die alte *downtime_file*-Dateivariable in der Hauptkonfigurationsdatei wird nicht länger unterstützt, weil Einträge von geplanten Ausfallzeiten (downtimes) nun in der Aufbewahrungsdatei (retention file) gespeichert werden. Um bestehende Einträge zu erhalten, stoppen Sie Nagios 2.x und hängen Sie den Inhalt Ihrer alten Downtime-Datei an das "retention file".
- Die alte *comment_file*-Dateivariable in der Hauptkonfigurationsdatei wird nicht länger unterstützt, weil Kommentare nun in der Aufbewahrungsdatei (retention file) gespeichert werden. Um bestehende Einträge zu erhalten, stoppen Sie Nagios 2.x und hängen Sie den Inhalt Ihrer alten Kommentar-Datei an die "Aufbewahrungsdatei" (retention file).

Stellen Sie außerdem sicher, dass Sie den "Was gibt's Neues"-Abschnitt in der Dokumentation lesen. Er beschreibt all die Änderungen am Nagios 3-Code, die seit der letzten stabilen Nagios 2.x-Version gemacht wurden. Es wurde einiges geändert, also werfen Sie einen Blick darauf.

aktualisieren einer RPM-Installation

Wenn Sie momentan eine RPM- oder Debian/Ubuntu-APT-paketbasierte Nagios-Installation haben und nun den Übergang zu einer Installation aus dem offiziellen Quellcode machen wollen, dann sind hier die grundlegenden Schritte:

1. Sichern Sie Ihre existierende Nagios-Installation
   - Konfigurationsdateien
     - Hauptkonfigurationsdatei (normalerweise *nagios.cfg*)
     - Ressource-Konfigurationsdatei (normalerweise *resource.cfg*)
     - CGI-Konfigurationsdatei (normalerweise *cgi.cfg*)
     - all Ihre Objektdefinitionsdateien
   - Aufbewahrungsdatei (normalerweise *retention.dat*)
   - die aktuelle Nagios-Protokolldatei (normalerweise *nagios.log*)
   - archivierte Nagios-Protokolldateien
2. Deinstallieren Sie die originalen RPM- oder APT-Pakete

3. Installieren Sie Nagios vom Quellcode, indem Sie der Schnellstartanleitung folgen
4. Sichern Sie Ihre Original-Nagios-Konfigurationsdateien, Aufbewahrungs- und Protokolldateien wieder zurück
5. Überprüfen Sie Ihre Konfiguration und starten Sie Nagios

Beachten Sie, dass verschiedene RPM- oder APT-Pakete Nagios auf verschiedene Weisen oder an verschiedenen Orten installieren. Stellen Sie sicher, dass Sie all Ihre kritischen Nagios-Dateien gesichert haben, bevor Sie das Original-RPM- oder APT-Paket entfernen, so dass Sie darauf zurückgreifen können, wenn Sie auf Probleme stoßen.

Windows-Maschinen überwachen

Hoch zu: Inhalt

Siehe auch: Schnellstart-Installationsanleitung, öffentlich zugängliche Dienste überwachen

Einführung

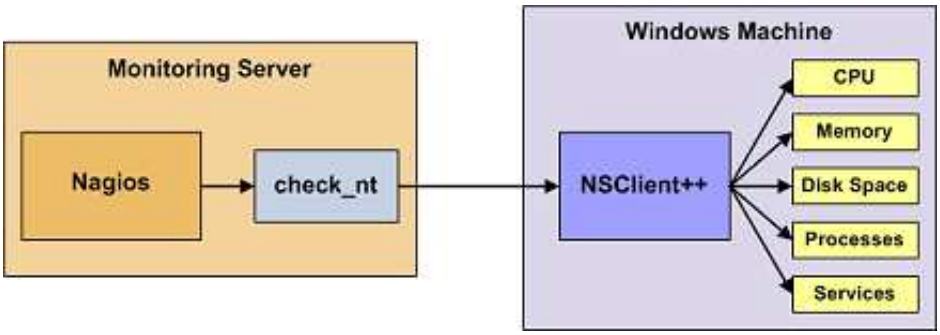
Dieses Dokument beschreibt, wie Sie "private" und Dienste und Attribute von Windows-Rechnern überwachen können, wie z.B.:

- Speicherbelegung
- CPU-Auslastung
- Plattenbelegung
- Zustände von Diensten
- laufende Prozesse
- etc.

Öffentlich nutzbare Dienste, die von Windows-Rechnern zur Verfügung gestellt werden (HTTP, FTP, POP3, etc.), können einfach mit Hilfe der Dokumentation öffentlich zugängliche Dienste überwachen kontrolliert werden.

Anmerkung: Diese Anweisungen gehen davon aus, dass Sie Nagios anhand der Schnellstartanleitung installiert haben. Die nachfolgenden Beispiel-Konfigurationseinträge beziehen sich auf Objekte, die in den Beispiel-Konfigurationsdateien (*commands.cfg*, *templates.cfg*, etc.) definiert sind. Diese Dateien werden installiert, wenn Sie der Schnellstartanleitung folgen.

Überblick



Die Überwachung von privaten Diensten oder Attributen eines Windows-Rechners erfordert die Installation eines Agenten. Dieser Agent dient als ein Bindeglied zwischen der Überwachung und dem eigentlichen Dienst oder Attribut auf dem Windows-Rechner. Ohne diesen Agenten wäre Nagios nicht in der Lage, private Dienste oder Attribute auf dem Window-Rechner zu überwachen.

Für dieses Beispiel installieren wir das NSClient++-Addon auf dem Windows-Rechner und werden das *check_nt*-Plugin zur Kommunikation mit dem NSClient++-Addon benutzen. Das *check_nt*-Plugin sollte bereits auf dem Nagios-Server installiert sein, wenn Sie der Schnellstartanleitung gefolgt sind.

Andere Windows-Agenten (wie NC_Net) können statt NSClient++ genutzt werden, wenn Sie möchten - vorausgesetzt, Sie passen die Befehls- und Service-Definitionen usw. entsprechend an. Aus Gründen der Einfachheit werde ich nur das NSClient++-Addon in diesen Anweisungen berücksichtigen.

Schritte

Es gibt einige Schritte, die Sie durchführen müssen, um einen neuen Windows-Rechner zu überwachen. Das sind:

1. erfüllen Sie einmalige Voraussetzungen
2. installieren Sie einen Überwachungsagenten auf dem Windows-Rechner
3. erstellen Sie neue Host- und Service-Definitione zur Überwachung des Windows-Rechners
4. starten Sie den Nagios-Daemon neu

Was bereits für Sie vorbereitet wurde

Um Ihnen das Leben ein wenig zu erleichtern, wurden bereits ein paar Konfigurationsaufgaben für Sie erledigt:

- Eine *check_nt*-Befehlsdefinition ist in der *commands.cfg*-Datei vorhanden. Das erlaubt Ihnen die Nutzung des *check_nt*-Plugins zur Überwachung von Windows-Diensten.
- Eine Host-Vorlage für Windows-Server (namens *windows-server*) wurde bereits in der *templates.cfg*-Datei erstellt. Das erlaubt es Ihnen, Windows-Host-Definitionen auf einfache Weise hinzuzufügen.

Die o.g. Konfigurationsdateien finden Sie im */usr/local/nagios/etc/objects/*-Verzeichnis. Sie können diese und andere Definitionen anpassen, damit Sie Ihren Anforderungen besser entsprechen. Allerdings empfehle ich Ihnen, noch ein wenig damit zu warten, bis Sie besser mit der Konfiguration von Nagios vertraut sind. Für den Moment folgen Sie einfach den nachfolgenden Anweisungen und Sie werden im Nu Ihre Windows-Rechner überwachen.

Voraussetzungen

Wenn Sie Nagios das erste Mal konfigurieren, um einen Windows-Rechner zu überwachen, dann müssen Sie ein paar zusätzliche Dinge tun. Denken Sie daran, dass Sie dies nur für den *ersten* Windows-Rechner machen müssen, den Sie überwachen wollen.

Editieren Sie die Hauptkonfigurationsdatei.

```
vi /usr/local/nagios/etc/nagios.cfg
```

Entfernen Sie das führende Hash-(#)-Zeichen der folgenden Zeile in der Hauptkonfigurationsdatei:

```
#cfg_file=/usr/local/nagios/etc/objects/windows.cfg
```

Speichern Sie die Datei und verlassen den Editor.

Was haben Sie gerade getan? Sie haben Nagios mitgeteilt, in der */usr/local/nagios/etc/objects/windows.cfg*-Datei nach weiteren Objektdefinitionen zu schauen. Dort werden Sie Host- und Service-Definitionen für Windows-Rechner einfügen. Diese Konfigurationsdatei enthält bereits einige Beispiel-Host-, Hostgroup- und Service-Definitionen. Für den *ersten* Windows-Rechner, den Sie überwachen, passen Sie einfach die Beispiel-Host- und Service-Definitionen an, statt neue zu erstellen.

Installation des Windows-Agenten

Bevor Sie mit der Überwachung von privaten Diensten und Attributen von Windows-Rechnern beginnen, müssen Sie einen Agenten auf diesen Rechnern installieren. Ich empfehle das NSClient++-Addon zu nutzen, das Sie unter http://sourceforge.net/projects/nscplus finden. Diese Anweisungen werden Sie durch eine Basisinstallation des NSClient++-Addons und die Nagios-Konfiguration für die Überwachung des Windows-Rechners führen.

1. Laden Sie die letzte stabile Version des NSClient++-Addons von
http://sourceforge.net/projects/nscplus

2. Entpacken Sie die NSClient++-Dateien in ein neues C:\NSClient++-Verzeichnis

3. Gehen Sie auf die Kommandozeile und wechseln Sie in das C:\NSClient++-Verzeichnis

4. Registrieren Sie den NSClient++-Dienst mit dem folgenden Befehl:

```
nsclient++ /install
```

5. Installieren Sie ein NSClient++-Icon im Infobereich (Systray) mit dem folgenden Befehl (Groß- und Kleinschreibung ist wichtig!)

```
nsclient++ SysTray
```

6. Öffnen Sie die Dienste-Applikation und stellen Sie sicher, dass der NSClient++-Dienst mit dem Desktop kommunizieren darf (Reiter "Anmelden", Häkchen bei "Datenaustausch zwischen Dienst und Desktop zulassen" gesetzt). Setzen Sie ggf. das Häkchen.

7. Editieren Sie die NSC.INI-Datei (im C:\NSClient++-Verzeichnis) und machen Sie folgende Änderungen:

- entfernen Sie die Kommentarzeichen (;) im [modules]-Abschnitt, außer für CheckWMI.dll und RemoteConfiguration.dll
- definieren Sie optional ein Passwort für Clients, indem Sie die 'password'-Option im [Settings]-Abschnitt setzen.
- entfernen Sie das Kommentarzeichen (;) vor der 'allowed_hosts'-Option im [Settings]-Abschnitt. Fügen Sie die IP-Adresse des Nagios-Servers ein oder lassen Sie diese Angabe leer, so dass sich alle Hosts verbinden können.
- entfernen Sie ggf. das Kommentarzeichen vor der 'port'-Option im [NSClient]-Abschnitt und setzen Sie den Wert auf '12489' (Standard).

8. Starten Sie den NSClient++-Dienst mit dem folgenden Befehl:

```
nsclient++ /start
```

9. Wenn es sauber installiert ist, sollte ein neues Icon in Ihrem Infobereich auftauchen, ein gelber Kreis mit einem schwarzen 'M'.

10. Geschafft! Der Windows-Rechner kann nun der Nagios-Überwachungskonfiguration hinzugefügt werden...

Nagios konfigurieren

Nun ist es Zeit, einige Objektdefinitionen in Ihren Nagios-Konfigurationsdateien anzulegen, um den neuen Windows-Rechner zu überwachen.

Editieren Sie die *windows.cfg*-Datei.

```
vi /usr/local/nagios/etc/objects/windows.cfg
```

Fügen Sie eine neue Host-Definition für den Windows-Rechner hinzu, den Sie überwachen möchten. Wenn dies der *erste* Windows-Rechner ist, den Sie überwachen, dann können Sie einfach die Beispiel-Definitionen in der *windows.cfg*-Datei anpassen. Ändern Sie die *host_name*-, *alias*- und *address*-Felder auf die entsprechenden Werte des Windows-Rechners.

```
define host{
        use             windows-server  ; Standard-Werte von einer Windows-Server-Vorlage erben (diese Zeile nicht löschen!)
        host_name       winserver
        alias           My Windows Server
        address         192.168.1.2
        }
```

Gut. Nun können Sie (in der gleichen Konfigurationsdatei) einige Service-Definitionen hinzufügen, um Nagios mitzuteilen, welche Dinge auf dem Windows-Server zu überwachen sind. Wenn dies der *erste* Windows-Rechner ist, den Sie überwachen, dann können Sie einfach die Beispiel-Definitionen in der *windows.cfg*-Datei anpassen.

Anmerkung: Ersetzen Sie "*winserver*" in den folgenden Beispiel-Definitionen durch den Namen, den Sie in der *host_name*-Direktive der Host-Definitionen angegeben haben, die Sie gerade hinzugefügt haben.

Fügen Sie die folgende Service-Definition hinzu, um die Version des NSClient++-Addons zu überwachen, das auf dem Windows-Rechner läuft. Dies ist nützlich, wenn Sie Ihre Windows-Server mit einer neueren Version des Addons aktualisieren möchten, weil Sie sehen können, welche Windows-Rechner noch auf die neueste Version des NSClient++-Addon aktualisiert werden muss.

```
define service{
        use                     generic-service
        host_name               winserver
        service_description     NSClient++ Version
        check_command           check_nt!CLIENTVERSION
        }
```

Fügen Sie die folgende Service-Definition hinzu, um die Laufzeit des Windows-Servers zu überwachen.

```
define service{
        use                     generic-service
        host_name               winserver
        service_description     Uptime
        check_command           check_nt!UPTIME
        }
```

Fügen Sie die folgende Service-Definition hinzu, um die CPU-Belastung des Windows-Servers zu überwachen und einen CRITICAL-Alarm zu erzeugen, wenn die 5-Minuten-Belastung mindestens 90% beträgt oder einen WARNING-Alarm, wenn die 5-Minuten-Belastung mindestens 80% beträgt.

```
define service{
        use                     generic-service
        host_name               winserver
        service_description     CPU Load
        check_command           check_nt!CPULOAD!-l 5,80,90
        }
```

Fügen Sie die folgende Service-Definition hinzu, um die Speicherbelegung des Windows-Servers zu überwachen und einen CRITICAL-Alarm zu erzeugen, wenn die Belegung mindestens 90% beträgt oder einen WARNING-Alarm, wenn die Belegung mindestens 80% beträgt.

```
define service{
        use                     generic-service
        host_name               winserver
        service_description     Memory Usage
        check_command           check_nt!MEMUSE!-w 80 -c 90
        }
```

Fügen Sie die folgende Service-Definition hinzu, um die Plattenbelegung von Laufwerk C: des Windows-Servers zu überwachen und einen CRITICAL-Alarm zu erzeugen, wenn die Belegung mindestens 90% beträgt oder einen WARNING-Alarm, wenn die Belegung mindestens 80% beträgt.

```
define service{
        use                     generic-service
        host_name               winserver
        service_description     C:\ Drive Space
        check_command           check_nt!USEDDISKSPACE!-l c -w 80 -c 90
        }
```

Fügen Sie die folgende Service-Definition hinzu, um den W3SVC-Dienst des Windows-Servers zu überwachen und einen CRITICAL-Alarm zu erzeugen, wenn der Dienst gestoppt ist.

```
define service{
        use                     generic-service
        host_name               winserver
        service_description     W3SVC
        check_command           check_nt!SERVICESTATE!-d SHOWALL -l W3SVC
        }
```

Fügen Sie die folgende Service-Definition hinzu, um den Explorer.exe-Prozess des Windows-Servers zu überwachen und einen CRITICAL-Alarm zu erzeugen, wenn der Prozess nicht läuft.

```
define service{
        use                     generic-service
        host_name               winserver
        service_description     Explorer
        check_command           check_nt!PROCSTATE!-d SHOWALL -l Explorer.exe
        }
```

Das war es vorerst. Sie haben einige grundlegende Dienste hinzugefügt, die auf dem Windows-Rechner überwacht werden sollen. Speichern Sie die Konfigurationsdatei.

Passwortschutz

Wenn Sie ein Passwort in der NSClient++-Konfigurationsdatei auf dem Windows-Rechner angegeben haben, dann müssen Sie die *check_nt*-Befehlsdefinition anpassen, damit sie das Passwort enthält. Öffnen Sie die *commands.cfg*-Datei.

```
vi /usr/local/nagios/etc/objects/commands.cfg
```

Ändern Sie die Definition des *check_nt*-Befehls, damit sie das "-s <PASSWORD>"-Argument enthält (wobei PASSWORD das Passwort ist, das Sie auf dem Windows-Rechner angegeben haben):

```
define command{
        command_name    check_nt
        command_line    $USER1$/check_nt -H $HOSTADDRESS$ -p 12489 -s PASSWORD -v $ARG1$ $ARG2$
        }
```

Speichern Sie die Datei

Nagios neu starten

Sie sind fertig mit der Anpassung der Nagios-Konfiguration, so dass Sie nun die Konfigurationsdateien überprüfen und Nagios neu starten müssen.

Wenn die Überprüfung irgendwelche Fehler enthält, dann müssen Sie diese beheben, bevor Sie fortfahren. Stellen Sie sicher, dass Sie Nagios nicht (erneut) starten, bevor die Überprüfung ohne Fehler durchgelaufen ist!

Nagios®

Linux/Unix-Rechner überwachen

Hoch zu: Inhalt

Siehe auch: Schnellstart-Installationsanleitung, öffentlich zugängliche Dienste überwachen

Einführung

Dieses Dokument beschreibt, wie Sie "private" Dienste und Attribute auf Linux/UNIX-Servern überwachen, wie z.B.:

- CPU-Auslastung
- Speichernutzung
- Plattenbelegung
- angemeldete Benutzer
- laufende Prozesse
- etc.

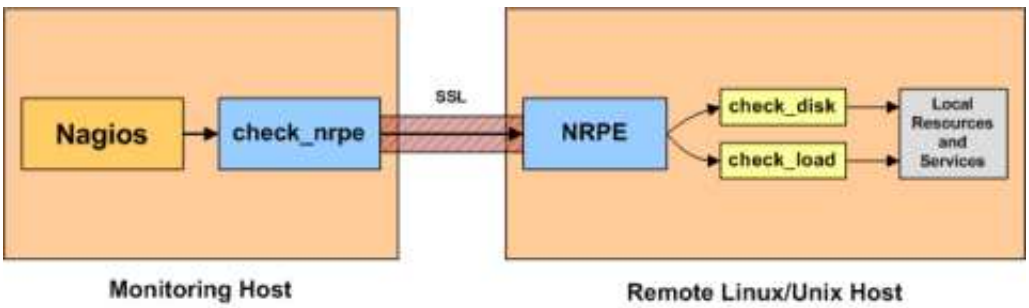
Öffentlich nutzbare Dienste, die von Linux-Servern zur Verfügung gestellt werden (HTTP, FTP, SSH, SMTP, etc.), können einfach mit Hilfe der Dokumentation öffentlich zugängliche Dienste überwachen kontrolliert werden.

Anmerkung: Diese Anweisungen gehen davon aus, dass Sie Nagios anhand der Schnellstartanleitung installiert haben. Die nachfolgenden Beispiel-Konfigurationseinträge beziehen sich auf Objekte, die in den Beispiel-Konfigurationsdateien (*commands.cfg*, *templates.cfg*, etc.) definiert sind. Diese Dateien werden installiert, wenn Sie der Schnellstartanleitung folgen.

Überblick

[Anmerkung: Dieses Dokument ist noch nicht vollständig. Ich würde empfehlen, die Dokumentation zum NRPE-Addon zu lesen, um zu sehen, wie ein entfernter Linux/Unix-Server zu überwachen ist.]

Es gibt verschiedene Wege, Attribute oder entfernte Linux/Unix-Server zu überwachen. Einer benutzt gemeinsame SSH-Schlüssel und das *check_by_ssh*-Plugin auf entfernten Servern. Diese Methode wird hier nicht behandelt, kann aber zu hoher Last auf Ihrem Überwachungs-Server führen, wenn Sie hunderte oder tausende von Services überwachen. Der Overhead durch das Auf- und Abbauen von SSH-Verbindungen ist der Grund dafür.



Eine andere gebräuchliche Methode der Überwachung von entfernten Linux/Unix-Hosts ist die Nutzung des NRPE-Addons. NRPE erlaubt Ihnen, Plugins auf entfernten Linux/Unix-Hosts auszuführen. Das ist nützlich, wenn Sie lokale Ressourcen/Attribute wie z.B. Plattenbelegung, CPU-Auslastung, Speichernutzung auf einem entfernten Host überwachen wollen.

Netware-Server überwachen

Hoch zu: Inhalt

Siehe auch: Schnellstart-Installationsanleitung, öffentlich zugängliche Dienste überwachen

Einführung

Dieses Dokument enthält Informationen, wie Sie Novell-Netware-Servern überwachen können.

Externe Ressourcen

Sie finden Informationen zur Überwachung von Netware-Servern mit Nagios auf der Novell-Cool Solutions-Website, darunter:

- MRTGEXT: NLM module for MRTG and Nagios
- Nagios: Host and Service Monitoring Tool
- Nagios and NetWare: SNMP-based Monitoring
- Monitor DirXML/IDM Driver States with Nagios
- Check NDS Login ability with Nagios
- NDPS/iPrint Print Queue Monitoring by Nagios
- check_gwiaRL Plugin for Nagios 2.0

Hinweis: Wenn Sie Novells Cool Solutions-Site besuchen, suchen Sie nach "Nagios", um mehr Artikel und Software-Komponenten zu finden, die sich auf Überwachung beziehen.

Dank an Christian Mies, Rainer Brunold und andere, die auf der Novell-Site Nagios- und Netware-Dokumentation, Addons usw. beigetragen haben!

Nagios®

Netzwerk-Drucker überwachen

Hoch zu: Inhalt

Siehe auch: Öffentlich zugängliche Dienste überwachen

Einführung

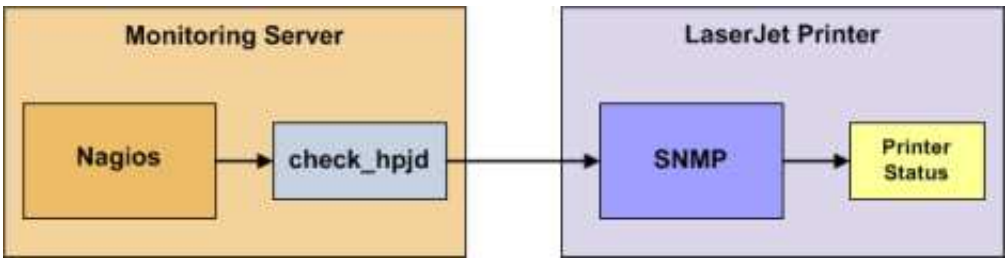
Dieses Dokument beschreibt, wie Sie den Status von Netzwerkdruckern überwachen können. HP-Drucker haben interne/externe JetDirect-Karten/Devices, andere Print-Server (wie der Troy PocketPro 100S oder der Netgear PS101) unterstützen das JetDirect-Protokoll.

Das *check_hpjd*-Plugin (das Bestandteil der Nagios-Distribution ist), erlaubt Ihnen die Überwachung des Zustands von JetDirect-fähigen Druckern, auf denen SNMP aktiviert ist. Das Plugin kann die folgenden Druckerzustände erkennen:

- Papierstau
- Kein Papier mehr
- Drucker Offline
- Benutzereingriff erforderlich
- Tonerstand niedrig
- Speicher unzureichend
- Klappe offen
- Ausgabefach voll
- und weitere...

Anmerkung: Diese Anweisungen gehen davon aus, dass Sie Nagios anhand der Schnellstartanleitung installiert haben. Die nachfolgenden Beispiel-Konfigurationseinträge beziehen sich auf Objekte, die in den Beispiel-Konfigurationsdateien (*commands.cfg*, *templates.cfg*, etc.) definiert sind. Diese Dateien werden installiert, wenn Sie der Schnellstartanleitung folgen.

Überblick



Die Überwachung des Zustands eines Netzwerkdruckers ist ziemlich einfach. Bei JetDirect-fähigen Druckern ist normalerweise SNMP aktiviert, so dass Nagios ihren Zustand mit Hilfe des *check_hpjd*-Plugins überwachen kann.

Das *check_hpjd*-Plugin wird nur dann kompiliert und installiert, wenn Sie die net-snmp- und net-snmp-utils-Pakete auf Ihrem System haben. Stellen Sie sicher, dass das Plugin im */usr/local/nagios/libexec*-Verzeichnis existiert, bevor Sie fortfahren. Falls nicht, installieren Sie net-snmp und net-snmp-utils und kompilieren und installieren Sie die Nagios-Plugins erneut.

Schritte

Es gibt einige Schritte, die Sie durchführen müssen, um einen neuen Netzwerkdrucker zu überwachen. Das sind:

1. erfüllen Sie einmalige Voraussetzungen
2. erstellen Sie neue Host- und Service-Definitionen zur Überwachung des Druckers
3. starten Sie den Nagios-Daemon neu

Was bereits für Sie vorbereitet wurde

Um Ihnen das Leben ein wenig zu erleichtern, wurden bereits ein paar Konfigurationsaufgaben für Sie erledigt:

- Eine *check_hpjd*-Befehlsdefinition ist in der *commands.cfg*-Datei vorhanden. Das erlaubt Ihnen die Nutzung des *check_hpjd*-Plugins zur Überwachung von Netzwerkdruckern.
- Eine Host-Vorlage für Drucker (namens *generic-printer*) wurde bereits in der *templates.cfg*-Datei erstellt. Das erlaubt es Ihnen, Drucker-Host-Definitionen auf einfache Weise hinzuzufügen.

Die o.g. Konfigurationsdateien finden Sie im */usr/local/nagios/etc/objects/*-Verzeichnis. Sie können diese und andere Definitionen anpassen, damit Sie Ihren Anforderungen besser entsprechen. Allerdings empfehle ich Ihnen, noch ein wenig damit zu warten, bis Sie besser mit der Konfiguration von Nagios vertraut sind. Für den Moment folgen Sie einfach den nachfolgenden Anweisungen und Sie werden im Nu Ihre Netzwerkdrucker überwachen.

Voraussetzungen

Wenn Sie Nagios das erste Mal konfigurieren, um einen Netzwerkdrucker zu überwachen, dann müssen Sie ein paar zusätzliche Dinge tun. Denken Sie daran, dass Sie dies nur für den *ersten* Netzwerkdrucker machen müssen, den Sie überwachen wollen.

Editieren Sie die Hauptkonfigurationsdatei.

```
vi /usr/local/nagios/etc/nagios.cfg
```

Entfernen Sie das führende Hash-(#)-Zeichen der folgenden Zeile in der Hauptkonfigurationsdatei:

```
#cfg_file=/usr/local/nagios/etc/objects/printer.cfg
```

Speichern Sie die Datei und verlassen den Editor.

Was haben Sie gerade getan? Sie haben Nagios mitgeteilt, in der */usr/local/nagios/etc/objects/printer.cfg*-Datei nach weiteren Objektdefinitionen zu schauen. Dort werden Sie Drucker-Host- und Service-Definitionen einfügen. Diese Konfigurationsdatei enthält bereits einige Beispiel-Host-, Hostgroup- und Service-Definitionen. Für den *ersten* Netzwerkdrucker, den Sie überwachen, passen Sie einfach die Beispiel-Host- und Service-Definitionen an, statt neue zu erstellen.

Nagios konfigurieren

Sie müssen einige Objektdefinitionen anlegen, um einen neuen Drucker zu überwachen.

Öffnen Sie die *printer.cfg*-Datei.

```
vi /usr/local/nagios/etc/objects/printer.cfg
```

Fügen Sie eine neue Host-Definition für den Netzwerkdrucker hinzu, den Sie überwachen möchten. Wenn dies der *erste* Netzwerkdrucker ist, den Sie überwachen, dann können Sie einfach die Beispiel-Definitionen in der *printer.cfg*-Datei anpassen. Ändern Sie die *host_name*-, *alias*- und *address*-Felder auf die entsprechenden Werte des Netzwerkdruckers.

```
define host{
        use             generic-printer         ; Inherit default values from a template
        host_name       hplj2605dn              ; The name we're giving to this printer
        alias           HP LaserJet 2605dn      ; A longer name associated with the printer
        address         192.168.1.30            ; IP address of the printer
        hostgroups      allhosts                ; Host groups this printer is associated with
        }
```

Nun können Sie (in der gleichen Konfigurationsdatei) einige Service-Definitionen hinzufügen, um Nagios mitzuteilen, welche Dinge auf dem Drucker zu überwachen sind. Wenn dies der *erste* Drucker ist, den Sie überwachen, dann können Sie einfach die Beispiel-Definitionen in der *printer.cfg*-Datei anpassen.

 Anmerkung: Ersetzen Sie "*hplj2605dn*" in der folgenden Beispiel-Definition durch den Namen, den Sie in der *host_name*-Direktive der Host-Definition angegeben haben, die Sie gerade hinzugefügt haben.

Fügen Sie die folgende Service-Definition hinzu, um den Zustand des Druckers zu prüfen. Der Service benutzt das *check_hpjd*-Plugin, um den Drucker alle zehn Minuten zu prüfen. Der Wert für die SNMP-Community lautet in diesem Beispiel "public".

```
define service{
        use                     generic-service         ; Inherit values from a template
        host_name               hplj2605dn              ; The name of the host the service is associated with
        service_description     Printer Status          ; The service description
        check_command           check_hpjd!-C public    ; The command used to monitor the service
        normal_check_interval   10                      ; Check the service every 10 minutes under normal conditions
        retry_check_interval    1                       ; Re-check the service every minute until its final/hard state is determined
        }
```

Fügen Sie die folgende Service-Definition hinzu, um alle zehn Minuten einen Ping an den Drucker zu senden. Das ist nützlich, um die generelle Netzwerkverbindung und Werte für RTA und Paketverlust zu überwachen.

```
define service{
        use                     generic-service
        host_name               hplj2605dn
        service_description     PING
        check_command           check_ping!3000.0,80%!5000.0,100%
        normal_check_interval   10
        retry_check_interval    1
        }
```

Speichern Sie die Datei.

Nagios neu starten

Sobald Sie die neuen Host- und Service-Definitionen in der *printer.cfg*-Datei hinzugefügt haben, sind Sie bereit, mit der Überwachung des Druckers zu beginnen. Um dies zu tun, müssen Sie die Konfigurationsdateien überprüfen und Nagios neu starten.

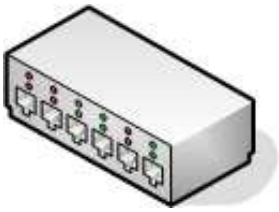
Wenn die Überprüfung irgendwelche Fehler enthält, dann müssen Sie diese beheben, bevor Sie fortfahren. Stellen Sie sicher, dass Sie Nagios nicht (erneut) starten, bevor die Überprüfung ohne Fehler durchgelaufen ist!

Router und Switches überwachen

Hoch zu: Inhalt

Siehe auch: Öffentlich zugängliche Dienste überwachen

Einführung



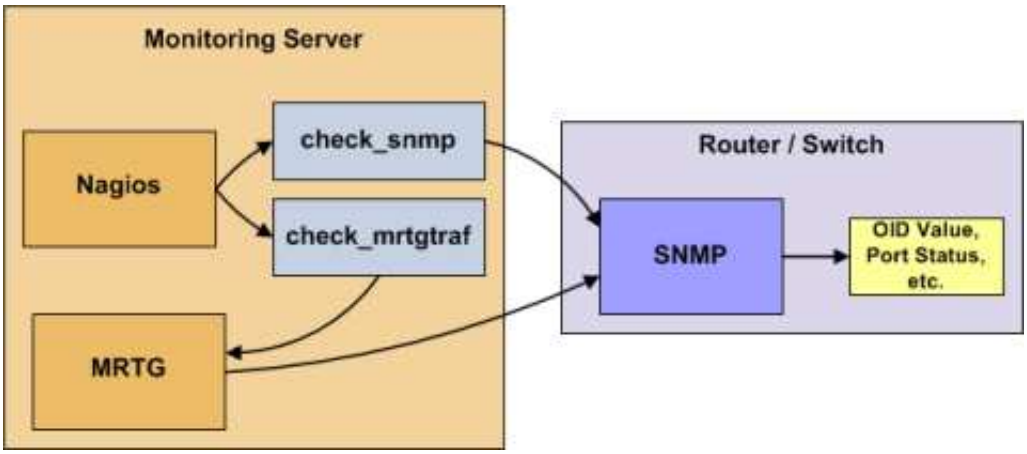
Dieses Dokument beschreibt, wie Sie den Zustand von Netzwerk-Switches und Routern überwachen können. Einige preiswerte "unmanaged" Switches und Router haben keine IP-Adresse und sind in Ihrem Netzwerk nicht sichtbar, so dass es keinen Weg gibt, um sie zu überwachen. Teurere Switches und Router haben eigene Adressen und können durch Ping überwacht oder über SNMP nach Statusinformationen abgefragt werden.

Ich werde beschreiben, wie Sie die folgenden Dinge auf "managed" Switches, Hubs und Routern überwachen können:

- Paketverlust, durchschnittliche Umlaufzeiten (round trip average, RTA)
- SNMP-Statusinformationen
- Bandbreite / Übertragungsrate (traffic rate)

Anmerkung: Diese Anweisungen gehen davon aus, dass Sie Nagios anhand der Schnellstartanleitung installiert haben. Die nachfolgenden Beispiel-Konfigurationseinträge beziehen sich auf Objekte, die in den Beispiel-Konfigurationsdateien (*commands.cfg*, *templates.cfg*, etc.) definiert sind. Diese Dateien werden installiert, wenn Sie der Schnellstartanleitung folgen.

Überblick



Die Überwachung von Switches und Routern kann entweder einfach oder auch aufwändiger sein - abhängig davon, welches Equipment Sie haben und was Sie überwachen wollen. Da es sich um kritische Infrastrukturkomponenten handelt, werden Sie diese ohne Zweifel mindestens in grundlegender Art und Weise überwachen.

Switches und Router können einfach per "Ping" überwacht werden, um Paketverlust, RTA usw. zu ermitteln. Wenn Ihr Switch SNMP unterstützt, können Sie mit dem *check_snmp*-Plugin z.B. den Port-Status und (wenn Sie MRTG benutzen) mit dem *check_mrtgtraf*-Plugin die Bandbreite überwachen.

Das *check_snmp*-Plugin wird nur dann kompiliert und installiert, wenn Sie die net-snmp- und net-snmp-utils-Pakete auf Ihrem System haben. Stellen Sie sicher, dass das Plugin im */usr/local/nagios/libexec*-Verzeichnis existiert, bevor Sie fortfahren. Falls nicht, installieren Sie net-snmp und net-snmp-utils und kompilieren und installieren Sie die Nagios-Plugins erneut.

Schritte

Es gibt einige Schritte, die Sie durchführen müssen, um einen neuen Router oder Switch zu überwachen. Das sind:

1. erfüllen Sie einmalige Voraussetzungen
2. erstellen Sie neue Host- und Service-Definitionen zur Überwachung des Geräts
3. starten Sie den Nagios-Daemon neu

Was bereits für Sie vorbereitet wurde

Um Ihnen das Leben ein wenig zu erleichtern, wurden bereits ein paar Konfigurationsaufgaben für Sie erledigt:

- Zwei Befehlsdefinitionen (*check_snmp* und *check_local_mrtgtraf*) sind bereits in der *commands.cfg*-Datei vorhanden. Das erlaubt Ihnen die Nutzung des *check_snmp*- bzw. *check_mrtgtraf*-Plugins zur Überwachung von Routern und Switches.
- Eine Host-Vorlage für Switches (namens *generic-switch*) wurde bereits in der *templates.cfg*-Datei erstellt. Das erlaubt es Ihnen, Router/Switch-Host-Definitionen auf einfache Weise hinzuzufügen.

Die o.g. Konfigurationsdateien finden Sie im */usr/local/nagios/etc/objects/*-Verzeichnis. Sie können diese und andere Definitionen anpassen, damit Sie Ihren Anforderungen besser entsprechen. Allerdings empfehle ich Ihnen, noch ein wenig damit zu warten, bis Sie besser mit der Konfiguration von Nagios vertraut sind. Für den Moment folgen Sie einfach den nachfolgenden Anweisungen und Sie werden im Nu Ihre Router/Switches überwachen.

Voraussetzungen

Wenn Sie Nagios das erste Mal konfigurieren, um einen Netzwerk-Switch zu überwachen, dann müssen Sie ein paar zusätzliche Dinge tun. Denken Sie daran, dass Sie dies nur für den *ersten* Switch machen müssen, den Sie überwachen wollen.

Editieren Sie die Hauptkonfigurationsdatei.

```
vi /usr/local/nagios/etc/nagios.cfg
```

Entfernen Sie das führende Hash-(#)-Zeichen der folgenden Zeile in der Hauptkonfigurationsdatei:

```
#cfg_file=/usr/local/nagios/etc/objects/switch.cfg
```

Speichern Sie die Datei und verlassen den Editor.

Was haben Sie gerade getan? Sie haben Nagios mitgeteilt, in der */usr/local/nagios/etc/objects/switch.cfg*-Datei nach weiteren Objektdefinitionen zu schauen. Dort werden Sie Host- und Service-Definitionen für Router- und Switches einfügen. Diese Konfigurationsdatei enthält bereits einige Beispiel-Host-, Hostgroup- und Service-Definitionen. Für den *ersten* Router/Switch, den Sie überwachen, passen Sie einfach die Beispiel-Host- und Service-Definitionen an, statt neue zu erstellen.

Nagios konfigurieren

Sie müssen einige Objektdefinitionen anlegen, um einen neuen Router/Switch zu überwachen.

Öffnen Sie die *switch.cfg*-Datei.

```
vi /usr/local/nagios/etc/objects/switch.cfg
```

Fügen Sie eine neue Host-Definition für den Switch hinzu, den Sie überwachen möchten. Wenn dies der *erste* Switch ist, den Sie überwachen, dann können Sie einfach die Beispiel-Definitionen in der *switch.cfg*-Datei anpassen. Ändern Sie die *host_name*-, *alias*- und *address*-Felder auf die entsprechenden Werte des Switches.

```
define host{
        use             generic-switch          ; Inherit default values from a template
        host_name       linksys-srw224p         ; The name we're giving to this switch
        alias           Linksys SRW224P Switch  ; A longer name associated with the switch
        address         192.168.1.253           ; IP address of the switch
        hostgroups      allhosts,switches       ; Host groups this switch is associated with
        }
```

Services überwachen

Nun können Sie einige Service-Definitionen hinzufügen (in der gleichen Konfigurationsdatei), um Nagios mitzuteilen, welche Dinge auf dem Switch zu überwachen sind. Wenn dies der *erste* Switch ist, den Sie überwachen, dann können Sie einfach die Beispiel-Definitionen in der *switch.cfg*-Datei anpassen.

Anmerkung: Ersetzen Sie "*linksys-srw224p*" in der folgenden Beispiel-Definition durch den Namen, den Sie in der *host_name*-Direktive der Host-Definition angegeben haben, die Sie gerade hinzugefügt haben.

Paketverlust und RTA überwachen

Fügen Sie die folgende Service-Definition hinzu, um unter normalen Bedingungen alle fünf Minuten Paketverlust und Round-Trip-Average zwischen dem Nagios-Host und dem Switch zu überwachen.

```
define service{
        use                     generic-service ; Inherit values from a template
        host_name               linksys-srw224p ; The name of the host the service is associated with
        service_description     PING            ; The service description
        check_command           check_ping!200.0,20%!600.0,60%  ; The command used to monitor the service
        normal_check_interval   5               ; Check the service every 5 minutes under normal conditions
        retry_check_interval    1               ; Re-check the service every minute until its final/hard state is determined
        }
```

Dieser Service wird:

- CRITICAL, falls der Round-Trip-Average (RTA) größer als 600 Millisekunden oder der Paketverlust 60% oder mehr ist
- WARNING, falls der Round-Trip-Average (RTA) größer als 200 Millisekunden oder der

Paketverlust 20% oder mehr ist
- OK, falls der Round-Trip-Average (RTA) kleiner als 200 Millisekunden oder der Paketverlust kleiner als 20% ist

SNMP-Statusinformationen überwachen

Wenn Ihr Switch oder Router SNMP unterstützt, können Sie eine Menge an Informationen mit dem *check_snmp*-Plugin überwachen. Wenn nicht, dann überspringen Sie diesen Abschnitt.

Fügen Sie die folgende Service-Definition hinzu, um die Laufzeit des Switches zu überwachen.

```
define service{
        use                     generic-service ; Inherit values from a template
        host_name               linksys-srw224p
        service_description     Uptime
        check_command           check_snmp!-C public -o sysUpTime.0
        }
```

In der *check_command*-Direktive der obigen Service-Definition sagt "-C public", dass der zu benutzende SNMP-Community-Name "public" lautet und "-o sysUpTime.0" gibt an, welche OID überprüft werden soll.

Wenn Sie sicherstellen wollen, dass sich ein bestimmter Port/ein bestimmtes Interface des Switches in einem "UP"-Zustand befindet, dann sollten Sie eine Service-Definition hinzufügen:

```
define service{
        use                     generic-service ; Inherit values from a template
        host_name               linksys-srw224p
        service_description     Port 1 Link Status
        check_command           check_snmp!-C public -o ifOperStatus.1 -r 1 -m RFC1213-MIB
        }
```

In dem obigen Beispiel bezieht sich "-o ifOperStatus.1" auf die OID des Betriebszustands von Port 1 des Switches. Die "-r 1"-Option teilt dem *check_snmp*-Plugin mit, einen OK-Zustand zurückzuliefern, wenn "1" im SNMP-Ergebnis gefunden wird (1 deutet einen "UP"-Zustand des Ports an) und CRITICAL, wenn es nicht gefunden wird. "-m RFC1213-MIB" ist optional und teilt dem *check_snmp*-Plugin mit, nur die "RFC1213-MIB" zu laden statt jeder einzelnen MIB, die auf Ihrem System installiert ist, was die Dinge beschleunigen kann.

Das war's mit dem SNMP-Überwachungsbeispiel. Es gibt eine Million Dinge, die mit SNMP überwacht werden können, also liegt es an Ihnen zu entscheiden, was Sie brauchen und was Sie überwachen wollen. Viel Erfolg!

 Hinweis: Normalerweise können Sie mit dem folgenden Befehl die OIDs eines Switches (oder eines anderen SNMP-fähigen Gerätes) herausfinden, die überwacht werden können (ersetzen Sie *192.168.1.253* durch die IP-Adresse des Switches): *snmpwalk -v1 -c public 192.168.1.253 -m ALL .1*

Bandbreite / Übertragungsrate überwachen

Wenn Sie die Bandbreitennutzung Ihres Switches oder Routers mit MRTG überwachen, dann können Sie durch Nagios alarmiert werden, wenn die Übertragungsraten Schwellwerte überschreiten, die Sie angeben. Mit dem *check_mrtgtraf*-Plugin (das in der Nagios-Plugin-Distribution enthalten ist) können Sie das tun.

Sie müssen dem *check_mrtgtraf*-Plugin mitteilen, in welcher Log-Datei die MRTG-Daten gespeichert sind, zusammen mit Schwellwerten, usw. In meinem Beispiel überwache ich einen Port eines Linksys-Switches. Die MRTG-Log-Datei ist abgelegt unter */var/lib/mrtg/192.168.1.253_1.log*. Hier ist die Service-Definition, die ich benutze, um die Bandbreitendaten zu überwachen, die in der Log-Datei gespeichert sind...

```
define service{
        use                     generic-service ; Inherit values from a template
        host_name               linksys-srw224p
        service_description     Port 1 Bandwidth Usage
        check_command           check_local_mrtgtraf!/var/lib/mrtg/192.168.1.253_1.log!AVG!1000000,2000000!5000000,5000000!10
        }
```

In dem obigen Beispiel teilt "/var/lib/mrtg/192.168.1.253_1.log" im *check_local_mrtgtraf*-Befehl dem Plugin mit, welche MRTG-Log-Datei auszulesen ist. Die "AVG"-Option gibt an, dass Durchschnitts-Bandbreitenstatistiken verwendet werden sollen. "1000000,2000000" sind die Schwellwerte (in Bytes) für Warnungen bei eingehenden Übertragungsraten. "5000000,5000000" sind die kritischen Schwellwerte (in Bytes) bei ausgehenden Übertragungsraten. "10" gibt an, dass das Plugin einen CRITICAL-Zustand zurückliefern soll, wenn die MRTG-Log-Datei älter als zehn Minuten ist (sie sollte alle fünf Minuten aktualisiert werden).

Speichern Sie die Datei.

Nagios neu starten

Sobald Sie die neuen Host- und Service-Definitionen in der *switch.cfg*-Datei hinzugefügt haben, sind Sie bereit, mit der Überwachung des Routers/Switches zu beginnen. Um dies zu tun, müssen Sie die Konfigurationsdateien überprüfen und Nagios neu starten.

Wenn die Überprüfung irgendwelche Fehler enthält, dann müssen Sie diese beheben, bevor Sie fortfahren. Stellen Sie sicher, dass Sie Nagios nicht (erneut) starten, bevor die Überprüfung ohne Fehler durchgelaufen ist!

Nagios®

Öffentlich zugängliche Dienste überwachen

Hoch zu: Inhalt

Siehe auch: Schnellstart-Installationsanleitung

Einführung

Dieses Dokument beschreibt, wie Sie öffentlich zugängliche Dienste, Applikationen und Protokolle überwachen können. Mit "öffentlich" meine ich Dienste, die über das Netzwerk zugänglich sind - entweder das lokale Netzwerk oder das größere Internet. Beispiele von öffentlichen Diensten umfassen u.a. HTTP, POP3, IMAP, FTP und SSH. Es gibt viele weitere öffentliche Dienste, die Sie wahrscheinlich jeden Tag benutzen. Diese Dienste und Applikationen, genau wie ihre zu Grunde liegenden Protokolle, können normalerweise mit Nagios ohne spezielle Zugangsvoraussetzungen überwacht werden.

Private Dienste können im Gegensatz dazu nicht ohne einen dazwischen geschalteten Agenten überwacht werden. Beispiele von mit Hosts verbundenen privaten Diensten sind Dinge wie CPU-Auslastung, Speicherbelegung, Plattenbelegung, angemeldete Benutzer, Prozessinformationen usw. Diese privaten Dienste oder Attribute von Hosts werden normalerweise nicht an externe Clients offengelegt. Diese Situation erfordert, dass ein zwischengeschalteter Überwachungsagent auf jedem Host installiert wird, den Sie überwachen müssen. Mehr Informationen zur Überwachung von privaten Diensten auf verschiedenen Arten von Hosts finden Sie in der Dokumentation zu:

- Windows-Rechner überwachen
- Netware-Server überwachen
- Linux/Unix-Rechner überwachen

Hinweis: Gelegentlich werden Sie feststellen, dass Informationen zu privaten Diensten und Applikationen mit SNMP überwacht werden können. Der SNMP-Agent erlaubt Ihnen, entfernt liegende anderenfalls private (und unzugängliche) Informationen des Hosts zu überwachen. Mehr Informationen zur Überwachung von Diensten mit SNMP finden Sie in der Dokumentation zur Überwachung von Switches und Routern.

Anmerkung: Diese Anweisungen gehen davon aus, dass Sie Nagios anhand der Schnellstartanleitung installiert haben. Die nachfolgenden Beispiel-Konfigurationseinträge beziehen sich auf Objekte, die in den Beispiel-Konfigurationsdateien (*commands.cfg* und *localhost.cfg*) definiert sind. Diese Dateien werden installiert, wenn Sie der Schnellstartanleitung folgen.

Plugins zur Überwachung von Services

Wenn Sie feststellen, dass Sie eine bestimmte Applikation, einen Service oder ein Protokoll überwachen müssen, dann stehen die Chancen gut, dass bereits ein Plugin existiert. Die offizielle Nagios-Plugin-Distribution enthält Plugins, mit denen eine Reihe von Services und Protokollen überwacht werden können. Es gibt auch eine große Zahl von Plugins, die andere Leute beigetragen haben, die Sie im *contrib/*-Unterverzeichnis der Plugin-Distribution finden. Die NagiosExchange.org-Website stellt eine Reihe von zusätzlichen Plugins bereit, die andere Benutzer geschrieben haben, also schauen Sie vorbei, wenn Sie Zeit finden.

Wenn Sie zufällig kein entsprechendes Plugin für das finden, was Sie überwachen möchten, dann können Sie immer Ihr eigenes schreiben. Plugins sind einfach zu schreiben, also lassen Sie sich nicht von diesem Gedanken abschrecken. Lesen Sie dazu die Dokumentation über die Entwicklung von Plugins.

Ich werde Sie durch die Überwachung von einigen grundlegenden Diensten führen, die Sie vielleicht früher oder später brauchen. Jeder dieser Services kann mit einem der Plugins überwacht werden, die als Teil der Nagios-Plugin-Distribution installiert werden. Lassen Sie uns beginnen...

erstellen einer Host-Definition

Bevor Sie einen Service überwachen können, müssen Sie einen Host definieren, der mit dem Service verbunden ist. Sie können Host-Definitionen in jeder Objektkonfigurationsdatei platzieren, die mit einer cfg_file-Direktive definiert ist oder in einem Verzeichnis, das in einer cfg_dir-Direktive angegeben ist. Wenn Sie bereits eine Host-Definition angelegt haben, dann können Sie diesen Schritt überspringen.

Lassen Sie uns für dieses Beispiel annehmen, dass Sie eine Reihe von Services auf einem entfernten Host überwachen wollen. Lassen Sie uns diesen Host *remotehost* nennen. Die Host-Definition kann in einer eigenen Datei ablegt oder zu einer bereits existierenden Objektkonfigurationsdatei hinzugefügt werden. Hier nun, wie die Host-Definition für *remotehost* aussehen könnte:

```
define host{
        use             generic-host            ; Inherit default values from a template
        host_name       remotehost              ; The name we're giving to this host
        alias           Some Remote Host        ; A longer name associated with the host
        address         192.168.1.50            ; IP address of the host
        hostgroups      allhosts                ; Host groups this host is associated with
        }
```

Nachdem für den Host eine Definition hinzugefügt wurde, können wir mit der Definition von zu überwachenden Services beginnen. Genau wie Host-Definitionen können auch Service-Definitionen in jeder Objektkonfigurationdatei abgelegt werden.

erstellen von Service-Definitionen

Für jeden Service, den Sie überwachen wollen, müssen Sie in Nagios einen Service definieren, der mit der Host-Definition verbunden ist, die Sie gerade angelegt haben. Sie können Host-Definitionen in jeder Objektkonfigurationsdatei platzieren, die mit einer cfg_file-Direktive definiert ist oder in einem Verzeichnis, das in einer cfg_dir-Direktive angegeben ist.

Einige Beispiel-Service-Definitionen zur Überwachung von gebräuchlichen Services (HTTP, FTP, usw.) finden Sie nachfolgend.

HTTP überwachen

Wahrscheinlich werden Sie zu irgendeinem Zeitpunkt Web-Server überwachen wollen - entweder Ihre eigenen oder die von anderen. Das *check_http*-Plugin macht genau das. Es versteht HTTP und kann Antwortzeiten, Fehler-Codes, Zeichenketten im zurückgelieferten HTML, Server-Zertifikate und vieles mehr überwachen.

Die *commands.cfg*-Datei enthält eine Befehlsdefinition für das *check_http*-Plugin. Sie lautet:

```
define command{
        name            check_http
        command_name    check_http
        command_line    $USER1$/check_http -I $HOSTADDRESS$ $ARG1$
        }
```

Eine einfache Service-Definition, um den HTTP-Service auf dem *remotehost*-Rechner zu überwachen, würde so aussehen:

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     HTTP
        check_command           check_http
        }
```

Diese einfache Service-Definition wird den auf *remotehost* laufenden HTTP-Service überwachen. Es werden Alarme erzeugt, wenn der Web-Server nicht innerhalb von 10 Sekunden antwortet bzw. wenn HTTP-Fehler-Codes (403, 404, usw.) zurückgeliefert werden. Das ist alles, was Sie für eine einfache Überwachung brauchen. Ziemlich simpel, oder?

 Hinweis: Für eine erweiterte Überwachung starten Sie das *check_http*-Plugin manuell mit *--help* als Kommandozeilenargument, um alle Optionen zu sehen, die das Plugin unterstützt. Diese *--help*-Syntax funktioniert bei allen Plugins, die ich in diesem Dokument behandeln werde.

Eine fortgeschrittenere Definition zur Überwachung des HTTP-Service finden Sie nachfolgend. Diese Service-Definition wird prüfen, ob der URI /download/index.php die Zeichenkette "latest-version.tar.gz" enthält. Falls die Zeichenkette nicht gefunden wird, der URI nicht gültig ist oder der Web-Server länger als fünf Sekunden für die Antwort braucht, wird ein Fehler erzeugt.

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     Product Download Link
        check_command           check_http!-u /download/index.php -t 5 -s "latest-version.tar.gz"
        }
```

FTP überwachen

Wenn Sie FTP-Server überwachen müssen, können Sie das *check_ftp*-Plugin benutzen. Die *commands.cfg*-Datei enthält eine Befehlsdefinition für das *check_ftp*-Plugin. Sie lautet:

```
define command{
        command_name    check_ftp
        command_line    $USER1$/check_ftp -H $HOSTADDRESS$ $ARG1$
        }
```

Eine einfache Service-Definition, um den FTP-Server auf dem *remotehost*-Rechner zu überwachen, würde so aussehen:

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     FTP
        check_command           check_ftp
        }
```

Diese Service-Definition wird den FTP-Service überwachen und Alarme erzeugen, wenn der FTP-Server nicht innerhalb von 10 Sekunden antwortet.

Eine fortgeschrittenere Definition finden Sie nachfolgend. Dieser Service wird den FTP-Server prüfen, der auf Port 1023 auf *remotehost* läuft. Falls der FTP-Server nicht innerhalb von fünf Sekunden antwortet oder die Server-Antwort nicht die Zeichenkette "Pure-FTPd [TLS]" enthält, wird ein Fehler erzeugt.

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     Special FTP
        check_command           check_ftp!-p 1023 -t 5 -e "Pure-FTPd [TLS]"
        }
```

SSH überwachen

Wenn Sie SSH-Server überwachen müssen, können Sie das *check_ssh*-Plugin benutzen. Die *commands.cfg*-Datei enthält eine Befehlsdefinition für das *check_ssh*-Plugin. Sie lautet:

```
define command{
        command_name    check_ssh
        command_line    $USER1$/check_ssh $ARG1$ $HOSTADDRESS$
        }
```

Eine einfache Service-Definition, um den SSH-Server auf dem *remotehost*-Rechner zu überwachen, würde so aussehen:

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     SSH
        check_command           check_ssh
        }
```

Diese Service-Definition wird den SSH-Service überwachen und Alarme erzeugen, wenn der SSH-Server nicht innerhalb von 10 Sekunden antwortet.

Eine fortgeschrittenere Definition finden Sie nachfolgend. Dieser Service wird den SSH-Server prüfen und einen Fehler erzeugen, wenn der Server nicht innerhalb von fünf Sekunden antwortet oder die Server-Antwort nicht mit der Zeichenkette "OpenSSH_4.2" übereinstimmt.

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     SSH Version Check
        check_command           check_ssh!-t 5 -r "OpenSSH_4.2"
        }
```

SMTP

Das *check_smtp*-Plugin kann genutzt werden, um Ihren e-Mail-Server zu überwachen. Die *commands.cfg*-Datei enthält eine Befehlsdefinition für das *check_smtp*-Plugin. Sie lautet:

```
define command{
        command_name    check_smtp
        command_line    $USER1$/check_smtp -H $HOSTADDRESS$ $ARG1$
        }
```

Eine einfache Service-Definition, um den SMTP-Server auf dem *remotehost*-Rechner zu überwachen, würde so aussehen:

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     SMTP
        check_command           check_smtp
        }
```

Diese Service-Definition wird den SMTP-Service überwachen und Alarme erzeugen, wenn der SMTP-Server nicht innerhalb von 10 Sekunden antwortet.

Eine fortgeschrittenere Definition finden Sie nachfolgend. Dieser Service wird den SMTP-Server prüfen und einen Fehler erzeugen, wenn der Server nicht innerhalb von fünf Sekunden antwortet oder die Server-Antwort nicht die Zeichenkette "mygreatmailserver" enthält.

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     SMTP Response Check
        check_command           check_smtp!-t 5 -e "mygreatmailserver.com"
        }
```

POP3 überwachen

Das *check_pop*-Plugin kann genutzt werden, um den POP3-Service Ihres e-Mail-Servers zu überwachen. Die *commands.cfg*-Datei enthält eine Befehlsdefinition für das *check_pop*-Plugin. Sie lautet:

```
define command{
        command_name    check_pop
        command_line    $USER1$/check_pop -H $HOSTADDRESS$ $ARG1$
        }
```

Eine einfache Service-Definition, um den POP3-Service auf dem *remotehost*-Rechner zu überwachen, würde so aussehen:

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     POP3
        check_command           check_pop
        }
```

Diese Service-Definition wird den POP3-Service überwachen und Alarme erzeugen, wenn der POP3-Server nicht innerhalb von 10 Sekunden antwortet.

Eine fortgeschrittenere Definition finden Sie nachfolgend. Dieser Service wird den POP3-Service prüfen und einen Fehler erzeugen, wenn der Server nicht innerhalb von fünf Sekunden antwortet oder die Server-Antwort nicht die Zeichenkette "mygreatmailserver.com" übereinstimmt.

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     POP3 Response Check
        check_command           check_pop!-t 5 -e "mygreatmailserver.com"
        }
```

IMAP überwachen

Das *check_imap*-Plugin kann genutzt werden, um den IMAP4-Service Ihres e-Mail-Servers zu überwachen. Die *commands.cfg*-Datei enthält eine Befehlsdefinition für das *check_imap*-Plugin. Sie lautet:

```
define command{
        command_name    check_imap
        command_line    $USER1$/check_imap -H $HOSTADDRESS$ $ARG1$
        }
```

Eine einfache Service-Definition, um den IMAP4-Server auf dem *remotehost*-Rechner zu überwachen, würde so aussehen:

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     IMAP
        check_command           check_imap
        }
```

Diese Service-Definition wird den IMAP4-Service überwachen und Alarme erzeugen, wenn der IMAP-Server nicht innerhalb von 10 Sekunden antwortet.

Eine fortgeschrittenere Definition finden Sie nachfolgend. Dieser Service wird den IMAP4-Service prüfen und einen Fehler erzeugen, wenn der Server nicht innerhalb von fünf Sekunden antwortet oder die Server-Antwort nicht die Zeichenkette "mygreatmailserver.com" enthält.

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     IMAP4 Response Check
        check_command           check_imap!-t 5 -e "mygreatmailserver.com"
        }
```

Nagios erneut starten

Sobald Sie die neuen Host- und Service-Definitionen zu Ihrer/n Konfigurationsdatei(en) hinzugefügt haben, sind Sie bereit, sie zu überwachen. Um dies zu tun, müssen Sie die Konfiguration überprüfen und Nagios erneut starten.

Wenn der Überprüfungsprozess irgendwelche Fehler produziert, dann verbessern Sie Ihre Konfigurationsdatei, bevor Sie fortfahren. Stellen Sie sicher, dass Sie Nagios nicht erneut starten, bevor der Überprüfungsprozess ohne Fehler durchläuft!

Konfigurationsüberblick

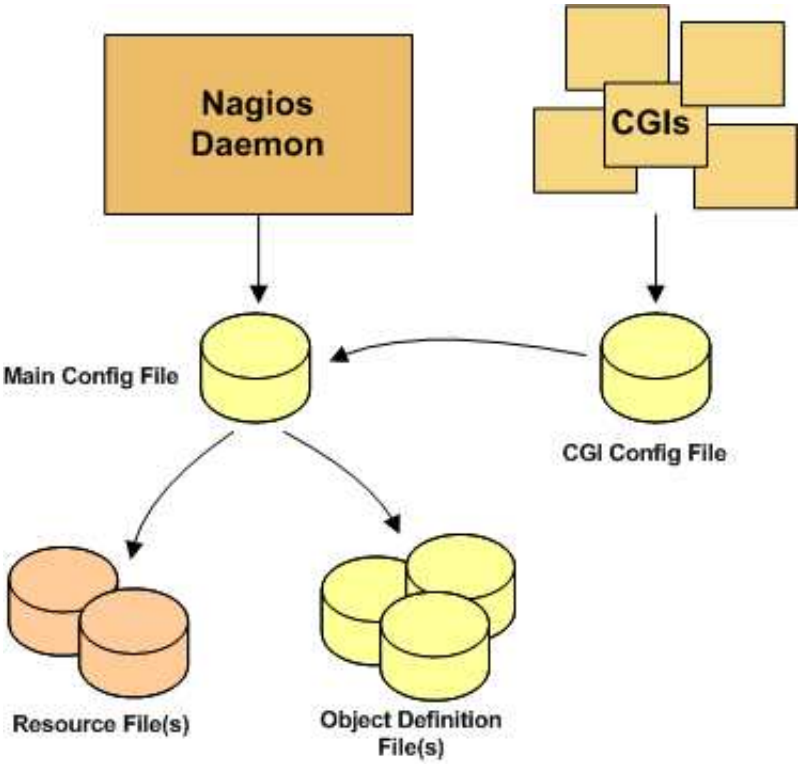
Hoch zu: Inhalt

Siehe auch: Hauptkonfigurationsdatei, Überblick Objektkonfiguration, CGI-Konfigurationsdatei

Einführung

Es gibt verschiedene Konfigurationsdateien, die Sie erstellen oder editieren müssen, bevor Sie irgendetwas überwachen können. Haben Sie Geduld! Nagios zu konfigurieren kann eine Weile dauern, besonders wenn Sie ein Neuling sind. Sobald Sie herausgefunden haben, wie die Dinge funktionieren, werden Sie feststellen, dass es die Mühe wert ist. :-)

Anmerkung: Beispiel-Konfigurationsdateien werden im */usr/local/nagios/etc/*-Verzeichnis installiert, wenn Sie der Schnellstart-Installationsanleitung folgen.



Hauptkonfigurationsdatei

Die Hauptkonfigurationsdatei enthält eine Reihe von Direktiven, die die Arbeitsweise des Nagios-Daemon beeinflussen. Diese Konfigurationsdatei wird vom Nagios-Daemon und den CGIs gelesen. Hier werden Sie in Ihr Konfigurationsabenteuer starten wollen.

Dokumentation zur Hauptkonfigurationsdatei finden Sie hier.

Ressource-Datei(en)

Ressource-Dateien können zur Speicherung von benutzerdefinierten Makros genutzt werden. Der Hauptgrund für Ressource-Dateien liegt darin, dass sie genutzt werden können, um sensible Informationen (wie z.B. Passworte) zu speichern, ohne dass sie für CGIs zugänglich sind (weil diese Dateien nicht von den CGIs gelesen werden, A.d.Ü.).

Sie können eine oder mehrere optionale Ressource-Dateien mit Hilfe der resource_file-Direktive in Ihrer Hauptkonfigurationsdatei angeben.

Objektdefinitionen-Dateien

Objektdefinitionen-Dateien werden genutzt, um Hosts, Services, Hostgruppen, Kontakte, Kontaktgruppen, Befehle usw. zu definieren. Hier definieren Sie, welche Dinge Sie überwachen wollen und wie Sie diese überwachen wollen.

Sie können eine oder mehrere Objektdefinitionen-Dateien mit Hilfe der cfg_file- und/oder cfg_dir-Direktiven in Ihrer Hauptkonfigurationsdatei angeben.

Eine Einführung zu Objektdefinitionen und wie sie in Beziehung zu einander stehen, finden Sie hier.

CGI-Konfigurationsdatei

Die CGI-Konfigurationsdatei enthält eine Reihe von Direktiven, die die Arbeitsweise der CGIs beeinflussen. Sie enthält auch einen Verweis auf die Hauptkonfigurationsdatei, so dass die CGIs wissen, wie Sie Nagios konfiguriert haben und wo Ihre Objektdefinitionen gespeichert sind.

Dokumentation zur CGI-Konfigurationsdatei finden Sie hier.

Optionen der Hauptkonfigurationsdatei

Hoch zu: Inhalt

Anmerkungen

Bei der Erstellung und/oder Änderung von Konfigurationsdateien sollten Sie folgendes beachten:

1. Zeilen, die mit einem '#'-Zeichen beginnen, werden als Kommentar angesehen und nicht verarbeitet
2. Variablennamen müssen am Anfang der Zeile beginnen - "White space" vor dem Namen sind nicht erlaubt
3. Variablennamen sind abhängig von Groß- und Kleinschreibung (case-sensitive)

Beispiel-Konfigurationsdatei

Hinweis: eine Beispiel-Hauptkonfigurationsdatei (*/usr/local/nagios/etc/nagios.cfg*) wird installiert, wenn Sie der Schnellstartanleitung folgen.

Position der Konfigurationsdatei

Die Hauptkonfigurationsdatei heißt üblicherweise *nagios.cfg* und ist im */usr/local/nagios/etc/*-Verzeichnis zu finden.

Variablen der Konfigurationsdatei

Nachfolgend finden Sie Beschreibungen jeder Option der Nagios-Hauptkonfigurationsdatei...

Protokolldatei (Log File)

Format: **log_file=<file_name>**

Beispiel: **log_file=/usr/local/nagios/var/nagios.log**

Diese Variable gibt an, wo Nagios die Hauptprotokolldatei anlegen soll. Dies sollte die erste Variable sein, die Sie in Ihrer Konfigurationsdatei definieren, weil Nagios versucht, dorthin die Fehler zu schreiben, die es in den übrigen Konfigurationsdaten findet. Wenn Sie Log-Rotation aktiviert haben, dann wird diese Datei automatisch jede Stunde, jeden Tag, jede Woche oder jeden Monat rotiert.

Objektkonfigurationsdatei (Object Configuration File)

Format: **cfg_file=<file_name>**

Beispiel: **cfg_file=/usr/local/nagios/etc/hosts.cfg**
cfg_file=/usr/local/nagios/etc/services.cfg
cfg_file=/usr/local/nagios/etc/commands.cfg

Diese Direktive wird benutzt, um eine Objektkonfigurationsdatei anzugeben, die Objektdefinitionen enthält, die Nagios zur Überwachung nutzen soll. Objektkonfigurationsdateien enthalten Definitionen für Hosts, Hostgruppen, Kontakte, Kontaktgruppe, Services, Befehle usw. Sie können Ihre Konfigurationsinformationen in verschiedene Dateien aufteilen und mehrere *cfg_file*=-Einträge angeben, um jede einzelne zu verarbeiten.

Objektkonfigurationsverzeichnis (Object Configuration Directory)

Format: **cfg_dir=<directory_name>**

Beispiel:
cfg_dir=/usr/local/nagios/etc/commands
cfg_dir=/usr/local/nagios/etc/services
cfg_dir=/usr/local/nagios/etc/hosts

Diese Direktive wird benutzt, um ein Verzeichnis anzugeben, das Objektkonfigurationsdateien enthält, die Nagios zur Überwachung nutzen soll. Alle Dateien in dem Verzeichnis mit einer *.cfg*-Endung werden als Objektkonfigurationsdateien verarbeitet. Zusätzlich wird Nagios rekursiv alle Konfigurationsdateien in den Unterverzeichnissen des Verzeichnisses verarbeiten, das Sie angegeben haben. Sie können Ihre Konfigurationsinformationen in verschiedene Verzeichnisse aufteilen und mehrere *cfg_dir*=-Einträge angeben, um alle Konfigurationsdateien jedes einzelnen Verzeichnisses zu verarbeiten.

Objekt-Cache-Datei (Object Cache File)

Format: **object_cache_file=<file_name>**

Beispiel: **object_cache_file=/usr/local/nagios/var/objects.cache**

Diese Direktive wird benutzt, um eine Datei anzugeben, in der eine zwischengespeicherte (cached) Kopie der Objektdefinitionen abgelegt wird. Die Cache-Datei wird jedes Mal (erneut) angelegt, wenn Nagios (erneut) gestartet wird, und wird von den CGIs benutzt. Sie ist dazu gedacht, die Zwischenspeicherung der Konfigurationsdateien in den CGIs zu beschleunigen und es Ihnen zu erlauben, die Objektkonfigurationsdateien zu editieren, während Nagios läuft, ohne die Ausgaben der CGIs zu beeinflussen.

vorgespeicherte Objektdatei (Precached Object File)

Format: **precached_object_file=<file_name>**

Beispiel: **precached_object_file=/usr/local/nagios/var/objects.precache**

Diese Direktive wird benutzt, um eine Datei anzugeben, die eine vorverarbeitete (pre-processed), vorgespeicherte (pre-cached) Kopie von Objektdefinitionen enthält. Diese Datei kann genutzt werden, um drastisch den Startvorgang in großen/komplexen Nagios-Installationen zu beschleunigen. Lesen Sie hier, wie der Startvorgang beschleunigt werden kann.

Ressource-Datei (Resource File)

Format: **resource_file=<file_name>**

Beispiel: **resource_file=/usr/local/nagios/etc/resource.cfg**

Dies wird benutzt, um eine optionale Ressource-Datei anzugeben, die $USERn$-Makro-Definitionen enthalten kann. $USER$-Makros sind sinnvoll zur Speicherung von Benutzernamen, Passwörtern und Objekten, die häufig in Befehlsdefinitionen (wie z.B. Verzeichnispfade) benutzt werden. Die CGIs werden *nicht* versuchen, Ressource-Dateien zu lesen, so dass Sie die Berechtigungen beschränken können (600 oder 660), um sensible Informationen zu schützen. Sie können mehrere Ressource-Dateien angeben, indem Sie mehrere resource_file-Einträge in die Hauptkonfigurationsdatei aufnehmen. Nagios wird sie alle verarbeiten. Schauen Sie in die resource.cfg-Datei im *sample-config/*-Unterverzeichnis der Nagios-Distribution, um ein Beispiel für die Definition von $USER$-Makros zu sehen.

temporäre Datei (Temp File)

Format: **temp_file=<file_name>**

Beispiel: **temp_file=/usr/local/nagios/var/nagios.tmp**

Dies ist eine temporäre Datei, die Nagios periodisch anlegt, wenn Kommentardaten, Statusdaten usw. aktualisiert werden. Die Datei wird gelöscht, wenn sie nicht länger benötigt wird.

temporärer Pfad (Temp Path)

Format: **temp_path=<dir_name>**

Beispiel: **temp_path=/tmp**

Dies ist ein Verzeichnis, das Nagios als "Schmierblock" (scratch space) benutzen kann, um während des Überwachungsprozesses temporäre Dateien anlegen zu können. Sie sollten *tmpwatch* oder ein ähnliches Programm ausführen, um in diesem Verzeichnis Dateien zu löschen, die älter als 24 Stunden sind.

Status-Datei (Status File)

Format: **status_file=<file_name>**

Beispiel: **status_file=/usr/local/nagios/var/status.dat**

Dies ist die Datei, die Nagios nutzt, um den aktuellen Zustand, Kommentar- und Ausfallzeitinformationen zu speichern. Diese Datei wird von den CGIs genutzt, so dass der aktuelle Überwachungszustand über ein Web-Interface berichtet werden kann. Die CGIs müssen Lesezugriff auf diese Datei haben, um richtig funktionieren zu können. Diese Datei wird jedes Mal gelöscht, wenn Nagios endet und neu angelegt, wenn Nagios startet.

Statusdatei-Aktualisierungsintervall (Status File Update Interval)

Format: **status_update_interval=<seconds>**

Beispiel: **status_update_interval=15**

Diese Einstellung legt fest, wie oft (in Sekunden) Nagios Statusdaten in der Statusdatei aktualisiert. Das kleinste Aktualisierungsintervall ist eine Sekunde.

Nagios-Benutzer (Nagios User)

Format: **nagios_user=<username/UID>**

Beispiel: **nagios_user=nagios**

Dies wird benutzt, um den "eigentlichen" (effective) Benutzer zu setzen, mit dem der Nagios-Prozess laufen soll. Nach dem anfänglichen Programmstart und bevor irgendeine Überwachung beginnt, wird Nagios die vorhandenen Berechtigungen "fallen lassen" (drop) und als dieser Benutzer laufen. Sie können entweder einen Benutzernamen oder eine UID angeben.

Nagios-Gruppe (Nagios Group)

Format: **nagios_group=<groupname/GID>**

Beispiel: **nagios_group=nagios**

Dies wird benutzt, um die "eigentliche" (effective) Gruppe zu setzen, mit der der Nagios-Prozess laufen soll. Nach dem anfänglichen Programmstart und bevor irgendeine Überwachung beginnt, wird Nagios die vorhandenen Berechtigungen "fallen lassen" (drop) und als diese Gruppe laufen. Sie können entweder einen Gruppennamen oder eine GID angeben.

Benachrichtigungsoption (Notifications Option)

Format: **enable_notifications=<0/1>**

Beispiel: **enable_notifications=1**

Diese Option legt fest, ob Nagios Benachrichtigungen versendet. Wenn diese Optionen deaktiviert ist, wird Nagios nach dem (Neu-) Start keine Benachrichtigungen für irgendeinen Host oder Service versenden. Anmerkung: Wenn Sie Statusinformationsaufbewahrung (retain state information) aktiviert haben, wird Nagios diese Einstellung ignorieren, wenn es (erneut) startet und die letzte bekannte Einstellung dieser Option nutzen (wie sie in der Statusaufbewahrungsdatei abgelegt ist), *es sei denn*, Sie haben die use_retained_program_state-Option deaktiviert. Wenn Sie diese Option ändern möchten, während die Statusaufbewahrung aktiviert ist (und die Option use_retained_program_state aktiviert ist), müssen Sie den entsprechenden externen Befehl benutzen oder sie über das Web-Interface ändern. Die Werte sind wie folgt:

- 0 = Benachrichtigungen deaktivieren
- 1 = Benachrichtigungen aktivieren (Default)

Option Service-Prüfungen ausführen (Service Check Execution Option)

Format: **execute_service_checks=<0/1>**

Beispiel: **execute_service_checks=1**

Diese Option legt fest, ob Nagios nach dem (Neu-) Start Service-Prüfungen ausführt. Wenn diese Option deaktiviert ist, wird Nagios nicht aktiv irgendwelche Service-Prüfungen ausführen und in einer Art von "Schlafmodus" verbleiben (es kann weiterhin passive Prüfungen empfangen, es sei denn, Sie haben diese deaktiviert). Diese Option wird oft benutzt, wenn Ersatz-Überwachungs-Server (backup monitoring server) konfiguriert werden, wie dies in der Dokumentation zu Redundanz beschrieben ist, oder wenn Sie eine verteilte-Überwachungsumgebung aufbauen. Anmerkung: wenn Sie Statusinformationsaufbewahrung (retain state information) aktiviert haben, wird Nagios diese Einstellung ignorieren, wenn es (erneut) startet und die letzte bekannte Einstellung dieser Option nutzen (wie sie in der Statusaufbewahrungsdatei abgelegt ist), *es sei denn*, Sie haben die use_retained_program_state-Option deaktiviert. Wenn Sie diese Option ändern möchten, während die Statusaufbewahrung aktiviert ist (und die Option use_retained_program_state aktiviert ist), müssen Sie den entsprechenden externen Befehl benutzen oder sie über das Web-Interface ändern. Die Werte sind wie folgt:

- 0 = keine Service-Prüfungen ausführen
- 1 = Service-Prüfungen ausführen (Default)

Option passive Service-Prüfungen akzeptieren (Passive Service Check Acceptance Option)

Format: **accept_passive_service_checks=<0/1>**

Beispiel: **accept_passive_service_checks=1**

Diese Option legt fest, ob Nagios nach dem (Neu-) Start passive Service-Prüfungen akzeptiert. Wenn diese Option deaktiviert ist, wird Nagios keine passiven Service-Prüfungen akzeptieren. Anmerkung: wenn Sie Statusinformationsaufbewahrung (retain state information) aktiviert haben, wird Nagios diese Einstellung ignorieren, wenn es (erneut) startet und die letzte bekannte Einstellung dieser Option nutzen (wie sie in der Statusaufbewahrungsdatei abgelegt ist), *es sei denn*, Sie haben die use_retained_program_state-Option deaktiviert. Wenn Sie diese Option ändern möchten, während die Statusaufbewahrung aktiviert ist (und die Option use_retained_program_state aktiviert ist), müssen Sie den entsprechenden externen Befehl benutzen oder sie über das Web-Interface ändern. Die Werte sind wie folgt:

- 0 = keine passiven Service-Prüfungen akzeptieren
- 1 = passive Service-Prüfungen akzeptieren (Default)

Option Host-Prüfungen ausführen (Host Check Execution Option)

Format: **execute_host_checks=<0/1>**

Beispiel: **execute_host_checks=1**

Diese Option legt fest, ob Nagios nach dem (Neu-) Start nach Bedarf oder regelmäßig geplante Host-Prüfungen ausführt. Wenn diese Option deaktiviert ist, wird Nagios nicht aktiv irgendwelche Host-Prüfungen ausführen, obwohl es weiterhin passive Host-Prüfungen empfangen wird, es sei denn, Sie haben diese deaktiviert). Diese Option wird am meisten genutzt, wenn Ersatz-Überwachungs-Server (backup monitoring server) konfiguriert werden, wie dies in der Dokumentation zu Redundanz beschrieben ist, oder wenn Sie eine verteilte-Überwachungsumgebung aufbauen. Anmerkung: wenn Sie Statusinformationsaufbewahrung (retain state information) aktiviert haben, wird Nagios diese Einstellung ignorieren, wenn es (erneut) startet und die letzte bekannte Einstellung dieser Option nutzen (wie sie in der Statusaufbewahrungsdatei abgelegt ist), *es sei denn*, Sie haben die use_retained_program_state-Option deaktiviert. Wenn Sie diese Option ändern möchten, während die

Statusaufbewahrung aktiviert ist (und die Option use_retained_program_state aktiviert ist), müssen Sie den entsprechenden externen Befehl benutzen oder sie über das Web-Interface ändern. Die Werte sind wie folgt:

- 0 = keine Host-Prüfungen ausführen
- 1 = Host-Prüfungen ausführen (Default)

Option passive Host-Prüfungen akzeptieren (Passive Host Check Acceptance Option)

Format: **accept_passive_host_checks=<0/1>**

Beispiel: **accept_passive_host_checks=1**

Diese Option legt fest, ob Nagios nach dem (Neu-) Start passive Host-Prüfungen akzeptiert. Wenn diese Option deaktiviert ist, wird Nagios keine passiven Host-Prüfungen akzeptieren. Anmerkung: wenn Sie Statusinformationsaufbewahrung (retain state information) aktiviert haben, wird Nagios diese Einstellung ignorieren, wenn es (erneut) startet und die letzte bekannte Einstellung dieser Option nutzen (wie sie in der Statusaufbewahrungsdatei abgelegt ist), *es sei denn*, Sie haben die use_retained_program_state-Option deaktiviert. Wenn Sie diese Option ändern möchten, während die Statusaufbewahrung aktiviert ist (und die Option use_retained_program_state aktiviert ist), müssen Sie den entsprechenden externen Befehl benutzen oder sie über das Web-Interface ändern. Die Werte sind wie folgt:

- 0 = keine passiven Host-Prüfungen akzeptieren
- 1 = passive Host-Prüfungen akzeptieren (Default)

Eventhandler-Option (Event Handler Option)

Format: **enable_event_handlers=<0/1>**

Beispiel: **enable_event_handlers=1**

Diese Option legt fest, ob Nagios nach dem (Neu-) Start Eventhandler ausführt. Wenn diese Option deaktiviert ist, wird Nagios keine Host- oder Service-Eventhandler ausführen. Anmerkung: wenn Sie Statusinformationsaufbewahrung (retain state information) aktiviert haben, wird Nagios diese Einstellung ignorieren, wenn es (erneut) startet und die letzte bekannte Einstellung dieser Option nutzen (wie sie in der Statusaufbewahrungsdatei abgelegt ist), *es sei denn*, Sie haben die use_retained_program_state-Option deaktiviert. Wenn Sie diese Option ändern möchten, während die Statusaufbewahrung aktiviert ist (und die Option use_retained_program_state aktiviert ist), müssen Sie den entsprechenden externen Befehl benutzen oder sie über das Web-Interface ändern. Die Werte sind wie folgt:

- 0 = Eventhandler deaktivieren
- 1 = Eventhandler aktivieren (Default)

Protokollrotationsmethode (Log Rotation Method)

Format: **log_rotation_method=<n/h/d/w/m>**

Beispiel: **log_rotation_method=d**

Dies ist die Rotationsmethode, die Nagios für Ihre Protokolldatei nutzen soll. Die Werte sind wie folgt:

- n = keine ("none" - die Protokolldatei nicht rotieren, das ist der Standard)
- h = stündlich ("hourly" - die Protokolldatei jede volle Stunde rotieren)
- d = täglich ("daily" - die Protokolldatei jeden Tag um Mitternacht rotieren)
- w = wöchentlich ("weekly" - die Protokolldatei jeden Samstag um Mitternacht rotieren)
- m = monatlich ("monthly" - die Protokolldatei am letzten Tag des Monats um Mitternacht rotieren)

Protokollarchiv-Pfad (Log Archiv Path)

Format: **log_archive_path=<path>**

Beispiel: **log_archive_path=/usr/local/nagios/var/archives/**

Dies ist das Verzeichnis, in dem Nagios die Protokolldateien ablegen soll, die rotiert wurden. Diese Option wird ignoriert, wenn Sie die Funktionalität der Protokollrotation (log rotation) nicht nutzen.

Option externe Befehle prüfen (External Command Check Option)

Format: **check_external_commands=<0/1>**

Beispiel: **check_external_commands=1**

Diese Option legt fest, ob Nagios das command file auf auszuführende Befehle prüfen soll. Diese Option muss aktiviert sein, wenn Sie planen, das Command-CGI zu nutzen, um Befehle über das Web-Interface zu erteilen. Mehr Informationen zu externen Befehlen finden Sie hier.

- 0 = nicht auf externe Befehle prüfen
- 1 = auf externe Befehle prüfen (Default)

Prüfintervall externe Befehle (External Command Check Interval)

Format: **command_check_interval=<xxx>[s]**

Beispiel: **command_check_interval=1**

Wenn Sie eine Zahl mit einem angehängten "s" angeben (z.B. 30s), dann ist dies die Zahl in *Sekunden*, die zwischen Prüfungen auf externe Befehle gewartet werden soll. Wenn Sie das "s" weglassen, ist dies die Zahl von "Zeiteinheiten", die zwischen den Prüfungen auf externe Befehle gewartet werden soll. Solange Sie nicht den Standardwert (60) der interval_length-Direktive geändert haben (wie weiter unten definiert), bedeutet dieser Wert Minuten.

Anmerkung: durch das Setzen dieses Wertes auf **-1** wird Nagios so oft wie möglich auf externe Befehle prüfen. Jedes Mal, wenn Nagios auf externe Befehle prüft, wird es alle im command file befindlichen Befehle lesen und verarbeiten, bevor es mit anderen Aufgaben fortfährt. Mehr Informationen zu externen Befehlen finden Sie hier.

externe Befehlsdatei (External Command File)

Format: **command_file=<file_name>**

Beispiel: **command_file=/usr/local/nagios/var/rw/nagios.cmd**

Dies ist die Datei, die Nagios auf zu verarbeitende externe Befehle prüfen wird. Das Command-CGI schreibt Befehle in diese Datei. Die externe Befehlsdatei ist als "named pipe" (FIFO) implementiert, die beim Start von Nagios angelegt und beim Herunterfahren wieder gelöscht wird. Wenn die Datei beim Start von Nagios existiert, wird der Nagios-Prozess mit einer Fehlermeldung enden. Mehr Informationen zu externen Befehlen finden Sie hier.

externe Befehlspuffer-Slots (External Command Buffer Slots)

Format: **external_command_buffer_slots=<#>**

Beispiel: **external_command_buffer_slots=512**

Anmerkung: dies ist ein fortgeschrittenes Feature. Diese Option legt fest, wie viele Puffer-Slots Nagios für die Zwischenspeicherung von externen Befehlen reserviert, die von einem "worker thread" aus der externen Befehlsdatei gelesen, aber noch nicht vom "main thread" des Nagios-Daemons verarbeitet wurden. Jeder Slot kann einen externen Befehl enthalten, so dass diese Option im Wesentlichen bestimmt, wie viele Befehle gepuffert werden können. Bei Installationen, wo Sie eine große Zahl von passiven Prüfungen verarbeiten (z.B. verteilten Setups), müssen Sie ggf. diese Zahl erhöhen. Sie sollten den Einsatz von MRTG erwägen, um die Nutzung der externen Befehlspuffer grafisch darzustellen. Mehr zur Konfiguration der grafischen Darstellung finden Sie hier.

Update-Prüfungen (Update Checks)

Format: **check_for_updates=<0/1>**

Beispiel: **check_for_updates=1**

Diese Option legt fest, ob Nagios automatisch prüft, ob neue Updates (Versionen) verfügbar sind. Es wird empfohlen, dass Sie diese Option aktivieren, damit Sie immer über die letzten kritischen Patches für Nagios informiert sind. Nagios ist kritisch für Sie - stellen Sie sicher, dass es sich in guter Form befindet. Nagios wird einmal am Tag auf neue Updates prüfen. Daten, die von Nagios Enterprises für den Update-Check gesammelt werden, werden gemäß unserer Privacy-Policy verarbeitet (Details siehe http://api.nagios.org).

Nur Update-Prüfungen (Bare Update Checks)

Format: **bare_update_checks=<0/1>**

Beispiel: **bare_update_checks=0**

Diese Option legt fest, welche Daten Nagios an api.nagios.org schickt, wenn es auf Updates prüft. Als Standard wird Nagios Informationen über die aktuell installierte Version schicken sowie einen Hinweis, ob es sich hierbei um eine neue oder eine aktualisierte Version handelt. Nagios Enterprises nutzt diese Daten, um die Anzahl der Nutzer bestimmter Versionen zu ermitteln. Aktivieren Sie diese Option, wenn Sie diese Informationen nicht senden möchten.

Sperrdatei (Lock File)

Format: **lock_file=<file_name>**

Beispiel: **lock_file=/tmp/nagios.lock**

Diese Option gibt die Position der Sperrdatei an, die Nagios anlegen sollte, wenn es als Daemon läuft (wenn es mit der -d Kommandozeilenoption gestartet wurde). Diese Datei enthält die Prozess-ID (PID) des laufenden Nagios-Prozesses.

Statusaufbewahrungsoption (State Retention Option)

Format: **retain_state_information=<0/1>**

Beispiel: **retain_state_information=1**

Diese Option legt fest, ob Nagios Statusinformationen für Hosts und Services zwischen Programmneustarts aufbewahren soll. Wenn Sie diese Option aktivieren, sollten Sie ein Wert für die state_retention_file-Variable angeben. Wenn sie aktiviert ist, wird Nagios alle Statusinformationen für Hosts und Services sichern, bevor es beendet (oder neu gestartet) wird und vorher gespeicherte Statusinformationen einlesen, wenn es neu gestartet wird.

- 0 = Statusinformationen nicht aufbewahren
- 1 = Statusinformationen aufbewahren (Default)

Statusaufbewahrungsdatei (State Retention File)

Format: **state_retention_file=<file_name>**

Beispiel: **state_retention_file=/usr/local/nagios/var/retention.dat**

Dies ist die Datei, die Nagios für die Speicherung von Status-, Ausfallzeit- und Kommentarinformationen nutzt, bevor es endet. Wenn Nagios neu startet, wird es die in dieser Datei gespeicherten Informationen nutzen, um die anfänglichen Zustände von Services und Hosts zu setzen, bevor es mit der Überwachung beginnt. Damit Nagios Statusinformationen zwischen Programmneustarts aufbewahrt, müssen Sie die retain_state_information-Option aktivieren.

automatisches Statusaufbewahrungs-Aktualisierungsintervall (Automatic State Retention Update Interval)

Format: **retention_update_interval=<minutes>**

Beispiel: **retention_update_interval=60**

Diese Einstellung legt fest, wie oft (in Minuten) Nagios automatisch während des normalen Betriebs die Aufbewahrungsdaten aktualisiert. Wenn Sie einen Wert von Null angeben, wird Nagios nicht in regelmäßigen Intervallen die Aufbewahrungsdaten sichern, aber es wird die Aufbewahrungsdaten vor der Beendigung oder dem Neustart sichern. Wenn Sie die Statusaufbewahrung deaktiviert haben (mit der retain_state_information-Option), hat diese Option keine Auswirkung.

Option aufbewahrten Programmzustand nutzen (Use Retained Program State Option)

Format: **use_retained_program_state=<0/1>**

Beispiel: **use_retained_program_state=1**

Diese Einstellung legt fest, ob Nagios verschiedene programmweite Statusvariablen auf Basis der Aufbewahrungsdatei setzen soll. Einige dieser programmweiten Statusvariablen, die normalerweise über Programmstarts hinweg gesichert werden, wenn Statusaufbewahrung aktiviert ist, umfassen die enable_notifications-, enable_flap_detection-, enable_event_handlers-, execute_service_checks- und accept_passive_service_checks-Optionen. Wenn Sie Statusaufbewahrung deaktiviert haben, hat diese Option keine Auswirkung.

- 0 = keinen aufbewahrten Programmzustand nutzen
- 1 = aufbewahrten Programmzustand nutzen (Default)

Option aufbewahrte Planungsinformationen nutzen (Use Retained Scheduling Info Option)

Format: **use_retained_scheduling_info=<0/1>**

Beispiel: **use_retained_scheduling_info=1**

Diese Einstellung legt fest, ob Nagios Planungsinformationen für Hosts und Services aufbewahrt, wenn es neu startet. Wenn Sie eine große Zahl (oder einen großen Anteil) von Hosts oder Services hinzufügen, empfehle ich diese Option zu deaktivieren, wenn Sie das erste Mal Nagios neu starten, weil es nachteilig die Verteilung von initialen Prüfungen beeinflussen kann. Andernfalls werden Sie diese Option wahrscheinlich aktiviert lassen.

- 0 = keine aufbewahrten Planungsinformationen nutzen
- 1 = aufbewahrten Planungsinformationen nutzen (Default)

aufbewahrte Host- und Service-Attributmasken (Retained Host and Service Attribute Masks)

Format: **retained_host_attribute_mask=<number>**
retained_service_attribute_mask=<number>

Beispiel: **retained_host_attribute_mask=0**
retained_service_attribute_mask=0

WARNUNG: dies ist ein fortgeschrittenes Feature. Sie müssen den Nagios-Quellcode lesen, um diese Option effizient nutzen zu können.

Diese Option legt fest, welche Host- oder Service-Attribute NICHT über Programmneustarts hinweg aufbewahrt werden. Die Werte für diese Optionen sind ein bitweises AND der durch die "MODATTR_"-Definitionen angegebenen Werte in den include/common.h-Quellcode-Dateien. Per Default werden alle Host- und Service-Attribute aufbewahrt.

aufbewahrte Prozessattributmasken (Retained Process Attribute Masks)

Format: **retained_process_host_attribute_mask=<number>**
retained_process_service_attribute_mask=<number>

Beispiel: **retained_process_host_attribute_mask=0**
retained_process_service_attribute_mask=0

WARNUNG: dies ist ein fortgeschrittenes Feature. Sie müssen den Nagios-Quellcode lesen, um diese Option effizient nutzen zu können.

Diese Option legt fest, welche Prozessattribute NICHT über Programmneustarts hinweg aufbewahrt werden. Es gibt zwei Masken, weil es oft separate Host- und Service-Prozessattribute gibt, die geändert werden können. Beispielsweise können Host-Prüfungen auf Programmebene deaktiviert werden, während Service-Prüfungen weiterhin aktiviert sind. Die Werte für diese Optionen sind ein bitweises AND der durch die "MODATTR_"-Definitionen angegebenen Werte in den include/common.h-Quellcode-Dateien. Per Default werden alle Prozessattribute aufbewahrt.

aufbewahrte Kontaktattributmasken (Retained Contact Attribute Masks)

Format: **retained_contact_host_attribute_mask=<number>**
retained_contact_service_attribute_mask=<number>

Beispiel: **retained_contact_host_attribute_mask=0**
retained_contact_service_attribute_mask=0

WARNUNG: dies ist ein fortgeschrittenes Feature. Sie müssen den Nagios-Quellcode lesen, um diese Option effizient nutzen zu können.

Diese Option legt fest, welche Kontaktattribute NICHT über Programmneustarts hinweg aufbewahrt werden. Es gibt zwei Masken, weil es oft separate Host- und Service-Prozessattribute gibt, die geändert werden können. Die Werte für diese Optionen sind ein bitweises AND der durch die "MODATTR_"-Definitionen angegebenen Werte in den include/common.h-Quellcode-Dateien. Per Default werden alle Kontaktattribute aufbewahrt.

Syslog-Protokollierungsoption (Syslog Logging Option)

Format: **use_syslog=<0/1>**

Beispiel: **use_syslog=1**

Diese Variable legt fest, ob Meldungen im Syslog des lokalen Hosts protokolliert werden sollen. Die Werte sind wie folgt:

- 0 = Syslog nicht nutzen
- 1 = Syslog nutzen [Default]

Benachrichtigungsprotokollierungsoption (Notification Logging Option)

Format: **log_notifications=<0/1>**

Beispiel: **log_notifications=1**

Diese Variable legt fest, ob Benachrichtungsmeldungen protokolliert werden. Wenn Sie eine Menge von Kontakten oder ständigen Service-Ausfällen haben, dann wird Ihre Protokolldatei relativ schnell wachsen. Benutzen Sie diese Option, um die Protokollierung von (Kontakt-)Benachrichtigungen zu verhindern.

- 0 = keine Benachrichtigungen protokollieren
- 1 = Benachrichtigungen protokollieren [Default]

Option Service-Wiederholungsprüfungen protokollieren (Service Check Retry Logging Option)

Format: **log_service_retries=<0/1>**

Beispiel: **log_service_retries=1**

Diese Variable legt fest, ob Service-Wiederholungsprüfungen protokolliert werden.
Service-Wiederholungsprüfungen treten auf, wenn ein Service-Prüfergebnis einen nicht-OK-Status ergibt, Sie Nagios aber so konfiguriert haben, dass die Prüfung mehr als einmal wiederholt wird, bevor ein Fehler gemeldet wird. Services in diesem Zustand befinden sich in einem "Soft"-Status. Die Protokollierung von Service-Wiederholungsprüfungen ist meist dann sinnvoll, wenn Sie versuchen, Nagios zu debuggen, oder Service-Eventhandler zu testen.

- 0 = keine Service-Wiederholungsprüfungen protokollieren [Default-H]
- 1 = Service-Wiederholungsprüfungen protokollieren [Default-C]

Option Host-Wiederholungsprüfungen-protokollieren (Host Check Retry Logging Option)

Format: **log_host_retries=<0/1>**

Beispiel: **log_host_retries=1**

Diese Variable legt fest, ob Host-Wiederholungsprüfungen protokolliert werden. Die Protokollierung von Host-Wiederholungsprüfungen ist meist dann sinnvoll, wenn Sie versuchen, Nagios zu debuggen, oder Host-Eventhandler zu testen.

- 0 = keine Host-Wiederholungsprüfungen protokollieren [Default-H]
- 1 = Host-Wiederholungsprüfungen protokollieren [Default-C]

Option Eventhandler-protokollieren (Event Handler Logging Option)

Format: **log_event_handlers=<0/1>**

Beispiel: **log_event_handlers=1**

Diese Variable legt fest, ob Service- und Host-Eventhandlers protokolliert werden. Eventhandler sind optionale Befehle, die ausgeführt werden können, wenn sich der Zustand eines Hosts oder Service ändert. Die Protokollierung von Eventhandlern ist meist dann sinnvoll, wenn Sie versuchen, Nagios zu debuggen, oder Ihre Eventhandler-Scripts zu testen.

- 0 = Eventhandler nicht protokollieren
- 1 = Eventhandler protokollieren [Default]

Option initiale Zustände protokollieren (Initial States Logging Option)

Format: **log_initial_states=<0/1>**

Beispiel: **log_initial_states=1**

Diese Variable legt fest, ob Nagios alle anfänglichen Host- und Service-Zustände protokolliert, selbst wenn sie in einem OK-Zustand sind. Anfängliche Service- und Host-Zustände werden normalerweise nur dann protokolliert, wenn es bei der ersten Prüfung ein Problem gibt. Die Aktivierung dieser Option ist sinnvoll, wenn Sie eine Applikation benutzen, die die Protokolldatei abfragt, um Langzeit-Zustandsstatistiken für Services und Hosts zu erstellen.

- 0 = keine anfänglichen Zusände protokollieren (Default)
- 1 = anfängliche Zustände protokollieren

Option externe Befehle protokollieren (External Command Logging Option)

Format: **log_external_commands=<0/1>**

Beispiel: **log_external_commands=1**

Diese Variable legt fest, ob Nagios externe Befehle protokolliert, die es aus der externen Befehlsdatei erhält. Anmerkung: diese Option kontrolliert nicht, ob passive Service-Prüfungen (die eine Art von externen Befehlen sind) protokolliert werden. Zur Aktivierung oder Deaktivierung der Protokollierung von passiven Prüfungen nutzen Sie die log_passive_checks-Option.

- 0 = keine externen Befehle protokollieren
- 1 = externe Befehle protokollieren (Default)

Option passive Prüfungen protokollieren (Passive Check Logging Option)

Format: **log_passive_checks=<0/1>**

Beispiel: **log_passive_checks=1**

Diese Variable legt fest, ob Nagios passive Host- und Service-Prüfungen protokolliert, die es von der externen Befehlsdatei bekommt. Wenn Sie eine verteilte Überwachungsumgebung aufbauen oder planen, eine große Zahl von passiven Prüfungen auf einer regelmäßigen Basis zu behandeln, dann können Sie diese Option deaktivieren, damit Ihre Protokolldatei nicht zu groß wird.

- 0 = keine passiven Prüfungen protokollieren
- 1 = passive Prüfungen protokollieren (Default)

globale Host-Eventhandler Option (Global Host Event Handler Option)

Format: **global_host_event_handler=<command>**

Beispiel: **global_host_event_handler=log-host-event-to-db**

Diese Option erlaubt Ihnen, einen Host-Eventhandler-Befehl anzugeben, der für jeden Host-Zustandswechsel ausgeführt wird. Der globale Eventhandler wird direkt vor dem Eventhandler ausgeführt, den Sie optional in jeder Host-Definition angeben können. Das *Befehls*-Argument ist der Kurzname eines Befehls, den Sie in Ihrer Objektkonfigurationsdatei angeben. Die maximale Ausführungszeit dieses Befehls kann durch die event_handler_timeout-Option angegeben werden. Mehr Informationen zu Eventhandlern finden Sie hier.

Globale Service-Eventhandler-Option (Global Service Event Handler Option)

Format: **global_service_event_handler=<command>**

Beispiel: **global_service_event_handler=log-service-event-to-db**

Diese Option erlaubt Ihnen, einen Service-Eventhandler-Befehl anzugeben, der für jeden Service-Zustandswechsel ausgeführt wird. Der globale Eventhandler wird direkt vor dem Eventhandler ausgeführt, den Sie optional in jeder Service-Definition angeben können. Das *Befehls*-Argument ist der Kurzname eines Befehls, den Sie in Ihrer Objektkonfigurationsdatei angeben. Die maximale Ausführungszeit dieses Befehls kann durch die event_handler_timeout-Option angegeben werden. Mehr Informationen zu Eventhandlern finden Sie hier.

Ruhezeit zwischen Prüfungen (Inter-Check Sleep Time)

Format: **sleep_time=<seconds>**

Beispiel: **sleep_time=1**

Dies ist die Anzahl von Sekunden, die Nagios "schlafen" wird, bevor es in der Planungswarteschlange (scheduling queue) nach weiteren auszuführenden Host- oder Service-Prüfungen schaut. Beachten Sie, dass Nagios nur schlafen wird, nachdem es anstehende Service-Prüfungen erledigt hat, die in Verzug geraten waren.

Verzögerungsmethode für Service-Prüfungen (Service Inter-Check Delay Method)

Format: **service_inter_check_delay_method=<n/d/s/x.xx>**

Beispiel: **service_inter_check_delay_method=s**

Diese Option erlaubt Ihnen die Kontrolle darüber, wie Service-Prüfungen anfänglich in der Planungswarteschlange "ausgebreitet" werden. Die Verwendung einer "schlauen" Verzögerungsberechnung (der Standard) veranlasst Nagios, ein durchschnittliches Prüfintervall zu berechnen und die anfänglichen Prüfungen aller Services über dieses Intervall zu verteilen, um dadurch CPU-Lastspitzen zu eliminieren. Keine Verzögerung zu benutzen wird *nicht* empfohlen, weil es dafür sorgt, dass die Ausführung aller Service-Prüfungen zur gleichen Zeit geplant wird. Das bedeutet, dass Sie generell hohe CPU-Spitzen haben werden, wenn die Services alle parallel ausgeführt werden. Mehr Informationen dazu, wie die Verzögerung von Service-Prüfungen die Planung dieser Prüfungen beeinflusst, finden Sie hier. Die Werte sind wie folgt:

- n = keine (none) Verzögerung benutzen - planen, dass alle Service-Prüfungen sofort ausgeführt werden (d.h. zur gleichen Zeit!)
- d = eine "dumme" (dumb) Verzögerung von einer Sekunde zwischen Service-Prüfungen benutzen
- s = eine "schlaue" (smart) Verzögerungsberechnung verwenden, um die Service-Prüfungen

gleichmäßig zu verteilen (Default)
- x.xx = eine benutzerdefinierte Verzögerung von x.xx Sekunden zwischen den Prüfungen benutzen

maximale Service-Prüfungsverteilung (Maximum Service Check Spread)

Format: **max_service_check_spread=<minutes>**

Beispiel: **max_service_check_spread=30**

Diese Option legt die maximale Anzahl in Minuten fest vom Nagios-Start bis zur Ausführung aller (regelmäßig geplanten) Service-Prüfungen. Diese Option wird automatisch die Verzögerungsmethode für Service-Prüfungen anpassen (falls notwendig), um sicherzustellen, dass die anfänglichen Prüfungen aller Services in dem Zeitrahmen stattfinden, den Sie angeben. Generell wird diese Optionen keine Auswirkung auf die Planung von Service-Prüfungen haben, wenn die Planungsinformationen mit Hilfe der use_retained_scheduling_info-Option aufbewahrt werden. Standardwert ist **30** (Minuten).

Service-Verschachtelungsfaktor (Service Interleave Factor)

Format: **service_interleave_factor=<s | *x*>**

Beispiel: **service_interleave_factor=s**

Diese Variable legt fest, wie Service-Prüfungen verschachtelt werden. Verschachtelung erlaubt eine gleichmäßigere Verteilung von Service-Prüfungen, reduzierte Last auf entfernten Hosts und schnellere Erkennung von Host-Problemen. Das Setzen des Wertes auf 1 ist gleichbedeutend mit keiner Verschachtelung der Service-Prüfungen (so arbeiteten die Nagios-Version bis 0.0.5). Setzen Sie diesen Wert auf **s** (schlau/smart) für die automatische Berechnung, solange Sie keinen bestimmten Grund für die Änderung haben. Der beste Weg zum Verständnis, wie Verschachtelung funktioniert, ist der Blick auf das status CGI (detail view), während Nagios startet. Sie sollten sehen, dass die Service-Prüfergebnisse verteilt werden, während sie auftauchen. Mehr Informationen dazu, wie Verschachtelung funktioniert, finden Sie hier.

- *x* = eine Zahl gleich oder größer 1, die den zu benutzenden Verschachtelungsfaktor angibt. Ein Verschachtelungsfaktor von 1 bedeutet keine Verschachtelung von Service-Prüfungen
- s = eine "schlaue" (smart) Verschachtelungsberechnung benutzen (Default)

maximale Anzahl gleichzeitiger Service-Prüfungen (Maximum Concurrent Service Checks)

Format: **max_concurrent_checks=<max_checks>**

Beispiel: **max_concurrent_checks=20**

Diese Option erlaubt Ihnen die Angabe der maximalen Anzahl von Service-Prüfungen, die zu irgendeiner Zeit gleichzeitig ausgeführt werden. Das Angeben eines Wertes von 1 verhindert grundsätzlich die Ausführung von parallelen Service-Prüfungen. Der Wert 0 (der Standard) sorgt dafür, dass es keine Beschränkung der Anzahl von gleichzeitig ausgeführten Service-Prüfungen gibt. Sie müssen den Wert auf der Basis der Systemressourcen anpassen, die Ihr Nagios-Rechner zur Verfügung stellt, da er direkt die maximale Last des System beeinflusst (Prozessorauslastung, Speicher, usw.). Mehr Informationen dazu, wie viele parallele Prüfungen Sie zulassen sollten, finden Sie hier.

Prüfergebnis-Erntefrequenz (Check Result Reaper Frequency)

Format: **check_result_reaper_frequency=<frequency_in_seconds>**

Beispiel: **check_result_reaper_frequency=5**

Diese Option erlaubt Ihnen die Kontrolle über die Frequenz *in Sekunden* der Prüfergebnis-"Ernte"-Ereignisse. "Ernte"-Ereignisse verarbeiten die Ergebnisse von Host- und Service-Prüfungen, die beendet wurden. Diese Ereignisse bilden den Kern der Überwachungslogik von Nagios.

maximale Prüfergebnis-Erntezeit (Maximum Check Result Reaper Time)

Format: **max_check_result_reaper_time=<seconds>**

Beispiel: **max_check_result_reaper_time=30**

Diese Option erlaubt Ihnen die Kontrolle der maximalen Zeit *in Sekunden*, die Host- und Service-Prüfergebnis-"Ernte"-Ereignisse laufen dürfen. "Ernte"-Ereignisse verarbeiten die Ergebnisse von Host- und Service-Prüfungen, die beendet sind. Wenn es eine Menge von Ergebnissen zu verarbeiten gibt, können Ernte-Ereignisse lange brauchen, um zu enden, was die pünktliche Ausführung von neuen Host- und Service-Prüfungen verzögern könnte. Diese Variable erlaubt Ihnen die Begrenzung der Zeit, die ein einzelnes Ernte-Ereignis laufen darf, bevor es die Kontrolle an andere Teile der Nagios-Überwachungslogik zurückgibt.

Prüfergebnis-Pfad (Check Result Path)

Format: **check_result_path=<path>**

Beispiel: **check_result_path=/var/spool/nagios/checkresults**

Diese Option legt fest, welches Verzeichnis Nagios benutzen wird, um temporär Host- und Service-Prüfergebnisse zu speichern, bevor sie verarbeitet werden. Diese Verzeichnis sollte nicht benutzt werden, um andere Dateien dort zu speichern, weil Nagios dieses Verzeichnis periodisch von alten Dateien säubern wird (mehr Informationen dazu finden Sie bei der max_check_result_file_age-Option).

Anmerkung: stellen Sie sicher, dass nur eine einzelne Nagios-Instanz Zugriff auf den Prüfergebnispfad hat. Wenn mehrere Nagios-Instanzen Zugriff auf das gleiche Verzeichnis haben, werden Sie Probleme bekommen, weil Prüfergebnisse von der falschen Nagios-Instanz verarbeitet wurden!

maximales Alter der Prüfergebnisdatei (Max Check Result File Age)

Format: **max_check_result_file_age=<seconds>**

Beispiel: **max_check_result_file_age=3600**

Diese Option legt das maximale Alter in Sekunden fest, die Daten aus den Prüfergebnisdateien im check_result_path-Verzeichnis als gültig angesehen werden. Prüfergebnisdateien, die älter als dieser Schwellwert sind, werden von Nagios gelöscht und die darin enthaltenen Daten werden nicht verarbeitet. Durch die Angabe eines Wertes von Null (0) bei dieser Option wird Nagios alle

Prüfergebnisdateien verarbeiten - selbst wenn sie älter als Ihre Hardware sind :-).

Verzögerungsmethode für Host-Prüfungen (Host Inter-Check Delay Method)

Format: **host_inter_check_delay_method=<n/d/s/x.xx>**

Beispiel: **host_inter_check_delay_method=s**

Diese Option erlaubt Ihnen die Kontrolle darüber, wie Host-Prüfungen, *die für eine regelmäßige Prügung geplant sind*, anfänglich in der Planungswarteschlange "ausgebreitet" werden. Die Verwendung einer "schlauen" Verzögerungsberechnung (der Standard) veranlasst Nagios, ein durchschnittliches Prüfintervall zu berechnen und die anfänglichen Prüfungen aller Hosts über dieses Intervall zu verteilen, um dadurch CPU-Lastspitzen zu eliminieren. Keine Verzögerung zu benutzen wird generell *nicht* empfohlen, weil es dafür sorgt, dass die Ausführung aller Host-Prüfungen zur gleichen Zeit geplant wird. Mehr Informationen dazu, wie die Verzögerung von Host-Prüfungen die Planung dieser Prüfungen beeinflusst, finden Sie hier. Die Werte sind wie folgt:

- n = keine (none) Verzögerung benutzen - planen, dass alle Host-Prüfungen sofort ausgeführt werden (d.h. zur gleichen Zeit!)
- d = eine "dumme" (dumb) Verzögerung von einer Sekunde zwischen Host-Prüfungen benutzen
- s = eine "schlaue" (smart) Verzögerungsberechnung verwenden, um die Host-Prüfungen gleichmäßig zu verteilen (Default)
- x.xx = eine benutzerdefinierte Verzögerung von x.xx Sekunden zwischen den Prüfungen benutzen

maximale Host-Prüfungsverteilung (Maximum Host Check Spread)

Format: **max_host_check_spread=<minutes>**

Beispiel: **max_host_check_spread=30**

Diese Option legt die maximale Anzahl in Minuten fest vom Nagios-Start bis zur Ausführung aller (regelmäßig geplanten) Host-Prüfungen. Diese Option wird automatisch die Verzögerungsmethode für Host-Prüfungen anpassen (falls notwendig), um sicherzustellen, dass die anfänglichen Prüfungen aller Hosts in dem Zeitrahmen stattfinden, den Sie angeben. Generell wird diese Optionen keine Auswirkung auf die Planung von Host-Prüfungen haben, wenn die Planungsinformationen mit Hilfe der use_retained_scheduling_info-Option aufbewahrt werden. Standardwert ist **30** (Minuten).

Zeitintervalllänge (Timing Interval Length)

Format: **interval_length=<seconds>**

Beispiel: **interval_length=60**

Dies ist die Zahl von Sekunden pro "Zeitintervall" (unit interval), das für die Steuerung in der Planungswarteschlange, bei erneuten Benachrichtigungen usw. verwendet wird. "Zeitintervalle" werden in den Objektkonfigurationsdateien benutzt, um festzulegen, wie oft eine Service-Prüfung ausgeführt, ein Kontakt erneut informiert wird usw.

Wichtig: Der Standardwert hierfür ist auf 60 eingestellt, was bedeutet, dass ein "Zeitwert" von eins (1) in der Objektkonfigurationsdatei 60 Sekunden bedeutet (eine Minute). Ich habe keine anderen Werte für diese Variable ausprobiert, also machen Sie Änderungen auf Ihr eigenes Risiko, wenn Sie etwas anderes

einstellen!

automatische Wiedereinplanungs-Option (Auto-Rescheduling Option)

Format: **auto_reschedule_checks=<0/1>**

Beispiel: **auto_reschedule_checks=1**

Diese Option legt fest, ob Nagios versucht, aktive Host- und Service-Prüfungen automatisch erneut einzuplanen, um sie über die Zeit "auszugleichen" (smooth out). Dies kann helfen, die Last des Überwachungsrechners auszubalancieren, da es versucht, die Zeit zwischen aufeinander folgenden Prüfungen einheitlich zu halten, auf Kosten der Ausführung von Prüfungen mit einer rigideren Planung.

WARNUNG: DIES IST EIN EXPERIMENTELLES FEATURE UND KÖNNTE IN DER ZUKUNFT ENTFERNT WERDEN. DIE AKTIVIERUNG DIESER OPTION KANN DIE LEISTUNG REDUZIEREN - STATT SIE ZU ERHÖHEN - WENN SIE UNGEEIGNET BENUTZT WIRD!

automatisches Wiedereinplanungs-Intervall (Auto-Rescheduling Interval)

Format: **auto_rescheduling_interval=<seconds>**

Beispiel: **auto_rescheduling_interval=30**

Diese Option legt fest, wie oft (in Sekunden) Nagios versuchen wird, automatisch Prüfungen erneut einzuplanen. Diese Option ist nur wirksam, wenn die auto_reschedule_checks-Option aktiviert ist. Standard ist 30 Sekunden.

WARNUNG: DIES IST EIN EXPERIMENTELLES FEATURE UND KÖNNTE IN DER ZUKUNFT ENTFERNT WERDEN. DIE AKTIVIERUNG DIESER OPTION KANN DIE LEISTUNG REDUZIEREN - STATT SIE ZU ERHÖHEN - WENN SIE UNGEEIGNET BENUTZT WIRD!

automatisches Wiedereinplanungsfenster (Auto-Rescheduling Window)

Format: **auto_rescheduling_window=<seconds>**

Beispiel: **auto_rescheduling_window=180**

Diese Option legt das Zeit-"Fenster" fest (in Sekunden), auf das Nagios blickt, wenn es automatisch Prüfungen erneut einplant. Nur Host- und Service-Prüfungen, die in den nächsten X Sekunden (festgelegt durch diese Variable) stattfinden, werden erneut eingeplant. Diese Option ist nur wirksam, wenn die auto_reschedule_checks-Option aktiviert ist. Standard ist 180 Sekunden (3 Minuten).

WARNING: DIES IST EIN EXPERIMENTELLES FEATURE UND KÖNNTE IN DER ZUKUNFT ENTFERNT WERDEN. DIE AKTIVIERUNG DIESER OPTION KANN DIE LEISTUNG REDUZIEREN - STATT SIE ZU ERHÖHEN - WENN SIE UNGEEIGNET GENUTZT WIRD!

Option aggressive Host-Prüfung (Aggressive Host Checking Option)

Format: **use_aggressive_host_checking=<0/1>**

Beispiel: **use_aggressive_host_checking=0**

Nagios versucht, schlau zu sein, wie und wann es den Status von Hosts prüft. Im Allgemeinen wird die Deaktivierung dieser Option Nagios dazu veranlassen, einige schlauere Entscheidungen zu treffen und Hosts ein bisschen schneller zu prüfen. Die Aktivierung dieser Option wird den Zeitaufwand zur Prüfung von Hosts erhöhen, aber es mag die Zuverlässigkeit ein wenig steigern. Solange Sie keine Probleme damit haben, dass Nagios die Erholung eines Hosts nicht korrekt erkennt, würde ich empfehlen, diese Option **nicht** zu aktivieren.

- 0 = keine aggressive Host-Prüfung benutzen (Default)
- 1 = aggressive Host-Prüfung benutzen

Option passive Host-Prüfung übersetzen (Translate Passive Host Checks Option)

Format: **translate_passive_host_checks=<0/1>**

Beispiel: **translate_passive_host_checks=1**

Diese Option legt fest, ob Nagios DOWN/UNREACHABLE-Ergebnisse von passiven Host-Prüfungen in ihre "korrekten" Zustände vom Standpunkt der lokalen Nagios-Instanz aus übersetzt. Dies kann sehr nützlich in verteilten und Failover-Umgebungen sein. Mehr Informationen zur Übersetzung von passiven Prüfergebnissen finden Sie hier.

- 0 = Prüfübersetzung deaktivieren (Default)
- 1 = Prüfübersetzung aktivieren

Option passive Host-Prüfungen sind SOFT (Passive Host Checks Are SOFT Option)

Format: **passive_host_checks_are_soft=<0/1>**

Beispiel: **passive_host_checks_are_soft=1**

Diese Option legt fest, ob Nagios passive Host-Prüfungen als HARD- oder SOFT-Zustände behandelt. Per Default wird ein passives Prüfergebnis einen Host in einen HARD-Status setzen. Sie können dieses Verhalten durch aktivieren dieser Option verändern.

- 0 = passive Host-Prüfungen sind HARD (Default)
- 1 = passive Host-Prüfungen sind SOFT

Option vorausschauende Host-Abhängigkeitsprüfung (Predictive Host Dependency Checks Option)

Format: **enable_predictive_host_dependency_checks=<0/1>**

Beispiel: **enable_predictive_host_dependency_checks=1**

Diese Option legt fest, ob Nagios vorausschauende Prüfungen von Hosts ausführen soll, von denen andere abhängig sind (wie in Host-Abhängigkeiten definiert) für einen bestimmten Host, dessen Zustand wechselt. Vorausschauende Prüfungen helfen dabei, die Abhängigkeitslogik so genau wie

möglich zu machen. Mehr Informationen darüber, wie vorausschauende Prüfungen arbeiten, finden Sie hier.

- 0 = vorausschauende Prüfungen deaktivieren
- 1 = vorausschauende Prüfungen aktivieren (Default)

Option vorausschauende Service-Abhängigkeitsprüfung (Predictive Service Dependency Checks Option)

Format: **enable_predictive_service_dependency_checks=<0/1>**

Beispiel: **enable_predictive_service_dependency_checks=1**

Diese Option legt fest, ob Nagios vorausschauende Prüfungen von Services ausführen soll, von denen andere abhängig sind (wie in Service-Abhängigkeiten definiert) für einen bestimmten Service, dessen Zustand wechselt. Vorausschauende Prüfungen helfen dabei, die Abhängigkeitslogik so genau wie möglich zu machen. Mehr Informationen darüber, wie vorausschauende Prüfungen arbeiten, finden Sie hier.

- 0 = vorausschauende Prüfungen deaktivieren
- 1 = vorausschauende Prüfungen aktivieren (Default)

Horizont für zwischengespeicherte Host-Prüfungen (Cached Host Check Horizon)

Format: **cached_host_check_horizon=<seconds>**

Beispiel: **cached_host_check_horizon=15**

Diese Option legt die maximale Zeit (in Sekunden) fest, die der Zustand einer vorherigen Host-Prüfung als aktuell angesehen wird. Zwischengespeicherte Host-Zustände (von Host-Prüfungen, die aktueller sind als die in diesem Wert angegebene Zeit) können die Leistung von Host-Prüfungen immens steigern. Ein zu hoher Wert für diese Option kann in (vorübergehend) ungenauen Host-Zuständen resultieren, während ein niedriger Wert zu einem Leistungseinbruch bei Host-Prüfungen führen kann. Benutzen Sie einen Wert von 0, wenn Sie die Zwischenspeicherung von Host-Prüfungen deaktivieren wollen. Mehr Informationen zu zwischengespeicherten Prüfungen finden Sie hier.

Horizont für zwischengespeicherte Service-Prüfungen (Cached Service Check Horizon)

Format: **cached_service_check_horizon=<seconds>**

Beispiel: **cached_service_check_horizon=15**

Diese Option legt die maximale Zeit (in Sekunden) fest, die der Zustand einer vorherigen Service-Prüfung als aktuell angesehen wird. Zwischengespeicherte Service-Zustände (von Service-Prüfungen, die aktueller sind als die in diesem Wert angegebene Zeit) können die Leistung von Service-Prüfungen steigern, wenn eine Menge von Service-Abhängigkeiten benutzt werden. Ein zu hoher Wert für diese Option kann in (vorübergehend) ungenauen Service-Zuständen resultieren, während ein niedriger Wert zu einem Leistungseinbruch bei Service-Prüfungen führen kann. Benutzen Sie einen Wert von 0, wenn Sie die Zwischenspeicherung von Service-Prüfungen deaktivieren wollen. Mehr Informationen zu zwischengespeicherten Prüfungen finden Sie hier.

Option Verbesserungen für große Installationen (Large Installation Tweaks Option)

Format: **use_large_installation_tweaks=<0/1>**

Beispiel: **use_large_installation_tweaks=0**

Diese Option legt fest, ob der Nagios-Daemon verschiedene Abkürzungen nimmt, um die Leistung zu steigern. Diese Abkürzungen resultieren im Verlust einiger Features, aber große Installationen werden wahrscheinlich einen hohen Nutzen davon haben. Mehr Informationen, welche Optimierungen vorgenommen werden, wenn Sie diese Option aktivieren, finden Sie hier.

- 0 = keine Verbesserungen verwenden (Default)
- 1 = Verbesserungen verwenden

Kindprozess-Speicher-Option (Child Process Memory Option)

Format: **free_child_process_memory=<0/1>**

Beispiel: **free_child_process_memory=0**

Diese Option legt fest, ob Nagios Speicher in Kindprozessen freigibt, wenn sie vom Hauptprozess ge"fork()"ed werden. Per Default gibt Nagios den Speicher frei. Wenn allerdings die use_large_installation_tweaks-Option aktiviert ist, wird der Speicher nicht freigegeben. Durch Definition dieser Option in Ihrer Konfigurationsdatei sind Sie in der Lage, das Verhalten einzustellen, das Sie möchten.

- 0 = Speicher nicht freigeben
- 1 = Speicher freigeben

Kindprozesse zweimal "fork()"en

Format: **child_processes_fork_twice=<0/1>**

Beispiel: **child_processes_fork_twice=0**

Diese Option legt fest, ob Nagios Kindprozesse zweimal "fork()"ed, wenn es Host- und Service-Prüfungen ausführt. Per Default "fork()"ed Nagios zweimal. Wenn allerdings die use_large_installation_tweaks-Option aktiviert ist, "fork()"ed Nagios nur einmal. Durch Definition dieser Option in Ihrer Konfigurationsdatei sind Sie in der Lage, das Verhalten einzustellen, das Sie möchten.

- 0 = nur einmal "fork()"en
- 1 = zweimal "fork()"en

Umgebungsmakros-Option

Format: **enable_environment_macros=<0/1>**

Beispiel: **enable_environment_macros=0**

Diese Option legt fest, ob Nagios alle Standard-Makros als Umgebungsvariablen für Prüfungen, Benachrichtigungen, Eventhandler usw. zur Verfügung stellt. In großen Installationen kann dies problematisch sein, weil es zusätzlichen Speicher (und wichtiger) mehr CPU benötigt, um die Werte aller Makros zu berechnen und sie der Umgebung zur Verfügung zu stellen.

- 0 = Makros nicht als Umgebungsvariablen verfügbar machen
- 1 = Makros als Umgebungsvariablen verfügbar machen (Default)

Flattererkennungsoption

Format: **enable_flap_detection=<0/1>**

Beispiel: **enable_flap_detection=0**

Diese Option legt fest, ob Nagios versucht festzustellen, ob Hosts und Services "flattern". Flattern tritt auf, wenn ein Host oder Service zu schnell zwischen verschiedenen Zuständen wechselt, was in einem Bombardement von Benachrichtigungen resultiert. Wenn Nagios erkennt, dass ein Host oder Service flattert, wird es vorübergehend Benachrichtigungen für diesen Host/Service unterdrücken, bis das Flattern endet. Flattererkennung ist sehr experimentell zu diesem Zeitpunkt, also benutzen Sie diese Option mit Vorsicht! Mehr Informationen dazu, wie Flattererkennung und Behandlung funktionieren, finden Sie hier. Anmerkung: Wenn Sie Statusaufbewahrung aktiviert haben, wird Nagios diese Einstellung ignorieren, wenn es (erneut) startet und die letzte bekannte Einstellung dieser Option nutzen (wie sie in der Statusaufbewahrungsdatei abgelegt ist), *es sei denn*, Sie haben die use_retained_program_state-Option deaktiviert. Wenn Sie diese Option ändern möchten, während die Statusaufbewahrung aktiviert ist (und die Option use_retained_program_state aktiviert ist), müssen Sie den entsprechenden externen Befehl benutzen oder sie über das Web-Interface ändern.

- 0 = Flattererkennung deaktivieren (Default)
- 1 = Flattererkennung aktivieren

niedriger Service-Flatterschwellwert (Low Service Flap Threshold)

Format: **low_service_flap_threshold=<percent>**

Beispiel: **low_service_flap_threshold=25.0**

Diese Option wird benutzt, um den niedrigen Schwellwert für die Erkennung von Service-Flattern zu setzen. Mehr Informationen dazu, wie Flattererkennung und Behandlung funktionieren (und wie diese Option Dinge beeinflusst), finden Sie hier.

hoher Service-Flatterschwellwert (High Service Flap Threshold)

Format: **high_service_flap_threshold=<percent>**

Beispiel: **high_service_flap_threshold=50.0**

Diese Option wird benutzt, um den hohen Schwellwert für die Erkennung von Service-Flattern zu setzen. Mehr Informationen dazu, wie Flattererkennung und Behandlung funktionieren (und wie diese Option Dinge beeinflusst), finden Sie hier.

niedriger Host-Flatterschwellwert (Low Host Flap Threshold)

Format: **low_host_flap_threshold=<percent>**

Beispiel: **low_host_flap_threshold=25.0**

Diese Option wird benutzt, um den niedrigen Schwellwert für die Erkennung von Host-Flattern zu setzen. Mehr Informationen dazu, wie Flattererkennung und Behandlung funktionieren (und wie diese Option Dinge beeinflusst), finden Sie hier.

hoher Host-Flatterschwellwert (High Host Flap Threshold)

Format: **high_host_flap_threshold=<percent>**

Beispiel: **high_host_flap_threshold=50.0**

Diese Option wird benutzt, um den hohen Schwellwert für die Erkennung von Host-Flattern zu setzen. Mehr Informationen dazu, wie Flattererkennung und Behandlung funktionieren (und wie diese Option Dinge beeinflusst), finden Sie hier.

Soft-Statusabhängigkeitsoption (Soft State Dependencies Option)

Format: **soft_state_dependencies=<0/1>**

Beispiel: **soft_state_dependencies=0**

Diese Option legt fest, ob Nagios Soft-Statusinformationen benutzen soll, um Host- und Serviceabhängigkeiten zu prüfen. Normalerweise wird Nagios nur den letzten Hard-Status des Hosts oder Service verwenden, wenn Abhängigkeiten geprüft werden. Wenn Sie möchten, dass der letzte Zustand (unabhängig davon, ob es ein Soft- oder Hard-Zustandstyp ist), dann aktivieren Sie diese Option.

- 0 = keine Soft-Status-Abhängigkeiten benutzen (Default)
- 1 = Soft-Status-Abhängigkeiten benutzen

Service-Prüfungs-Zeitüberschreitung (Service Check Timeout)

Format: **service_check_timeout=<seconds>**

Beispiel: **service_check_timeout=60**

Dies ist die maximale Zahl von Sekunden, die Service-Prüfungen laufen dürfen. Wenn Prüfungen diese Grenze überschreiten, werden sie abgebrochen (killed) und ein CRITICAL-Zustand wird zurückgeliefert. Außerdem wird ein Fehler protokolliert.

Es gibt oft eine weitverbreitete Verwirrung, was diese Option wirklich tut. Es ist als allerletzter Versuch gedacht, wenn Plugins sich "daneben benehmen" und nicht innerhalb einer bestimmten Zeit enden. Sie sollte auf einen hohen Wert (z.B. 60 Sekunden oder mehr) gesetzt werden, so dass jede Service-Prüfung normalerweise innerhalb dieser Zeit beendet ist. Wenn eine Service-Prüfung länger läuft, dann wird Nagios diese Prüfung abbrechen, weil es denkt, dass es sich um einen außer Kontrolle geratenen Prozess

handelt.

Host-Prüfungs-Zeitüberschreitung (Host Check Timeout)

Format: **host_check_timeout=<seconds>**

Beispiel: **host_check_timeout=60**

Dies ist die maximale Zahl von Sekunden, die Host-Prüfungen laufen dürfen. Wenn Prüfungen diese Grenze überschreiten, werden sie abgebrochen (killed), ein CRITICAL-Zustand wird zurückgeliefert und der Host als "DOWN" angesehen. Außerdem wird ein Fehler protokolliert.

Es gibt oft eine weitverbreitete Verwirrung, was diese Option wirklich tut. Es ist als allerletzter Versuch gedacht, wenn Plugins sich "daneben benehmen" und nicht innerhalb einer bestimmten Zeit enden. Sie sollte auf einen hohen Wert (z.B. 60 Sekunden oder mehr) gesetzt werden, so dass jede Host-Prüfung normalerweise innerhalb dieser Zeit beendet ist. Wenn eine Host-Prüfung länger läuft, dann wird Nagios diese Prüfung abbrechen, weil es denkt, dass es sich um einen außer Kontrolle geratenen Prozess handelt.

Eventhandler-Zeitüberschreitung

Format: **event_handler_timeout=<seconds>**

Beispiel: **event_handler_timeout=60**

Dies ist die maximale Zahl von Sekunden, die Eventhandler laufen dürfen. Wenn ein Eventhandler diese Grenze überschreitet, wird er abgebrochen (killed) und eine Warnung protokolliert.

Es gibt oft eine weitverbreitete Verwirrung, was diese Option wirklich tut. Es ist als allerletzter Versuch gedacht, wenn Befehle sich "daneben benehmen" und nicht innerhalb einer bestimmten Zeit enden. Sie sollte auf einen hohen Wert (z.B. 60 Sekunden oder mehr) gesetzt werden, so dass jeder Eventhandler normalerweise innerhalb dieser Zeit beendet ist. Wenn ein Eventhandler länger läuft, dann wird Nagios diesen Eventhandler abbrechen, weil es denkt, dass es sich um einen außer Kontrolle geratenen Prozess handelt.

Benachrichtigungs-Zeitüberschreitung

Format: **notification_timeout=<seconds>**

Beispiel: **notification_timeout=60**

Dies ist die maximale Zahl von Sekunden, die Benachrichtigungsbefehle laufen dürfen. Wenn ein Benachrichtigungsbefehl diese Grenze überschreitet, wird er abgebrochen (killed) und eine Warnung protokolliert.

Es gibt oft eine weitverbreitete Verwirrung, was diese Option wirklich tut. Es ist als allerletzter Versuch gedacht, wenn Befehle sich "daneben benehmen" und nicht innerhalb einer bestimmten Zeit enden. Sie sollte auf einen hohen Wert (z.B. 60 Sekunden oder mehr) gesetzt werden, so dass jeder Benachrichtigungsbefehl normalerweise innerhalb dieser Zeit beendet ist. Wenn ein Benachrichtigungsbefehl länger läft, dann wird Nagios diesen Benachrichtigungsbefehl abbrechen, weil es denkt, dass es sich um einen außer Kontrolle geratenen Prozess handelt.

Zwangsverfolgungs-Service-Prozessor-Zeitüberschreitung (Obsessive Compulsive Service Processor Timeout)

Format: **ocsp_timeout=<seconds>**

Beispiel: **ocsp_timeout=5**

Dies ist die maximale Zahl von Sekunden, die ein Zwangsverfolgungs-Service-Prozessor-Befehl (obsessive compulsive service processor command) laufen darf. Wenn ein Befehl diese Grenze überschreitet, wird er abgebrochen (killed) und eine Warnung protokolliert.

Zwangsverfolgungs-Host-Prozessor-Zeitüberschreitung (Obsessive Compulsive Host Processor Timeout)

Format: **ochp_timeout=<seconds>**

Beispiel: **ochp_timeout=5**

Dies ist die maximale Zahl von Sekunden, die ein Zwangsverfolgungs-Host-Prozessor-Befehl (obsessive compulsive host processor command) laufen darf. Wenn ein Befehl diese Grenze überschreitet, wird er abgebrochen (killed) und eine Warnung protokolliert.

Performance-Daten-Prozessorbefehls-Zeitüberschreitung (Performance Data Processor Command Timeout)

Format: **perfdata_timeout=<seconds>**

Beispiel: **perfdata_timeout=5**

Dies ist die maximale Zahl von Sekunden, die ein Host-Performance-Daten-Prozessorbefehl (host performance data processor command) oder Service-Performance-Daten-Prozessorbefehl (service performance data processor command) laufen darf. Wenn ein Befehl diese Grenze überschreitet, wird er abgebrochen (killed) und eine Warnung protokolliert.

Verfolgung-von-Services-Option (Obsess Over Services Option)

Format: **obsess_over_services=<0/1>**

Beispiel: **obsess_over_services=1**

Dieser Wert legt fest, ob Nagios Service-Prüfergebnisse "verfolgt" (obsess) und den Zwangsverfolgungs-Service-Prozessorbefehl ausführt, den Sie angeben. Ich weiß - komischer Name, aber das ist alles, was mir eingefallen ist. Diese Option ist nützlich, um verteilte Überwachung durchzuführen. Wenn Sie keine verteilte Überwachung machen, dann aktivieren Sie diese Option nicht.

- 0 = Services nicht verfolgen (Default)
- 1 = Services verfolgen

Zwangsverfolgungs-Service-Prozessorbefehl (Obsessive Compulsive Service Processor Command)

Format: **ocsp_command=<command>**

Beispiel: **ocsp_command=obsessive_service_handler**

Diese Option erlaubt Ihnen einen Befehl anzugeben, der nach *jeder* Service-Prüfung ausgeführt wird, was bei verteilter Überwachung nützlich sein kann. Dieser Befehl wird nach Eventhandler- oder Benachrichtigungs-Befehlen ausgeführt. Das *command*-Argument ist der Kurzname einer command-Definition, die Sie in Ihrer Objektkonfigurationsdatei angeben. Die maximale Zeit, die dieser Befehl laufen darf, wird mit der ocsp_timeout-Option kontrolliert. Mehr Informationen zu verteilter Überwachung finden Sie hier. Dieser Befehl wird nur ausgeführt, wenn die obsess_over_services-Option global aktiviert ist und wenn die *obsess_over_service*-Direktive in der Service-Definition aktiviert ist.

Verfolgung-von-Hosts-Option (Obsess Over Hosts Option)

Format: **obsess_over_hosts=<0/1>**

Beispiel: **obsess_over_hosts=1**

Dieser Wert legt fest, ob Nagios Host-Prüfergebnisse "verfolgt" (obsess) und den Zwangsverfolgungs-Host-Prozessorbefehl ausführt, den Sie angeben. Ich weiß - komischer Name, aber das ist alles, was mir eingefallen ist. Diese Option ist nützlich, um verteilte Überwachung durchzuführen. Wenn Sie keine verteilte Überwachung machen, dann aktivieren Sie diese Option nicht.

- 0 = Hosts nicht verfolgen (Default)
- 1 = Hosts verfolgen

Zwangsverfolgungs-Host-Prozessorbefehl (Obsessive Compulsive Host Processor Command)

Format: **ochp_command=<command>**

Beispiel: **ochp_command=obsessive_host_handler**

Diese Option erlaubt Ihnen einen Befehl anzugeben, der nach *jeder* Host-Prüfung ausgeführt wird, was bei verteilter Überwachung nützlich sein kann. Dieser Befehl wird nach Eventhandler- oder Benachrichtigungs-Befehlen ausgeführt. Das *command*-Argument ist der Kurzname einer command-Definition, die Sie in Ihrer Objektkonfigurationsdatei angeben. Die maximale Zeit, die dieser Befehl laufen darf, wird mit der ochp_timeout-Option kontrolliert. Mehr Informationen zu verteilter Überwachung finden Sie hier. Dieser Befehl wird nur ausgeführt, wenn die obsess_over_hosts-Option global aktiviert ist und wenn die *obsess_over_hosts*-Direktive in der Host-Definition aktiviert ist.

Performance-Daten-Verarbeitungsoption (Performance Data Processing Option)

Format: **process_performance_data=<0/1>**

Beispiel: **process_performance_data=1**

Dieser Wert legt fest, ob Nagios Performance-Daten von Host- und Service-Prüfungen verarbeitet.

- 0 = keine Performance-Daten verarbeiten (Default)
- 1 = Performance-Daten verarbeiten

Host-Performance-Daten-Verarbeitungsbefehl (Host Performance Data Processing Command)

Format: **host_perfdata_command=<command>**

Beispiel: **host_perfdata_command=process-host-perfdata**

Diese Option erlaubt es Ihnen, einen Befehl anzugeben, der nach *jeder* Host-Prüfung ausgeführt wird, um Host-Performance-Daten zu verarbeiten, die von der Prüfung zurückgeliefert worden sein könnten. Das *command*-Argument ist der Kurzname einer command-Definition, die Sie in Ihrer Objektkonfigurationsdatei angeben. Dieser Befehl wird nur ausgeführt, wenn die process_performance_data-Option global aktiviert ist und wenn die *process_perf_data*-Direktive in der Host-Definition aktiviert ist.

Service-Performance-Daten-Verarbeitungsbefehl (Service Performance Data Processing Command)

Format: **service_perfdata_command=<command>**

Beispiel: **service_perfdata_command=process-service-perfdata**

Diese Option erlaubt es Ihnen, einen Befehl anzugeben, der nach *jeder* Service-Prüfung ausgeführt wird, um Service-Performance-Daten zu verarbeiten, die von der Prüfung zurückgeliefert worden sein könnten. Das *command*-Argument ist der Kurzname einer command-Definition, die Sie in Ihrer Objektkonfigurationsdatei angeben. Dieser Befehl wird nur ausgeführt, wenn die process_performance_data-Option global aktiviert ist und wenn die *process_perf_data*-Direktive in der Service-Definition aktiviert ist.

Host-Performance-Daten-Datei (Host Performance Data File)

Format: **host_perfdata_file=<file_name>**

Beispiel: **host_perfdata_file=/usr/local/nagios/var/host-perfdata.dat**

Diese Option erlaubt es Ihnen, eine Datei anzugeben, in die nach jeder Host-Prüfung Performance-Daten geschrieben werden. Daten werden in die Performance-Datei geschrieben, wie es in der host_perfdata_file_template-Option angegeben ist. Performance-Daten werden nur in diese Datei geschrieben, wenn die process_performance_data-Option global aktiviert ist und wenn die *process_perf_data*-Direktive in der Host-Definition aktiviert ist.

Service-Performance-Daten-Datei (Service Performance Data File)

Format: **service_perfdata_file=<file_name>**

Beispiel: **service_perfdata_file=/usr/local/nagios/var/service-perfdata.dat**

Diese Option erlaubt es Ihnen, eine Datei anzugeben, in die nach jeder Service-Prüfung Performance-Daten geschrieben werden. Daten werden in die Performance-Datei geschrieben, wie es in der service_perfdata_file_template-Option angegeben ist. Performance-Daten werden nur in diese Datei geschrieben, wenn die process_performance_data-Option global aktiviert ist und wenn die *process_perf_data*-Direktive in der Service-Definition aktiviert ist.

Host-Performance-Daten-Dateivorlage (Host Performance Data File Template)

Format: **host_perfdata_file_template=<template>**

Beispiel: **host_perfdata_file_template=[HOSTPERFDATA]\t$TIMET$\t$HOSTNAME$\t$HOSTEXECUTIONTIME$\t$HOSTOUTPUT$\t$HOSTPERFDATA$**

Diese Option legt fest, welche (und wie) Daten in die Host-Performancedaten-Datei geschrieben werden. Diese Vorlage kann Makros, Sonderzeichen (\t für Tabulator, \r für Wagenrücklauf (carriage return), \n für Zeilenvorschub (newline)) und reinen Text enthalten. Nach jedem Schreibvorgang wird ein Zeilenvorschub an die Performancedaten-Datei angehängt.

Service-Performance-Daten-Dateivorlage (Service Performance Data File Template)

Format: **service_perfdata_file_template=<template>**

Beispiel: **service_perfdata_file_template=[SERVICEPERFDATA]\t$TIMET$\t$HOSTNAME$\t$SERVICEDESC$\t$SERVICEEXECUTIONTIME$\t$SERVICELATENCY$\t$SERVICEOUTPUT$\t$SERVICEPERFDATA$**

Diese Option legt fest, welche (und wie) Daten in die Service-Performancedaten-Datei geschrieben werden. Diese Vorlage kann Makros, Sonderzeichen (\t für Tabulator, \r für Wagenrücklauf (carriage return), \n für Zeilenvorschub (newline)) und reinen Text enthalten. Nach jedem Schreibvorgang wird ein Zeilenvorschub an die Performancedaten-Datei angehängt.

Host-Performance-Daten-Dateimodus (Host Performance Data File Mode)

Format: **host_perfdata_file_mode=<mode>**

Beispiel: **host_perfdata_file_mode=a**

Diese Option legt fest, wie die Host-Performance-Datendatei geöffnet wird. Solange die Datei keine "named pipe" ist, werden Sie diese wahrscheinlich im append-Modus (anhängen) öffnen wollen.

- a = Datei im append-Modus öffnen (Default)
- w = Datei im Write-Modus öffnen
- p = Datei im nicht-blockierenden Schreib-/Lesemodus öffnen (nützlich, wenn man in Pipes schreibt)

Service-Performance-Daten-Dateimodus (Service Performance Data File Mode)

Format: **service_perfdata_file_mode=<mode>**

Beispiel: **service_perfdata_file_mode=a**

Diese Option legt fest, wie die Service-Performance-Datendatei geöffnet wird. Solange die Datei keine "named pipe" ist, werden Sie diese wahrscheinlich im append-Modus (anhängen) öffnen wollen.

- a = Datei im append-Modus öffnen (Default)
- w = Datei im Write-Modus öffnen
- p = Datei im nicht-blockierenden Schreib-/Lesemodus öffnen (nützlich, wenn man in Pipes schreibt)

Host-Performance-Daten-Dateiverarbeitungsintervall (Host Performance Data File Processing Interval)

Format: **host_perfdata_file_processing_interval=<seconds>**

Beispiel: **host_perfdata_file_processing_interval=0**

Diese Option erlaubt es Ihnen, das Intervall (in Sekunden) anzugeben, in dem die Host-Performance-Daten-Datei mit dem Host-Performance-Daten-Dateiverarbeitungsbefehl verarbeitet wird. Ein Wert von 0 gibt an, dass die Performance-Daten-Datei nicht in regelmäßigen Intervallen verarbeiten werden soll.

Service-Performance-Daten-Dateiverarbeitungsintervall (Service Performance Data File Processing Interval)

Format: **service_perfdata_file_processing_interval=<seconds>**

Beispiel: **service_perfdata_file_processing_interval=0**

Diese Option erlaubt es Ihnen, das Intervall (in Sekunden) anzugeben, in dem die Service-Performance-Daten-Datei mit dem Service-Performance-Daten-Dateiverarbeitungsbefehl verarbeitet wird. Ein Wert von 0 gibt an, dass die Performance-Daten-Datei nicht in regelmäßigen Intervallen verarbeiten werden soll.

Host-Performance-Daten-Dateiverarbeitungsbefehl (Host Performance Data File Processing Command)

Format: **host_perfdata_file_processing_command=<command>**

Beispiel: **host_perfdata_file_processing_command=process-host-perfdata-file**

Diese Option erlaubt es Ihnen, den Befehl anzugeben, der ausgeführt werden soll, um die Host-Performance-Daten-Datei zu verarbeiten. Das *command*-Argument ist der Kurzname einer command-Definition, die Sie in Ihrer Objektkonfigurationsdatei angeben. Das Intervall, in dem dieser Befehl ausgeführt wird, ist durch die host_perfdata_file_processing_interval-Direktive festgelegt.

Service-Performance-Daten-Dateiverarbeitungsbefehl (Service Performance Data File Processing Command)

Format: **service_perfdata_file_processing_command=<command>**

Beispiel: **service_perfdata_file_processing_command=process-service-perfdata-file**

Diese Option erlaubt es Ihnen, den Befehl anzugeben, der ausgeführt werden soll, um die Service-Performance-Daten-Datei zu verarbeiten. Das *command*-Argument ist der Kurzname einer command-Definition, die Sie in Ihrer Objektkonfigurationsdatei angeben. Das Intervall, in dem dieser Befehl ausgeführt wird, ist durch die service_perfdata_file_processing_interval-Direktive festgelegt.

verwaiste Service-Prüfungsoption (Orphaned Service Check Option)

Format: **check_for_orphaned_services=<0/1>**

Beispiel: **check_for_orphaned_services=1**

Diese Option erlaubt es Ihnen, Prüfungen auf verwaiste Service-Prüfungen zu aktivieren oder zu deaktivieren. Verwaiste Service-Prüfungen sind Prüfungen, die ausgeführt und aus der Ereigniswarteschlange entfernt wurden, aber während langer Zeit keine Ergebnisse geliefert haben. Weil keine Ergebnisse für den Service zurückgeliefert wurden, wird er nicht erneut in der Ereigniswarteschlange eingeplant. Das kann dazu führen, dass Service-Prüfungen nicht mehr ausgeführt werden. Normalerweise passiert das sehr selten - es kann dann auftreten, wenn ein externer Benutzer oder Prozess den Prozess abgebrochen (killed) hat, der benutzt wurde, um eine Service-Prüfung auszuführen. Wenn diese Option aktiviert ist und Nagios feststellt, dass eine bestimmte Service-Prüfung kein Ergebnis geliefert hat, dann wird es einen Fehler protokollieren und die Service-Prüfung erneut einplanen. Wenn Sie feststellen, dass Service-Prüfungen anscheinend nie erneut eingeplant werden, dann aktivieren Sie diese Option und schauen Sie nach Protokollmeldungen zu verwaisten Services.

- 0 = nicht auf verwaiste Service-Prüfungen prüfen
- 1 = auf verwaiste Service-Prüfungen prüfen (Default)

verwaiste Host-Prüfungsoption (Orphaned Host Check Option)

Format: **check_for_orphaned_hosts=<0/1>**

Beispiel: **check_for_orphaned_hosts=1**

Diese Option erlaubt es Ihnen, Prüfungen auf verwaiste Host-Prüfungen zu aktivieren oder zu deaktivieren. Verwaiste Host-Prüfungen sind Prüfungen, die ausgeführt und aus der Ereigniswarteschlange entfernt wurden, aber während langer Zeit keine Ergebnisse geliefert haben. Weil keine Ergebnisse für den Host zurückgeliefert wurden, wird er nicht erneut in der Ereigniswarteschlange eingeplant. Das kann dazu führen, dass Host-Prüfungen nicht mehr ausgeführt werden. Normalerweise passiert das sehr selten - es kann dann auftreten, wenn ein externer Benutzer oder Prozess den Prozess abgebrochen (killed) hat, der benutzt wurde, um eine Host-Prüfung auszuführen. Wenn diese Option aktiviert ist und Nagios feststellt, dass eine bestimmte Host-Prüfung kein Ergebnis geliefert hat, dann wird es einen Fehler protokollieren und die Host-Prüfung erneut einplanen. Wenn Sie feststellen, dass Host-Prüfungen anscheinend nie erneut eingeplant werden, dann aktivieren Sie diese Option und schauen Sie nach Protokollmeldungen zu verwaisten Hosts.

- 0 = nicht auf verwaiste Host-Prüfungen prüfen
- 1 = auf verwaiste Host-Prüfungen prüfen (Default)

Service-Frischeprüfungsoption (Service Freshness Checking Option)

Format: **check_service_freshness=<0/1>**

Beispiel: **check_service_freshness=0**

Diese Option legt fest, ob Nagios regelmäßig die "Frische" (freshness) von Service-Prüfungen prüft. Das Aktivieren dieser Option ist nützlich, um sicherzustellen, dass passive Service-Prüfungen rechtzeitig empfangen werden. Mehr Informationen zur Frische-Prüfung finden Sie hier.

- 0 = Service-Frische nicht prüfen
- 1 = Service-Frische prüfen (Default)

Service-Frische-Prüfintervall (Service Freshness Check Interval)

Format: **service_freshness_check_interval=<seconds>**

Beispiel: **service_freshness_check_interval=60**

Diese Einstellung legt fest, wie oft (in Sekunden) Nagios regelmäßig die "Frische" (freshness) von Service-Prüfergebnissen prüfen wird. Wenn Sie die Service-Frische-Prüfung (mit der check_service_freshness-Option) deaktiviert haben, dann hat diese Option keine Auswirkungen. Mehr Informationen zur Frische-Prüfung finden Sie hier.

Host-Frischeprüfungsoption (Host Freshness Checking Option)

Format: **check_host_freshness=<0/1>**

Beispiel: **check_host_freshness=0**

Diese Option legt fest, ob Nagios regelmäßig die "Frische" (freshness) von Host-Prüfungen prüft. Das Aktivieren dieser Option ist nützlich, um sicherzustellen, dass passive Host-Prüfungen rechtzeitig empfangen werden. Mehr Informationen zur Frische-Prüfung finden Sie hier.

- 0 = Host-Frische nicht prüfen
- 1 = Host-Frische prüfen (Default)

Host-Frische-Prüfintervall (Host Freshness Check Interval)

Format: **host_freshness_check_interval=<seconds>**

Beispiel: **host_freshness_check_interval=60**

Diese Einstellung legt fest, wie oft (in Sekunden) Nagios regelmäßig die "Frische" (freshness) von Host-Prüfergebnissen prüfen wird. Wenn Sie die Host-Frische-Prüfung (mit der check_host_freshness-Option) deaktiviert haben, dann hat diese Option keine Auswirkungen. Mehr Informationen zur Frische-Prüfung finden Sie hier.

zusätzliche Frische-Latenzschwellwert-Option (Additional Freshness Threshold Latency Option)

Format: **additional_freshness_latency=<#>**

Beispiel: **additional_freshness_latency=15**

Diese Option legt die Anzahl von Sekunden fest, die Nagios zu jedem Host- oder Service-Frischeschwellwert hinzurechnet, den es automatisch berechnet (d.h., die nicht explizit durch den Benutzer angegeben wurden). Mehr Informationen zur Frische-Prüfung finden Sie hier.

eingebetteter-Perl-Interpreter-Option (Embedded Perl Interpreter Option)

Format: **enable_embedded_perl=<0/1>**

Beispiel: **enable_embedded_perl=1**

Diese Einstellung legt fest, ob der eingebettete Perl-Interpreter auf programmweiter Basis aktiviert ist. Nagios muss mit Unterstützung für eingebettetes Perl kompiliert sein, damit diese Option eine Auswirkung hat. Mehr Informationen zum eingebauten Perl-Interpreter finden Sie hier.

Option implizite Nutzung des eingebetteten Perl-Interpreters (Embedded Perl Implicit Use Option)

Format: **use_embedded_perl_implicitly=<0/1>**

Beispiel: **use_embedded_perl_implicitly=1**

Diese Einstellung legt fest, ob der eingebettete Perl-Interpreter für Perl-Plugins/Scripte benutzt werden soll, die ihn nicht explizit aktiviert/deaktiviert haben. Nagios muss mit Unterstützung für eingebettetes Perl kompiliert sein, damit diese Option eine Auswirkung hat. Mehr Informationen zum eingebauten Perl-Interpreter finden Sie hier.

Datumformat (Date Format)

Format: **date_format=<option>**

Beispiel: **date_format=us**

Diese Option erlaubt es Ihnen anzugeben, welche Art von Datums-/Zeitformat Nagios im Web-Interface und in den Datums-/Zeit-Makros benutzen soll. Mögliche Optionen (zusammen mit einer Beispielausgabe) umfassen u.a.:

Option	Ausgabeformat	Beispielausgabe
us	MM/DD/YYYY HH:MM:SS	06/30/2002 03:15:00
euro	DD/MM/YYYY HH:MM:SS	30/06/2002 03:15:00
iso8601	YYYY-MM-DD HH:MM:SS	2002-06-30 03:15:00
strict-iso8601	YYYY-MM-DDTHH:MM:SS	2002-06-30T03:15:00

Zeitzonen-Option (Timezone Option)

Format: **use_timezone=<tz>**

Beispiel: **use_timezone=US/Mountain**

Diese Option erlaubt es Ihnen, die Standard-Zeitzone zu überschreiben, in der die Nagios-Instanz läuft. Das ist nützlich, wenn Sie mehrere Nagios-Instanzen haben, die auf dem gleichen Server laufen, aber mit verschiedenen lokalen Zeiten verbunden sind. Wenn nichts angegeben ist, wird Nagios die Zeitzone des Rechners benutzen.

 Anmerkung: wenn Sie diese Option nutzen, um eine eigene Zeitzone anzugeben, müssen Sie auch die Apache-Konfigurationsdirektiven für die CGIs auf die Zeitzone ändern, die Sie haben möchten. Beispiel:

<Directory "/usr/local/nagios/sbin/">
SetEnv TZ "US/Mountain"
...
</Directory>

Illegale Zeichen für Objektnamen (Illegal Object Name Characters)

Format: **illegal_object_name_chars=<chars...>**

Beispiel: **illegal_object_name_chars='~!$%^&*"|'<>?,()=**

Diese Option erlaubt es Ihnen, illegale Zeichen anzugeben, die nicht in Host-Namen, Service-Beschreibungen oder Namen anderen Objektarten benutzt werden können. Nagios gestattet Ihnen, die meisten Zeichen in Objektdefinitionen zu benutzen, aber ich rate Ihnen, die im Beispiel gezeigten Zeichen nicht zu nutzen. Wenn Sie es dennoch tun, können Sie Probleme im Web-Interface, in Benachrichtigungsbefehlen usw. bekommen.

illegale Zeichen für Makroausgaben (Illegal Macro Output Characters)

Format: **illegal_macro_output_chars=<chars...>**

Beispiel: **illegal_macro_output_chars='~$^&"|'<>**

Diese Option erlaubt es Ihnen, illegale Zeichen anzugeben, die aus Makros entfernt werden, bevor diese in Benachrichtigungen, Eventhandlern und anderen Befehlen benutzt werden. Dies betrifft NICHT Makros, die in Service- oder Host-Prüfbefehlen benutzt werden. Sie können sich entscheiden, die Zeichen im Beispiel nicht zu entfernen, aber ich rate Ihnen, das nicht zu tun. Einige dieser Zeichen werden von der Shell interpretiert (z.B. der "Backtick") und können zu Sicherheitsproblemen führen. Die folgenden Makros werden von den Zeichen gereinigt, die Sie angeben:

$HOSTOUTPUT$, **$HOSTPERFDATA$**, **$HOSTACKAUTHOR$**, **$HOSTACKCOMMENT$**, **$SERVICEOUTPUT$**, **$SERVICEPERFDATA$**, **$SERVICEACKAUTHOR$**, and **$SERVICEACKCOMMENT$**

Option Anpassung regulärer Ausdrücke (Regular Expression Matching Option)

Format: **use_regexp_matching=<0/1>**

Beispiel: **use_regexp_matching=0**

Diese Option legt fest, ob verschiedene Direktiven in Ihren Objektdefinitionen als reguläre Ausdrücke verarbeitet werden. Mehr Informationen dazu, wie das funktioniert, finden Sie hier.

- 0 = keine angepassten regulären Ausdrücke benutzen (Default)
- 1 = angepasste reguläre Ausdrücke benutzen

Option wahre reguläre Ausdru;cksanpassung (True Regular Expression Matching Option)

Format: **use_true_regexp_matching=<0/1>**

Beispiel: **use_true_regexp_matching=0**

Wenn Sie reguläre Ausdrücke von verschiedenen Objektdirektiven mit der use_regexp_matching-Option aktiviert haben, dann wird diese Option festlegen, wann Objektdirektiven als reguläre Ausdrücke behandelt werden. Wenn diese Option deaktiviert ist (der Standard), dann werden Direktiven nur dann als reguläre Ausdrücke behandelt, wenn sie *****, **?**, **+** oder **\.** enthalten. Wenn diese Option aktiviert ist, werden alle entsprechenden Direktiven als reguläre Ausdrücke behandelt - seien Sie vorsichtig bei der Aktivierung! Mehr Informationen darüber, wie das funktioniert, finden Sie hier.

- 0 = keine Anpassung wahrer regul¨rer Ausdrücke benutzen (Default)
- 1 = Anpassung wahrer regul¨rer Ausdrücke benutzen

Administrator-e-Mail-Adresse (Administrator Email Address)

Format: **admin_email=<email_address>**

Beispiel: **admin_email=root@localhost.localdomain**

Dies ist die e-Mail-Adresse des Administrators der lokalen Maschine (d.h. die, auf der Nagios läuft). Dieser Wert kann mit Hilfe des **$ADMINEMAIL$**-Makros in Benachrichtigungsbefehlen benutzt werden.

Administrator-Pager (Administrator Pager)

Format: **admin_pager=<pager_number_or_pager_email_gateway>**

Beispiel: **admin_pager=pageroot@localhost.localdomain**

Dies ist die Pager-Nummer (oder die des Pager-e-Mail-Gateways) des Administrators der lokalen Maschine (d.h. die, auf der Nagios läuft). Die Pager-Nummer/Adresse kann mit Hilfe des **$ADMINPAGER$**-Makros in Benachrichtigungsbefehlen benutzt werden.

Ereignisvermittler-Optionen (Event Broker Options)

Format: **event_broker_options=<#>**

Beispiel: **event_broker_options=-1**

Diese Option kontrolliert, welche Daten an den Ereignisvermittler gesandt werden und damit an jedes geladene Ereignisvermittler-Modul. Dies ist ein fortgeschrittenes Feature. Falls Sie im Zweifel sind, vermitteln Sie entweder gar nichts (wenn Sie keine Ereignisvermittler-Module benutzen) oder alles (wenn Sie Ereignisvermittler-Module benutzen). Mögliche Werte sehen Sie nachfolgend.

- 0 = nichts vermitteln
- -1 = alles vermitteln
- # = sehen Sie sich die BROKER_*-Definitionen im Quellcode an (include/broker.h), um andere Werte zu ermitteln

Ereignisvermittler-Module (Event Broker Modules)

Format: **broker_module=<modulepath> [moduleargs]**

Beispiel: **broker_module=/usr/local/nagios/bin/ndomod.o cfg_file=/usr/local/nagios/etc/ndomod.cfg**

Diese wird benutzt, um ein Ereignisvermittler-Modul anzugeben, das beim Nagios-Start geladen werden soll. Benutzen Sie mehrere Direktiven, wenn Sie mehrere Module laden wollen. An das Modul zu übergebende Argumente werden durch ein Leerzeichen vom Pfad des Moduls getrennt.

!!! WARNUNG !!!

Überschreiben Sie KEINE Module, während sie von Nagios genutzt werden, oder Nagios wird mit einem SEGFAULT-Feuerwerk abstürzen. Dies ist ein Fehler/eine Begrenzung entweder in dlopen(), dem Kernel, und/oder dem Filesystem. Und vielleicht Nagios...

Der korrekte/sichere Weg, ein Modul zu aktualisieren, ist eine der folgenden Methoden:

1. stoppen Sie Nagios, ersetzen Sie das Modul und starten Sie Nagios erneut
2. während Nagios läuft... löschen Sie die originale Moduldatei, schieben Sie die neue Moduldatei an den richtigen Platz und starten Sie Nagios erneut

Debug-Datei (Debug File)

Format: **debug_file=<file_name>**

Beispiel: **debug_file=/usr/local/nagios/var/nagios.debug**

Diese Option legt fest, wohin Nagios Debugging-Informationen schreiben soll. Welche Informationen (falls überhaupt) geschrieben werden, wird durch die debug_level- und debug_verbosity-Optionen festgelegt. Sie können die Debug-Datei automatisch rotieren lassen, wenn sie eine bestimmte Größe überschreitet, die Sie über die max_debug_file_size-Option festlegen können.

Debug-Stufe (Debug Level)

Format: **debug_level=<#>**

Beispiel: **debug_level=24**

Diese Option legt fest, welche Arten von Informationen Nagios in das debug_file schreiben soll. Dieser Wert ist ein logisches ODER der nachfolgenden Werte:

- -1 = alles protokollieren
- 0 = nichts protokollieren (Default)
- 1 = Informationen zu Funktionsbeginn/Ende
- 2 = Konfigurationsinformationen
- 4 = Prozessinformationen
- 8 = geplante Ereignisinformationen
- 16 = Host-/Service-Prüfinformationen
- 32 = Benachrichtigungsinformationen
- 64 = Ereignisvermittlerinformationen

Debug-Ausführlichkeit (Debug Verbosity)

Format: **debug_verbosity=<#>**

Beispiel: **debug_verbosity=1**

Diese Option legt fest, wie viel Debugging-Informationen Nagios in das debug_file schreiben soll.

- 0 = grundlegende Informationen
- 1 = detailliertere Informationen (Default)
- 2 = sehr detaillierte Informationen

maximale Debug-Dateigröße (Maximum Debug File Size)

Format: **max_debug_file_size=<#>**

Beispiel: **max_debug_file_size=1000000**

Diese Option legt die maximale Größe (in Bytes) der Debug-Datei fest. Wenn die Datei die Größe überschreitet, dann wird ".old" als Erweiterung angehängt. Wenn bereits eine Datei mit der ".old"-Erweiterung existiert, wird diese gelöscht. Das stellt sicher, dass Ihr Plattenplatz nicht außer Kontrolle gerät, wenn Sie Nagios debuggen.

Nagios®

Überblick Objektkonfiguration

Hoch zu: Inhalt

Siehe auch: Konfigurationsüberblick, Objektdefinitionen

Was sind Objekte?

Objekte sind alle Elemente, die an der Überwachungs- und Benachrichtigungslogik beteiligt sind. Objekttypen umfassen:

- Services
- Servicegruppen
- Hosts
- Hostgruppen
- Kontakte
- Kontaktgruppen
- Befehle
- Zeitfenster
- Benachrichtigungseskalationen
- Benachrichtigungs- und Ausführungsabhängigkeiten

Mehr Informationen darüber, was Objekte sind und wie sie in Beziehung zueinander stehen, finden Sie nachstehend.

Wo werden Objekte definiert?

Objekte können in einer oder mehreren Konfigurationsdateien und/oder Verzeichnissen definiert werden, die Sie mit den cfg_file- und/oder cfg_dir-Direktiven in der Hauptkonfigurationsdatei angeben.

Hinweis: Wenn Sie der Schnellstart-Installationsanleitung folgen, werden verschiedene Beispiel-Objektkonfigurationsdateien in */usr/local/nagios/etc/objects/* abgelegt. Sie können diese Beispieldateien benutzen, um zu sehen, wie Objektvererbung funktioniert und lernen, wie Sie Ihre eigenen Objektdefinitionen anlegen.

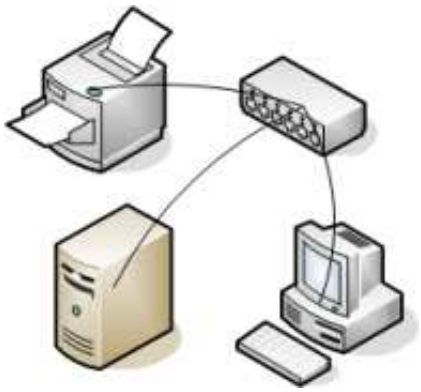
Wie werden Objekte definiert?

Objekte werden in einem flexiblen Vorlagenformat definiert, das es viel einfacher machen kann, Ihre Nagios-Konfiguration auf lange Sicht zu verwalten. Grundlegende Informationen, wie Objekte in Ihren Konfigurationsdateien definiert werden, finden Sie hier.

Sobald Sie mit den Grundlagen vertraut sind, wie Objekte zu definieren sind, sollten Sie bei Objektvererbung weiterlesen, weil es Ihre Konfiguration robuster für die Zukunft macht. Erfahrene Benutzer können einige fortgeschrittene Möglichkeiten der Objektdefinition ausnutzen, die in der Dokumentation zu Objekt-Tricks beschrieben sind.

Objekte erklärt

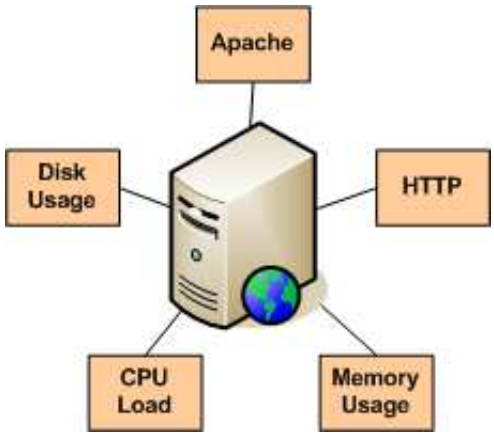
Einige der Hauptobjekttypen werden nachfolgend genauer erklärt...



Hosts sind eins der zentralen Objekte in der Überwachungslogik. Wichtige Attribute von Hosts sind:

- Hosts sind normaler Weise physikalische Geräte in Ihrem Netzwerk (Server, Workstations, Router, Switches, Drucker usw.).
- Hosts haben eine Adresse irgendeiner Art (z.B. eine IP- oder MAC-Adresse).
- Hosts haben einen oder mehrere Services, die mit ihm verbunden sind.
- Hosts können Eltern/Kind-Beziehungen mit anderen Hosts haben, oftmals dargestellt durch reale Netzwerkverbindungen, die in der Netzwerk-Erreichbarkeits-Logik benutzt wird.

Hostgruppen sind Gruppen von einem oder mehreren Hosts. Hostgruppen können es einfacher machen, (1) den Status von in Beziehung stehenden Hosts im Nagios-Web-Interface anzusehen und (2) Ihre Konfiguration mit Hilfe von Objekt-Tricks zu vereinfachen.



Services sind eins der zentralen Objekte in der Überwachungslogik. Services sind mit Hosts verbunden und können:

- Attribute eines Hosts sein (CPU-Auslastung, Plattenbelegung, Laufzeit, usw.)
- Services sein, die durch den Host zur Verfügung gestellt werden (HTTP, POP3, FTP, SSH, usw.)
- andere Dinge sein, die mit dem Host verbunden sind (DNS records, usw.)

Servicegruppen sind Gruppen von einem oder mehreren Services. Servicegruppen können es einfacher machen, (1) den Status von in Beziehung stehenden Services im Nagios-Web-Interface anzusehen und (2) Ihre Konfiguration mit Hilfe von Objekt-Tricks zu vereinfachen.

Kontakte sind Leute, die am Benachrichtigungsprozess beteiligt sind:

- Kontakte haben eine oder mehrere Benachrichtigungsmethoden (Handy, Pager, e-Mail, Sofortnachrichten, usw.)
- Kontakte erhalten Benachrichtigungen zu Hosts und Services, für die sie verantwortlich sind

Kontaktgruppen sind Gruppen von einem oder mehreren Kontakten. Kontaktgruppen können es einfacher machen, alle Leute zu definieren, die informiert werden, wenn bestimmte Host- oder Serviceprobleme auftreten.

Zeitfenster werden benutzt, um zu kontrollieren:

- wann Hosts und Services überwacht werden
- wann Kontakte Benachrichtigungen erhalten

Informationen darüber, wie Zeitfenster arbeiten, finden Sie hier.

Befehle werden benutzt, um Nagios mitzuteilen, welche Programme, Scripte usw. es ausführen soll:

- Host- und Service-Prüfungen
- Benachrichtigungen
- Eventhandler
- und mehr...

Nagios®

Objektdefinitionen

Hoch zu: Inhalt

Siehe auch: Überblick Objektkonfiguration, Objekt-Tricks, Objektvererbung, maßgeschneiderte Objektvariablen

Einführung

Eine der Möglichkeiten des Objektkonfigurationsformats von Nagios besteht darin, dass Sie Objektkonfigurationsdefinitionen erstellen können, die Eigenschaften von anderen Objektdefinitionen erben. Eine Erklärung, wie Objektvererbung funktioniert, finden Sie hier. Ich empfehle Ihnen dringend, dass Sie sich mit Objektvererbung beschäftigen, nachdem Sie die folgende Dokumentation überflogen haben, weil sie Ihnen die Arbeit bei der Erstellung und Wartung der Objektdefinitionen viel einfacher macht. Lesen Sie außerdem die Objekt-Tricks, die Ihnen Abkürzungsmöglichkeiten für andernfalls langwierige Konfigurationsaufgaben bieten.

Wenn Sie Konfigurationsdateien anlegen und/oder editieren, dann behalten Sie das Folgende im Hinterkopf:

1. Zeilen, die mit einem '#'-Zeichen beginnen, werden als Kommentare angesehen und nicht verarbeitet
2. Direktivennamen sind "case-sensitive", die Beachtung von Groß- und Kleinschreibung ist wichtig!
3. Zeichen nach einem Semikolon (;) in Konfigurationszeilen werden als Kommentar angesehen und deshalb nicht verarbeitet

Anmerkungen zur Aufbewahrung (Retention Notes)

Es ist wichtig anzumerken, dass einige Direktiven in Host-, Service- und Kontaktdefinitionen möglicherweise keine Wirkung zeigen, wenn Sie diese in den Konfigurationsdateien ändern. Objektdirektiven, die dieses Verhalten zeigen, sind mit einem Stern gekennzeichnet (*). Der Grund für dieses Verhalten liegt darin, dass Nagios Werte in der Statusaufbewahrungsdatei denen aus den Konfigurationsdateien vorzieht, wenn Sie Statusaufbewahrung auf programmweiter Basis aktiviert haben *und* den Wert der Direktive während der Laufzeit mit einem externen Befehl geändert haben.

Ein Weg, um dieses Problem zu umgehen, ist das Deaktivieren der Aufbewahrung von nicht-Statusinformationen mit Hilfe der *retain_nonstatus_information*-Direktive in den Host-, Service- und Kontaktdefinitionen. Durch das Deaktivieren dieser Direktive wird Nagios dazu veranlasst, beim (Neu-)Start die initialen Werte für diese Direktiven aus Ihren Konfigurationsdateien zu nehmen, statt die aus der Statusaufbewahrungsdatei zu verwenden.

Beispielkonfigurationsdatei

Anmerkung: Beispielobjektkonfigurationsdateien werden im */usr/local/nagios/etc/*-Verzeichnis installiert, wenn Sie der Schnellstart-Anleitung gefolgt sind.

Objekttypen

Host-Definition

Beschreibung:

Eine Host-Definition wird benutzt, um einen Server, eine Workstation, ein Gerät usw. zu definieren, die sich in Ihrem Netzwerk befinden.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

```
define host{
        host_name                       host_name
        alias                           alias
        display_name                    display_name
        address                         address
        parents                         host_names
        hostgroups                      hostgroup_names
        check_command                   command_name
        initial_state                   [o,d,u]
        max_check_attempts              #
        check_interval                  #
        retry_interval                  #
        active_checks_enabled           [0/1]
        passive_checks_enabled          [0/1]
        check_period                    timeperiod_name
```

obsess_over_host	[0/1]
check_freshness	[0/1]
freshness_threshold	#
event_handler	*command_name*
event_handler_enabled	[0/1]
low_flap_threshold	#
high_flap_threshold	#
flap_detection_enabled	[0/1]
flap_detection_options	[o,d,u]
process_perf_data	[0/1]
retain_status_information	[0/1]
retain_nonstatus_information	[0/1]
contacts	*contacts*
contact_groups	*contact_groups*
notification_interval	#
first_notification_delay	#
notification_period	*timeperiod_name*
notification_options	[d,u,r,f,s]
notifications_enabled	[0/1]
stalking_options	[o,d,u]
notes	*note_string*
notes_url	*url*
action_url	*url*
icon_image	*image_file*
icon_image_alt	*alt_string*
vrml_image	*image_file*
statusmap_image	*image_file*
2d_coords	*x_coord,y_coord*
3d_coords	*x_coord,y_coord,z_coord*

}

Beispieldefinition:

```
define host{
        host_name                       bogus-router
        alias                           Bogus Router #1
        address                         192.168.1.254
        parents                         server-backbone
        check_command                   check-host-alive
        check_interval                  5
        retry_interval                  1
        max_check_attempts              5
        check_period                    24x7
        process_perf_data               0
        retain_nonstatus_information    0
        contact_groups                  router-admins
        notification_interval           30
        notification_period             24x7
        notification_options            d,u,r
        }
```

Beschreibung der Direktiven:

host_name: Diese Direktive wird benutzt, um einen Kurznamen zu definieren, der den Host identifiziert. Er wird in Hostgruppen- und Service-Definitionen benutzt, um auf diesen bestimmten Host zu verweisen. Hosts können mehrere Services haben (die überwacht werden), die mit ihm verbunden sind.

alias: Diese Direktive wird benutzt, um einen längeren Namen oder eine Beschreibung zu definieren, der/die den Host identifiziert. Er/sie wird angeboten, damit Sie den Host einfacher identifizieren können. Bei korrekter Anwendung wird das $HOSTALIAS$-Makro diesen Alias/diese Beschreibung enthalten.

address: Diese Direktive wird benutzt, um die Adresse des Hosts zu definieren. Normalerweise ist dies die IP-Adresse des Hosts, obwohl es eigentlich alles sein kann, was Sie wollen (solange es genutzt werden kann, um den Status des Hosts zu prüfen). Sie können einen vollqualifizierten Domänennamen (FQDN) statt einer IP-Adresse benutzen, um den Host zu identifizieren, aber wenn keine DNS-Dienste verfügbar sind, kann dies zu Problemen führen. Bei korrekter Anwendung wird das $HOSTADDRESS$-Makro diese Adresse enthalten. **Anmerkung:** Wenn Sie keine Adress-Direktive in einer Host-Definition benutzen, wird der Name des Hosts statt der Adresse benutzt. Trotzdem ein Wort der Warnung: wenn DNS ausfällt, werden die meisten Ihrer Service-Prüfungen fehlschlagen, weil die Plugins nicht in der Lage sind, den Host-Namen aufzulösen.

display_name: Diese Direktive wird benutzt, um einen alternativen Namen zu definieren, der im Web-Interface für den Host angezeigt wird. Wenn nicht angegeben, wird statt dessen der Wert der *host_name*-Direktive benutzt. Anmerkung: die aktuellen CGIs nutzen diese Option nicht, zukünftige Versionen des Web-Interfaces werden das tun.

parents: Diese Direktive wird benutzt, um eine Komma-separierte Liste von Kurznamen der "Eltern"-Hosts dieses bestimmten Hosts zu definieren. Eltern-Hosts sind typischerweise Router, Switches, Firewalls usw. Ein Router, Switch usw., der am nächsten zum entfernten Host ist, wird als "Eltern" dieses Hosts angesehen. Lesen Sie weitere Informationen im Dokument "Festlegen des Zustands und der Erreichbarkeit von Netzwerk-Hosts", das Sie hier finden. Wenn dieser Host im gleichen Netzwerksegment wie der überwachende Host ist (ohne dazwischen liegende Router usw.), wird der Host als im lokalen Netzwerk befindlich angesehen und hat deshalb keinen Eltern-Host. Lassen Sie diesen Wert leer, wenn der Host keinen Eltern-Host hat (d.h. wenn er im gleichen Segment wie der Nagios-Host ist). Die Reihenfolge, in der Sie Eltern-Hosts angeben, hat keinen Einfluss darauf, wie Dinge überwacht werden.

hostgroups: Diese Direktive wird benutzt, um den/die *Kurznamen* der Hostgruppe(n) anzugeben, zu dem/denen der Host gehört. Mehrere Hostgruppen werden durch Kommata von einander getrennt. Diese Direktive kann als Alternative (oder zusätzlich) zur *members*-Direktive in den hostgroup-Definitionen genutzt werden.

check_command: Diese Direktive wird benutzt, um den *Kurznamen* des Befehls anzugeben, mit dem geprüft wird, ob der Host funktioniert oder nicht. Typischerweise wird dieser Befehl versuchen, den Host per "ping" zu prüfen, ob er "lebt". Der Befehl muss den Status OK (0) zurückliefern, denn sonst wird Nagios annehmen, dass der Host "down" ist. Wenn Sie diesen Wert leer lassen, wird der Host *nicht* aktiv geprüft. Dadurch wird Nagios höchstwahrscheinlich annehmen, dass der Host "up" ist (und ihn ggf. als "PENDING" im Web-Interface anzeigen). Das ist nützlich, wenn Sie Drucker oder andere Geräte überwachen, die regelmäßig ausgeschaltet werden. Die maximale Zeit, die der Prüfbefehl laufen darf, wird durch die host_check_timeout-Option kontrolliert.

initial_state: Als Default nimmt Nagios an, dass sich alle Hosts im UP-Zustand befinden, wenn es startet. Sie können mit dieser Direktive den initialen Zustand eines Hosts übersteuern. Gültige Optionen sind: **o** = UP, **d** = DOWN und **u** = UNREACHABLE.

max_check_attempts: Diese Direktive wird benutzt, um zu definieren, wie oft Nagios den Host-Prüfbefehl wiederholt, wenn er einen anderen als einen OK-Zustand zurückliefert. Bei einem Wert von 1 wird Nagios einen Alarm generieren, ohne den Host erneut zu prüfen. Anmerkung: wenn Sie den Zustand des Hosts nicht prüfen wollen, müssen Sie den Wert trotzdem mindestens auf 1 setzen. Lassen Sie die *check_command*-Option leer, um die Host-Prüfung zu umgehen.

check_interval: Diese Direktive wird benutzt, um die Anzahl von "Zeiteinheiten" zwischen regelmäßig geplanten Prüfungen zu definieren. Solange Sie die interval_length-Direktive mit einem Default-Wert von 60 nicht verändert haben, wird diese Zahl Minuten bedeuten. Mehr Informationen zu diesem Wert finden Sie in der check scheduling-Dokumentation.

retry_interval: Diese Direktive wird benutzt, um die Anzahl von "Zeiteinheiten" zu definieren, die zwischen erneuten Überprüfungen gewartet werden sollen. Erneute Überprüfungen für den Host werden mit dem Wiederholungsintervall eingeplant, wenn dieser in einen nicht-UP-Zustand gewechselt ist. Sobald der Host **max_check_attempts**-Mal ohne eine Zustandsänderung geprüft wurde, wird die Planung zum "normalen" Wert zurückkehren, der durch den **check_interval**-Wert angegeben wird. Solange Sie die interval_length-Direktive mit einem Default-Wert von 60 nicht verändert haben, wird diese Zahl Minuten bedeuten. Mehr Informationen zu diesem Wert finden Sie in der check scheduling-Dokumentation.

active_checks_enabled *: Diese Direktive wird benutzt, um festzulegen, ob aktive Prüfungen (entweder regelmäßig geplant oder nach Bedarf) für diesen Host aktiviert sind oder nicht. Werte: 0 = keine aktiven Host-Prüfungen, 1 = aktive Host-Prüfungen.

passive_checks_enabled *: Diese Direktive wird benutzt, um festzulegen, ob passive Prüfungen für diesen Host aktiviert sind oder nicht. Werte: 0 = passive Host-Prüfungen deaktivieren, 1 = passive Host-Prüfungen aktivieren

check_period: Diese Direktive wird benutzt, um den Kurznamen des Zeitfensters anzugeben, in dem aktive Prüfungen für diesen Host ausgeführt werden.

obsess_over_host *: Diese Direktive legt fest, ob Prüfungen für den Host über ochp_command "verfolgt" werden sollen.

check_freshness *: Diese Direktive wird benutzt, um festzulegen, ob Frische-Prüfungen (freshness checks) für diesen Host aktiviert sind oder nicht. Werte: 0 = Frische-Prüfungen deaktivieren, 1 = Frische-Prüfungen aktivieren.

freshness_threshold: Diese Direktive wird benutzt, um den Frische-Schwellwert (freshness threshold) (in Sekunden) für diesen Host festzulegen. Wenn Sie einen Wert von Null für diese Direktive setzen, wird Nagios automatisch einen Frische-Schwellwert festlegen.

event_handler: Diese Direktive wird benutzt, um den *Kurznamen* des Befehls anzugeben, der jedes Mal ausgeführt werden soll, sobald ein Statuswechsel für den Host erkannt wird (d.h. er "down" geht oder sich wieder erholt). Lesen Sie die Dokumentation zu Eventhandlern für eine detailliertere Erklärung, wie Scripte zur Behandlung von Ereignissen geschrieben werden. Die maximale Zeit, die ein Eventhandler-Befehl dauern darf, wird durch die event_handler_timeout-Option kontrolliert.

event_handler_enabled *: Diese Direktive wird benutzt, um festzulegen, ob der Eventhandler für diesen Host aktiviert ist oder nicht. Werte: 0 = Host-Eventhandler deaktivieren, 1 = Host-Eventhandler aktivieren

low_flap_threshold: Diese Direktive wird benutzt, um den unteren Zustandsänderungsschwellwert zu definieren, der in der Flattererkennung für diesen Host benutzt wird. Mehr Informationen zur Flattererkennung finden Sie hier. Wenn Sie diese Direktive auf 0 setzen, wird der programmweite Wert aus der low_host_flap_threshold-Direktive benutzt.

high_flap_threshold: Diese Direktive wird benutzt, um den oberen Zustandsänderungsschwellwert zu definieren, der in der Flattererkennung für diesen Host benutzt wird. Mehr Informationen zur Flattererkennung finden Sie hier. Wenn Sie diese Direktive auf 0 setzen, wird der programmweite Wert aus der high_host_flap_threshold-Direktive benutzt.

flap_detection_enabled *: Diese Direktive wird benutzt, um festzulegen, ob Flattererkennung für diesen Host aktiviert ist. Mehr Informationen zur Flattererkennung finden Sie hier. Werte: 0 = Host-Flattererkennung deaktivieren, 1 = Host-Flattererkennung aktivieren.

flap_detection_options: Diese Direktive wird benutzt, um festzulegen, welche Host-Zustände die Flattererkennungslogik für diesen Host benutzen wird. Gültige Optionen sind Kombinationen von einem oder mehreren folgender Werte: **o** = UP-Zustände, **d** = DOWN-Zustände, **u** = UNREACHABLE-Zustände.

process_perf_data *: Diese Direktive wird benutzt, um festzulegen, ob die Verarbeitung von Performance-Daten für diesen Host aktiviert ist. Werte: 0 = Verarbeitung von Performance-Daten deaktiviert, 1 = Verarbeitung von Performance-Daten aktiviert.

retain_status_information: Diese Direktive wird benutzt, um festzulegen, ob zustandsbezogene Informationen zu diesem Host über Programmneustarts hinweg aufbewahrt wird. Das ist nur sinnvoll, wenn Sie Statusaufbewahrung über die retain_state_information-Direktive aktiviert haben. Werte: 0 = Aufbewahrung von Statusinformationen deaktivieren, 1 = Aufbewahrung von Statusinformationen aktivieren.

retain_nonstatus_information: Diese Direktive wird benutzt, um festzulegen, ob nicht-zustandsbezogene Informationen zu diesem Host über Programmneustarts hinweg aufbewahrt wird. Das ist nur sinnvoll, wenn Sie Statusaufbewahrung über die retain_state_information-Direktive aktiviert haben. Werte: 0 = Aufbewahrung von nicht-Statusinformationen deaktivieren, 1 = Aufbewahrung von nicht-Statusinformationen aktivieren.

contacts: Dies ist eine Liste der *Kurznamen* der Kontakte, die über Probleme (oder Erholungen) dieses Hosts informiert werden sollen. Mehrere Kontakte werden jeweils durch ein Komma voneinander getrennt. Nützlich, wenn Benachrichtigungen nur an ein paar Leute gehen sollen und Sie dafür keine Kontaktgruppen definieren wollen. Sie müssen mindestens einen Kontakt oder eine Kontaktgruppe in jeder Host-Definition angeben.

contact_groups: Dies ist eine Liste der *Kurznamen* der Kontaktgruppen, die über Probleme (oder Erholungen) dieses Hosts informiert werden sollen. Mehrere Kontaktgruppen werden durch ein Komma voneinander getrennt. Sie müssen mindestens einen Kontakt oder eine Kontaktgruppe in jeder Host-Definition angeben.

notification_interval: Diese Direktive wird benutzt, um die Anzahl von "Zeiteinheiten" anzugeben, die gewartet werden soll, bevor ein Kontakt erneut darüber informiert werden soll, dass dieser Host *immer noch* "down" oder unerreichbar ist. Solange Sie nicht die interval_length-Direktive auf einen anderen als den Standardwert von 60 verändert haben, bedeutet diese Zahl Minuten. Wenn Sie diesen Wert auf 0 setzen, wird Nagios die Kontakte *nicht* erneut über Probleme dieses Hosts informieren - nur eine Problembenachrichtigung wird versandt.

first_notification_delay: Diese Direktive wird benutzt, um die Anzahl von "Zeiteinheiten" anzugeben, die gewartet werden soll, bevor die erste Problembenachrichtigung versandt wird, wenn dieser Host in einen nicht-UP-Zustand wechselt. Solange Sie nicht die interval_length-Direktive auf einen anderen als den Standardwert von 60 verändert haben, bedeutet diese Zahl Minuten. Wenn Sie diesen Wert auf 0 setzen, wird Nagios sofort Benachrichtigungen versenden.

notification_period: Diese Direktive wird benutzt, um den Kurznamen des Zeitfensters anzugeben, in dem Benachrichtigungen zu Ereignissen dieses Hosts an Kontakte versandt werden. Wenn ein Host zu einer Zeit "down" geht, unerreichbar wird oder sich wieder erholt, die nicht in diesem Zeitfenster liegt, werden keine Benachrichtigungen versandt.

notification_options: Diese Direktive wird benutzt, um festzulegen, wann Benachrichtigungen für diesen Host versandt werden. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte: **d** = Benachrichtigungen bei einem DOWN-Zustand versenden, **u** = Benachrichtigungen bei einem UNREACHABLE-Zustand versenden, **r** = Benachrichtigungen bei Erholungen (OK-Zustand) versenden, **f** = Benachrichtigungen versenden, wenn der Host mit Flattern anfängt bzw. aufhört und **s** = Benachrichtigungen versenden, wenn eine geplante Ausfallzeit anfängt oder aufhört. Wenn Sie **n** (none) als Option angeben, werden keine Host-Benachrichtigungen versandt. Wenn Sie keine Benachrichtigungsoptionen angeben, geht Nagios davon aus, dass Sie Benachrichtigungen zu allen möglichen Zuständen haben möchten. Beispiel: wenn Sie **d,r** in diesem Feld angeben, werden Benachrichtigungen nur dann versandt, wenn der Host in einen DOWN-Zustand geht und sich wieder von einem DOWN-Zustand erholt.

notifications_enabled *: Diese Direktive wird benutzt, um festzulegen, ob Benachrichtigungen für diesen Host aktiviert sind oder nicht. Werte: 0 = Host-Benachrichtigungen deaktivieren, 1 = Host-Benachrichtigungen aktivieren.

stalking_options: Diese Direktive legt fest, für welche Host-Zustände "Verfolgung" (stalking) aktiviert ist. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte: **o** = verfolgen von UP-Zuständen, **d** = verfolgen von DOWN-Zuständen und **u** = verfolgen von UNREACHABLE-Zuständen. Mehr Informationen zur Statusverfolgung finden Sie hier.

notes: Diese Direktive wird benutzt, um optional einen Text mit Informationen zu diesem Host anzugeben. Wenn Sie hier Anmerkungen angeben, werden Sie diese in der extended information-CGI sehen (wenn Sie Informationen zu dem entsprechenden Host ansehen).

notes_url: Diese Variable wird benutzt, um einen optionalen URL anzugeben, der verwendet werden kann, um weitere Informationen zu diesem Host zu liefern. Wenn Sie einen URL angeben, werden Sie ein rotes Verzeichnis-Icon in den CGIs sehen (wenn Sie Host-Informationen betrachten), das auf den URL verweist, den Sie hier angeben. Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*). Dies kann sehr nützlich sein, wenn Sie detaillierte Informationen zu diesem Host, Notfallkontaktmethoden usw. für anderes Support-Personal zur Verfügung stellen wollen.

action_url: Diese Direktive wird benutzt, um einen optionalen URL anzugeben, der verwendet werden kann, um weitere Aktionen für diesen Host zu ermöglichen. Wenn Sie einen URL angeben, werden Sie einen roten "Klecks" in den CGIs sehen (wenn Sie Host-Informationen betrachten). Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*).

icon_image: Diese Variable wird benutzt, um den Namen eines GIF-, PNG oder JPG-Images anzugeben, das mit diesem Host verbunden werden soll. Dieses Bild wird an verschiedenen Stellen in den CGIs angezeigt. Das Bild wird am besten aussehen, wenn es 40x40 Pixel groß ist. Bilder für Hosts werden im **logos/**-Unterverzeichnis Ihres HTML-Images-Verzeichnis gesucht (d.h. */usr/local/nagios/share/images/logos*).

icon_image_alt: Diese Variable wird benutzt, um eine optionale Zeichenkette anzugeben, die für den ALT-Tag des Bildes benutzt wird, das durch das <*icon_image*>-Argument angegeben wurde.

vrml_image: Diese Variable wird benutzt, um den Namen eines GIF-, PNG- oder JPG-Images anzugeben, das mit diesem Host verbunden werden soll. Dieses Bild wird als Textur-Map für den angegebenen Host in der statuswrl-CGI benutzt. Anders als das Bild, das Sie in der <*icon_image*>-Variable angeben, sollte dieses möglichst *keinerlei* Transparenz haben. Wenn es das tut, wird das Host-Objekt ein wenig komisch aussehen. Bilder für Hosts werden im **logos/**-Unterverzeichnis Ihres HTML-Images-Verzeichnis gesucht (d.h. */usr/local/nagios/share/images/logos*).

statusmap_image: Diese Variable wird benutzt, um den Namen eines Bildes anzugeben, das mit diesem Host im statusmap-CGI verbunden werden soll. Sie können ein JPG-, PNG- oder GIF-Bild angeben, aber ich würde zu einem Bild im GD2-Format raten, weil andere Bildformate zu hohen CPU-Belastungen führen können, wenn die Statusmap generiert wird. PNG-Bilder können mit Hilfe des **pngtogd2**-Utilitys (das in Thomas Boutell's gd library enthalten ist) ins GD2-Format umgewandelt werden. Die GD2-Bilder werden im *unkomprimierten* Format erstellt, um die CPU-Belastung zu minimieren, während das Statusmap-CGI das Netzwerkkartenbild erstellt. Das Bild wird am besten aussehen, wenn es 40x40 Pixel groß ist. Sie können diese Option leer lassen, wenn Sie das Statusmap-CGI nicht nutzen. Bilder für Hosts werden im **logos/**-Unterverzeichnis Ihres HTML-Images-Verzeichnis gesucht (d.h. */usr/local/nagios/share/images/logos*).

2d_coords:	Diese Variable wird benutzt, um Koordinaten anzugeben, wenn der Host im statusmap-CGI gezeichnet wird. Koordinaten sollen in positiven Ganzzahlen angegeben werden, weil sie physischen Pixeln im generierten Bild entsprechen. Der Ursprung (0,0) für die Zeichnung ist die linke, obere Ecke des Bildes, das sich in die positive X-Richtung (nach rechts) und in die positive Y-Richtung (nach unten) erstreckt. Die Größe der Icons ist normalerweise etwa 40x40 Pixel (Text benötigt etwas mehr Platz). Die Koordinaten, die Sie angeben, beziehen sich auf die linke, obere Ecke des Icons. Anmerkung: Machen Sie sich keine Sorgen über die maximalen X- und Y-Koordinaten, die Sie benutzen können. Das CGI wird automatisch die maximale Größe des zu erstellenden Bildes aufgrund der größten X- und Y-Koordinaten festlegen, die Sie angegeben haben.
3d_coords:	Diese Variable wird benutzt, um Koordinaten anzugeben, die beim Zeichnen des Hosts im statuswrl-CGI verwendet werden. Koordinaten können positive oder negative reelle Zahlen sein. Der Ursprung für die Zeichnung ist (0.0,0.0,0.0). Die Größe des Host-Kubus ist 0,5 Einheiten auf jeder Seite (Text benötigt etwas mehr Platz). Die Koordinaten, die Sie hier angeben, beziehen sich auf das Zentrum des Host-Kubus.

Hostgruppen-Definition

Beschreibung:

Eine Hostgruppen-Definition wird benutzt, um einen oder mehrere Hosts zu gruppieren, um die Konfiguration mit Objekt-Tricks zu vereinfachen oder für Anzeigezwecke in den CGIs.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

```
define hostgroup{
        hostgroup_name      hostgroup_name
        alias               alias
        members             hosts
        hostgroup_members   hostgroups
        notes               note_string
        notes_url           url
        action_url          url
        }
```

Beispieldefinition:

```
define hostgroup{
        hostgroup_name          novell-servers
        alias                   Novell Servers
        members                 netware1,netware2,netware3,netware4
        }
```

Beschreibung der Direktiven:

hostgroup_name: Diese Direktive wird benutzt, um einen Kurznamen zu definieren, der die Hostgruppe identifiziert.

alias: Diese Direktive wird benutzt, um einen längeren Namen oder eine Beschreibung zu definieren, der die Hostgruppen identifiziert. Er/sie wird angeboten, damit Sie eine bestimmte Hostgruppe einfacher identifizieren können.

members: Dies ist eine Liste von *Kurznamen* der Hosts, die in dieser Gruppe enthalten sein sollen. Mehrere Hostnamen sollten jeweils durch Komma von einander getrennt werden. Diese Direktive kann als Alternative (oder als Zusatz) zu der *hostgroups*-Direktive in den Host-Definitionen verwendet werden.

hostgroup_members: Diese optionale Direktive kann genutzt werden, um Hosts aus anderen "sub"-Hostgruppen in diese Hostgruppe aufzunehmen. Geben Sie eine komma-separierte Liste von Kurznamen anderer Hostgruppen an, deren Mitglieder in diese Gruppe aufgenommen werden sollen.

notes: Diese Direktive wird benutzt, um optional einen Text mit Informationen zu dieser Hostgruppe anzugeben. Wenn Sie hier Anmerkungen angeben, werden Sie diese in der extended information-CGI sehen (wenn Sie Informationen zu der entsprechenden Hostgruppe ansehen).

notes_url: Diese Variable wird benutzt, um einen optionalen URL anzugeben, der verwendet werden kann, um weitere Informationen zu dieser Hostgruppe zu liefern. Wenn Sie einen URL angeben, werden Sie ein rotes Verzeichnis-Icon in den CGIs sehen (wenn Sie Hostgruppen-Informationen betrachten), das auf den URL verweist, den Sie hier angeben. Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*). Dies kann sehr nützlich sein, wenn Sie detaillierte Infomationen zu dieser Hostgruppe, Notfallkontaktmethoden usw. für anderes Support-Personal zur Verfügung stellen wollen.

action_url: Diese Direktive wird benutzt, um einen optionalen URL anzugeben, der verwendet werden kann, um weitere Aktionen für diese Hostgruppe zu ermöglichen. Wenn Sie einen URL angeben, werden Sie einen roten "Klecks" in den CGIs sehen (wenn Sie Hostgruppen-Informationen betrachten). Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*).

Service-Definition

Beschreibung:

Eine Service-Definition wird benutzt, um einen "Service" zu identifizieren, der auf einem Host läuft. Der Begriff "Service" wird sehr locker benutzt. Es kann sich um einen realen Service auf einem Host handeln (POP, SMTP, HTTP, etc.) oder eine andere Art von Metrik, die mit dem Host verbunden ist (Antwort auf einen Ping, Anzahl der angemeldeten Benutzer, freier Plattenplatz usw.). Die verschiedenen Parameter einer Service-Definition sind nachfolgend dargestellt.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

define service{

host_name	*host_name*
hostgroup_name	*hostgroup_name*
service_description	*service_description*
display_name	*display_name*
servicegroups	servicegroup_names
is_volatile	[0/1]
check_command	*command_name*
initial_state	[o,w,u,c]
max_check_attempts	#
check_interval	#
retry_interval	#
active_checks_enabled	[0/1]
passive_checks_enabled	[0/1]
check_period	*timeperiod_name*
obsess_over_service	[0/1]
check_freshness	[0/1]
freshness_threshold	#
event_handler	*command_name*
event_handler_enabled	[0/1]
low_flap_threshold	#
high_flap_threshold	#
flap_detection_enabled	[0/1]
flap_detection_options	[o,w,c,u]
process_perf_data	[0/1]
retain_status_information	[0/1]

retain_nonstatus_information	[0/1]
notification_interval	#
first_notification_delay	#
notification_period	*timeperiod_name*
notification_options	[w,u,c,r,f,s]
notifications_enabled	[0/1]
contacts	*contacts*
contact_groups	*contact_groups*
stalking_options	[o,w,u,c]
notes	*note_string*
notes_url	*url*
action_url	*url*
icon_image	*image_file*
icon_image_alt	*alt_string*

}

Beispieldefinition:

```
define service{
        host_name               linux-server
        service_description     check-disk-sda1
        check_command           check-disk!/dev/sda1
        max_check_attempts      5
        check_interval          5
        retry_interval          3
        check_period            24x7
        notification_interval   30
        notification_period     24x7
        notification_options    w,c,r
        contact_groups          linux-admins
        }
```

Beschreibung der Direktiven:

host_name: Diese Direktive wird benutzt, um den *Kurznamen* des/der Hosts anzugeben, auf denen der Service "läuft" bzw. mit dem/denen er verbunden ist. Mehrere Hosts sind jeweils durch Komma von einander zu trennen.

hostgroup_name: Diese Direktive wird benutzt, um den/die *Kurznamen* der Hostgruppe(n) anzugeben, auf der/denen der Service "läuft" bzw. mit der/denen er verbunden ist. Mehrere Hostgruppen werden durch Kommata von einander getrennt. Der hostgroup_name kann anstatt oder zusätzlich zur host_name-Direktive benutzt werden.

service_description: Diese Direktive wird benutzt, um eine Beschreibung des Service zu definieren, die Leerzeichen, Bindestriche und Doppelpunkte enthalten kann (Semikolon, Apostroph und Fragezeichen sollten vermieden werden). Keine zwei Services des gleichen Hosts können die gleiche Beschreibung haben. Services werden eindeutig durch die *host_name* und *service_description*-Direktiven identifiziert.

display_name: Diese Direktive wird benutzt, um einen alternativen Namen zu definieren, der im Web-Interface für diesen Service angezeigt wird. Falls nicht angegeben, wird der Wert aus der *service_description*-Direktive benutzt. Anmerkung: Die aktuellen CGIs nutzen diese Option nicht, zukünftige Versionen des Web-Interfaces werden dies tun.

servicegroups: Diese Direktive wird benutzt, um den/die *Kurznamen* der Servicegruppe(n) anzugeben, zu der/denen der Service gehört. Mehrere Servicegruppen werden durch Kommata von einander getrennt. Diese Direktive kann als Alternative (oder zusätzlich) zur *members*-Direktive in den servicegroup-Definitionen genutzt werden.

is_volatile: Diese Direktive wird benutzt, um zu kennzeichnen, dass der Service "sprunghaft" (volatile) ist. Services sind normalerweise *nicht* sprunghaft. Mehr Informationen zu sprunghaften Services und wie sie sich von normalen Services unterscheiden, finden Sie hier. Werte: 0 = Services sind nicht sprunghaft, 1 = Service sind sprunghaft.

check_command: Diese Direktive wird benutzt, um den *Kurznamen* des Befehls anzugeben, mit dem der Zustand des Service geprüft wird. Die maximale Zeit, die der Prüfbefehl laufen darf, wird durch die service_check_timeout-Option kontrolliert.

initial_state: Als Default nimmt Nagios an, dass sich alle Services im OK-Zustand befinden, wenn es startet. Sie können mit dieser Direktive den initialen Zustand eines Service übersteuern. Gültige Optionen sind: **o** = OK, **w** = WARNING, *u* = UNKNOWN und **c** = CRITICAL.

max_check_attempts: Diese Direktive wird benutzt, um zu definieren, wie oft Nagios den Service-Prüfbefehl wiederholt, wenn er einen anderen als einen OK-Zustand zurückliefert. Bei einem Wert von 1 wird Nagios einen Alarm generieren, ohne den Service erneut zu prüfen.

check_interval: Diese Direktive wird benutzt, um die Anzahl von "Zeiteinheiten" zwischen "regulär" geplanten Prüfungen zu definieren. "Reguläre" Prüfungen sind solche, die stattfinden, wenn sich der Service in einem OK-Zustand befindet oder wenn sich der Service in einem nicht-OK-Zustand befindet, aber mehr als **max_check_attemps**-mal erneut geprüft wurde. Solange Sie die interval_length-Direktive mit einem Default-Wert von 60 nicht verändert haben, wird diese Zahl Minuten bedeuten. Mehr Informationen zu diesem Wert finden Sie in der check scheduling-Dokumentation.

retry_interval: Diese Direktive wird benutzt, um die Anzahl von "Zeiteinheiten" zu definieren, die zwischen erneuten Überprüfungen des Service gewartet werden sollen. Erneute Überprüfungen für Services werden mit dem Wiederholungsintervall eingeplant, wenn diese in einen nicht-OK-Zustand gewechselt sind. Sobald der Service **max_check_attempts**-Mal ohne eine Zustandsänderung geprüft wurde, wird die Planung zum "normalen" Wert zurückkehren, der durch den **check_interval**-Wert angegeben wird. Solange Sie die interval_length-Direktive mit einem Default-Wert von 60 nicht verändert haben, wird diese Zahl Minuten bedeuten. Mehr Informationen zu diesem Wert finden Sie in der check scheduling-Dokumentation.

active_checks_enabled *: Diese Direktive wird benutzt, um festzulegen, ob aktive Prüfungen (entweder regelmäßig geplant oder nach Bedarf) für diesen Service aktiviert sind oder nicht. Werte: 0 = keine aktiven Service-Prüfungen, 1 = aktive Service-Prüfungen.

passive_checks_enabled *: Diese Direktive wird benutzt, um festzulegen, ob passive Prüfungen für diesen Service aktiviert sind oder nicht. Werte: 0 = passive Service-Prüfungen deaktivieren, 1 = passive Service-Prüfungen aktivieren

check_period: Diese Direktive wird benutzt, um den Kurznamen des Zeitfensters anzugeben, in dem aktive Prüfungen für diesen Service ausgeführt werden.

obsess_over_service *: Diese Direktive legt fest, ob Prüfungen für den Service über ocsp_command "verfolgt" werden sollen.

check_freshness *: Diese Direktive wird benutzt, um festzulegen, ob Frische-Prüfungen (freshness checks) für diesen Service aktiviert sind oder nicht. Werte: 0 = Frische-Prüfungen deaktivieren, 1 = Frische-Prüfungen aktivieren.

freshness_threshold: Diese Direktive wird benutzt, um den Frische-Schwellwert (freshness threshold) (in Sekunden) für diesen Service festzulegen. Wenn Sie einen Wert von Null für diese Direktive setzen, wird Nagios automatisch einen Frische-Schwellwert festlegen.

event_handler: Diese Direktive wird benutzt, um den *Kurznamen* des Befehls anzugeben, der jedes Mal ausgeführt werden soll, sobald ein Statuswechsel für den Service erkannt wird (d.h. er in einen nicht-OK-Zustand geht oder sich wieder erholt). Lesen Sie die Dokumentation zu Eventhandlern für eine detailliertere Erklärung, wie Scripte zur Behandlung von Ereignissen geschrieben werden. Die maximale Zeit, die ein Eventhandler-Befehl dauern darf, wird durch die event_handler_timeout-Option kontrolliert.

event_handler_enabled *: Diese Direktive wird benutzt, um festzulegen, ob der Eventhandler für diesen Service aktiviert ist oder nicht. Werte: 0 = Service-Eventhandler deaktivieren, 1 = Service-Eventhandler aktivieren

low_flap_threshold: Diese Direktive wird benutzt, um den unteren Zustandsänderungsschwellwert zu definieren, der in der Flattererkennung für diesen Service benutzt wird. Mehr Informationen zur Flattererkennung finden Sie hier. Wenn Sie diese Direktive auf 0 setzen, wird der programmweite Wert aus der low_service_flap_threshold-Direktive benutzt.

high_flap_threshold: Diese Direktive wird benutzt, um den oberen Zustandsänderungsschwellwert zu definieren, der in der Flattererkennung für diesen Service benutzt wird. Mehr Informationen zur Flattererkennung finden Sie hier. Wenn Sie diese Direktive auf 0 setzen, wird der programmweite Wert aus der high_service_flap_threshold-Direktive benutzt.

flap_detection_enabled *: Diese Direktive wird benutzt, um festzulegen, ob Flattererkennung für diesen Service aktiviert ist. Mehr Informationen zur Flattererkennung finden Sie hier. Werte: 0 = Service-Flattererkennung deaktivieren, 1 = Service-Flattererkennung aktivieren.

flap_detection_options: Diese Direktive wird benutzt, um festzulegen, welche Service-Zustände die Flattererkennungslogik für diesen Service benutzen wird. Gültige Optionen sind Kombinationen von einem oder mehreren folgender Werte: **o** = OK-Zustände, **w** = WARNING-Zustände, **c** = CRITICAL-Zustände, **u** = UNKNOWN-Zustände.

process_perf_data *: Diese Direktive wird benutzt, um festzulegen, ob die Verarbeitung von Performance-Daten für diesen Service aktiviert ist. Werte: 0 = Verarbeitung von Performance-Daten deaktiviert, 1 = Verarbeitung von Performance-Daten aktiviert.

retain_status_information: Diese Direktive wird benutzt, um festzulegen, ob zustandsbezogene Informationen zu diesem Service über Programmneustarts hinweg aufbewahrt werden. Das ist nur sinnvoll, wenn Sie Statusaufbewahrung über die retain_state_information-Direktive aktiviert haben. Werte: 0 = Aufbewahrung von Statusinformationen deaktivieren, 1 = Aufbewahrung von Statusinformationen aktivieren.

retain_nonstatus_information: Diese Direktive wird benutzt, um festzulegen, ob nicht-zustandsbezogene Informationen zu diesem Service über Programmneustarts hinweg aufbewahrt werden. Das ist nur sinnvoll, wenn Sie Status-Beibehaltung über die retain_state_information-Direktive aktiviert haben. Werte: 0 = Aufbewahrung von nicht-Statusinformationen deaktivieren, 1 = Aufbewahrung von nicht-Statusinformationen aktivieren.

notification_interval: Diese Direktive wird benutzt, um die Anzahl von "Zeiteinheiten" anzugeben, die gewartet werden soll, bevor ein Kontakt erneut darüber informiert werden soll, dass dieser Service *immer noch* in einem nicht-OK-Zustand ist. Solange Sie nicht die interval_length-Direktive auf einen anderen als den Standardwert von 60 verändert haben, bedeutet diese Zahl Minuten. Wenn Sie diesen Wert auf 0 setzen, wird Nagios die Kontakte *nicht* erneut über Probleme dieses Service informieren - nur eine Problembenachrichtigung wird versandt.

first_notification_delay: Diese Direktive wird benutzt, um die Anzahl von "Zeiteinheiten" anzugeben, die gewartet werden soll, bevor die erste Problembenachrichtigung versandt wird, wenn dieser Service in einen nicht-OK-Zustand wechselt. Solange Sie nicht die interval_length-Direktive auf einen anderen als den Standardwert von 60 verändert haben, bedeutet diese Zahl Minuten. Wenn Sie diesen Wert auf 0 setzen, wird Nagios sofort Benachrichtigungen versenden.

notification_period: Diese Direktive wird benutzt, um den Kurznamen des Zeitfensters anzugeben, in dem Benachrichtigungen zu Ereignissen dieses Service an Kontakte versandt werden. Zu Zeiten, die nicht in diesem Zeitfenster liegen, werden keine Benachrichtigungen versandt.

notification_options: Diese Direktive wird benutzt, um festzulegen, wann Benachrichtigungen für diesen Service versandt werden. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte: **w** = Benachrichtigungen bei einem WARNING-Zustand versenden, **u** = Benachrichtigungen bei einem UNKNOWN-Zustand versenden, **r** = Benachrichtigungen bei Erholungen (OK-Zustand) versenden, **f** = Benachrichtigungen versenden, wenn der Service mit Flattern anfängt bzw. aufhört und **s** = Benachrichtigungen versenden, wenn eine geplante Ausfallzeit anfängt oder aufhört. Wenn Sie **n** (none) als Option angeben, werden keine Service-Benachrichtigungen versandt. Wenn Sie keine Benachrichtigungsoptionen angeben, geht Nagios davon aus, dass Sie Benachrichtigungen zu allen möglichen Zuständen haben möchten. Beispiel: wenn Sie **w,r** in diesem Feld angeben, werden Benachrichtigungen nur dann versandt, wenn der Service in einen WARNING-Zustand geht und sich wieder von einem WARNING-Zustand erholt.

notifications_enabled *: Diese Direktive wird benutzt, um festzulegen, ob Benachrichtigungen für diesen Service aktiviert sind oder nicht. Werte: 0 = Service-Benachrichtigungen deaktivieren, 1 = Service-Benachrichtigungen aktivieren.

contacts: Dies ist eine Liste der *Kurznamen* der Kontakte, die über Probleme (oder Erholungen) dieses Service informiert werden sollen. Mehrere Kontakte werden jeweils durch Kommata voneinander getrennt. Nützlich, wenn Benachrichtigungen nur an ein paar Leute gehen sollen und Sie dafür keine Kontaktgruppen definieren wollen. Sie müssen mindestens einen Kontakt oder eine Kontaktgruppe in jeder Service-Definition angeben.

contact_groups: Dies ist eine Liste der *Kurznamen* der Kontaktgruppen, die über Probleme (oder Erholungen) dieses Service informiert werden sollen. Mehrere Kontaktgruppen werden durch Kommata voneinander getrennt. Sie müssen mindestens einen Kontakt oder eine Kontaktgruppe in jeder Service-Definition angeben.

stalking_options: Diese Direktive legt fest, für welche Service-Zustände "Verfolgung" (stalking) aktiviert ist. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte: **o** = verfolgen von OK-Zuständen, **w** = verfolgen von WARNING-Zuständen, **c** = verfolgen von CRITICAL-Zuständen und **u** = verfolgen von UNKNOWN-Zuständen. Mehr Informationen zur Statusverfolgung finden Sie hier.

notes: Diese Direktive wird benutzt, um optional einen Text mit Informationen zu diesem Service anzugeben. Wenn Sie hier Anmerkungen angeben, werden Sie diese in der extended information-CGI sehen (wenn Sie Informationen zu dem entsprechenden Service ansehen).

notes_url: Diese Variable wird benutzt, um einen optionalen URL anzugeben, der benutzt werden kann, um weitere Informationen zu diesem Service zu liefern. Wenn Sie einen URL angeben, werden Sie ein rotes Verzeichnis-Icon in den CGIs sehen (wenn Sie Service-Informationen betrachten), das auf den URL verweist, den Sie hier angeben. Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*). Dies kann sehr nützlich sein, wenn Sie detaillierte Informationen zu diesem Service, Notfallkontaktmethoden usw. für anderes Support-Personal zur Verfügung stellen wollen.

action_url: Diese Direktive wird benutzt, um einen optionalen URL anzugeben, der benutzt werden kann, um weitere Aktionen für diesen Service zu ermöglichen. Wenn Sie einen URL angeben, werden Sie einen roten "Klecks" in den CGIs sehen (wenn Sie Host-Informationen betrachten). Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*).

icon_image: Diese Variable wird benutzt, um den Namen eines GIF-, PNG oder JPG-Images anzugeben, das mit diesem Service verbunden werden soll. Dieses Bild wird an verschiedenen Stellen in den CGIs angezeigt. Das Bild wird am besten aussehen, wenn es 40x40 Pixel groß ist. Bilder für Services werden im **logos/**-Unterverzeichnis Ihres HTML-Images-Verzeichnis gesucht (d.h. */usr/local/nagios/share/images/logos*).

icon_image_alt: Diese Variable wird benutzt, um eine optionale Zeichenkette anzugeben, die für den ALT-Tag des Bildes benutzt wird, das durch das <*icon_image*>-Argument angegeben wurde.

Servicegruppen-Definition

Beschreibung:

Eine Servicegruppen-Definition wird benutzt, um einen oder mehrere Services zu gruppieren, um die Konfiguration mit Objekt-Tricks zu vereinfachen oder für Anzeigezwecke in den CGIs.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

define servicegroup{

servicegroup_name	*servicegroup_name*
alias	*alias*
members	*services*
servicegroup_members	*servicegroups*
notes	*note_string*
notes_url	*url*
action_url	*url*

}

Beispieldefinition:

```
define servicegroup{
        servicegroup_name       dbservices
        alias                   Database Services
        members                 ms1,SQL Server,ms1,SQL Server Agent,ms1,SQL DTC
        }
```

Beschreibung der Direktiven:

servicegroup_name: Diese Direktive wird benutzt, um einen Kurznamen zu definieren, der die Servicegruppe identifiziert.

alias: Diese Direktive wird benutzt, um einen längeren Namen oder eine Beschreibung zu definieren, der die Servicegruppen identifiziert. Er/sie wird angeboten, damit Sie ein bestimmte Servicegruppe einfacher identifizieren können.

members: Dies ist eine Liste von *Kurznamen* der Services (und der Namen der entsprechenden Hosts), die in dieser Gruppe enthalten sein sollen. Host- und Service-Namen sollten jeweils durch Komma von einander getrennt werden. Diese Direktive kann als Alternative (oder als Zusatz) zu der *servicegroups*-Direktive in den Service-Definitionen verwendet werden. Das Format der member-Direktive ist wie folgt (beachten Sie, dass ein Host-Name einem Service-Namen/einer Service-Beschreibung vorangestellt werden muss):

members=<host1>,<service1>,<host2>,<service2>,...,<host*n*>,<service*n*>

servicegroup_members: Diese optionale Direktive kann genutzt werden, um Services aus anderen "sub"-Servicegruppen in diese Servicegruppe aufzunehmen. Geben Sie eine komma-separierte Liste von Kurznamen anderer Servicegruppen an, deren Mitglieder in diese Gruppe aufgenommen werden sollen.

notes: Diese Direktive wird benutzt, um optional einen Text mit Informationen zu dieser Servicegruppe anzugeben. Wenn Sie hier Anmerkungen angeben, werden Sie diese in der extended information-CGI sehen (wenn Sie Informationen zu der entsprechenden Servicegruppe ansehen).

notes_url: Diese Variable wird benutzt, um einen optionalen URL anzugeben, der benutzt werden kann, um weitere Informationen zu dieser Servicegruppe zu liefern. Wenn Sie einen URL angeben, werden Sie ein rotes Verzeichnis-Icon in den CGIs sehen (wenn Sie Servicegruppen-Informationen betrachten), das auf den URL verweist, den Sie hier angeben. Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*). Dies kann sehr nützlich sein, wenn Sie detaillierte Infomationen zu dieser Servicegruppe, Notfallkontaktmethoden usw. für anderes Support-Personal zur Verfügung stellen wollen.

action_url: Diese Direktive wird benutzt, um einen optionalen URL anzugeben, der benutzt werden kann, um weitere Aktionen für diese Servicegruppe zu ermöglichen. Wenn Sie einen URL angeben, werden Sie einen roten "Klecks" in den CGIs sehen (wenn Sie Servicegruppen-Informationen betrachten). Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*).

Kontakt-Definition

Beschreibung:

Eine Kontakt-Definition wird benutzt, um jemanden zu identifizieren, der im Falle eines Problems in Ihrem Netzwerk informiert werden soll. Die verschiedenen Parameter einer Kontakt-Definition werden nachfolgend beschrieben.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

define contact{

contact_name	*contact_name*
alias	*alias*
contactgroups	*contactgroup_names*
host_notifications_enabled	[0/1]
service_notifications_enabled	[0/1]
host_notification_period	*timeperiod_name*
service_notification_period	*timeperiod_name*
host_notification_options	[d,u,r,f,s,n]
service_notification_options	[w,u,c,r,f,s,n]
host_notification_commands	*command_name*
service_notification_commands	*command_name*
email	*email_address*
pager	*pager_number or pager_email_gateway*
address*x*	*additional_contact_address*
can_submit_commands	[0/1]
retain_status_information	[0/1]
retain_nonstatus_information	[0/1]

}

Beispieldefinition:

```
define contact{
        contact_name                    jdoe
        alias                           John Doe
        host_notifications_enabled      1
        service_notifications_enabled   1
        service_notification_period     24x7
        host_notification_period        24x7
        service_notification_options    w,u,c,r
        host_notification_options       d,u,r
        service_notification_commands   notify-by-email
        host_notification_commands      host-notify-by-email
        email                           jdoe@localhost.localdomain
        pager                           555-5555@pagergateway.localhost.localdomain
```

```
address1                        xxxxx.xyyy@icq.com
address2                        555-555-5555
can_submit_commands             1
}
```

Beschreibung der Direktiven:

contact_name: Diese Direktive wird benutzt, um einen Kurznamen zu definieren, der den Kontakt identifiziert. Er wird in Kontaktgruppen-Definitionen benutzt. Bei korrekter Anwendung wird das $CONTACTNAME$-Makro diesen Wert enthalten.

alias: Diese Direktive wird benutzt, um einen längeren Namen oder eine Beschreibung zu definieren, der/die den Kontakt identifiziert. Bei korrekter Anwendung wird das $CONTACTALIAS$-Makro diesen Alias/diese Beschreibung enthalten. Falls nicht angegeben, wird der *contact_name* als Alias verwendet.

contactgroups: Diese Direktive wird benutzt, um den/die *Kurznamen* der Kontaktgruppe(n) anzugeben, zu dem/denen der Kontakt gehört. Mehrere Kontaktgruppen werden durch Kommata von einander getrennt. Diese Direktive kann als Alternative (oder zusätzlich) zur *members*-Direktive in den contactgroup-Definitionen genutzt werden.

host_notifications_enabled: Diese Direktive wird benutzt, um festzulegen, ob der Kontakt Benachrichtigungen über Host-Probleme und Erholungen bekommt. Werte: 0 = keine Benachrichtigungen versenden, 1 = Benachrichtigungen versenden

service_notifications_enabled: Diese Direktive wird benutzt, um festzulegen, ob der Kontakt Benachrichtigungen über Service-Probleme und Erholungen bekommt. Werte: 0 = keine Benachrichtigungen versenden, 1 = Benachrichtigungen versenden

host_notification_period: Diese Direktive wird benutzt, um den Kurznamen des Zeitfensters anzugeben, in dem der Kontakt über Host-Probleme oder Erholungen informiert wird. Sie können dies als "Bereitschafts"-Zeiten dieses Kontakts für Host-Benachrichtigungen ansehen. Lesen Sie die Dokumentation zu Zeitfenstern, um mehr Informationen darüber zu erhalten, wie diese funktionieren und welche potenziellen Probleme durch unsachgemäßen Gebrauch entstehen können.

service_notification_period: Diese Direktive wird benutzt, um den Kurznamen des Zeitfensters anzugeben, in dem der Kontakt über Service-Probleme oder Erholungen informiert wird. Sie können dies als "Bereitschafts"-Zeiten dieses Kontakts für Service-Benachrichtigungen ansehen. Lesen Sie die Dokumentation zu Zeitfenstern, um mehr Informationen darüber zu erhalten, wie diese funktionieren und welche potenziellen Probleme durch unsachgemäßen Gebrauch entstehen können.

host_notification_commands: Diese Direktive wird benutzt, um *Kurznamen* von Befehlen anzugeben, die zur Benachrichtigung von Kontakten über *Host*-Probleme oder Erholungen benutzt werden. Mehrere Benachrichtigungsbefehle sollten durch Kommata von einander getrennt werden. Alle Benachrichtigungsbefehle werden ausgeführt, wenn der Kontakt informiert werden muss. Die maximale Zeit, die der Benachrichtigungsbefehl laufen darf, wird durch die notification_timeout-Option kontrolliert.

host_notification_options: Diese Direktive wird benutzt, um die Host-Zustände festzulegen, bei denen Benachrichtigungen an den Kontakt versandt werden. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte: **d** = Benachrichtigungen bei einem DOWN-Zustand versenden, **u** = Benachrichtigungen bei einem UNREACHABLE-Zustand versenden, **r** = Benachrichtigungen bei Erholungen (UP-Zustand) versenden, **f** = Benachrichtigungen versenden, wenn der Host mit Flattern anfängt bzw. aufhört und **s** = Benachrichtigungen versenden, wenn eine geplante Ausfallzeit anfängt oder aufhört. Wenn Sie **n** (none) als Option angeben, werden keine Host-Benachrichtigungen versandt.

service_notification_options: Diese Direktive wird benutzt, um die Service-Zustände festzulegen, bei denen Benachrichtigungen an den Kontakt versandt werden. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte: **w** = Benachrichtigungen bei einem WARNING-Zustand versenden, **u** = Benachrichtigungen bei einem UNKNOWN-Zustand versenden, **r** = Benachrichtigungen bei Erholungen (OK-Zustand) versenden, **f** = Benachrichtigungen versenden, wenn der Service mit Flattern anfängt bzw. aufhört und **s** = Benachrichtigungen versenden, wenn eine geplante Ausfallzeit anfängt oder aufhört. Wenn Sie **n** (none) als Option angeben, werden keine Service-Benachrichtigungen versandt.

service_notification_commands: Diese Direktive wird benutzt, um *Kurznamen* von Befehlen anzugeben, die zur Benachrichtigung von Kontakten über *Service*-Probleme oder Erholungen benutzt werden. Mehrere Benachrichtigungsbefehle sollten durch Kommata von einander getrennt werden. Alle Benachrichtigungsbefehle werden ausgeführt, wenn der Kontakt informiert werden muss. Die maximale Zeit, die der Benachrichtigungsbefehl laufen darf, wird durch die notification_timeout-Option kontrolliert.

email: Diese Direktive wird benutzt, um ein e-Mail-Adresse für den Kontakt zu definieren. Abhängig von Ihren Benachrichtigungsbefehlen kann sie benutzt werden, um eine Alarm-Mail an den Kontakt zu versenden. Bei korrekter Anwendung wird das $CONTACTEMAIL$-Makro diesen Wert enthalten.

pager:	Diese Direktive wird benutzt, um eine Pager-Nummer für den Kontakt zu definieren. Sie kann auch eine e-Mail-Adresse eines Pager-Gateways (z.B. pagejoe@pagenet.com) sein. Abhängig von Ihren Benachrichtigungsbefehlen kann sie benutzt werden, um eine Alarm-Page an den Kontakt zu versenden. Bei korrekter Anwendung wird das $CONTACTPAGER$-Makro diesen Wert enthalten.
address*x*:	Adress-Direktiven werden benutzt, um zusätzliche "Adressen" für den Kontakt zu definieren. Diese Adressen können alles sein - Mobiltelefonnummern, Instant-Messaging-Adressen usw. Abhängig von Ihren Benachrichtigungsbefehlen kann sie benutzt werden, um einen Alarm an den Kontakt zu versenden. Bis zu sechs Adressen können mit Hilfe dieser Direktiven definiert werden (*address1* bis *address6*). Das $CONTACTADDRESS*x*$-Makro wird diesen Wert enthalten.
can_submit_commands:	Diese Direktive wird benutzt, um festzulegen, ob dieser Kontakt über die CGIs externe Befehle an Nagios erteilen kann. Werte: 0 = dem Kontakt die Erteilung von Befehlen verweigern, 1 = dem Kontakt die Erteilung von Befehlen erlauben.
retain_status_information:	Diese Direktive wird benutzt, um festzulegen, ob zustandsbezogene Informationen zu diesem Kontakt über Programmneustarts hinweg aufbewahrt wird. Das ist nur sinnvoll, wenn Sie Statusaufbewahrung über die retain_state_information-Direktive aktiviert haben. Werte: 0 = Aufbewahrung von Statusinformationen deaktivieren, 1 = Aufbewahrung von Statusinformationen aktivieren.
retain_nonstatus_information:	Diese Direktive wird benutzt, um festzulegen, ob nicht-zustandsbezogene Informationen zu diesem Kontakt über Programmneustarts hinweg aufbewahrt wird. Das ist nur sinnvoll, wenn Sie Statusaufbewahrung über die retain_state_information-Direktive aktiviert haben. Werte: 0 = Aufbewahrung von nicht-Statusinformationen deaktivieren, 1 = Aufbewahrung von nicht-Statusinformationen aktivieren.

Kontaktgruppen-Definition

Beschreibung:

Eine Kontaktgruppen-Definition wird benutzt, um einen oder mehrere Kontakte zu gruppieren, um Alarm-/Erholungs-Benachrichtigungen zu versenden.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

define contactgroup{

contactgroup_name	*contactgroup_name*
alias	*alias*
members	*contacts*
contactgroup_members	*contactgroups*

}

Beispieldefinition:

```
define contactgroup{
        contactgroup_name       novell-admins
        alias                   Novell Administrators
        members                 jdoe,rtobert,tzach
        }
```

Beschreibung der Direktiven:

contactgroup_name:	Diese Direktive wird benutzt, um einen Kurznamen zu definieren, der die Kontaktgruppe identifiziert.
alias:	Diese Direktive wird benutzt, um einen längeren Namen oder eine Beschreibung zu definieren, der die Kontaktgruppe identifiziert.
members:	Dies ist eine Liste von *Kurznamen* der Kontakte, die in dieser Gruppe enthalten sein sollen. Mehrere Kontaktnamen sollten jeweils durch Komma von einander getrennt werden. Diese Direktive kann als Alternative (oder als Zusatz) zu der *contactgroups*-Direktive in den Kontakt-Definitionen verwendet werden.
contactgroup_members:	Diese optionale Direktive kann genutzt werden, um Kontakte aus anderen "sub"-Kontaktgruppen in diese Kontaktgruppe aufzunehmen. Geben Sie eine komma-separierte Liste von Kurznamen anderer Kontaktgruppen an, deren Mitglieder in diese Gruppe aufgenommen werden sollen.

Zeitfenster-Definition

Beschreibung:

Ein Zeitfenster ist eine Liste von Zeiten an verschiedenen Tagen, die als "gültige" Zeiten für Benachrichtigungen und Service-Prüfungen angesehen werden. Es besteht aus Zeitbereichen für jeden Tag der Woche. Verschiedene Ausnahmen zu den normalen wöchentlichen Zeiten werden unterstützt, u.a.: bestimmte Wochentage, bestimmte Tage eines Monats, Tage eines bestimmten Monats und Kalendertage.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

define timeperiod{

timeperiod_name	*timeperiod_name*
alias	*alias*
[weekday]	*timeranges*
[exception]	*timeranges*
exclude	[*timeperiod1,timeperiod2,...,timeperiodn*]

}

Beispiel-Definitionen:

```
define timeperiod{
        timeperiod_name         nonworkhours
        alias                   Non-Work Hours
        sunday                  00:00-24:00                     ; jeder Sonntag jeder Woche
        monday                  00:00-09:00,17:00-24:00         ; jeder Montag jeder Woche
        tuesday                 00:00-09:00,17:00-24:00         ; jeder Dienstag jeder Woche
        wednesday               00:00-09:00,17:00-24:00         ; jeder Mittwoch jeder Woche
        thursday                00:00-09:00,17:00-24:00         ; jeder Donnerstag jeder Woche
        friday                  00:00-09:00,17:00-24:00         ; jeder Freitag jeder Woche
        saturday                00:00-24:00                     ; jeder Samstag jeder Woche
        }

define timeperiod{
        timeperiod_name         misc-single-days
        alias                   Misc Single Days
        1999-01-28              00:00-24:00             ; 28. Januar 1999
        monday 3                00:00-24:00             ; 3. Montag im Monat
        day 2                   00:00-24:00             ; 2. Tag im Monat
        february 10             00:00-24:00             ; 10. Februar im Jahr
        february -1             00:00-24:00             ; letzter Tag im Februar
        friday -2               00:00-24:00             ; vorletzer Freitag im Monat
        thursday -1 november    00:00-24:00             ; letzter Donnerstag im November
        }

define timeperiod{
        timeperiod_name         misc-date-ranges
        alias                   Misc Date Ranges
        2007-01-01 - 2008-02-01         00:00-24:00             ; 1. Januar 2007 bis zum 1. Februar 2008
        monday 3 - thursday 4           00:00-24:00             ; 3. Montag bis 4. Donnerstag
        day 1 - 15                      00:00-24:00             ; 1. bis 15. Tag
        day 20 - -1                     00:00-24:00             ; 20. Tag bis Monatsende
        july 10 - 15                    00:00-24:00             ; 10. - 15. Juli
        april 10 - may 15               00:00-24:00             ; 10. April bis zum 15. Mai
        tuesday 1 april - friday 2 may  00:00-24:00             ; 1. Dienstag im April bis zum 2. Freitag im Mai
        }

define timeperiod{
        timeperiod_name         misc-skip-ranges
        alias                   Misc Skip Ranges
        2007-01-01 - 2008-02-01 / 3             00:00-24:00     ; jeder dritte Tag vom 1. Januar 2008 bis zum 1. Februar 2008
        2008-04-01 / 7                          00:00-24:00     ; jeder 7. Tag ab dem 1. April 2008 (ohne Endedatum)
        monday 3 - thursday 4 / 2               00:00-24:00     ; jeder zweite Tag vom 3. Montag bis zum 4. Donnerstag des Monats
        day 1 - 15 / 5                          00:00-24:00     ; jeder 5. Tag vom 1. bis zum 15. Tag des Monats
        july 10 - 15 / 2                        00:00-24:00     ; jeder zweite Tag vom 10. Juli bis zum 15.Juli
        tuesday 1 april - friday 2 may / 6      00:00-24:00     ; jeder sechste Tag vom 1. Dienstag im April bis zum 2. Freitag im Mai
        }
```

Beschreibung der Direktiven:

timeperiod_name: Diese Direktive ist der Kurzname, der benutzt wird, um das Zeitfenster zu identifizieren.

alias: Diese Direktive ist ein längerer Name oder eine Beschreibung zur Identifizierung des Zeitfensters.

[*weekday*]: Die Wochentags-Direktiven ("*sunday*" bis "*saturday*") sind komma-separierte Listen von Zeitbereichen, die "gültige" Zeiten für einen bestimmten Tag der Woche sind. Beachten Sie, dass es sieben verschiedene Tage gibt, für die Sie Zeitbereiche angeben können ("Sunday" bis "Saturday"). Jeder Zeitbereich hat die Form **HH:MM-HH:MM**, wobei die Stunden in einem 24-Stunden-Format angegeben werden. Wenn Sie einen kompletten Tag aus dem Zeitfenster ausschließen wollen, dann geben Sie ihn einfach nicht an.

[*exception*]: Sie können verschiedene Arten von Ausnahmen zum Standard-Wochentagsplan angeben. Ausnahmen können eine Reihe von verschiedenen Formen annehmen, u.a. einzelne Tage eines bestimmten oder jeden Monats, einzelne Wochentage eines Monats oder einzelner Kalenderdaten. Sie können auch einen Bereich von Tagen/Daten und sogar bestimmte Intervalle überspringen, um Funktionalitäten wie "alle drei Tage zwischen diesen Daten" zu erreichen. Statt alle möglichen von Ausnahmen anzugeben, zeige ich Ihnen anhand der o.g. Beispieldefinitionen, was möglich ist. :-) Wochentage und verschiedene Arten von Ausnahmen haben alle verschiedene Vorrangebenen, so dass es wichtig ist zu verstehen, wie sie sich gegenseitig beeinflussen. Mehr Informationen dazu finden Sie in der Dokumentation zu Zeitfenstern.

exclude: Diese Direktive wird benutzt, um die Kurznamen von anderen Zeitfenstern abzugeben, deren Zeitbereiche in diesem Zeitfenster ausgeschlossen werden sollen. Mehrere Zeitfensternamen sind durch Kommata von einander zu trennen.

Befehls-Definition

Beschreibung:

Eine Befehls-Definition ist genau das. Sie definiert einen Befehl. Befehle, die definiert werden können, umfassen u.a. Service-Prüfungen, Host-Benachrichtigungen und Host-Eventhandler.
Befehls-Definitionen können Makros enthalten, aber Sie müssen sicherstellen, dass Sie nur solche Makros verwenden, die unter den gegebenen Umständen "gültig" sind. Mehr Informationen dazu, welche Makros verfügbar und wann sie "gültig" sind, finden Sie hier. Die verschiedenen Argumente einer Befehls-Definition sehen Sie nachfolgend.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

```
define command{
        command_name    command_name
        command_line    command_line
        }
```

Beispieldefinition:

```
define command{
        command_name    check_pop
        command_line    /usr/local/nagios/libexec/check_pop -H $HOSTADDRESS$
        }
```

Beschreibung der Direktiven:

command_name: Diese Direktive ist der Kurzname, der zur Identifizierung des Befehls benutzt wird. Er wird u.a. in Kontakt-, Host- und Service-Definitionen (in notification-, check-, und event handler-Direktiven) verwendet.

command_line: Diese Direktive wird benutzt, um zu definieren, was wirklich durch Nagios ausgeführt wird, wenn der Befehl für Service- oder Host-Prüfungen, Benachrichtigungen oder Eventhandler benutzt wird. Vor der Ausführung der Kommandozeile werden alle gültigen Makros durch die entsprechenden Werte ersetzt. Lesen Sie die Dokumentation, um festzustellen, welche verschiedenen Makros Sie benutzen können. Beachten Sie, dass die Kommandozeile *nicht* von Anführungszeichen eingeschlossen wird. Achten Sie auch darauf, dass Sie bei der Übergabe eines Dollarzeichens ($) ein weiteres Dollarzeichen zur Maskierung benutzen müssen (aus bar$foo muss bar$$foo werden).

ANMERKUNG: Sie dürfen kein **Semikolon** (;) in der *command_line*-Direktive einsetzen, weil alles danach als Kommentar angesehen wird. Sie können diese Begrenzung umgehen, indem Sie eines der **$USER$**-Makros in Ihrem resource file mit einem Semikolon füllen und dann in der *command_line*-Direktive auf das entsprechende $USER$-Makro verweisen.

Wenn Sie während der Laufzeit Parameter an Befehle übergeben wollen, können Sie die **$ARGn$**-Makros in der *command_line*-Direktive der Befehlsdefinition benutzen und in den Objektdefinitions-Direktiven (Host-Prüfbefehl, Service-Eventhandler, usw.), die auf den Befehl verweisen, einzelne Argumente durch Ausrufezeichen (!) vom Befehlsnamen (und von einander) trennen. Mehr Informationen, wie Argumente in Befehlsdefinitionen während der Laufzeit verarbeitet werden, finden Sie in der Dokumentation zu Makros.

Service-Abhängigkeits-Definition

Beschreibung:

Service-Abhängigkeiten sind ein fortgeschrittenes Feature von Nagios, das es Ihnen erlaubt, Benachrichtungen und aktive Prüfungen von Services in Abhängigkeit vom Status eines oder mehrerer Services zu unterdrücken. Service-Abhängigkeiten sind optional und zielen hauptsächlich auf fortgeschrittene Benutzer mit komplizierten Überwachungsumgebungen. Mehr Informationen, wie Service-Abhängigkeiten arbeiten (lesen Sie dies!), finden Sie hier.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional. Trotz allem müssen Sie mindestens ein Kriterium angeben, damit die Definition von Nutzen ist.

define servicedependency{

dependent_host_name	*host_name*
dependent_hostgroup_name	*hostgroup_name*
dependent_service_description	*service_description*
host_name	*host_name*
hostgroup_name	*hostgroup_name*
service_description	*service_description*
inherits_parent	[0/1]
execution_failure_criteria	[o,w,u,c,p,n]
notification_failure_criteria	[o,w,u,c,p,n]
dependency_period	timeperiod_name

}

Beispieldefinition:

```
define servicedependency{
        host_name                       WWW1
        service_description             Apache Web Server
        dependent_host_name             WWW1
        dependent_service_description   Main Web Site
        execution_failure_criteria      n
        notification_failure_criteria   w,u,c
        }
```

Beschreibung der Direktiven:

dependent_host: Diese Direktive wird benutzt, um den/die *Kurznamen* des/der Hosts anzugeben, auf dem der *abhängige* Service "läuft" oder mit dem er verbunden ist. Mehrere Hosts werden durch Kommata von einander getrennt. Bleibt die Direktive leer, so kann dadurch eine "same host"-Abhängigkeit erstellt werden.

dependent_hostgroup: Diese Direktive wird benutzt, um den/die *Kurznamen* der Hostgruppe(n) anzugeben, auf der/denen der *abhängige* Service "läuft" oder mit dem er verbunden ist. Mehrere Hostgruppen werden durch Kommata von einander getrennt. Die dependent_hostgroup-Direktive kann statt der (oder zusätzlich zur) dependent_host-Direktive benutzt werden.

dependent_service_description: Diese Direktive wird benutzt, um die *Beschreibung* des *abhängigen* Service anzugeben.

host_name: Diese Direktive wird benutzt, um den/die *Kurznamen* des/der Hosts anzugeben, auf dem/denen der Service "läuft" oder mit dem/denen er verbunden ist, von dem "es" *abhängt* (auch als Master-Service bezeichnet). Mehrere Hosts werden durch Kommata von einander getrennt.

hostgroup_name: Diese Direktive wird benutzt, um den/die *Kurznamen* der Hostgruppe(n) anzugeben, auf der/denen der Service "läuft" oder mit der/denen er verbunden ist, von dem "es" *abhängt* (auch als Master-Service bezeichnet). Mehrere Hostgruppen werden durch Kommata von einander getrennt. Der hostgroup_name kann statt oder zusätzlich zur host_name-Direktive benutzt werden.

service_description: Diese Direktive wird benutzt, um die *Beschreibung* des Service anzugeben, von dem "es" *abhängt* (auch als Master-Service bezeichnet).

inherits_parent: Diese Direktive gibt an, ob die abhängige Definition Abhängigkeiten von dem Service erbt, von dem sie *abhängt* (auch als Master-Service bezeichnet). In anderen Worten, wenn der Master-Service von anderen Services abhängt und eine der Abhängigkeiten fehlschlägt, dann wird auch diese Abhängigkeit fehlschlagen.

execution_failure_criteria: Diese Direktive wird benutzt, um die Kriterien festzulegen, wann der abhängige Service *nicht* aktiv geprüft werden soll. Wenn der Master-Service in einem der Zustände ist, die wir angeben, wird der *abhängige* Service nicht aktiv geprüft. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte (mehrere Werte werden durch Kommata von einander getrennt): **o** = fehlschlagen bei einem OK-Zustand, **w** = fehlschlagen bei einem WARNING-Zustand, **u** = fehlschlagen bei einem UNKNOWN-Zustand, **c** = fehlschlagen bei einem CRITICAL-Zustand und **p** = fehlschlagen bei einem PENDING-Zustand (d.h. der Service wurde bisher noch nie geprüft). Wenn Sie **n** (none) als Option angeben, wird die Ausführungsabhängigkeit nie fehlschlagen und die Prüfungen des abhängigen Service werden immer erfolgen (falls andere Bedingungen das erlauben). Beispiel: wenn Sie **o,c,u** in diesem Feld angeben, dann wird der *abhängige* Service nicht aktiv geprüft, wenn der *Master*-Service sich in einem OK-, CRITICAL- oder UNKNOWN-Zustand befindet.

notification_failure_criteria:	Diese Direktive wird benutzt, um die Kriterien festzulegen, wann *keine* Benachrichtigungen für den abhängigen Service versandt werden sollen. Wenn der Master-Service in einem der Fehler-Zustände ist, die wir angeben, werden keine Benachrichtigungen für den *abhängigen* Service an die Kontakte versandt. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte: **o** = fehlschlagen bei einem OK-Zustand, **w** = fehlschlagen bei einem WARNING-Zustand, **u** = fehlschlagen bei einem UNKNOWN-Zustand, **c** = fehlschlagen bei einem CRITICAL-Zustand und **p** = fehlschlagen bei einem PENDING-Zustand (d.h. der Service wurde bisher noch nie geprüft). Wenn Sie **n** (none) als Option angeben, wird die Benachrichtigungsabhängigkeit nie fehlschlagen und die Benachrichtungen für den abhängigen Service werden immer erfolgen. Beispiel: wenn Sie **w** in diesem Feld angeben, dann werden die Benachrichtigungen für den *abhängigen* Service nicht versandt, wenn der *Master*-Service sich in einem WARNING-Zustand befindet.
dependency_period:	Diese Direktive wird benutzt, um den Kurznamen eines Zeitfensters anzugeben, in welchem diese Abhängigkeit gültig ist. Wenn diese Direktive nicht angegeben wird, ist die Abhängigkeit zu allen Zeiten gültig.

Serviceeskalations-Definition

Beschreibung:

Serviceeskalationen sind *komplett optional* und werden benutzt, um Benachrichtigungen für einen bestimmten Service zu eskalieren. Mehr Informationen, wie Eskalationen arbeiten, finden Sie hier.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

define serviceescalation{

host_name	*host_name*
hostgroup_name	*hostgroup_name*
service_description	*service_description*
contacts	*contacts*
contact_groups	*contactgroup_name*
first_notification	#
last_notification	#
notification_interval	#
escalation_period	timeperiod_name
escalation_options	[w,u,c,r]

}

Beispieldefinition:

```
define serviceescalation{
        host_name               nt-3
        service_description     Processor Load
        first_notification      4
        last_notification       0
        notification_interval   30
        contact_groups          all-nt-admins,themanagers
        }
```

Beschreibung der Direktiven:

host_name: Diese Direktive wird benutzt, um den/die *Kurznamen* des/der Hosts anzugeben, für den die Service-Eskalation gilt oder mit dem/denen er verbunden ist.

hostgroup_name: Diese Direktive wird benutzt, um den/die *Kurznamen* der Hostgruppen anzugeben, für den die Service-Eskalation gilt oder mit der/denen er verbunden ist. Mehrere Hostgruppen werden durch Kommata von einander getrennt. Der hostgroup_name kann statt oder zusätzlich zur host_name-Direktive benutzt werden.

service_description: Diese Direktive wird benutzt, um die *Beschreibung* des Service zu identifizieren, auf den die Eskalation zutreffen soll.

first_notification: Diese Direktive ist eine Zahl, die die *erste* Benachrichtigung angibt, für die diese Eskalation gilt. Wenn Sie beispielsweise den Wert auf 3 setzen, dann wird diese Eskalation nur dann benutzt, wenn der Service lang genug in einem nicht-OK-Zustand ist, damit eine dritte Benachrichtigung versandt wird.

last_notification: Diese Direktive ist eine Zahl, die die *letzte* Benachrichtigung angibt, für die diese Eskalation gilt. Wenn Sie beispielsweise den Wert auf 5 setzen, dann wird diese Eskalation nicht benutzt, wenn mehr als fünf Benachrichtigungen für den Service versandt werden. Wenn der Wert auf Null gesetzt wird, wird dieser Eskalationseintrag immer benutzt (unabhängig davon, wie viele Benachrichtigungen versandt werden.)

contacts: Dies ist eine Liste von *Kurznamen* der Kontakte, die informiert werden sollen, wenn es Probleme (oder Erholungen) für diesen Service gibt. Mehrere Kontakte werden durch Kommata von einander getrennt. Das ist nützlich, wenn Sie Benachrichtigungen nur an ein paar Leute verschicken wollen und keine Kontaktgruppen definieren wollen. Sie müssen mindestens einen Kontakt oder eine Kontaktgruppe in jeder Serviceeskalations-Definition angeben.

contact_groups: Dies ist eine Liste von *Kurznamen* der Kontaktgruppen, die informiert werden sollen, wenn die Service-Benachrichtigung eskaliert. Mehrere Kontaktgruppen werden durch Kommata von einander getrennt. Sie müssen mindestens einen Kontakt oder eine Kontaktgruppe in jeder Serviceeskalations-Definition angeben.

notification_interval: Diese Direktive wird benutzt, um das Intervall festzulegen, in dem Benachrichtigungen versandt werden, wenn diese Eskalation gültig ist. Wenn Sie einen Wert von 0 für dieses Intervall angeben, wird Nagios die erste Benachrichtigung versenden, wenn diese Eskalation gültig wird, dann aber verhindern, dass weitere Benachrichtigungen versandt werden. Benachrichtigungen werden wieder versandt, bis sich der Host erholt. Dies ist nützlich, wenn Sie nach einer bestimmten Zeit keine weiteren Benachrichtigungen versenden wollen. Anmerkung: Wenn mehrere Eskalationseinträge eines Hosts für ein oder mehr Benachrichtigungsbereiche überlappen, wird das kürzeste Benachrichtigungsintervall aller Eskalationseinträge benutzt.

escalation_period: Diese Direktive wird benutzt, um den Kurznamen des Zeitfensters anzugeben, in dem diese Eskalation gültig ist. Wenn diese Direktive nicht angegeben wird, ist diese Eskalation zu allen Zeiten gültig.

escalation_options: Diese Direktive wird benutzt, um die Kriterien festzulegen, wann diese Service-Eskalation benutzt wird. Diese Eskalation wird nur benutzt, wenn der Service in einem der Zustände ist, die in dieser Direktive angeben werden. Wenn diese Direktive nicht in einer Service-Eskalation angegeben wird, ist die Eskalation für alle Service-Zustände gültig. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte: **r** = eskalieren bei einem OK-(Erholungs)-Zustand, **w** = eskalieren bei einem WARNING-Zustand, **u** = eskalieren bei einem UNKNOWN-Zustand, und **c** = eskalieren bei einem CRITICAL-Zustand. Beispiel: wenn Sie **w** in diesem Feld angeben, dann wird die Eskalation nur benutzt, wenn sich der Service in einem WARNING-Zustand befindet.

Host-Abhängigkeits-Definition

Beschreibung:

Host-Abhängigkeiten sind ein fortgeschrittenes Feature von Nagios, das es Ihnen erlaubt, Benachrichtungen von Hosts in Abhängigkeit vom Status eines oder mehrerer Hosts zu unterdrücken. Host-Abhängigkeiten sind optional und zielen hauptsächlich auf fortgeschrittene Benutzer mit komplizierten Überwachungsumgebungen. Mehr Informationen, wie Host-Abhängigkeiten arbeiten (lesen Sie dies!), finden Sie hier.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

define hostdependency{

dependent_host_name	*host_name*
dependent_hostgroup_name	*hostgroup_name*
host_name	*host_name*
hostgroup_name	*hostgroup_name*
inherits_parent	[0/1]
execution_failure_criteria	[o,d,u,p,n]
notification_failure_criteria	[o,d,u,p,n]
dependency_period	timeperiod_name

}

Beispieldefinition:

```
define hostdependency{
        host_name                       WWW1
        dependent_host_name             DBASE1
        notification_failure_criteria   d,u
        }
```

Beschreibung der Direktiven:

dependent_host_name: Diese Direktive wird benutzt, um den/die *Kurznamen* des/der *abhängigen* Hosts zu identifizieren. Mehrere Hosts werden durch Kommata von einander getrennt.

dependent_hostgroup_name: Diese Direktive wird benutzt, um den/die *Kurznamen* der *abhängigen* Hostgruppe(n) zu identifizieren. Mehrere Hostgruppen werden durch Kommata von einander getrennt. Der dependent_hostgroup_name kann statt oder zusätzlich zur dependent_host_name-Direktive benutzt werden.

host_name: Diese Direktive wird benutzt, um den/die *Kurznamen* des/der Hosts anzugeben, von dem "es" *abhängt* (auch als Master-Host bezeichnet). Mehrere Hosts werden durch Kommata von einander getrennt.

hostgroup_name: Diese Direktive wird benutzt, um den/die *Kurznamen* der Hostgruppe(n) anzugeben, von dem "es" *abhängt* (auch als Master-Host bezeichnet). Mehrere Hostgruppen werden durch Kommata von einander getrennt. Der hostgroup_name kann statt oder zusätzlich zur host_name-Direktive benutzt werden.

inherits_parent: Diese Direktive gibt an, ob die abhängige Definition Abhängigkeiten von dem Host erbt, von dem sie *abhängt* (auch als Master-Host bezeichnet). In anderen Worten, wenn der Master-Host von anderen Hosts abhängt und eine der Abhängigkeiten fehlschlägt, dann wird auch diese Abhängigkeit fehlschlagen.

execution_failure_criteria: Diese Direktive wird benutzt, um die Kriterien festzulegen, wann der abhängige Host *nicht* aktiv geprüft werden soll. Wenn der Master-Host in einem der Zustände ist, die wir angeben, wird der *abhängige* Host nicht aktiv geprüft. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte (mehrere Werte werden durch Kommata von einander getrennt): **o** = fehlschlagen bei einem UP-Zustand, **u** = fehlschlagen bei einem UNREACHABLE-Zustand und **p** = fehlschlagen bei einem PENDING-Zustand (d.h. der Host wurde bisher noch nie geprüft). Wenn Sie **n** (none) als Option angeben, wird die Ausführungsabhängigkeit nie fehlschlagen und die Prüfungen des abhängigen Host werden immer erfolgen (falls andere Bedingungen das erlauben). Beispiel: wenn Sie **u,d** in diesem Feld angeben, dann wird der *abhängige* Host nicht aktiv geprüft, wenn der *Master*-Service sich in einem UNREACHABLE- oder DOWN-Zustand befindet.

notification_failure_criteria: Diese Direktive wird benutzt, um die Kriterien festzulegen, wann *keine* Benachrichtigungen für den abhängigen Host versandt werden sollen. Wenn der Master-Host in einem der Fehler-Zustände ist, die wir angeben, werden keine Benachrichtigungen für den *abhängigen* Host an die Kontakte versandt. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte: **o** = fehlschlagen bei einem UP-Zustand, **d** = fehlschlagen bei einem DOWN-Zustand, **u** = fehlschlagen bei einem UNREACHABLE-Zustand, und **p** = fehlschlagen bei einem PENDING-Zustand (d.h. der Host wurde bisher noch nie geprüft). Wenn Sie **n** (none) als Option angeben, wird die Benachrichtigungsabhängigkeit nie fehlschlagen und die Benachrichtungen für den abhängigen Host werden immer erfolgen. Beispiel: wenn Sie **d** in diesem Feld angeben, dann werden die Benachrichtigungen für den *abhängigen* Host nicht versandt, wenn der *Master*-Host sich in einem DOWN-Zustand befindet.

dependency_period: Diese Direktive wird benutzt, um den Kurznamen eines Zeitfensters anzugeben, in welchem diese Abhängigkeit gültig ist. Wenn diese Direktive nicht angegeben wird, ist die Abhängigkeit zu allen Zeiten gültig.

Host-Eskalations-Definition

Beschreibung:

Host-Eskalationen sind *komplett optional* und werden benutzt, um Benachrichtigungen für einen bestimmten Host zu eskalieren. Mehr Informationen, wie Eskalationen arbeiten, finden Sie hier.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional.

define hostescalation{

host_name	*host_name*
hostgroup_name	*hostgroup_name*
contacts	*contacts*
contact_groups	*contactgroup_name*
first_notification	#
last_notification	#
notification_interval	#
escalation_period	timeperiod_name
escalation_options	[d,u,r]

}

Beispieldefinition:

```
define hostescalation{
        host_name               router-34
        first_notification      5
        last_notification       8
        notification_interval   60
        contact_groups          all-router-admins
        }
```

Beschreibung der Direktiven:

host_name: Diese Direktive wird benutzt, um den/die *Kurznamen* des/der Hosts anzugeben, für den die Eskalation gilt.

hostgroup_name: Diese Direktive wird benutzt, um den/die *Kurznamen* der Hostgruppen anzugeben, für den die Eskalation gilt. Mehrere Hostgruppen werden durch Kommata von einander getrennt. Wenn diese Direktive benutzt wird, trifft die Eskalation auf alle Hosts zu, die Mitglied der angegebenen Hostgruppe(n) sind.

first_notification: Diese Direktive ist eine Zahl, die die *erste* Benachrichtigung angibt, für die diese Eskalation gilt. Wenn Sie beispielsweise den Wert auf 3 setzen, dann wird diese Eskalation nur dann benutzt, wenn der Host lang genug "down" oder unerreichbar ist, damit eine dritte Benachrichtigung versandt wird.

last_notification: Diese Direktive ist eine Zahl, die die *letzte* Benachrichtigung angibt, für die diese Eskalation gilt. Wenn Sie beispielsweise den Wert auf 5 setzen, dann wird diese Eskalation nicht benutzt, wenn mehr als fünf Benachrichtigungen für den Host versandt werden. Wenn der Wert auf Null gesetzt wird, wird dieser Eskalationseintrag immer benutzt (unabhängig davon, wie viele Benachrichtigungen versandt werden.)

contacts: Dies ist eine Liste von *Kurznamen* der Kontakte, die informiert werden sollen, wenn es Probleme (oder Erholungen) für diesen Host gibt. Mehrere Kontakte werden durch Kommata von einander getrennt. Das ist nützlich, wenn Sie Benachrichtigungen nur an ein paar Leute verschicken wollen und keine Kontaktgruppen definieren wollen. Sie müssen mindestens einen Kontakt oder eine Kontaktgruppe in jeder Hosteskalations-Definition angeben.

contact_groups: Dies ist eine Liste von *Kurznamen* der Kontaktgruppen, die informiert werden sollen, wenn die Host-Benachrichtigung eskaliert. Mehrere Kontaktgruppen werden durch Kommata von einander getrennt. Sie müssen mindestens einen Kontakt oder eine Kontaktgruppe in jeder Hosteskalations-Definition angeben.

notification_interval: Diese Direktive wird benutzt, um das Intervall festzulegen, in dem Benachrichtigungen versandt werden, wenn diese Eskalation gültig ist. Wenn Sie einen Wert von 0 für dieses Intervall angeben, wird Nagios die erste Benachrichtigung versenden, wenn diese Eskalation gültig wird, dann aber verhindern, dass weitere Benachrichtigungen versandt werden. Benachrichtigungen werden wieder versandt, bis sich der Host erholt. Dies ist nützlich, wenn Sie nach einer bestimmten Zeit keine weiteren Benachrichtigungen versenden wollen. Anmerkung: Wenn mehrere Eskalationseinträge eines Hosts für ein oder mehr Benachrichtungsbereiche überlappen, wird das kürzeste Benachrichtigungsintervall aller Eskalationseinträge benutzt.

escalation_period: Diese Direktive wird benutzt, um den Kurznamen des Zeitfensters anzugeben, in dem diese Eskalation gültig ist. Wenn diese Direktive nicht angegeben wird, ist diese Eskalation zu allen Zeiten gültig.

escalation_options: Diese Direktive wird benutzt, um die Kriterien festzulegen, wann diese Host-Eskalation benutzt wird. Diese Eskalation wird nur benutzt, wenn der Host in einem der Zustände ist, die in dieser Direktive angeben werden. Wenn diese Direktive nicht in einer Host-Eskalation angegeben wird, ist die Eskalation für alle Host-Zustände gültig. Gültige Optionen sind eine Kombination von einem oder mehreren folgender Werte: **r** = eskalieren bei einem UP-(Erholungs)-Zustand, **d** = eskalieren bei einem DOWN-Zustand und **u** = eskalieren bei einem UNREACHABLE-Zustand. Beispiel: wenn Sie **d** in diesem Feld angeben, dann wird die Eskalation nur benutzt, wenn sich der Host in einem DOWN-Zustand befindet.

erweiterte Hostinformations-Definition

Beschreibung:

Einträge für erweiterte Hostinformationen sind grundsätzlich dazu gedacht, die Ausgaben der status-, statusmap-, statuswrl- und extinfo-CGIs schöner aussehen zu lassen. Sie haben keinen Einfluss auf die Überwachung und sind vollständig optional.

 Hinweis: Ab Nagios 3.x sind alle Direktiven der erweiterten Hostinformations-Definition auch in den Host-Definitionen verfügbar. Dadurch können Sie entscheiden, die nachstehenden Direktiven in Ihren Host-Definitionen zu benutzen, wenn es Ihre Konfigurationen vereinfacht. Separate erweiterte Hostinformations-Definitionen werden weiterhin unterstützt, um Rückwärtskompatibilität zu gewährleisten.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional. Trotz allem müssen Sie mindestens ein Kriterium angeben, damit die Definition von Nutzen ist.

define hostextinfo{

host_name	*host_name*
notes	*note_string*
notes_url	*url*
action_url	*url*
icon_image	*image_file*
icon_image_alt	*alt_string*
vrml_image	*image_file*
statusmap_image	*image_file*
2d_coords	*x_coord,y_coord*
3d_coords	*x_coord,y_coord,z_coord*

}

Beispieldefinition:

```
define hostextinfo{
        host_name       netware1
        notes           This is the primary Netware file server
        notes_url       http://webserver.localhost.localdomain/hostinfo.pl?host=netware1
        icon_image      novell40.png
        icon_image_alt  IntranetWare 4.11
        vrml_image      novell40.png
        statusmap_image novell40.gd2
        2d_coords       100,250
        3d_coords       100.0,50.0,75.0
        }
```

Variablenbeschreibungen:

host_name: Diese Variable wird benutzt, um den/die *Kurznamen* des/der Hosts zu identifizieren, mit dem/denen diese Daten verbunden sind.

notes: Diese Direktive wird benutzt, um eine optionale Zeichenkette mit Anmerkungen zu definieren, die den Host betreffen. Wenn Sie hier eine Anmerkung angeben, werden Sie diese im extended Information-CGI sehen (wenn Sie Informationen zu dem bestimmten Host ansehen).

notes_url: Diese Variable wird benutzt, um einen optionalen URL zu definieren, der mehr Informationen zu diesem Host bereitstellt. Wenn Sie einen URL angeben, werden Sie im extended information-CGI einen Link namens "Extra Host Notes" sehen (wenn Sie Informationen zu dem bestimmten Host ansehen). Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*). Dies kann sehr nützlich sein, wenn Sie detaillierte Informationen zu diesem Host, Notfallkontaktmethoden usw. für anderes Support-Personal zur Verfügung stellen wollen.

action_url: Diese Variable wird benutzt, um einen optionalen URL zu definieren, der mehr Aktionen für diesen Host bereitstellt. Wenn Sie einen URL angeben, werden Sie im extended information-CGI einen Link namens "Extra Host Notes" sehen (wenn Sie Informationen zu dem bestimmten Host ansehen). Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*). Dies kann sehr nützlich sein, wenn Sie detaillierte Informationen zu diesem Host, Notfallkontaktmethoden usw. für anderes Support-Personal zur Verfügung stellen wollen.

icon_image: Diese Variable wird benutzt, um den Namen eines GIF-, PNG- oder JPG-Images anzugeben, das mit diesem Host verbunden werden soll. Dieses Bild wird in den status- und extended information-CGIs angezeigt. Das Bild wird am besten aussehen, wenn es 40x40 Pixel groß ist. Bilder für Hosts werden im **logos/**-Unterverzeichnis Ihres HTML-Images-Verzeichnis gesucht (d.h. */usr/local/nagios/share/images/logos*).

icon_image_alt: Diese Variable wird benutzt, um eine optionale Zeichenkette anzugeben, die für den ALT-Tag des Bildes benutzt wird, das durch das <*icon_image*>-Argument angegeben wurde. Das ALT-Tag wird in den status-, extended information- und statusmap-CGIs benutzt.

vrml_image: Diese Variable wird benutzt, um den Namen eines GIF-, PNG- oder JPG-Images anzugeben, das mit diesem Host verbunden werden soll. Dieses Bild wird als Textur-Map für den angegebenen Host in der statuswrl-CGI benutzt. Anders als das Bild, das Sie in der <*icon_image*>-Variable angeben, sollte dieses möglichst *keinerlei* Transparenz haben. Wenn es das tut, wird das Host-Objekt ein wenig komisch aussehen. Bilder für Hosts werden im **logos/**-Unterverzeichnis Ihres HTML-Images-Verzeichnis gesucht (d.h. */usr/local/nagios/share/images/logos*).

statusmap_image: Diese Variable wird benutzt, um den Namen eines Bildes anzugeben, das mit diesem Host im statusmap-CGI verbunden werden soll. Sie können ein JPG-, PNG- oder GIF-Bild angeben, aber ich würde zu einem Bild im GD2-Format raten, weil andere Bildformate zu hohen CPU-Belastungen führen können, wenn die Statusmap generiert wird. PNG-Bilder können mit Hilfe des **pngtogd2**-Utilitys (das in Thomas Boutell's gd library enthalten ist) ins GD2-Format umgewandelt werden. Die GD2-Bilder werden im *unkomprimierten* Format erstellt, um die CPU-Belastung zu minimieren, während das Statusmap-CGI das Netzwerkkartenbild erstellt. Das Bild wird am besten aussehen, wenn es 40x40 Pixel groß ist. Sie können diese Option leer lassen, wenn Sie das Statusmap-CGI nicht nutzen. Bilder für Hosts werden im **logos/**-Unterverzeichnis Ihres HTML-Images-Verzeichnis gesucht (d.h. */usr/local/nagios/share/images/logos*).

2d_coords: Diese Variable wird benutzt, um Koordinaten anzugeben, wenn der Host im statusmap-CGI gezeichnet wird. Koordinaten sollen in positiven Ganzzahlen angegeben werden, weil sie physischen Pixeln im generierten Bild entsprechen. Der Ursprung (0,0) für die Zeichnung ist die linke, obere Ecke des Bildes, das sich in die positive X-Richtung (nach rechts) und in die positive Y-Richtung (nach unten) erstreckt. Die Größe der Icons ist normalerweise etwa 40x40 Pixel (Text benötigt etwas mehr Platz). Die Koordinaten, die Sie angeben, beziehen sich auf die linke, obere Ecke des Icons. Anmerkung: Machen Sie sich keine Sorgen über die maximalen X- und Y-Koordinaten, die Sie benutzen können. Das CGI wird automatisch die maximale Größe des zu erstellenden Bildes aufgrund der größten X- und Y-Koordinaten festlegen, die Sie angegeben haben.

3d_coords: Diese Variable wird benutzt, um Koordinaten anzugeben, die beim Zeichnen des Hosts im statuswrl-CGI verwendet werden. Koordinaten können positive oder negative reelle Zahlen sein. Der Ursprung für die Zeichnung ist (0.0,0.0,0.0). Die Größe des Host-Kubus ist 0,5 Einheiten auf jeder Seite (Text benötigt etwas mehr Platz). Die Koordinaten, die Sie hier angeben, beziehen sich auf das Zentrum des Host-Kubus.

erweiterte Serviceinformations-Definition

Beschreibung:

Einträge für erweiterte Serviceinformationen sind grundsätzlich dazu gedacht, die Ausgaben der status- und extinfo-CGIs schöner aussehen zu lassen. Sie haben keinen Einfluss auf die Überwachung und sind vollständig optional.


Hinweis: Ab Nagios 3.x sind alle Direktiven der erweiterten Serviceinformations-Definition auch in den Service-Definitionen verfügbar. Dadurch können Sie entscheiden, die nachstehenden Direktiven in Ihren Service-Definitionen zu benutzen, wenn es Ihre Konfigurationen vereinfacht. Separate erweiterte Serviceinformations-Definitionen werden weiterhin unterstützt, um Rückwärtskompatibilität zu gewährleisten.

Format der Definition:

Anmerkung: Direktiven in rot werden benötigt, die in schwarz sind optional. Trotz allem müssen Sie mindestens ein Kriterium angeben, damit die Definition von Nutzen ist.

define serviceextinfo{

host_name	*host_name*
service_description	*service_description*
notes	*note_string*
notes_url	*url*
action_url	*url*
icon_image	*image_file*
icon_image_alt	*alt_string*

}

Beispieldefinition:

```
define serviceextinfo{
        host_name               linux2
        service_description     Log Anomalies
        notes                   Security-related log anomalies on secondary Linux server
        notes_url               http://webserver.localhost.localdomain/serviceinfo.pl?host=linux2&service=Log+Anomalies
        icon_image              security.png
        icon_image_alt          Security-Related Alerts
        }
```

Variablenbeschreibungen:

host_name: Diese Variable wird benutzt, um den/die *Kurznamen* des/der Hosts zu identifizieren, mit dem/denen der Service verbunden sind.

service_description: Diese ist die Beschreibung des Service, mit dem/denen diese Daten verbunden sind.

notes: Diese Direktive wird benutzt, um eine optionale Zeichenkette mit Anmerkungen zu definieren, die den Service betreffen. Wenn Sie hier eine Anmerkung angeben, werden Sie diese im extended Information-CGI sehen (wenn Sie Informationen zu dem bestimmten Service ansehen).

notes_url: Diese Variable wird benutzt, um einen optionalen URL zu definieren, der mehr Informationen zu diesem Service bereitstellt. Wenn Sie einen URL angeben, werden Sie im extended information-CGI einen Link namens "Extra Service Notes" sehen (wenn Sie Informationen zu dem bestimmten Service ansehen). Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*). Dies kann sehr nützlich sein, wenn Sie detaillierte Informationen zu diesem Host, Notfallkontaktmethoden usw. für anderes Support-Personal zur Verfügung stellen wollen.

action_url: Diese Variable wird benutzt, um einen optionalen URL zu definieren, der mehr Aktionen für diesen Service bereitstellt. Wenn Sie einen URL angeben, werden Sie im extended information-CGI einen Link namens "Extra Service Notes" sehen (wenn Sie Informationen zu dem bestimmten Service ansehen). Jeder gültige URL kann benutzt werden. Wenn Sie relative Pfade benutzen, wird der Basis-Pfad der gleiche sein, der benutzt wird, um auf die CGIs zuzugreifen (d.h. */cgi-bin/nagios/*). Dies kann sehr nützlich sein, wenn Sie detaillierte Informationen zu diesem Host, Notfallkontaktmethoden usw. für anderes Support-Personal zur Verfügung stellen wollen.

icon_image: Diese Variable wird benutzt, um den Namen eines GIF-, PNG- oder JPG-Images anzugeben, das mit diesem Service verbunden werden soll. Dieses Bild wird in den status- und extended information-CGIs angezeigt. Das Bild wird am besten aussehen, wenn es 40x40 Pixel groß ist. Bilder für Service werden im **logos/**-Unterverzeichnis Ihres HTML-Images-Verzeichnis gesucht (d.h. */usr/local/nagios/share/images/logos*).

icon_image_alt: Diese Variable wird benutzt, um eine optionale Zeichenkette anzugeben, die für den ALT-Tag des Bildes benutzt wird, das durch das <*icon_image*>-Argument angegeben wurde. Das ALT-Tag wird in den status-, extended information- und statusmap-CGIs benutzt.

Optionen CGI-Konfigurationsdatei

Hoch zu: Inhalt

Siehe auch: Konfigurationsübersicht, Informationen zu den CGIs, Authentifizierung und Autorisierung in den CGIs, CGI-Kopf- und Fußzeilen

Anmerkungen

Wenn Sie Konfigurationsdateien erstellen oder anpassen, beachten Sie bitte folgendes:

1. Zeilen, die mit einem '#'-Zeichen beginnen, werden als Kommentar betrachtet und nicht verarbeitet
2. Variablennamen müssen am Zeilenanfang beginnen - "white space" sind vor dem Namen NICHT erlaubt
3. Variablennamen sind abhängig von Groß- und Kleinschreibung

Beispielkonfiguration

Hinweis: eine Beispiel-CGI-Konfigurationsdatei (*/usr/local/nagios/etc/cgi.cfg*) wird für Sie installiert, wenn Sie der Schnellstart-Anleitung folgen.

Position der Konfigurationsdatei

Als Standard erwartet Nagios, dass die CGI-Konfigurationsdatei **cgi.cfg** heißt und zusammen mit der Hauptkonfigurationsdatei im Verzeichnis für die Konfigurationsdateien liegt. Wenn Sie den Namen der Datei oder die Position ändern müssen, dann können Sie Apache so konfigurieren, dass eine Umgebungsvariable namens NAGIOS_CGI_CONFIG übergeben wird (die auf die korrekte Position der CGIs verweist). Lesen Sie in der Apache-Dokumentation nach, wie das zu tun ist.

Variablen der Konfigurationsdatei

Nachfolgend finden Sie Beschreibungen jeder Option der Hauptkonfigurationsdatei...

Position der Hauptkonfigurationsdatei

Format: **main_config_file=<file_name>**

Beispiel: **main_config_file=/usr/local/nagios/etc/nagios.cfg**

Dies gibt die Position Ihrer Hauptkonfigurationsdatei an. Die CGIs müssen wissen, wo sie zu finden ist, um Informationen zu Konfigurationsinformationen, aktuellen Host- und Service-Zuständen usw. zu bekommen.

vollständiger (physical) HTML-Pfad

Format: **physical_html_path=<Pfad>**

Beispiel: **physical_html_path=/usr/local/nagios/share**

Dies ist der *vollständige* Pfad zu den HTML-Dateien von Nagios auf Ihrer Workstation oder Ihrem Server. Nagios nimmt an, dass die Dokumentation und die Bilddateien (die von den CGIs benutzt werden) in Unterverzeichnissen namens *docs/* und *images/* gespeichert sind.

URL-HTML-Pfad

Format: **url_html_path=<Pfad>**

Beispiel: **url_html_path=/nagios**

Wenn Sie Nagios über einen Web-Browser mit einem URL wie **http://www.myhost.com/nagios**, aufrufen, sollte dieser Wert */nagios* sein. Grundsätzlich ist es der Pfadanteil des URL, der zum Aufruf der Nagios-HTML-Seiten benutzt wird.

Nutzung der Authentifizierung

Format: **use_authentication=<0/1>**

Beispiel: **use_authentication=1**

Diese Option kontrolliert, ob die CGIs die Authentifizierungs- und Autorisierungsfunktionalität nutzen, um den Zugang von Benutzern zu Informationen und Befehlen zu prüfen oder nicht. Ich möchte dringend raten, dass Sie die Authentifizierungsfunktionalität für die CGIs nutzen. Wenn Sie sich entscheiden, die Authentifizierung nicht zu nutzen, dann stellen Sie sicher, dass das command CGI entfernt wird, um nicht autorisierte Benutzer an der Ausführung von Nagios-Befehlen zu hindern. Das CGI wird keine Befehle ausführen, wenn Authentifizierung deaktiviert ist, aber ich würde trotzdem dazu raten, das CGI zu entfernen, damit man auf der sicheren Seite ist. Mehr Informationen zur Einstellung der Authentifizierung und der Konfiguration von Autorisierung für die CGIs finden Sie hier.

- 0 = die Authentifizierungsfunktionalität nicht nutzen
- 1 = die Authentifizierungs- und Autorisierungsfunktionalität nutzen (Default)

Standard-Benutzername

Format: **default_user_name=<username>**

Beispiel: **default_user_name=guest**

Das Setzen dieser Variable definiert einen Standard-Benutzernamen, der die CGIs aufrufen kann. Dies erlaubt es Leuten in einer sicheren Domäne (d.h. hinter einer Firewall) die CGIs aufzurufen, ohne dass sie sich am Web-Server authentifizieren müssen. Sie können das benutzen, um die Basis-Authentifizierung zu verhindern, wenn Sie keinen sicheren Server einsetzen, weil Basis-Authentifizierung Passwörter im Klartext über das Internet überträgt.

Wichtig: Definieren Sie *keinen* Standard-Benutzernamen, solange Sie nicht einen sicheren Web-Server haben und sicher sind, dass sich jeder, der die CGIs aufruft, in irgendeiner Weise authentifiziert hat. Wenn Sie diese Variable definieren, dann wird jeder, der sich am Web-Server authentifiziert, alle Rechte dieses Benutzers erben!

Zugang zu System/Prozessinformationen

Format: **authorized_for_system_information=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_system_information=nagiosadmin,theboss**

Dies ist eine Komma-separierte Liste von Namen von *authentifizierten Benutzern*, die System/Prozessinformationen im extended information CGI ansehen können. Benutzer in dieser Liste sind *nicht* automatisch autorisiert, System/Prozessbefehle zu erteilen. Wenn Sie möchten, dass Benutzer auch System/Prozessbefehle erteilen können, dann müssen Sie diese der authorized_for_system_commands-Variable hinzufügen. Mehr Informationen, wie man Authentifizierung einrichtet und Autorisierung für die CGIs konfiguriert, finden Sie hier.

Zugang zu System/Prozessbefehlen

Format: **authorized_for_system_commands=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_system_commands=nagiosadmin**

Dies ist eine Komma-separierte Liste von Namen von *authentifizierten Benutzern*, die System/Prozessbefehle über das command CGI erteilen können. Benutzer in dieser Liste sind *nicht* automatisch autorisiert, System/Prozessinformationen anzusehen. Wenn Sie möchten, dass Benutzer auch System/Prozessinformationen ansehen können, dann müssen Sie diese der authorized_for_system_information-Variable hinzufügen. Mehr Informationen, wie man Authentifizierung einrichtet und Autorisierung für die CGIs konfiguriert, finden Sie hier.

Zugang zu Konfigurationsinformationen

Format: **authorized_for_configuration_information=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_configuration_information=nagiosadmin**

Dies ist eine Komma-separierte Liste von Namen von *authentifizierten Benutzern*, die Konfigurationsinformationen im configuration CGI ansehen können. Benutzer in dieser Liste können Informationen zu allen konfigurierten Hosts, Hostgruppen, Kontakten, Kontaktgruppen, Zeitfenstern und Befehlen ansehen. Mehr Informationen, wie man Authentifizierung einrichtet und Autorisierung für die CGIs konfiguriert, finden Sie hier.

Global Host Information Access

Format: **authorized_for_all_hosts=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_all_hosts=nagiosadmin,theboss**

Dies ist eine Komma-separierte Liste von Namen von *authentifizierten Benutzern*, die Status- und Konfigurationsinformationen im für alle Hosts ansehen können. Benutzer in dieser Liste sind automatisch autorisiert, Informationen zu allen Services anzusehen. Benutzer in dieser Liste sind *nicht* automatisch berechtigt, Befehle für alle Hosts oder Services zu erteilen. Wenn Sie möchten, dass Benutzer auch Befehle für alle Hosts oder Services erteilen können, dann müssen Sie diese der authorized_for_all_host_commands-Variable hinzufügen. Mehr Informationen, wie man Authentifizierung einrichtet und Autorisierung für die CGIs konfiguriert, finden Sie hier.

Zugang zu globalen Host-Befehlen

Format: **authorized_for_all_host_commands=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_all_host_commands=nagiosadmin**

Dies ist eine Komma-separierte Liste von Namen von *authentifizierten Benutzern*, die Befehle für alle Hosts über das command CGI erteilen können. Benutzer in dieser Liste sind auch automatisch autorisiert, Befehle für alle Services zu erteilen. Benutzer in dieser Liste sind *nicht* automatisch berechtigt, Status- oder Konfigurationsinformationen für alle Hosts oder Services anzusehen. Wenn Sie möchten, dass Benutzer auch Status- und Konfigurationsinformationen für alle Hosts oder Services ansehen können, dann müssen Sie diese der authorized_for_all_hosts-Variable hinzufügen. Mehr Informationen, wie man Authentifizierung einrichtet und Autorisierung für die CGIs konfiguriert, finden Sie hier.

Zugang zu globalen Service-Informationen

Format: **authorized_for_all_services=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_all_services=nagiosadmin,theboss**

Dies ist eine Komma-separierte Liste von Namen von *authentifizierten Benutzern*, die Status- und Konfigurationsinformationen für alle Services ansehen können. Benutzer in dieser Liste sind *nicht* automatisch autorisiert, Informationen zu allen Hosts anzusehen. Benutzer in dieser Liste sind *nicht* automatisch berechtigt, Befehle für alle Services zu erteilen. Wenn Sie möchten, dass Benutzer auch Befehle für alle Services erteilen können, dann müssen Sie diese der authorized_for_all_service_commands-Variable hinzufügen. Mehr Informationen, wie man Authentifizierung einrichtet und Autorisierung für die CGIs konfiguriert, finden Sie hier.

Zugang zu globalen Service-Befehlen

Format: **authorized_for_all_service_commands=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_all_service_commands=nagiosadmin**

Dies ist eine Komma-separierte Liste von Namen von *authentifizierten Benutzern*, die Befehle für alle Services über das command CGI erteilen können. Benutzer in dieser Liste sind *nicht* automatisch autorisiert, Befehle für alle Hosts zu erteilen. Benutzer in dieser Liste sind *nicht* automatisch berechtigt, Status- oder Konfigurationsinformationen für alle Hosts anzusehen. Wenn Sie möchten, dass Benutzer auch Status- und Konfigurationsinformationen für alle Services ansehen können, dann müssen Sie diese der authorized_for_all_services-Variable hinzufügen. Mehr Informationen, wie man Authentifizierung einrichtet und Autorisierung für die CGIs konfiguriert, finden Sie hier.

Autorennamen sperren

Format: **lock_author_names=[0/1]**

Beispiel: **lock_author_names=1**

Diese Option erlaubt es Ihnen, Benutzer daran zu hindern, den Autorennamen zu ändern, wenn sie Kommentare, Bestätigungen und geplanten Ausfallzeiten über das Web-Interface eingeben. Wenn diese Option aktiviert ist, können Benutzer nicht mit der Befehlsanfrage verbundene Autorennamen ändern.

- 0 = Benutzern erlauben, den Autorennamen zu bei der Erteilung von Befehlen zu ändern
- 1 = Benutzer daran hindern, den Autorennamen zu ändern (default)

Statusmap CGI Background Image

Format: **statusmap_background_image=<image_file>**

Beispiel: **statusmap_background_image=smbackground.gd2**

Diese Option erlaubt es Ihnen, ein Bild anzugeben, das als Hintergrund im statusmap CGI benutzt wird, wenn Sie die Layout-Methode mit benutzerdefinierten Koordinaten benutzen. Das Hintergrundbild ist nicht in anderen Layout-Methoden verfügbar. Es wird angenommen, dass sich das Bild im HTML-Image-Pfad befindet (d.h. in /usr/local/nagios/share/images). Dieser Pfad wird automatisch durch das Anhängen von "/images" an den in der physical_html_path-Direktive ermittelt. Anmerkung: Die Bilddatei kann im GIF-, JPEG-, PNG- oder GD2-Format sein. Das GD2-Format (vorzugsweise im unkomprimierten Format) wird empfohlen, weil es die CPU-Belastung reduziert, wenn das CGI das Kartenbild generiert.

Standard-Statusmap-Layout-Methode

Format: **default_statusmap_layout=<layout_number>**

Beispiel: **default_statusmap_layout=4**

Gültige Werte sind:

<layout_number>-Wert	Layout-Methode
0	User-defined coordinates
1	Depth layers
2	Collapsed tree
3	Balanced tree
4	Circular
5	Circular (Marked Up)
6	Circular (Balloon)

Statuswrl CGI Include World

Format: **statuswrl_include=<vrml_file>**

Beispiel: **statuswrl_include=myworld.wrl**

Diese Option erlaubt es Ihnen, Ihre eigenen Objekte in der generierten VRML-Welt einzusetzen. Es wird angenommen, dass sich die Datei in dem Pfad befindet, der in der physical_html_path-Direktive angegeben ist. Anmerkung: diese Datei muss eine vollqualifizierte VRML-Welt sein (d.h. Sie können sie in einem VRML-Browser ansehen).

Standard-Statuswrl-Layout-Methode

Format: **default_statuswrl_layout=<layout_number>**

Beispiel: **default_statuswrl_layout=4**

Diese Option erlaubt es Ihnen, die Standard-Layout-Methode anzugeben, die in der statuswrl CGI benutzt wird. Gültige Optionen sind:

<layout_number>-Wert	Layout-Methode
0	User-defined coordinates
2	Collapsed tree
3	Balanced tree
4	Circular

CGI-Aktualisierungsrate

Format: **refresh_rate=<rate_in_seconds>**

Beispiel: **refresh_rate=90**

Diese Option erlaubt es Ihnen, die Anzahl von Sekunden zwischen Seitenaktualisierungen für die status-, statusmap- und extinfo-CGIs festzulegen.

Audio-Alarme

Format: **host_unreachable_sound=<sound_file>**
host_down_sound=<sound_file>
service_critical_sound=<sound_file>
service_warning_sound=<sound_file>
service_unknown_sound=<sound_file>

Beispiele: **host_unreachable_sound=hostu.wav**
host_down_sound=hostd.wav
service_critical_sound=critical.wav
service_warning_sound=warning.wav
service_unknown_sound=unknown.wav

Diese Option erlaubt es Ihnen, eine Audio-Datei anzugeben, die in Ihrem Browser abgespielt wird, wenn es ein Problem gibt, während Sie das status CGI ansehen. Wenn es mehrere Probleme gibt, wird die Datei für das kritischste Problem abgespielt. Das kritischste Problem sind ein oder mehrere nicht erreichbare Host, während das am wenigsten kritische Problem Services in einem UNKNOWN-Zustand sind (beachten Sie die Reihenfolge im obigen Beispiel). Audio-Dateien werden im **media**-Unterverzeichnis Ihres HTML-Verzeichnisses erwartet (d.h. */usr/local/nagios/share/media*).

Ping-Syntax

Format: **ping_syntax=<command>**

Beispiel: **ping_syntax=/bin/ping -n -U -c 5 $HOSTADDRESS$**

Diese Option legt fest, welche Syntax benutzt wird, wenn ein Host vom WAP-Interface aus ange"ping"t wird (mit Hilfe des statuswml CGI. Sie müssen den kompletten Pfad zum ping-Binary zusammen mit allen benötigten Optionen angeben. Das $HOSTADDRESS$-Makro wird durch die Adresse des Hosts ersetzt, bevor der Befehl ausgeführt wird.

Escape HTML Tags Option

Format: **escape_html_tags=[0/1]**

Beispiel: **escape_html_tags=1**

Diese Option legt fest, ob HTML-Tags in Host- und Service-(Plugin-)Ausgaben in CGIs unberücksichtigt bleiben oder nicht. Wenn Sie diese Option aktivieren, wird die Plugin-Ausgabe keine anklickbaren Hyperlinks enthalten.

Notes URL Target

Format: **notes_url_target=[target]**

Beispiel: **notes_url_target=_blank**

Diese Option legt den Namen des Ziel-Frames fest, in dem Anmerkungs-URLs angezeigt werden sollen. Gültige Optionen umfassen *_blank*, *_self*, *_top*, *_parent* oder jeden anderen gültigen Zielnamen.

Action URL Target

Format: **action_url_target=[target]**

Beispiel: **action_url_target=_blank**

Diese Option legt den Namen des Ziel-Frames fest, in dem Aktions-URLs angezeigt werden sollen. Gültige Optionen umfassen *_blank*, *_self*, *_top*, *_parent* oder jeden anderen gültigen Zielnamen.

Splunk-Integrationsoption

Format: **enable_splunk_integration=[0/1]**

Beispiel: **enable_splunk_integration=1**

Diese Option legt fest, ob die Integration mit Splunk im Web-Interface aktiviert ist oder nicht. Wenn sie aktiviert ist, werden an verschiedenen Stellen "Splunk It"-Links in den CGIs angezeigt (Log-Datei, Alarmhistorie, Host-/Service-Details, usw.). Das ist nützlich, wenn Sie nach den Ursachen suchen, warum ein bestimmtes Problem auftrat. Für mehr Informationen über Splunk besuchen Sie http://www.splunk.com/.

Splunk-URL

Format: **splunk_url=<path>**

Beispiel: **splunk_url=http://127.0.0.1:8000/**

Diese Option wird benutzt, um den Basis-URL zu Ihrem Splunk-Interface zu definieren. Dieser URL wird von den CGIs benutzt, wenn Links erzeugt werden, falls die enable_splunk_integration-Option aktiviert ist.

Authentifizierung und Autorisierung in den CGIs

Hoch zu: Inhalt

Siehe auch: Optionen CGI-Konfigurationdatei, Informationen zu den CGIs

Einführung

Dieses Dokument beschreibt, wie die Nagios-CGIs entscheiden, wer die Überwachungs- und Konfigurationsinformationen sehen darf und wer über das Web-Interface Befehle an den Nagios-Daemon erteilen darf.

Definitionen

Bevor wir fortfahren, ist es wichtig, dass Sie die Bedeutung und den Unterschied zwischen authentifizierten Benutzern und authentifizierten Kontakten verstehen:

- Ein **authentifizierter Benutzer** ist jemand, der sich dem Web-Server gegenüber mit Benutzer und Passwort authentifiziert hat und dem Zugang zum Nagios-Web-Interface gewährt wurde.
- Ein **authentifizierter Kontakt** ist ein authentifizierter Benutzer, dessen Benutzername mit dem Kurznamen einer Kontakt-Definition übereinstimmt.

Erstellen von authentifizierten Benutzern

Wenn wir annehmen, dass Sie Ihren Web-Server wie in der Schnellstart-Anleitung konfiguriert haben, dann sollte er Sie dazu auffordern, sich zu authentifizieren, bevor Sie die Nagios-CGIs benutzen können. Sie sollten außerdem ein Benutzerkonto (*nagiosadmin*) haben, das Zugang zu den CGIs hat.

Während Sie weitere Kontakte definieren, um Host- und Service-Benachrichtigungen zu erhalten, möchten Sie wahrscheinlich auch, dass sie Zugang zum Nagios-Web-Interface haben. Sie können den folgenden Befehl benutzen, um zusätzliche Benutzer hinzuzufügen, die sich bei den CGI authentifizieren können. Ersetzen Sie <username> durch den Benutzernamen, den Sie hinzufügen möchten. In den meisten Fällen sollte der Benutzername mit dem Kurznamen eines Kontakts übereinstimmen, den Sie definiert haben.

```
htpasswd /usr/local/nagios/etc/htpasswd.users <username>
```

aktivieren der Authentifizierungs/Autorisierungsfunktionalität in den CGIs

Als nächstes sollten Sie sicherstellen, dass die CGI so konfiguriert sind, dass sie die Authentifizierungs- und Autorisierungsfunktionalität nutzen, um festzulegen, welchen Zugang Benutzer zu Informationen und/oder Befehlen haben. Dies wird durch die use_authentication-Variable in der CGI-Konfigurationsdatei erreicht, die einen Wert ungleich Null haben muss. Beispiel:

```
use_authentication=1
```

Okay, nun sind Sie fertig mit dem Einstellen der grundlegenden Authentifizierungs- und Autorisierungsfunktionalität in den CGIs.

Standardberechtigungen für CGI-Informationen

Welche Standardberechtigungen haben Benutzer in den CGIs, wenn die Authentifizierungs- und Autorisierungsfunktionalität aktiviert ist?

CGI-Daten	Authentifizierte Kontakte*	andere authentifizierte Benutzer*
Host-Statusinformationen	Ja	Nein
Host-Konfigurationsinformationen	Ja	Nein
Host-Verlauf	Ja	Nein
Host-Benachrichtigungen	Ja	Nein
Host-Befehle	Ja	Nein
Service-Statusinformationen	Ja	Nein
Service-Konfigurationsinformationen	Ja	Nein
Service-Verlauf	Ja	Nein
Service-Benachrichtigungen	Ja	Nein
Service-Befehle	Ja	Nein
Alle Konfigurationsinformationen	Nein	Nein
System/Prozessinformationen	Nein	Nein
System/Prozessbefehle	Nein	Nein

*Authentifizierten Kontakten** werden die folgenden Berechtigungen für jeden **Service** gewährt, bei dem sie als Kontakt eingetragen sind (aber "Nein" für Services, bei denen sie nicht als Kontakt eingetragen sind)...

- Autorisierung, um Service-Statusinformationen zu sehen
- Autorisierung, um Service-Konfigurationsinformationen zu sehen
- Autorisierung, um Verlauf und Benachrichtigungen für den Service zu sehen
- Autorisierung, um Service-Befehle zu erteilen

*Authentifizierten Kontakten** werden die folgenden Berechtigungen für jeden **Host** gewährt, bei dem sie als Kontakt eingetragen sind (aber "Nein" für Hosts, bei denen sie nicht als Kontakt eingetragen sind)...

- Autorisierung, um Host-Statusinformationen zu sehen
- Autorisierung, um Host-Konfigurationsinformationen zu sehen
- Autorisierung, um Verlauf und Benachrichtigungen für den Host zu sehen
- Autorisierung, um Host-Befehle zu erteilen
- Autorisierung, um Statusinformationen für alle Services des Hosts zu sehen
- Autorisierung, um Konfigurationsinformationen für alle Services des Hosts zu sehen
- Autorisierung, um Verlauf und Benachrichtigungen für alle Services des Host zu sehen
- Autorisierung, um Befehle für alle Services des Hosts zu erteilen

Es ist wichtig anzumerken, dass als Grundeinstellung **keiner** autorisiert ist, das Folgende zu tun:

- die Log-Datei über das showlog CGI anzusehen
- Nagios-Prozessinformationen über das extended information CGI anzusehen
- Nagios-Prozessbefehle über das command CGI zu erteilen
- Definitionen für Hostgruppen, Kontakte, Kontaktgruppen, Zeitfenster und Befehle über das configuration CGI anzusehen

Sie werden unzweifelhaft Zugang zu diesen Informationen haben wollen, so dass Sie wie unten beschrieben zusätzliche Rechte für sich (und vielleicht andere Benutzer) zuweisen möchten.

Zusätzliche Berechtigungen zu CGI-Informationen gewähren

Mir ist klar, dass die verfügbaren Optionen es nicht erlauben, sehr genau auf bestimmte Berechtigungen einzugehen, aber es ist besser als nichts...

Benutzern können zusätzliche Autorisierungen gegeben werden, indem sie den folgenden Variablen in der CGI-Konfigurationsdatei hinzugefügt werden...

- authorized_for_system_information
- authorized_for_system_commands
- authorized_for_configuration_information
- authorized_for_all_hosts
- authorized_for_all_host_commands
- authorized_for_all_services
- authorized_for_all_service_commands

CGI-Autorisierungsanforderungen

Wenn Sie verwirrt sind, welche Autorisierung Sie benötigen, um Zugang zu verschiedenen Informationen in den CGIs zu bekommen, lesen Sie hier den Abschnitt ***Autorisierungsanforderungen***, in dem jedes CGI beschrieben ist.

Authentifizierung auf sicheren Web-Servern

Wenn Ihr Web-Server in einer sicheren Domäne steht (d.h. hinter einer Firewall) oder wenn Sie SSL benutzen, dann können Sie einen Standard-Benutzernamen definieren, der verwendet werden kann, um die CGI aufzurufen. Dies wird durch die Definition der default_user_name-Option in der CGI-Konfigurationsdatei erreicht. Durch die Definition eines Standard-Benutzernamens, der die CGIs aufrufen kann, können Sie Benutzern erlauben, die CGIs aufzurufen, ohne dass sie sich am Web-Server authentifizieren müssen. Sie möchten das vielleicht nutzen, um die Verwendung der Basis-Web-Authentifizierung zu verhindern, weil diese Passwörter im Klartext über das Internet überträgt.

Wichtig: Definieren Sie *keinen* Standard-Benutzernamen, solange Sie nicht einen sicheren Web-Server haben und sicher sind, dass sich jeder, der die CGIs aufruft, in irgendeiner Weise authentifiziert hat. Wenn Sie diese Variable definieren, dann wird jeder, der sich am Web-Server authentifiziert, alle Rechte dieses Benutzers erben!

Überprüfen Ihrer Konfiguration

Hoch zu: Inhalt

Siehe auch: Konfigurationsübersicht, Nagios starten und stoppen

Überprüfen Ihrer Konfiguration

Jedes Mal, nachdem Sie Ihre Konfigurationsdateien verändert haben, sollten Sie sie überprüfen. Es ist wichtig, das zu tun, bevor Sie Nagios (neu)starten, weil Nagios herunterfährt, wenn Ihre Konfiguration Fehler enthält.

Um Ihre Konfiguration zu überprüfen, starten Sie Nagios mit der **-v**-Option wie folgt:

```
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

Wenn Sie vergessen haben, kritische Daten einzugeben oder Dinge fehlkonfiguriert haben, spuckt Nagios eine Warnung oder eine Fehlermeldung aus, die Ihnen die Stelle des Problems zeigen sollte. Fehlermeldungen geben grundsätzlich die Zeile der Konfigurationsdatei aus, die die Ursache des Problems zu sein scheint. Bei Fehlern wird Nagios oft sofort die Überprüfung beenden und bereits nach der Ausgabe des ersten Fehlers zur Kommandozeile zurückkehren. Das geschieht, weil dieser erste Fehler im Laufe der restlichen Konfigurationdatei(en) weitere Fehler nach sich ziehen könnte. Sobald Sie Fehlermeldungen bekommen, müssen Sie Ihre Konfigurationsdateien editieren, um das Problem zu beheben. Warnungen können *generell* problemlos ignoriert werden, weil sie lediglich Empfehlungen darstellen und keine Erfordernisse für den Betrieb.

Sobald Sie Ihre Konfigurationsdateien überprüft und eventuelle Fehler bereinigt haben, können Sie Nagios (neu)starten

Nagios starten und stoppen

Hoch zu: Inhalt

Siehe auch: Überprüfen Ihrer Konfiguration

Es gibt mehr als einen Weg, um Nagios zu starten, zu stoppen und erneut zu starten. Hier einige der üblichen...

Hinweis: Stellen Sie immer sicher, dass Sie Ihre Konfiguration überprüfen, bevor Sie Nagios (neu)starten.

Nagios starten

1. **Init-Script:** Der einfachste Weg, den Nagios-Daemon zu starten, ist die Nutzung des Init-Scripts:

   ```
   /etc/rc.d/init.d/nagios start
   ```

2. **manuell:** Sie können Nagios manuell mit der **-d**-Kommandozeilenoption wie folgt starten:

   ```
   /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg
   ```

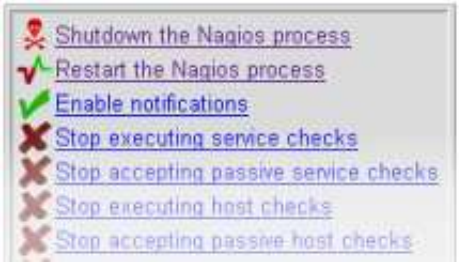
Nagios erneut starten

Ein Neustart bzw. neu laden ist notwendig, wenn Sie Ihre Konfigurationsdateien verändert haben und diese Änderungen aktiv werden sollen.

1. **Init-Script:** Der einfachste Weg, den Nagios-Daemon neu zu starten, ist die Nutzung des Init-Scripts:

   ```
   /etc/rc.d/init.d/nagios reload
   ```

2. **Web-Interface:** Sie können Nagios mit Hilfe des Web-Interfaces neu starten. Klicken Sie auf den "Process Info"-Navigations-Link und wählen Sie "Restart the Nagios process":

3. **manuell:** Sie können den Nagios-Prozess durch Senden eines SIGHUP-Signals neu starten:

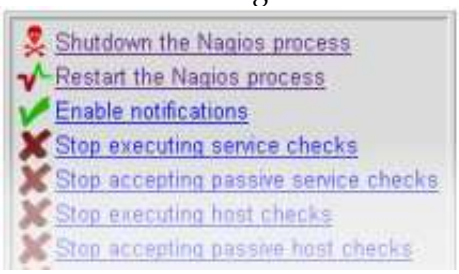
   ```
   kill -HUP <nagios_pid>
   ```

Nagios stoppen

1. **Init-Script:** Der einfachste Weg, den Nagios-Daemon zu stoppen, ist die Nutzung des Init-Script:

```
/etc/rc.d/init.d/nagios stop
```

2. **Web-Interface:** Sie können Nagios mit Hilfe des Web-Interfaces stoppen. Klicken Sie auf den "Process Info"-Navigations-Link und wählen Sie "Shutdown the Nagios process":



3. **manuell:** Sie können den Nagios-Prozess durch Senden eines SIGTERM-Signals stoppen:

```
kill <nagios_pid>
```

Nagios Plugins

Hoch zu: Inhalt

Siehe auch: Plugin API, Embedded Perl Interpreter Übersicht, Aktive Prüfungen

Einführung

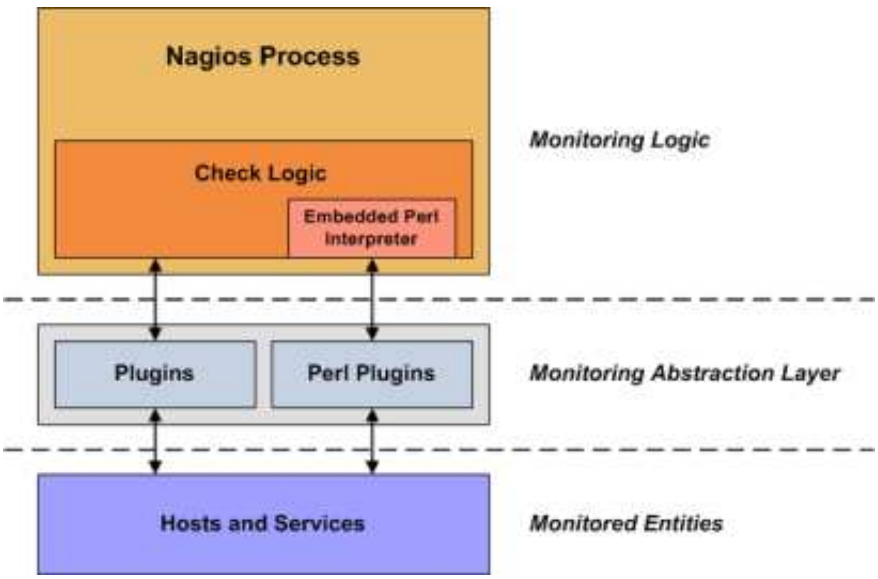
Nagios enthält nicht, wie viele andere Überwachungs-Tools, interne Mechanismen zur Prüfung des Zustands von Hosts und Services in Ihrem Netzwerk. Nagios verlässt sich statt dessen auf externe Programme (Plugins genannt), die all die schmutzige Arbeit tun.

Was sind Plugins?

Plugins sind kompilierte Programme oder Scripts (Perl-Scripts, Shell-Scripts, usw.), die von einer Kommandozeile aus laufen können, um den Status eines Hosts oder Service zu prüfen. Nagios benutzt die Ergebnisse von Plugins, um den aktuellen Status von Hosts oder Services in Ihrem Netzwerk zu ermitteln.

Nagios wird ein Plugin immer dann ausführen, wenn die Notwendigkeit besteht, den Status eines Hosts oder Service zu prüfen. Das Plugin tut *etwas* (beachten Sie den sehr allgemeinen Ausdruck), um die Prüfung auszuführen und dann einfach die Ergebnisse an Nagios zurückzuliefern. Nagios wird die Ergebnisse verarbeiten, die es vom Plugin erhält, und dann notwendige Aktionen ausführen (starten von Eventhandlern, senden von Benachrichtigungen, etc).

Plugins als eine Abstraktionsschicht



Plugins arbeiten wie eine Abstraktionsschicht zwischen der Überwachungslogik im Nagios-Dämon und den eigentlichen Services und Hosts, die überwacht werden.

Der Vorteil dieses Typs von Plugin-Architektur ist, dass Sie fast alles überwachen können, was Ihnen einfällt. Wenn Sie den Prozess der Überwachung automatisieren können, können Sie es mit Nagios überwachen. Es gibt bereits eine Menge von Plugins, die erzeugt wurden, um grundlegende Ressourcen wie z.B. Prozessorauslastung, Plattenbelegung, Ping-Raten usw. zu überwachen. Wenn Sie etwas anderes überwachen möchten, werfen Sie einen Blick in die Dokumentation zu Plugins schreiben und erstellen Sie ein eigenes. Es ist einfach!

Der Nachteil dieses Typs von Plugin-Architektur ist die Tatsache, dass Nagios absolut keine Ahnung davon hat, was Sie überwachen. Sie könnten Netzwerkverkehr-Statistiken, Datenfehler-Raten, Raumtemperatur, CPU-Spannung, Lüftergeschwindigkeit, Prozessorauslastung, Plattenbelegung überwachen oder die Fähigkeit Ihres superphantastischen Toasters, am Morgen Ihr Brot ordnungsgemäß zu bräunen... Nagios versteht nicht die Besonderheiten dessen, was überwacht wird - es verfolgt lediglich Veränderungen des *Zustands* dieser Ressourcen. Nur die Plugins selbst wissen genau, was sie überwachen und wie die eigentlichen Prüfungen auszuführen sind.

Welche Plugins sind verfügbar?

Es gibt bereits zahlreiche Plugins, um viele verschiedene Arten von Geräten und Services zu überwachen, u.a.:

- HTTP, POP3, IMAP, FTP, SSH, DHCP
- CPU-Auslastung, Plattenbelegung, Speicherauslastung, Anzahl Benutzer
- Unix/Linux, Windows- und Netware-Server
- Router und Switches
- etc.

Plugins beschaffen

Plugins werden nicht mit Nagios verteilt, aber Sie finden die offiziellen Nagios-Plugins zum Download und viele weitere Plugins, die von Nagios-Benutzern erstellt und gewartet werden, an folgenden Stellen:

- Nagios Plugins Project: http://nagiosplug.sourceforge.net/
- Nagios Downloads Page: http://www.nagios.org/download/
- NagiosExchange.org: http://www.nagiosexchange.org/

Wie benutze ich Plugin *X*?

Fast alle Plugins zeigen grundlegende Bedienungshinweise an, wenn sie von der Kommandozeile mit der Option '-h' oder '--help' aufgerufen werden. Wenn Sie z.B. wissen möchten, wie das Plugins check_http arbeitet bzw. welche Optionen es akzeptiert, sollten Sie folgenden Befehl ausprobieren:

```
./check_http --help
```

Plugin API

Informationen zu technischen Aspekten von Plugins sowie zur Erstellung Ihrer eigenen Plugins finden Sie hier.

Nagios®

Makros verstehen und wie sie arbeiten

Hoch zu: Inhalt

Siehe auch: Liste verfügbarer Makros

Makros

Eine der Haupteigenschaften, die Nagios so flexibel machen, ist die Fähigkeit, Makros in Befehlsdefinitionen zu benutzen. Makros erlauben Ihnen, Bezug auf Informationen von Hosts, Services und anderen Quellen zu nehmen.

Makroersetzungen - wie Makros arbeiten

Bevor Nagios einen Befehl ausführt, ersetzt es jedes Makro, das es in der Befehlsdefinition findet, durch den entsprechenden Wert. Diese Makroersetzung erfolgt für alle Arten von Befehlen, die Nagios ausführt - Host- und Service-Checks, Benachrichtigungen, Eventhandler usw.

Bestimmte Makros können wieder Makros enthalten. Dazu zählen die Makros $HOSTNOTES$, $HOSTNOTESURL$, $HOSTACTIONURL$, $SERVICENOTES$, $SERVICENOTESURL$ und $SERVICEACTIONURL$.

Beispiel 1: Host-Address Makro

Wenn Sie Host- und Service-Makros in Befehlsdefinitionen benutzen, beziehen sich diese auf Werte für den Host oder Service, für den der Befehl ausgeführt wird. Nehmen wir ein Beispiel. Angenommen, wir benutzen eine Host-Definition und einen *check_ping*-Befehl, die wie folgt definiert sind:

```
define host{
        host_name       linuxbox
        address         192.168.1.2
        check_command   check_ping
        ...
        }

define command{
        command_name    check_ping
        command_line    /usr/local/nagios/libexec/check_ping -H $HOSTADDRESS$ -w 100.0,90% -c 200.0,60%
        }
```

die erweiterte/endgültige auszuführende Befehlszeile für die Host-Prüfung würde so aussehen:

```
/usr/local/nagios/libexec/check_ping -H 192.168.1.2 -w 100.0,90% -c 200.0,60%
```

Ziemlich einfach, stimmt's? Die Schönheit liegt darin, dass Sie eine einzelne Befehlsdefinition für eine unbegrenzte Zahl von Hosts nutzen können. Jeder Host kann mit der selben Befehlsdefinition geprüft werden, weil jede Host-Adresse automatisch vor der Ausführung in der Befehlszeile ersetzt wird.

Beispiel 2: Befehlsargument-Makros

Sie können auch Argumente an Befehle übergeben, was recht handlich ist, wenn Sie Ihre Befehlsdefinitonen ziemlich generisch halten möchten. Argumente werden in der Objektdefinition (d.h. Host oder Service) angegeben, indem sie durch Ausrufezeichen (!) vom Befehlsnamen getrennt werden:

```
define service{
        host_name               linuxbox
        service_description     PING
        check_command           check_ping!200.0,80%!400.0,40%
        ...
        }
```

Im obigen Beispiel hat der Service-Check zwei Argumente (auf die mit $ARGn$-Makros zugegriffen werden kann). Das $ARG1$-Makro wird "200.0,80%" und $ARG2$ wird "400.0,40%" (beide ohne Anführungszeichen). Angenommen, wir benutzen die vorher angegebene Host-Definiton und einen wie folgt definierten *check_ping*-Befehl:

```
define command{
        command_name    check_ping
        command_line    /usr/local/nagios/libexec/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$
        }
```

die erweiterte/endgültige auszuführende Befehlszeile für die Service-Prüfung würde so aussehen:

```
        /usr/local/nagios/libexec/check_ping -H 192.168.1.2 -w 200.0,80% -c 400.0,40%
```

 Hinweis: Falls Sie Ausrufezeichen (!) in Ihren Argumenten übergeben müssen, dann können Sie das tun, indem Sie diese mit einem Backslash (\) maskieren. Falls Sie Backslashes in Ihren Argumenten einsetzen müssen, sind diese ebenfalls mit Backslashes zu maskieren.

On-Demand-Makros

Wenn Sie Host- und Service-Makros in Ihren Befehlsdefinitionen benutzen, dann beziehen sie sich normalerweise auf Werte des Hosts oder Service, für den der Befehl ausgeführt wird. Wenn beispielsweise eine Host-Prüfung für einen Host namens "linuxbox" ausgeführt wird, werden sich all die Standard-Host-Makros auf Werte für diesen Host beziehen ("linuxbox").

Wenn Sie möchten, dass sich die Werte eines Befehls auf einen anderen Host oder Service beziehen (für den der Befehl nicht ausgeführt wird), dann können Sie die sogenannten "On-Demand-Makros" benutzen. On-Demand-Makros sehen wie normale Makros aus, außer der Tatsache, dass sie einen Bezeichner für den Host oder Service enthalten, von dem sie ihren Wert erhalten sollen. Hier das grundsätzliche Format von On-Demand-Makros:

- $*HOSTMACRONAME*:*host_name*$
- $*SERVICEMACRONAME*:*host_name*:*service_description*$

Ersetzen Sie *HOSTMACRONAME* und *SERVICEMACRONAME* durch den Namen eines der Standard-Host- oder Service-Makros, die hier zu finden sind.

Beachten Sie, dass der Makroname durch einen Doppelpunkt (:) vom Host- oder Service-Bezeichner getrennt ist. Für On-Demand-Service-Makros besteht der Service-Bezeichner aus einem Host-Namen und einer Service-Beschreibung - sie sind ebenfalls durch einen Doppelpunkt (:) voneinander getrennt.

 Hinweis: On-Demand-Service-Makros können ein leeres Host-Namen-Feld enthalten. In diesem Fall wird automatisch der Name des Hosts benutzt, der mit dem Service verbunden ist.

Beispiele für On-Demand-Host- und Service-Makros folgen:

```
$HOSTDOWNTIME:myhost$                          <--- On-Demand-Host-Makro
$SERVICESTATEID:novellserver:DS Database$      <--- On-Demand-Service-Makro
$SERVICESTATEID::CPU Load$                     <--- On-Demand-Service-Makro mit leerem Host-Namen-Feld
```

On-Demand-Makros gibt es auch für hostgroup-, servicegroup-, contact- und contactgroup-Makros. Zum Beispiel:

```
$CONTACTEMAIL:john$                            <--- On-Demand-Contact-Makro
$CONTACTGROUPMEMBERS:linux-admins$             <--- On-Demand-Contactgroup-Makro
$HOSTGROUPALIAS:linux-servers$                 <--- On-Demand-Hostgroup-Makro
$SERVICEGROUPALIAS:DNS-Cluster$                <--- On-Demand-Servicegroup-Makro
```

On-Demand-Gruppen-Makros

Sie können die Werte eines Makros über alle Kontakte, Hosts oder Services in einer bestimmten Gruppe mit einem speziellen Format Ihrer On-Demand-Makrodeklaration erhalten. Sie tun dies, indem Sie auf eine bestimmte Hostgruppe, Servicegruppe oder Kontaktgruppe in einem On-Demand-Makro verweisen und zwar wie folgt:

- $*HOSTMACRONAME*:*hostgroup_name*:*delimiter*$
- $*SERVICEMACRONAME*:*servicegroup_name*:*delimiter*$
- $*CONTACTMACRONAME*:*contactgroup_name*:*delimiter*$

Ersetzen Sie *HOSTMACRONAME*, *SERVICEMACRONAME* und *CONTACTMACRONAME* durch den Namen eines der Standard-Host-, Service- oder Kontaktmakros, die Sie hier finden. Der Begrenzer (delimiter), den Sie angeben, wird benutzt, um Makrowerte der einzelnen Gruppenmitglieder von einander zu trennen.

Das folgende Makro wird beispielsweise eine komma-separierte Liste von Host-Status-IDs zurückliefern, die Mitglieder der *hg1*-Hostgruppe sind:

```
$HOSTSTATEID:hg1:,$
```

Diese Makrodefinition wird etwas zurückliefern, was etwa so aussieht:

```
0,2,1,1,0,0,2
```

Benutzervariablen-Makros

Jede Benutzerobjekt-Variable, die Sie in Host-, Service- oder Contact-Definitionen einsetzen, ist auch in Makros verfügbar. Benutzervariablen-Makros werden wie folgt benannt:

- $_HOST*varname*$
- $_SERVICE*varname*$
- $_CONTACT*varname*$

Nehmen Sie die folgende Host-Definition mit einer "_MACADDRESS" genannten Benutzervariablen...

```
define host{
        host_name       linuxbox
        address         192.168.1.1
        _MACADDRESS     00:01:02:03:04:05
        ...
        }
```

Die Benutzervariable _MACADRESS wäre in einem Makro $_HOSTMACADDRESS$ verfügbar. Weitere Informationen zu Benutzervariablen und wie sie in Makros eingesetzt werden können, finden Sie hier.

Makrobereinigung

Einige Makros werden von potenziell gefährlichen Shell-Metazeichen bereinigt, bevor Ersetzungen in Befehlen stattfinden. Welche Zeichen aus den Makros entfernt werden, hängt von den Einstellungen der illegal_macro_output_chars-Direktive ab. Die folgenden Makros werden von potenziell gefährlichen Zeichen bereinigt:

1. $HOSTOUTPUT$
2. $LONGHOSTOUTPUT$
3. $HOSTPERFDATA$
4. $HOSTACKAUTHOR$
5. $HOSTACKCOMMENT$
6. $SERVICEOUTPUT$
7. $LONGSERVICEOUTPUT$
8. $SERVICEPERFDATA$
9. $SERVICEACKAUTHOR$
10. $SERVICEACKCOMMENT$

Makros als Umgebungsvariablen

Die meisten Makros werden als Umgebungsvariablen zur Verfügung gestellt, um einen einfachen Einsatz in Scripts oder Befehlen zu ermöglichen, die von Nagios ausgeführt werden. Aus Gründen der Sicherheit und der Vernunft werden $USERn$ und "on-demand" Host- und Service-Makros nicht als Umgebungsvariablen zur Verfügung gestellt.

Umgebungsvariablen, die Standard-Makros enthalten, werden ebenso wie ihre entsprechenden Makronamen benannt (hier aufgeführt), wobei ihnen "NAGIOS_" vorangestellt wird. Beispielsweise wäre das $HOSTNAME$-Makro als Umgebungsvariable "NAGIOS_HOSTNAME" verfügbar.

Verfügbare Makros

Eine Liste aller in Nagios verfügbaren Makros sowie eine Tabelle, wann sie eingesetzt werden können, finden Sie hier.

Standard-Makros in Nagios

Hoch zu: Inhalt

Siehe auch: Wie Makros arbeiten

In Nagios verfügbare Standard-Makros sind hier aufgelistet. On-Demand-Makros und Makros für Benutzervariablen sind hier beschrieben.

Makro-Geltungsbereich

Obwohl Makros in allen Befehlen benutzt werden können, die Sie definieren, sind sie ggf. nicht "gültig" innerhalb eines bestimmten Befehlstyps. Zum Beispiel sind einige Makros vielleicht nur gültig bei Service-Benachrichtigungen, andere vielleicht nur bei Host-Prüfungen. Es gibt zehn Arten von Befehlen, die Nagios erkennt und unterschiedlich behandelt. Dies sind:

1. Service-Prüfungen
2. Service-Benachrichtigungen
3. Host-Prüfungen
4. Host-Benachrichtigungen
5. Service-Eventhandler und/oder ein globaler Service-Eventhandler
6. Host-Eventhandler und/oder ein globaler Host-Eventhandler
7. OCSP Befehl
8. OCHP Befehl
9. Service-Performance-Daten Befehle
10. Host-Performance-Daten Befehle

Die nachfolgenden Aufstellungen enthalten alle aktuell in Nagios verfügbaren Makros zusammen mit einer kurzen Beschreibung und den Befehlstypen, in denen sie gelten. Wenn ein Makro in einem Befehl benutzt wird, in dem es nicht gültig ist, wird es durch eine leere Zeichenkette ersetzt. Es ist zu beachten, dass Makros aus Großbuchstaben bestehen und in Dollarzeichen ($) eingeschlossen werden.

Makroverfügbarkeits-Aufstellung

Legende:

Nein	Das Makro ist nicht verfügbar
Ja	Das Makro ist verfügbar

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
Host-Makros: [3]								
$HOSTNAME$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTDISPLAYNAME$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTALIAS$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTADDRESS$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTSTATE$	Ja	Ja	Ja [1]	Ja	Ja	Ja	Ja	Ja
$HOSTSTATEID$	Ja	Ja	Ja [1]	Ja	Ja	Ja	Ja	Ja
$LASTHOSTSTATE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$LASTHOSTSTATEID$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja

$HOSTSTATETYPE$	Ja	Ja	Ja [1]	Ja	Ja	Ja	Ja	Ja
$HOSTATTEMPT$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$MAXHOSTATTEMPTS$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTEVENTID$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$LASTHOSTEVENTID$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTPROBLEMID$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$LASTHOSTPROBLEMID$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTLATENCY$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTEXECUTIONTIME$	Ja	Ja	Ja [1]	Ja	Ja	Ja	Ja	Ja
$HOSTDURATION$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTDURATIONSEC$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTDOWNTIME$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTPERCENTCHANGE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTGROUPNAME$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTGROUPNAMES$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$LASTHOSTCHECK$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$LASTHOSTSTATECHANGE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$LASTHOSTUP$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$LASTHOSTDOWN$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$LASTHOSTUNREACHABLE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTOUTPUT$	Ja	Ja	Ja [1]	Ja	Ja	Ja	Ja	Ja
$LONGHOSTOUTPUT$	Ja	Ja	Ja [1]	Ja	Ja	Ja	Ja	Ja
$HOSTPERFDATA$	Ja	Ja	Ja [1]	Ja	Ja	Ja	Ja	Ja
$HOSTCHECKCOMMAND$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTACKAUTHOR$ [8]	Nein	Nein	Nein	Ja	Nein	Nein	Nein	Nein
$HOSTACKAUTHORNAME$ [8]	Nein	Nein	Nein	Ja	Nein	Nein	Nein	Nein
$HOSTACKAUTHORALIAS$ [8]	Nein	Nein	Nein	Ja	Nein	Nein	Nein	Nein
$HOSTACKCOMMENT$ [8]	Nein	Nein	Nein	Ja	Nein	Nein	Nein	Nein
$HOSTACTIONURL$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTNOTESURL$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTNOTES$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$TOTALHOSTSERVICES$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$TOTALHOSTSERVICESOK$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$TOTALHOSTSERVICESWARNING$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$TOTALHOSTSERVICESUNKNOWN$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$TOTALHOSTSERVICESCRITICAL$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
Hostgroup-Makros:								
$HOSTGROUPALIAS$ [5]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTGROUPMEMBERS$ [5]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTGROUPNOTES$ [5]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTGROUPNOTESURL$ [5]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTGROUPACTIONURL$ [5]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
Service-Makros:								
$SERVICEDESC$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEDISPLAYNAME$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICESTATE$	Ja [2]	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICESTATEID$	Ja [2]	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LASTSERVICESTATE$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LASTSERVICESTATEID$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICESTATETYPE$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEATTEMPT$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$MAXSERVICEATTEMPTS$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEISVOLATILE$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEEVENTID$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LASTSERVICEEVENTID$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEPROBLEMID$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LASTSERVICEPROBLEMID$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICELATENCY$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEEXECUTIONTIME$	Ja [2]	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEDURATION$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEDURATIONSEC$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEDOWNTIME$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein

$SERVICEPERCENTCHANGE$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEGROUPNAME$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEGROUPNAMES$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LASTSERVICECHECK$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LASTSERVICESTATECHANGE$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LASTSERVICEOK$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LASTSERVICEWARNING$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LASTSERVICEUNKNOWN$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LASTSERVICECRITICAL$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEOUTPUT$	Ja [2]	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$LONGSERVICEOUTPUT$	Ja [2]	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEPERFDATA$	Ja [2]	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICECHECKCOMMAND$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICEACKAUTHOR$ [8]	Nein	Ja	Nein	Nein	Nein	Nein	Nein	Nein
$SERVICEACKAUTHORNAME$ [8]	Nein	Ja	Nein	Nein	Nein	Nein	Nein	Nein
$SERVICEACKAUTHORALIAS$ [8]	Nein	Ja	Nein	Nein	Nein	Nein	Nein	Nein
$SERVICEACKCOMMENT$ [8]	Nein	Ja	Nein	Nein	Nein	Nein	Nein	Nein
$SERVICEACTIONURL$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICENOTESURL$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein
$SERVICENOTES$	Ja	Ja	Nein	Nein	Ja	Nein	Ja	Nein

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
Servicegroup-Makros:								
$SERVICEGROUPALIAS$ [6]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$SERVICEGROUPMEMBERS$ [6]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$SERVICEGROUPNOTES$ [6]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$SERVICEGROUPNOTESURL$ [6]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$SERVICEGROUPACTIONURL$ [6]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
Contact-Makros:								
$CONTACTNAME$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$CONTACTALIAS$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$CONTACTEMAIL$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$CONTACTPAGER$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$CONTACTADDRESSn$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
Contactgroup-Makros:								
$CONTACTGROUPALIAS$ [7]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$CONTACTGROUPMEMBERS$ [7]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
Auswertungsmakros:								
$TOTALHOSTSUP$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALHOSTSDOWN$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALHOSTSUNREACHABLE$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALHOSTSDOWNUNHANDLED$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALHOSTSUNREACHABLEUNHANDLED$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALHOSTPROBLEMS$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALHOSTPROBLEMSUNHANDLED$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALSERVICESOK$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALSERVICESWARNING$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALSERVICESCRITICAL$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALSERVICESUNKNOWN$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALSERVICESWARNINGUNHANDLED$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALSERVICESCRITICALUNHANDLED$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALSERVICESUNKNOWNUNHANDLED$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALSERVICEPROBLEMS$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja
$TOTALSERVICEPROBLEMSUNHANDLED$ [10]	Ja	Ja [4]	Ja	Ja [4]	Ja	Ja	Ja	Ja

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
Benachrichtigungsmakros:								

$NOTIFICATIONTYPE$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$NOTIFICATIONRECIPIENTS$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$NOTIFICATIONISESCALATED$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$NOTIFICATIONAUTHOR$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$NOTIFICATIONAUTHORNAME$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$NOTIFICATIONAUTHORALIAS$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$NOTIFICATIONCOMMENT$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$HOSTNOTIFICATIONNUMBER$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$HOSTNOTIFICATIONID$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$SERVICENOTIFICATIONNUMBER$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein
$SERVICENOTIFICATIONID$	Nein	Ja	Nein	Ja	Nein	Nein	Nein	Nein

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
Datums-/Zeitmakros:								
$LONGDATETIME$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$SHORTDATETIME$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$DATE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$TIME$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$TIMET$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$ISVALIDTIME:$ [9]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$NEXTVALIDTIME:$ [9]	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
Dateimakros:								
$MAINCONFIGFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$STATUSDATAFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$COMMENTDATAFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja [5]
$DOWNTIMEDATAFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$RETENTIONDATAFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$OBJECTCACHEFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$TEMPFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$TEMPPATH$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$LOGFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$RESOURCEFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$COMMANDFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$HOSTPERFDATAFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$SERVICEPERFDATAFILE$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja

Makroname	Service-Prüfungen	Service-Benachrichtigungen	Host-Prüfungen	Host-Benachrichtigungen	Service-Eventhandler und OCSP	Host-Eventhandler und OCHP	Service-Perf-Daten	Host-Perf-Daten
verschiedene Makros:								
$PROCESSSTARTTIME$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$EVENTSTARTTIME$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$ADMINEMAIL$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$ADMINPAGER$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$ARGn$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
$USERn$	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja

Makrobeschreibungen

Host-Makros: [3]	
$HOSTNAME$	Kurzname für den Host (z.B. "biglinuxbox"). Dieser Wert wird aus der *host_name*-Direktive in der Host-Definition genommen.
$HOSTDISPLAYNAME$	Ein alternativer Anzeigename für den Host. Dieser Wert wird aus der *display_name*-Direktive in der Host-Definition genommen.
$HOSTALIAS$	Langname/Beschreibung für den Host. Dieser Wert wird aus der *alias*-Direktive in der Host-Definition genommen.

$HOSTADDRESS$	Adresse des Hosts. Dieser Wert wird aus der *address*-Direktive in der Host-Definition genommen.
$HOSTSTATE$	Eine Zeichenkette, die den aktuellen Status des Hosts angibt ("UP", "DOWN" oder "UNREACHABLE").
$HOSTSTATEID$	Eine Zahl, die dem aktuellen Status des Hosts entspricht: 0=UP, 1=DOWN, 2=UNREACHABLE.
$LASTHOSTSTATE$	Eine Zeichenkette, die den letzten Status des Hosts angibt ("UP", "DOWN" oder "UNREACHABLE").
$LASTHOSTSTATEID$	Eine Zahl, die dem letzten Status des Hosts entspricht: 0=UP, 1=DOWN, 2=UNREACHABLE.
$HOSTSTATETYPE$	Eine Zeichenkette, die den Statustyp der aktuellen Host-Prüfung angibt ("HARD" oder "SOFT"). Ein Soft-Status tritt auf, wenn eine Host-Prüfung einen nicht-OK (nicht-UP) Status zurückliefert und noch Wiederholungen anstehen. Ein Hard-Status liegt vor, wenn die Anzahl der Host-Prüfungs-Wiederholungen einen maximal definierten Wert erreicht hat.
$HOSTATTEMPT$	Die Anzahl der aktuellen Host-Prüfungs-Wiederholungen. Wenn dies beispielsweise das zweite Mal ist, dass der Host erneut geprüft wird, dann steht hier die Zahl zwei. Die aktuelle Wiederholungsanzahl ist eigentlich nur dann sinnvoll, wenn man Eventhandler für Soft-Zustände schreibt, die auf einer bestimmten Aktion für diese entsprechende Zahl basieren.
$MAXHOSTATTEMPTS$	Die max. Prüfversuche, wie sie für den aktuellen Host definiert sind. Nützlich, wenn man Host-Eventhandler für "Soft"-Zustände schreibt, die eine bestimmte Aktion ausführen basierend auf der Host-Wiederholungsanzahl.
$HOSTEVENTID$	Eine global eindeutige Zahl verbunden mit dem aktuellen Status des Hosts. Jedes Mal, wenn eine Host- (oder Service-) Statusänderung eintritt, wird eine globale Ereignis-ID-Nummer um eins (1) erhöht. Falls bei einem Host keine Statusänderung eintritt, wird dieses Makro auf Null (0) gesetzt.
$LASTHOSTEVENTID$	Die vorherige (global eindeutige) Ereigniszahl, die für den Host vergeben wurde.

$HOSTPROBLEMID$	Eine global eindeutige Zahl verbunden mit dem aktuellen Problemstatus des Hosts. Jedes Mal, wenn ein Host (oder Service) von einem UP- oder OK-Status in einen Problemzustand wechselt, wird eine globale Problem-ID um eins (1) erhöht. Dieses Makro wird ungleich Null sein, wenn der Host sich gerade in einem Zustand ungleich UP befindet. Statuswechsel zwischen Zuständen ungleich UP (z.B. DOWN oder UNREACHABLE) erhöhen diese Problem-ID nicht. Wenn sich der Host gerade in einem UP-Zustand befindet, wird dieses Makro auf Null (0) gesetzt. In Kombination mit Eventhandlern kann dieses Makro benutzt werden, um automatisch ein Trouble-Ticket zu eröffnen, wenn Hosts das erste Mal einen Problemzustand erreichen.
$LASTHOSTPROBLEMID$	Die vorherige (global eindeutige) Ereigniszahl, die für den Host vergeben wurde. In Kombination mit Eventhandlern kann dieses Makro benutzt werden, um automatisch ein Trouble-Ticket zu schließen, wenn Hosts in einen UP-Status zurückkehren.
$HOSTLATENCY$	Eine (Fließkomma-) Zahl, die die Anzahl von Sekunden angibt, um die eine *geplante* Host-Prüfung nach der eigentlichen Planungszeit stattfand. Wenn beispielsweise eine Prüfung für 03:14:15 geplant war und erst um 03:14:17 ausgeführt wurde, dann beträgt die Verzögerung 2.0 Sekunden. On-Demand-Host-Prüfungen haben eine Verzögerung von null Sekunden.
$HOSTEXECUTIONTIME$	Eine (Fließkomma-) Zahl, die die Dauer der Ausführung einer Host-Prüfung in Sekunden angibt.
$HOSTDURATION$	Eine Zeichenkette, die die Zeitdauer angibt, die sich der Host im aktuellen Status befindet. Das Format ist "XXh YYm ZZs" und gibt die Stunden, Minuten und Sekunden an.
$HOSTDURATIONSEC$	Eine Zahl, die die Zeitdauer in Sekunden angibt, die sich der Host im aktuellen Status befindet.
$HOSTDOWNTIME$	Eine Zahl, die die aktuelle "Downtime-Tiefe" für den Host angibt. Wenn dieser Host sich gerade in einer Phase einer geplanten Downtime befindet, ist dieser Wert größer als Null. Ist der Host nicht gerade in einer Downtime-Phase, ist dieser Wert Null.

$HOSTPERCENTCHANGE$	Eine (Fließkomma-) Zahl, die den prozentualen Statuswechsel angibt, dem der Host unterworfen war. Dieser Wert wird vom flap detection-Algorithmus benutzt.
$HOSTGROUPNAME$	Der Kurzname der Hostgruppe, zu der dieser Host gehört. Dieser Wert wird aus der *hostgroup_name*-Direktive in der hostgroup-Definition entnommen. Wenn der Host zu mehreren Hostgruppen gehört, enthält dieses Makro nur einen der Namen.
$HOSTGROUPNAMES$	Eine Komma-separierte Liste der Kurznamen aller Hostgruppen, zu denen dieser Host gehört.
$LASTHOSTCHECK$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt eine Prüfung des Hosts stattfand.
$LASTHOSTSTATECHANGE$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt ein Statuswechsel des Hosts stattfand.
$LASTHOSTUP$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt festgestellt wurde, dass sich der Host in einem UP-Zustand befand.
$LASTHOSTDOWN$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt festgestellt wurde, dass sich der Host in einem DOWN-Zustand befand.
$LASTHOSTUNREACHABLE$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt festgestellt wurde, dass sich der Host in einem UNREACHABLE-Zustand befand.
$HOSTOUTPUT$	Die erste Zeile von Textausgaben der letzten Host-Prüfung (z.B. "Ping OK")
$LONGHOSTOUTPUT$	Die vollständige Textausgabe (außer der ersten Zeile) der letzten Host-Prüfung.
$HOSTPERFDATA$	Dieses Makro enthält jegliche Performance-Daten, die von der letzten Host-Prüfung geliefert worden sein könnten.
$HOSTCHECKCOMMAND$	Dieses Makro enthält den Namen des Befehls (zusammen mit übergebenen Argumenten), der zur Host-Prüfung benutzt wurde.

$HOSTACKAUTHOR$ [8]	Eine Zeichenkette, die den Namen des Benutzers enthält, der das Host-Problem bestätigt hat. Dieses Makro ist nur gültig bei Benachrichtigungen, bei denen das Makro $NOTIFICATIONTYPE$ auf "ACKNOWLEDGEMENT" gesetzt ist.
$HOSTACKAUTHORNAME$ [8]	Eine Zeichenkette, die den Kurznamen der Kontaktperson (falls zutreffend) enthält, die das Host-Problem bestätigt hat. Dieses Makro ist nur gültig bei Benachrichtigungen, bei denen das Makro $NOTIFICATIONTYPE$ auf "ACKNOWLEDGEMENT" gesetzt ist.
$HOSTACKAUTHORALIAS$ [8]	Eine Zeichenkette, die den Alias der Kontaktperson (falls zutreffend) enthält, die das Host-Problem bestätigt hat. Dieses Makro ist nur gültig bei Benachrichtigungen, bei denen das Makro $NOTIFICATIONTYPE$ auf "ACKNOWLEDGEMENT" gesetzt ist.
$HOSTACKCOMMENT$ [8]	Eine Zeichenkette, die den Bestätigungskommentar enthält, den der Benutzer eingegeben hat, der das Host-Problem bestätigt hat. Dieses Makro ist nur gültig bei Benachrichtigungen, bei denen das Makro $NOTIFICATIONTYPE$ auf "ACKNOWLEDGEMENT" gesetzt ist.
$HOSTACTIONURL$	Der Action-URL für den Host. Dieses Makro kann andere Makros enthalten (z.B. $HOSTNAME$), was nützlich sein kann, wenn man den Hostnamen an eine Web-Seite übergeben will.
$HOSTNOTESURL$	Der Anmerkungs-URL für den Host. Dieses Makro kann andere Makros enthalten (z.B. $HOSTNAME$), was nützlich sein kann, wenn man den Hostnamen an eine Web-Seite übergeben will.
$HOSTNOTES$	Anmerkungen für den Host. Dieses Makro kann andere Makros enthalten (z.B. $HOSTNAME$), was nützlich sein kann, wenn man Hostspezifische Statusinformationen in der Beschreibung haben möchte.
$TOTALHOSTSERVICES$	Die Gesamtzahl von Services, die mit dem Host verbunden sind.
$TOTALHOSTSERVICESOK$	Die Gesamtzahl von Services im OK-Zustand, die mit dem Host verbunden sind.
$TOTALHOSTSERVICESWARNING$	Die Gesamtzahl von Services im WARNING-Zustand, die mit dem Host verbunden sind.

$TOTALHOSTSERVICESUNKNOWN$	Die Gesamtzahl von Services im UNKNOWN-Zustand, die mit dem Host verbunden sind.
$TOTALHOSTSERVICESCRITICAL$	Die Gesamtzahl von Services im CRITICAL-Zustand, die mit dem Host verbunden sind.
Hostgroup-Makros: [5]	
$HOSTGROUPALIAS$ [5]	Der Langname / Alias entweder des 1) Hostgruppennamens, der als On-Demand-Makro-Argument übergeben wurde oder 2) der primären Hostgruppe, die mit dem aktuellen Host verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde). Dieser Wert wird aus der *alias*-Direktive in der hostgroup-Definition genommen.
$HOSTGROUPMEMBERS$ [5]	Eine Komma-separierte Liste aller Hosts, die entweder 1) zu dem Hostgruppennamen gehören, der als On-Demand-Makro-Argument übergeben wurde, oder 2) zu der primären Hostgruppe gehören, die mit dem aktuellen Host verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde).
$HOSTGROUPNOTES$ [5]	Die Anmerkungen, die verbunden sind mit entweder 1) dem Hostgroup-Namen, der als On-Demand-Makro-Argument übergeben wurde oder 2) der primären Hostgruppe, die mit dem aktuellen Host verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde). Dieser Wert wird aus der *notes*-Direktive in der hostgroup-Definition genommen.
$HOSTGROUPNOTESURL$ [5]	Der Anmerkungs-URL, der verbunden ist mit entweder 1) dem Hostgroup-Namen, der als On-Demand-Makro-Argument übergeben wurde oder 2) der primären Hostgruppe, die mit dem aktuellen Host verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde). Dieser Wert wird aus der *notes_url*-Direktive in der hostgroup-Definition genommen.

$HOSTGROUPACTIONURL$ [5]	Der Action-URL, der verbunden ist mit entweder 1) dem Hostgroup-Namen, der als On-Demand-Makro-Argument übergeben wurde oder 2) der primären Hostgruppe, die mit dem aktuellen Host verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde). Dieser Wert wird aus der *action_url*-Direktive in der hostgroup-Definition genommen.
Service-Makros:	
$SERVICEDESC$	Der Langname/die Beschreibung des Service (z.B. "Main Website"). Dieser Wert wird aus der *description*-Direktive der Service-Definition genommen.
$SERVICEDISPLAYNAME$	Ein alternativer Anzeigename für den Service. Dieser Wert wird aus der *display_name*-Direktive der Service-Definition genommen.
$SERVICESTATE$	Eine Zeichenkette, die den aktuellen Status des Service anzeigt ("OK", "WARNING", "UNKNOWN" oder "CRITICAL").
$SERVICESTATEID$	Eine Zahl, die dem aktuellen Status des Service entspricht: 0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN.
$LASTSERVICESTATE$	Eine Zeichenkette, die den letzten Status des Service angibt ("OK", "WARNING", "UNKNOWN" oder "CRITICAL").
$LASTSERVICESTATEID$	Eine Zahl, die dem letzten Status des Service entspricht: 0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN.
$SERVICESTATETYPE$	Eine Zeichenkette, die den Statustyp für die aktuelle Service-Prüfung anzeigt ("HARD" oder "SOFT"). Ein Soft-Status tritt auf, wenn eine Service-Prüfung einen nicht-OK Status zurückliefert und noch Wiederholungen anstehen. Ein Hard-Status liegt vor, wenn die Anzahl der Service-Prüfungswiederholungen einen maximal definierten Wert erreicht hat.

$SERVICEATTEMPT$	Die Anzahl der aktuellen Service-Prüfungswiederholungen. Wenn dies beispielsweise das zweite Mal ist, dass der Service erneut geprüft wird, dann steht hier die Zahl zwei. Die aktuelle Wiederholungsanzahl ist eigentlich nur dann sinnvoll, wenn man Eventhandler für Soft-Zustände schreibt, die auf einer bestimmten Aktion für diese entsprechende Zahl basieren.
$MAXSERVICEATTEMPTS$	Die max. Prüfversuche, wie sie für den aktuellen Service definiert sind. Nützlich, wenn man Service-Eventhandler für "Soft"-Zustände schreibt, die eine bestimmte Aktion ausführen basierend auf der Service-Wiederholungsanzahl.
$SERVICEISVOLATILE$	Zeigt an, ob der Service als sprunghaft ("volatile") markiert ist: 0 = not volatile, 1 = volatile.
$SERVICEEVENTID$	Eine global eindeutige Zahl verbunden mit dem aktuellen Status des Service. Jedes Mal, wenn eine Service- (oder Host-) Statusänderung eintritt, wird eine globale Ereignis-ID-Nummer um eins (1) erhöht. Falls bei einem Service keine Statusänderung eintritt, wird dieses Makro auf Null (0) gesetzt.
$LASTSERVICEEVENTID$	Die vorherige (global eindeutige) Ereigniszahl, die für den Service vergeben wurde.
$SERVICEPROBLEMID$	Eine global eindeutige Zahl verbunden mit dem aktuellen Problemstatus des Service. Jedes Mal, wenn ein Service (oder Host) von einem UP- oder OK-Status in einen Problemzustand wechselt, wird eine globale Problem-ID um eins (1) erhöht. Dieses Makro wird ungleich Null sein, wenn der Service sich gerade in einem Zustand ungleich OK befindet. Statuswechsel zwischen Zuständen ungleich OK (z.B. DOWN oder UNREACHABLE) erhöhen diese Problem-ID nicht. Wenn sich der Service gerade in einem OK-Zustand befindet, wird dieses Makro auf Null (0) gesetzt. In Kombination mit Eventhandlern kann dieses Makro benutzt werden, um automatisch ein Trouble-Ticket zu eröffnen, wenn Services das erste Mal einen Problemzustand erreichen.
$LASTSERVICEPROBLEMID$	Die vorherige (global eindeutige) Ereigniszahl, die für den Service vergeben wurde. In Kombination mit Eventhandlern kann dieses Makro benutzt werden, um automatisch ein Trouble-Ticket zu schließen, wenn Services zu einem Up-Status zurückkehren.

$SERVICELATENCY$	Eine (Fließkomma-) Zahl, die die Anzahl von Sekunden angibt, um die eine *geplante* Service-Prüfung nach der eigentlichen Planungszeit stattfand. Wenn beispielsweise eine Prüfung für 03:14:15 geplant war und erst um 03:14:17 ausgeführt wurde, dann beträgt die Verzögerung 2.0 Sekunden.
$SERVICEEXECUTIONTIME$	Eine (Fließkomma-) Zahl, die die Dauer der Ausführung einer Service-Prüfung in Sekunden angibt.
$SERVICEDURATION$	Eine Zeichenkette, die die Zeitdauer angibt, die sich der Service im aktuellen Status befindet. Das Format ist "XXh YYm ZZs" und gibt die Stunden, Minuten und Sekunden an.
$SERVICEDURATIONSEC$	Eine Zahl, die die Zeitdauer in Sekunden angibt, die sich der Service im aktuellen Status befindet.
$SERVICEDOWNTIME$	Eine Zahl, die die aktuelle "Downtime-Tiefe" für den Service angibt. Wenn dieser Service sich gerade in einer Phase einer geplanten Downtime befindet, ist dieser Wert größer als Null. Ist der Service nicht gerade in einer Downtime-Phase, ist dieser Wert Null.
$SERVICEPERCENTCHANGE$	Eine (Fließkomma-) Zahl, die den prozentualen Statuswechsel angibt, der der Service unterworfen war. Dieser Wert wird vom flap detection-Algorithmus benutzt.
$SERVICEGROUPNAME$	Der Kurzname der Servicegruppe, zu der dieser Service gehört. Dieser Wert wird aus der *servicegroup_name*-Direktive in der servicegroup-Definition entnommen. Wenn der Service zu mehreren Servicegruppen gehört, enthält dieses Makro nur einen der Namen.
$SERVICEGROUPNAMES$	Eine Komma-separierte Liste von Kurznamen aller Servicegruppen, zu denen dieser Service gehört.
$LASTSERVICECHECK$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt eine Prüfung des Service stattfand.
$LASTSERVICESTATECHANGE$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt ein Statuswechsel des Service stattfand.

$LASTSERVICEOK$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt festgestellt wurde, dass sich der Service in einem OK-Zustand befand.
$LASTSERVICEWARNING$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt festgestellt wurde, dass sich der Service in einem WARNING-Zustand befand.
$LASTSERVICEUNKNOWN$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt festgestellt wurde, dass sich der Service in einem UNKNOWN-Zustand befand.
$LASTSERVICECRITICAL$	Dieses ist ein Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), der die Zeit angibt, zu der zuletzt festgestellt wurde, dass sich der Service in einem CRITICAL-Zustand befand.
$SERVICEOUTPUT$	Die erste Zeile von Textausgaben der letzten Service-Prüfung (z.B. "Ping OK")
$LONGSERVICEOUTPUT$	Die vollständige Textausgabe (außer der ersten Zeile) der letzten Service-Prüfung.
$SERVICEPERFDATA$	Dieses Makro enthält jegliche Performance-Daten, die von der letzten Service-Prüfung geliefert worden sein könnten.
$SERVICECHECKCOMMAND$	Dieses Makro enthält den Namen des Befehls (zusammen mit übergebenen Argumenten), der zur Service-Prüfung benutzt wurde.
$SERVICEACKAUTHOR$ [8]	Eine Zeichenkette, die den Namen des Benutzers enthält, der das Service-Problem bestätigt hat. Dieses Makro ist nur gültig bei Benachrichtigungen, bei denen das Makro $NOTIFICATIONTYPE$ auf "ACKNOWLEDGEMENT" gesetzt ist.
$SERVICEACKAUTHORNAME$ [8]	Eine Zeichenkette, die den Kurznamen der Kontaktperson (falls zutreffend) enthält, die das Service-Problem bestätigt hat. Dieses Makro ist nur gültig bei Benachrichtigungen, bei denen das Makro $NOTIFICATIONTYPE$ auf "ACKNOWLEDGEMENT" gesetzt ist.
$SERVICEACKAUTHORALIAS$ [8]	Eine Zeichenkette, die den Alias der Kontaktperson (falls zutreffend) enthält, die das Service-Problem bestätigt hat. Dieses Makro ist nur gültig bei Benachrichtigungen, bei denen das Makro $NOTIFICATIONTYPE$ auf "ACKNOWLEDGEMENT" gesetzt ist.

$SERVICEACKCOMMENT$ [8]	Eine Zeichenkette, die den Bestätigungskommentar enthält, den der Benutzer eingegeben hat, der das Service-Problem bestätigt hat. Dieses Makro ist nur gültig bei Benachrichtigungen, bei denen das Makro $NOTIFICATIONTYPE$ auf "ACKNOWLEDGEMENT" gesetzt ist.
$SERVICEACTIONURL$	Der Action-URL für den Service. Dieses Makro kann andere Makros enthalten (z.B. $HOSTNAME$ oder $SERVICEDESC$), was nützlich sein kann, wenn man den Servicenamen an eine Web-Seite übergeben will.
$SERVICENOTESURL$	Der Anmerkungs-URL für den Service. Dieses Makro kann andere Makros enthalten (z.B. $HOSTNAME$ oder $SERVICEDESC$), was nützlich sein kann, wenn man den Servicenamen an eine Web-Seite übergeben will.
$SERVICENOTES$	Anmerkungen für den Service. Dieses Makro kann andere Makros enthalten (z.B. $HOSTNAME$ oder $SERVICEDESC$), was nützlich sein kann, wenn man Servicespezifische Statusinformationen in der Beschreibung haben möchte.
Servicegroup-Makros: [6]	
$SERVICEGROUPALIAS$ [6]	Der Langname / Alias entweder des 1) Servicegroup-Namens, der als On-Demand-Makro-Argument übergeben wurde oder 2) der primären Servicegruppe, die mit dem aktuellen Service verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde). Dieser Wert wird aus der *alias*-Direktive in der servicegroup-Definition genommen.
$SERVICEGROUPMEMBERS$ [6]	Eine Komma-separierte Liste aller Services, die entweder 1) zu dem Servicegruppennamen gehören, der als On-Demand-Makro-Argument übergeben wurde, oder 2) zu der primären Servicegruppe gehören, die mit dem aktuellen Service verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde).

$SERVICEGROUPNOTES$ [6]	Die Anmerkungen, die verbunden sind mit entweder 1) dem Servicegroup-Namen, der als On-Demand-Makro-Argument übergeben wurde oder 2) der primären Servicegruppe, die mit dem aktuellen Host verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde). Dieser Wert wird aus der *notes*-Direktive in the servicegroup-Definition genommen.
$SERVICEGROUPNOTESURL$ [6]	Der Anmerkungs-URL, der verbunden sind mit entweder 1) dem Servicegroup-Namen, der als On-Demand-Makro-Argument übergeben wurde oder 2) der primären Servicegruppe, die mit dem aktuellen Service verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde). Dieser Wert wird aus der *notes_url*-Direktive in der servicegroup-Definition genommen.
$SERVICEGROUPACTIONURL$ [6]	Der Action-URL, der verbunden sind mit entweder 1) dem Servicegroup-Namen, der als On-Demand-Makro-Argument übergeben wurde oder 2) der primären Servicegruppe, die mit dem aktuellen Service verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde). Dieser Wert wird aus der *action_url*-Direktive in der servicegroup-Definition genommen.
Contact-Makros:	
$CONTACTNAME$	Kurzname für den Kontakt (z.B. "mmustermann"), der über ein Host- oder Service-Problem informiert wird. Dieser Wert wird aus der *contact_name*-Direktive in der contact-Definition genommen.
$CONTACTALIAS$	Langname/Beschreibung für den Kontakt, der informiert wird. Dieser Wert wird aus der *alias*-Direktive in der contact-Definition genommen.
$CONTACTEMAIL$	e-Mail-Adresse für den Kontakt, der informiert wird. Dieser Wert wird aus der *email*-Direktive in der contact-Definition genommen.
$CONTACTPAGER$	Pager-Nummer/-Adresse für den Kontakt, der informiert wird. Dieser Wert wird aus der *pager*-Direktive in der contact-Definition genommen.

$CONTACTADDRESSn$	Adresse für den Kontakt, der informiert wird. Jeder Kontakt kann sechs verschiedene Adressen haben (zusätzlich zur e-Mail-Adresse und Pager-Nummer). Die Makros für diese Adressen sind $CONTACTADDRESS1$ - $CONTACTADDRESS6$. Dieser Wert wird aus der *addressx*-Direktive in der contact-Definition genommen.
$CONTACTGROUPNAME$	Der Kurzname für die Kontaktgruppe, deren Mitglied der Kontakt ist. Dieser Wert wird aus der *contact_group*-Direktive in der contactgroup-Definition genommen.
$CONTACTGROUPNAMES$	Eine Komma-separierte Liste der Kurznamen aller Kontaktgruppen, deren Mitglied dieser Kontakt ist.
Contactgroup-Makros: [5]	
$CONTACTGROUPALIAS$ [7]	Der Langname / Alias entweder des 1) Contactgroup-Namens, der als On-Demand-Makro-Argument übergeben wurde oder 2) der primären Kontaktgruppe, die mit dem aktuellen Kontakt verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde). Dieser Wert wird aus der *alias*-Direktive in der contactgroup-Definition genommen.
$CONTACTGROUPMEMBERS$ [7]	Eine Komma-separierte Liste aller Kontakte, die entweder 1) zu dem Kontaktgruppennamen gehören, der als On-Demand-Makro-Argument übergeben wurde, oder 2) zu der primären Kontaktgruppe gehören, die mit dem aktuellen Kontakt verbunden ist (falls sie nicht im Zusammenhang mit einem On-Demand-Makro benutzt wurde).
Auswertungs-Makros:	
$TOTALHOSTSUP$	Dieses Makro gibt die Gesamtzahl der Hosts an, die sich in einem UP-Zustand befinden.
$TOTALHOSTSDOWN$	Dieses Makro gibt die Gesamtzahl der Hosts an, die sich in einem DOWN-Zustand befinden.
$TOTALHOSTSUNREACHABLE$	Dieses Makro gibt die Gesamtzahl der Hosts an, die sich in einem UNREACHABLE-Zustand befinden.

$TOTALHOSTSDOWNUNHANDLED$	Dieses Makro gibt die Gesamtzahl der Hosts an, die sich in einem DOWN-Status befinden und "unbehandelt" sind. Unbehandelte Host-Probleme sind solche, die nicht bestätigt sind, sich nicht in einer geplanten Downtime befinden, und für die Prüfungen momentan aktiviert sind.
$TOTALHOSTSUNREACHABLEUNHANDLED$	Dieses Makro gibt die Gesamtzahl der Hosts an, die sich in einem UNREACHABLE-Status befinden und "unbehandelt" sind. Unbehandelte Host-Probleme sind solche, die nicht bestätigt sind, sich nicht in einer geplanten Downtime befinden, und für die Prüfungen momentan aktiviert sind.
$TOTALHOSTPROBLEMS$	Dieses Makro gibt die Gesamtzahl der Hosts an, die sich in einem DOWN- oder UNREACHABLE-Status befinden.
$TOTALHOSTPROBLEMSUNHANDLED$	Dieses Makro gibt die Gesamtzahl der Hosts an, die sich in einem DOWN- oder UNREACHABLE-Status befinden und "unbehandelt" sind. Unbehandelte Host-Probleme sind solche, die nicht bestätigt sind, sich nicht in einer geplanten Downtime befinden, und für die Prüfungen momentan aktiviert sind.
$TOTALSERVICESOK$	Dieses Makro gibt die Gesamtzahl der Services an, die sich in einem OK-Status befinden.
$TOTALSERVICESWARNING$	Dieses Makro gibt die Gesamtzahl der Services an, die sich in einem WARNING-Status befinden.
$TOTALSERVICESCRITICAL$	Dieses Makro gibt die Gesamtzahl der Services an, die sich in einem CRITICAL-Status befinden.
$TOTALSERVICESUNKNOWN$	Dieses Makro gibt die Gesamtzahl der Services an, die sich in einem UNKNOWN-Status befinden.
$TOTALSERVICESWARNINGUNHANDLED$	Dieses Makro gibt die Gesamtzahl der Services an, die sich in einem WARNING-Status befinden und "unbehandelt" sind. Unbehandelte Service-Probleme sind solche, die nicht bestätigt sind, sich nicht in einer geplanten Downtime befinden, und für die Prüfungen momentan aktiviert sind.
$TOTALSERVICESCRITICALUNHANDLED$	Dieses Makro gibt die Gesamtzahl der Services an, die sich in einem CRITICAL-Status befinden und "unbehandelt" sind. Unbehandelte Service-Probleme sind solche, die nicht bestätigt sind, sich nicht in einer geplanten Downtime befinden, und für die Prüfungen momentan aktiviert sind.

$TOTALSERVICESUNKNOWNUNHANDLED$	Dieses Makro gibt die Gesamtzahl der Services an, die sich in einem UNKNOWN-Status befinden und "unbehandelt" sind. Unbehandelte Service-Probleme sind solche, die nicht bestätigt sind, sich nicht in einer geplanten Downtime befinden, und für die Prüfungen momentan aktiviert sind.
$TOTALSERVICEPROBLEMS$	Dieses Makro gibt die Gesamtzahl der Services an, die sich in einem WARNING-, CRITICAL- oder UNKNOWN-Status befinden.
$TOTALSERVICEPROBLEMSUNHANDLED$	Dieses Makro gibt die Gesamtzahl der Services an, die sich in einem WARNING-, CRITICAL- oder UNKNOWN-Status befinden und "unbehandelt" sind. Unbehandelte Service-Probleme sind solche, die nicht bestätigt sind, sich nicht in einer geplanten Downtime befinden, und für die Prüfungen momentan aktiviert sind.
Benachrichtigungs-Makros:	
$NOTIFICATIONTYPE$	Eine Zeichenkette, die den Typ der Benachrichtigung angibt, die versandt wird ("PROBLEM", "RECOVERY", "ACKNOWLEDGEMENT", "FLAPPINGSTART", "FLAPPINGSTOP", "FLAPPINGDISABLED", "DOWNTIMESTART", "DOWNTIMEEND" oder "DOWNTIMECANCELLED").
$NOTIFICATIONRECIPIENTS$	Eine Komma-separierte Liste der Kurznamen von allen Kontakten, die über den Host oder Service benachrichtigt werden.
$NOTIFICATIONISESCALATED$	Eine Ganzzahl, die angibt, ob diese Benachrichtigung an normale Kontakte für den Host oder Service versandt wurde, oder ob sie eskaliert wurde. 0 = normale (nicht-eskalierte) Benachrichtigung, 1 = eskalierte Benachrichtigung

$NOTIFICATIONAUTHOR$	Eine Zeichenkette, die den Namen des Benutzers angibt, der die Benachrichtigung geschrieben hat. Falls das $NOTIFICATIONTYPE$-Makro auf "DOWNTIMESTART" oder "DOWNTIMEEND" gesetzt ist, wird es der Name des Benutzers sein, der die Downtime für den Host oder Service geplant hat. Falls das $NOTIFICATIONTYPE$-Makro auf "ACKNOWLEDGEMENT" gesetzt ist, wird es der Name des Benutzers sein, der das Problem für den Host oder Service bestätigt hat. Falls das $NOTIFICATIONTYPE$-Makro auf "CUSTOM" gesetzt ist, wird es der Name des Benutzers sein, der die benutzerdefinierte Benachrichtigung für den Host oder Service ausgelöst hat.
$NOTIFICATIONAUTHORNAME$	Eine Zeichenkette, die den Kurznamen des Kontakts (falls zutreffend) enthält, der im Makro $NOTIFICATIONAUTHOR$ angegeben wurde.
$NOTIFICATIONAUTHORALIAS$	Eine Zeichenkette, die den Alias des Kontakts (falls zutreffend) enthält, der im Makro $NOTIFICATIONAUTHOR$ angegeben wurde.
$NOTIFICATIONCOMMENT$	Eine Zeichenkette, die den Kommentar des Benutzers angibt, der die Benachrichtigung geschrieben hat. Falls das $NOTIFICATIONTYPE$-Makro auf "DOWNTIMESTART" oder "DOWNTIMEEND" gesetzt ist, wird es der Kommentar des Benutzers sein, der die Downtime für den Host oder Service geplant hat. Falls das $NOTIFICATIONTYPE$-Makro auf "ACKNOWLEDGEMENT" gesetzt ist, wird es der Kommentar des Benutzers sein, der das Problem für den Host oder Service bestätigt hat. Falls das $NOTIFICATIONTYPE$-Makro auf "CUSTOM" gesetzt ist, wird es der Kommentar des Benutzers sein, der die benutzerdefinierte Benachrichtigung für den Host oder Service ausgelöst hat.
$HOSTNOTIFICATIONNUMBER$	Die aktuelle Benachrichtigungsnummer für den Host. Die Benachrichtigungsnummer wird jedes Mal um eins (1) erhöht, wenn eine neue Benachrichtigung für den Host versandt wird (außer bei Bestätigungen). Die Benachrichtigungsnummer wird auf Null (0) zurückgesetzt, wenn der Host wieder im UP-Zustand ist (*nachdem* die Benachrichtigung versandt wurde). Die Benachrichtungsnummer wird weder durch Bestätigungen noch durch Benachrichtigungen über "Flap detection" oder geplante Downtimes erhöht.

$HOSTNOTIFICATIONID$	Eine eindeutige Zahl, die eine Host-Benachrichtigung identifiziert. Benachrichtigungsnummern sind eindeutig sowohl für Host- als auch Service-Benachrichtigungen, so dass Sie diese eindeutige Zahl als primären Schlüssel in einer Benachrichtigungs-Datenbank benutzen können. Benachrichtigungsnummern sollten über den Restart des Nagios- Prozesses hinweg eindeutig bleiben, solange Sie "state retention" aktiviert haben. Die Benachrichtigungsnummer wird für jede neue Host-Benachrichtigung um eins (1) erhöht, unabhängig von der Anzahl der benachrichtigten Kontakte.
$SERVICENOTIFICATIONNUMBER$	Die aktuelle Benachrichtigungsnummer für den Service. Die Benachrichtigungsnummer wird jedes Mal um eins (1) erhöht, wenn eine neue Benachrichtigung für den Service versandt wird (außer bei Bestätigungen). Die Benachrichtigungsnummer wird auf Null (0) zurückgesetzt, wenn der Service wieder im OK-Zustand ist (*nachdem* die Benachrichtigung versandt wurde). Die Benachrichtungsnummer wird weder durch Bestätigungen noch durch Benachrichtigungen über "Flap detection" oder geplante Downtimes erhöht.
$SERVICENOTIFICATIONID$	Eine eindeutige Zahl, die eine Service-Benachrichtigung identifiziert. Benachrichtigungsnummern sind eindeutig sowohl für Host- als auch Service-Benachrichtigungen, so dass Sie diese eindeutige Zahl als primären Schlüssel in einer Benachrichtigungs-Datenbank benutzen können. Benachrichtigungsnummern sollten über den Restart des Nagios- Prozesses hinweg eindeutig bleiben, solange Sie "state retention" aktiviert haben. Die Benachrichtigungsnummer wird für jede neue Service-Benachrichtigung um eins (1) erhöht, unabhängig von der Anzahl der benachrichtigten Kontakte.
Datum-/Zeit-Makros:	
$LONGDATETIME$	Aktueller Datum-/Zeitstempel (z.B. *Fri Oct 13 00:30:28 CDT 2000*). Das Datum-Format ist festgelegt durch die date_format-Direktive.
$SHORTDATETIME$	Aktueller Datum-/Zeitstempel (z.B. *10-13-2000 00:30:28*). Das Datum-Format ist festgelegt durch die date_format-Direktive.

$DATE$	Aktueller Datumstempel (z.B. *10-13-2000*). Das Datum-Format ist festgelegt durch die date_format-Direktive.
$TIME$	Aktueller Zeitstempel (z.B. *00:30:28*).
$TIMET$	Aktueller Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche).
$ISVALIDTIME:$ [9]	Dies ist ein spezielles On-Demand-Makro, das 1 oder 0 zurückliefert, abhängig davon, ob eine bestimmte Zeit innerhalb einer angegebenen Zeitperiode gültig ist. Es gibt zwei Arten, dieses Makro zu benutzen: 1. **$ISVALIDTIME:24x7$** wird auf "1" gesetzt, wenn die aktuelle Zeit innerhalb der "24x7"-Zeitperiode gültig ist. Falls nicht, wird es auf "0" gesetzt. 2. **$ISVALIDTIME:24x7:*timestamp*$** wird auf "1" gesetzt, wenn die durch das "timestamp"-Argument angegebene Zeit (die im time_t-Format sein muss) innerhalb der "24x7"-Zeitperiode gültig ist. Falls nicht, wird es auf "0" gesetzt.
$NEXTVALIDTIME:$ [9]	Dies ist ein spezielles On-Demand-Makro, das die nächste gültige Zeit (im time_t-Format) für eine angebene Zeitperiode zurückliefert. Es gibt zwei Arten, dieses Makro zu benutzen: 1. **$NEXTVALIDTIME:24x7$** wird die nächste gültige Zeit zurückliefern - ab der aktuellen Zeit - innerhalb der "24x7"-Zeitperiode. 2. **$NEXTVALIDTIME:24x7:*timestamp*$** wird die nächste gültige Zeit zurückliefern - ab der durch das "timestamp"-Argument angegebenen Zeit (die im time_t-Format sein muss) - innerhalb der "24x7"-Zeitperiode. Falls keine gültige Zeit innerhalb der angegebenen Zeitperiode gefunden werden kann, wird das Makro auf "0" gesetzt.
Datei-Makros:	
$MAINCONFIGFILE$	Der Standort der Hauptkonfigurationsdatei (main config file).
$STATUSDATAFILE$	Der Standort der Statusdaten-Datei (main config file).
$COMMENTDATAFILE$	Der Standort der Kommentardaten-Datei (comment data file).

$DOWNTIMEDATAFILE$	Der Standort der Ausfallzeitendaten-Datei (downtime data file).
$RETENTIONDATAFILE$	Der Standort der Aufbewahrungsdaten-Datei (retention data file).
$OBJECTCACHEFILE$	Der Standort der Objektzwischenspeicherungs-Datei (object cache file).
$TEMPFILE$	Der Standort der temporären Datei (temp file).
$TEMPPATH$	Das durch die temp path-Variable festgelegte Verzeichnis.
$LOGFILE$	Der Standort der Protokolldatei (log file).
$RESOURCEFILE$	Der Standort der Ressource-Datei (resource file).
$COMMANDFILE$	Der Standort der Befehlsdatei (command file).
$HOSTPERFDATAFILE$	Der Standort der Host-Performancedaten-Datei (host performance data file; falls definiert).
$SERVICEPERFDATAFILE$	Der Standort der Service-Performancedaten-Datei (service performance data file; falls definiert).
Verschiedene Makros:	
$PROCESSSTARTTIME$	Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), die angibt, wann der Nagios-Prozess das letzte Mal neu/wieder gestartet wurde. Sie können die Laufzeit durch Subtraktion von $PROCESSSTARTTIME$ von $TIMET$ ermitteln.
$EVENTSTARTTIME$	Zeitstempel im time_t-Format (Sekunden seit der UNIX-Epoche), die angibt, wann der Nagios-Prozess begann, Ereignisse (Prüfungen, usw.) zu verarbeiten. Sie können die Zeit, die Nagios zum Start benötigte, durch Subtraktion von $PROCESSSTARTTIME$ von $EVENTSTARTTIMET$ ermitteln.
$ADMINEMAIL$	Globale administrative e-Mail-Adresse. Dieser Wert wird aus der admin_email-Direktive genommen.
$ADMINPAGER$	Globale administrative Pager-Nummer/-Adresse. Dieser Wert wird aus der admin_pager-Direktive genommen.

$ARGn$	Das *n*-te an den Befehl (Benachrichtigung, Eventhandler, Service-Prüfungen, usw.) übergebene Argument. Nagios unterstützt bis zu 32 Argument-Makros ($ARG1$ bis $ARG32$).
$USERn$	Das *n*-te benutzerdefinierbare Makro. Benutzermakros können in ein oder mehreren resource files definiert werden. Nagios unterstützt bis zu 32 User-Makros ($USER1$ bis $USER32$).

Anmerkungen

[1] Diese Makros sind nicht gültig für den Host, dem sie zugeordnet sind, wenn der Host gerade geprüft wird (denn die Werte konnten noch nicht ermittelt werden).

[2] Diese Makros sind nicht gültig für den Service, dem sie zugeordnet sind, wenn der Service gerade geprüft wird (denn die Werte konnten noch nicht ermittelt werden).

[3] Wenn Host-Makros in Service-bezogenen Befehlen benutzt werden (z.B. Service-Benachrichtigungen, Eventhandler, usw.) verweisen sie auf den Host, dem der Service zugeordnet ist.

[4] Wenn Host- und Service-Auswertungsmakros in Benachrichtigungen benutzt werden, werden die Summen gefiltert, um so nur die Hosts und Services zu berücksichtigen, für die die Kontaktperson berechtigt ist (z.B. Hosts und Services, für die sie Benachrichtigungen erhalten soll).

[5] Diese Makros sind normalerweise der ersten/primären Hostgruppe des aktuellen Hosts zugeordnet. Sie können deshalb in vielen Fällen als Host-Makros angesehen werden. Allerdings sind diese Makros nicht als On-Demand-Makros verfügbar. Statt dessen können sie als On-Demand-Hostgroup-Makros benutzt werden, wenn Sie den Namen einer Hostgruppe an das Makro übergeben. Beispielsweise würde $HOSTGROUPMEMBERS:hg1$ eine komma-separierte Liste aller (Host)-Mitglieder der Hostgruppe *hg1* zurückliefern.

[6] Diese Makros sind normalerweise der ersten/primären Servicegruppe des aktuellen Service zugeordnet. Sie können deshalb in vielen Fällen als Service-Makros angesehen werden. Allerdings sind diese Makros nicht als On-Demand-Makros verfügbar. Statt dessen können sie als On-Demand-Servicegroup-Makros benutzt werden, wenn Sie den Namen einer Servicegruppe an das Makro übergeben. Beispielsweise würde $SERVICEGROUPMEMBERS:sg1$ eine komma-separierte Liste aller (Service)-Mitglieder der Servicegruppe *sg1* zurückliefern.

[7] Diese Makros sind normalerweise der ersten/primären Kontaktgruppe des aktuellen Kontakts zugeordnet. Sie können deshalb in vielen Fällen als Kontakt-Makros angesehen werden. Allerdings sind diese Makros nicht als On-Demand-Makros verfügbar. Statt dessen können sie als On-Demand-Contaktgroup-Makros benutzt werden, wenn Sie den Namen einer Kontaktgruppe an das Makro übergeben. Beispielsweise würde $CONTACTGROUPMEMBERS:cg1$ eine komma-separierte Liste aller (Kontakt)-Mitglieder der Kontaktgruppe *cg1* zurückliefern.

[8] Diese Bestätigungsmakros sind veraltet. Nutzen Sie statt dessen die mehr generischen Makros $NOTIFICATIONAUTHOR$, $NOTIFICATIONAUTHORNAME$, $NOTIFICATIONAUTHORALIAS$ oder $NOTIFICATIONAUTHORCOMMENT$

[9] Diese Makros sind nur als On-Demand-Makros verfügbar - d.h. Sie müssen ein zusätzliches Argument übergeben, um sie zu nutzen. Diese Makros sind nicht als Umgebungsvariablen verfügbar.

[10] Auswertungsmakros sind nicht als Umgebungsvariablen verfügbar, wenn die Option use_large_installation_tweaks aktiviert ist, weil sie ziemlich CPU-intensiv zu berechnen sind.

Host-Prüfungen (Host checks)

Hoch zu: Inhalt

Siehe auch: Netzwerk-Erreichbarkeit, Aktive Prüfungen, Service-Prüfungen, Prüfungsplanung, vorausschauende Abhängigkeitsprüfungen

Einführung

Die grundlegenden Tätigkeiten von Host-Prüfungen werden hier beschrieben...

Wann werden Host-Prüfungen durchgeführt?

Hosts werden durch den Nagios-Daemon geprüft

- in regelmäßigen Intervallen, wie sie durch die *check_interval* und *retry_interval*-Optionen in Ihren Host-Definitionen festgelegt sind.
- nach Bedarf, wenn ein mit dem Host verbundener Service den Status wechselt.
- nach Bedarf als Teil der Host-Verfügbarkeits-Logik.
- nach Bedarf bei vorausschauenden Host-Abhängigkeitsprüfungen.

Regelmäßige Host-Prüfungen sind optional. Wenn Sie die *check_interval*-Option in Ihrer Host-Definition auf Null (0) setzen, wird Nagios keine Host-Prüfungen auf planmäßiger Basis durchführen. Es wird jedoch weiterhin nach Bedarf Prüfungen für den Host durchführen für andere Teile der Überwachungslogik.

Prüfungen nach Bedarf werden gemacht, wenn ein mit dem Host verbundener Service den Status wechselt, denn Nagios muss wissen, ob auch der Host den Status gewechselt hat. Services, die den Status wechseln, sind oft ein Indikator dafür, dass auch der Host den Status gewechselt hat. Wenn beispielsweise der mit einem Host verbundene HTTP-Service den Status von CRITICAL auf OK gewechselt hat, kann das bedeuten, dass der Host gerade einen Reboot beendet hat und nun wieder verfügbar ist.

Host-Prüfungen nach Bedarf werden auch als Teil der Host-Erreichbarkeit erledigt. Nagios ist so konstruiert, dass Netzwerkausfälle so schnell wie möglich erkannt werden und zwischen DOWN- und UNREACHABLE-Zuständen unterschieden werden kann. Das sind sehr unterschiedliche Zustände und es kann dem Admin helfen, schnell die Ursache für einen Netzwerkausfall zu finden.

Prüfungen nach Bedarf werden auch als Teil der vorausschauenden Host-Abhängigkeitsprüfungs-Logik durchgeführt.

zwischengespeicherte Host-Prüfungen (cached host checks)

Die Performance von Host-Prüfungen nach Bedarf kann signifikant durch den Einsatz von "cached checks" erhöht werden, die es Nagios erlauben, auf eine Host-Prüfung zu verzichten, wenn es feststellt, dass ein relativ frisches Prüfungsergebnis genügt. Mehr Informationen zu "cached checks" finden Sie hier.

Abhängigkeiten und Prüfungen

Sie können Host-Ausführungs-Abhängigkeiten definieren, die Nagios von der Statusprüfung eines Hosts abhalten in Abhängigkeit vom Status ein oder mehrerer anderer Hosts. Mehr Informationen zu Abhängigkeiten finden Sie hier.

Parallelisierung von Host-Prüfungen

Geplante Host-Prüfungen laufen parallel. Wenn Nagios eine geplante Host-Prüfung ausführt, wird es die Host-Prüfung veranlassen und dann zu anderen Arbeiten zurückkehren (Service-Prüfungen ausführen, etc.). Die Host-Prüfung läuft in einem Kind-Prozess, der vom Haupt-Nagios-Prozess aufgerufen wird ("fork()ed"). Wenn die Host-Prüfung beendet ist, wird der Kind-Prozess den Haupt-Nagios-Prozess (seinen Eltern-Prozess) über das Ergebnis informieren. Der Haupt-Nagios-Prozess wird dann das Prüfungsergebnis behandeln und geeignete Aktionen durchführen (Eventhandler starten, Benachrichtigungen senden, usw.).

Host-Prüfungen nach Bedarf laufen ebenfalls parallel, falls notwendig. Wie bereits vorher erwähnt kann Nagios auf die eigentliche Ausführung einer Host-Prüfung nach Bedarf verzichten, wenn es das gespeicherte Ergebnis einer relativ frischen Host-Prüfung benutzen kann.

Wenn Nagios die Ergebnisse von geplanten und nach Bedarf ausgeführten Host-Prüfungen verarbeitet, kann es (zusätzliche) Prüfungen anderer Hosts veranlassen. Diese Prüfungen können aus zwei Gründen veranlasst werden: vorausschauende Abhängigkeitsprüfungen und um den Status des Hosts mit Hilfe von Netzwerk-Erreichbarkeits-Logik festzustellen. Die zusätzlichen Prüfungen werden normalerweise parallel ausgeführt. Allerdings gibt es eine große Ausnahme, der Sie sich bewusst sein sollten, da sie einen negativen Einfluss auf die Performance haben kann...

Hosts, deren *max_check_attempts*-Wert auf **1** gesetzt sind, können schwerwiegende Performance-Probleme verursachen. Der Grund? Wenn Nagios den richtigen Status mit Hilfe der Netzwerk-Erreichbarkeits-Logik ermitteln muss (um zu sehen, ob sie DOWN oder UNREACHABLE sind), muss es **aufeinanderfolgende** Prüfungen für alle direkten Eltern des Hosts starten. Um es noch einmal zu wiederholen, diese Prüfungen laufen *nacheinander* statt parallel, also kann es zu einem Performance-Einbruch kommen. Aus diesem Grund würde ich empfehlen, dass Sie immer einen Wert größer als 1 für die *max_check_attempts*-Direktiven in Ihren Host-Definitionen benutzen.

Host-Zustände

Hosts, die geprüft werden, können in einem von drei unterschiedlichen Zuständen sein

- UP
- DOWN
- UNREACHABLE

Host-Statusermittlung

Host-Prüfungen werden mit Hilfe von Plugins durchgeführt, die den Status OK, WARNING, UNKNOWN oder CRITICAL zurückliefern können. Wie übersetzt Nagios diese Return-Codes der Plugins in die Host-Zustände UP, DOWN oder UNREACHABLE? Wir werden sehen...

Die nachfolgende Tabelle zeigt, wie sich die Return-Codes von Plugins mit vorläufigen Host-Zuständen decken. Einige Nachbearbeitung (die später beschrieben wird) ergibt den endgültigen Host-Zustand.

Plugin-Ergebnis	vorläufiger Host-Zustand
OK	UP
WARNING	UP oder DOWN*
UNKNOWN	DOWN
CRITICAL	DOWN

 Anmerkung: Das Ergebnis WARNING bedeutet normalerweise, dass der Host UP ist. Trotzdem werden WARNING-Ergebnisse so interpretiert, dass der Host DOWN ist, wenn die use_aggressive_host_checking-Option aktiviert ist.

Wenn der vorläufige Host-Status DOWN ist, wird Nagios versuchen festzustellen, ob der Host wirklich DOWN ist oder UNREACHABLE. Die Unterscheidung zwischen den Host-Zuständen DOWN und UNREACHABLE ist wichtig, weil es Admins erlaubt, die Grundursache von Netzwerkausfällen schneller zu ermitteln. Die folgende Tabelle zeigt, wie Nagios eine endgültige Zustandsermittlung basierend auf dem Zustand der Eltern des Hosts durchführt. Die Eltern eines Hosts werden in der *parents*-Direktive der Host-Definition festgelegt.

vorläufiger Host-Zustand	Zustand Host-Eltern	endgültiger Host-Zustand
DOWN	mindestens ein Elternteil ist UP	DOWN
DOWN	alle Eltern sind entweder DOWN oder UNREACHABLE	UNREACHABLE

Mehr Informationen, wie Nagios zwischen DOWN- und UNREACHABLE-Zuständen unterscheidet, finden Sie hier.

Host-Statusänderungen

Wie Ihnen wahrscheinlich bereits bewusst ist, bleiben Hosts nicht immer in einem Zustand. Dinge gehen kaputt, Patches werden eingespielt und Server müssen neu gestartet werden. Wenn Nagios den Status von Hosts prüft, ist es in der Lage festzustellen, wenn ein Host zwischen UP-, DOWN- und UNREACHABLE-Zuständen wechselt und geeignete Maßnahmen ergreifen. Diese Zustandsänderungen resultieren in verschiedenen Statustypen (HARD oder SOFT), was zum Auslösen von Eventhandlern und dem Versenden von Benachrichtigungen führen kann. Das Erkennen und Behandeln von Statusänderungen ist das, worum es sich bei Nagios handelt.

Wenn Host-Statusänderungen zu oft erfolgen, werden sie als "flatternd" (flapping) angesehen. Ein gutes Beispiel für einen flatternden Host wäre ein Server, der spontan jedes Mal neu startet, sobald das Betriebssystem lädt. Das ist immer ein spaßiges Szenario, mit dem man sich befassen muss. Nagios kann erkennen, wenn Hosts anfangen zu flattern, und kann Benachrichtigungen unterdrücken, bis das Flattern stoppt und sich der Host-Status stabilisiert. Mehr Informationen über die Erkennungslogik des Flatterns finden Sie hier.

Service-Prüfungen (Service Checks)

Hoch zu: Inhalt

Siehe auch: Aktive Prüfungen, Host-Prüfungen, Prüfungsplanung, vorausschauenden Abhängigkeitsprüfungen

Einführung

Die grundlegenden Tätigkeiten von Service-Prüfungen werden hier beschrieben...

Wann werden Service-Prüfungen durchgeführt?

Services werden durch den Nagios-Daemon geprüft

- in regelmäßigen Intervallen, wie sie durch die *check_interval* und *retry_interval*-Optionen in Ihren Service-Definitionen festgelegt sind.
- nach Bedarf bei vorausschauende Host-Abhängigkeitsprüfungen.

Prüfungen nach Bedarf werden als Teil der vorausschauenden Service-Abhängigkeitsprüfungs-Logik durchgeführt. Diese Prüfungen helfen sicherzustellen, dass die Abhängigkeitslogik so genau wie möglich ist. Falls Sie die Service-Abhängigkeiten nicht nutzen, wird Nagios keine Service-Prüfungen nach Bedarf durchführen.

zwischengespeicherte Service-Prüfungen (cached service checks)

Die Performance von Service-Prüfungen nach Bedarf kann signifikant durch den Einsatz von "cached checks" erhöht werden, die es Nagios erlauben, auf eine Service-Prüfung zu verzichten, wenn es feststellt, dass ein relativ frisches Prüfungsergebnis genügt. Cached checks werden nur dann einen Performance-Gewinn ergeben, wenn Sie Service-Abhängigkeiten nutzen. Mehr Informationen zu "cached checks" finden Sie hier.

Abhängigkeiten und Prüfungen

Sie können Service-Ausführungs-Abhängigkeiten definieren, die Nagios von der Statusprüfung eines Service abhalten in Abhängigkeit vom Status ein oder mehrerer anderer Services. Mehr Informationen zu Abhängigkeiten finden Sie hier.

Parallelisierung von Service-Prüfungen

Geplante Service-Prüfungen laufen parallel. Wenn Nagios eine geplante Service-Prüfung ausführt, wird es die Service-Prüfung veranlassen und dann zu anderen Arbeiten zurückkehren (Host-Prüfungen ausführen, etc.). Die Service-Prüfung läuft in einem Kind-Prozess, der vom Haupt-Nagios-Prozess aufgerufen wird ("fork()ed"). Wenn die Service-Prüfung beendet ist, wird der Kind-Prozess den Haupt-Nagios-Prozess (seinen Eltern-Prozess) über das Ergebnis informieren. Der
Haupt-Nagios-Prozess wird dann das Prüfungsergebnis behandeln und geeignete Aktionen durchführen (Eventhandler starten, Benachrichtigungen senden, usw.).

Service-Prüfungen nach Bedarf laufen ebenfalls parallel, falls notwendig. Wie bereits vorher erwähnt kann Nagios auf die eigentliche Ausführung einer Service-Prüfung nach Bedarf verzichten, wenn es das gespeicherte Ergebnis einer relativ frischen Service-Prüfung benutzen kann.

Service-Zustände

Services, die geprüft werden, können in einem von vier unterschiedlichen Zuständen sein

- OK
- WARNING
- UNKNOWN
- CRITICAL

Service-Statusermittlung

Service-Prüfungen werden mit Hilfe von Plugins durchgeführt, die den Status OK, WARNING, UNKNOWN oder CRITICAL zurückliefern können. Diese Return-Codes der Plugins werden direkt in die Service-Zustände übersetzt. Beispielsweise wird das WARNING-Ergebnis eines Plugins zu einem WARNING-Status eines Service führen.

Service-Statusänderungen

Wenn Nagios den Status von Services prüft, ist es in der Lage festzustellen, wenn ein Service zwischen OK-, WARNING-, UNKNOWN- und CRITICAL-Zuständen wechselt und geeignete Maßnahmen ergreifen. Diese Zustandsänderungen resultieren in verschiedenen Statustypen (HARD oder SOFT), was zum Auslösen von Eventhandlern und dem Versenden von Benachrichtigungen führen kann. Service-Statusänderungen können auch zum Auslösen von Host-Prüfungen nach Bedarf führen. Das Erkennen und Behandeln von Statusänderungen ist das, worum es sich bei Nagios handelt.

Wenn Service-Statusänderungen zu oft erfolgen, werden sie als "flatternd" (flapping) angesehen. Nagios kann erkennen, wenn Services anfangen zu flattern, und kann Benachrichtigungen unterdrücken, bis das Flattern stoppt und sich der Service-Status stabilisiert. Mehr Informationen über die Erkennungslogik des Flatterns finden Sie hier.

Aktive Prüfungen (Active Checks)

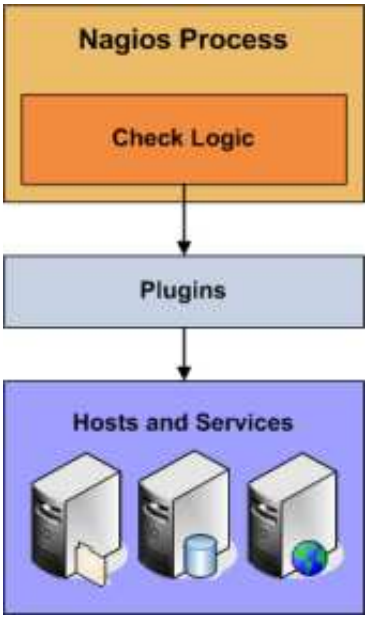
Hoch zu: Inhalt

Siehe auch: Passive Prüfungen, Plugins, Service-Prüfungen, Host-Prüfungen

Einführung

Nagios ist in der Lage, Hosts und Services auf zwei Arten zu überwachen: aktiv und passiv. Passive Prüfungen werden anderswo beschrieben, so dass wir uns hier auf aktive Prüfungen konzentrieren. Aktive Prüfungen sind die gebräuchlichste Methode zur Überwachung von Hosts und Services. Die Hauptmerkmale von aktiven Prüfungen sind

- aktive Prüfungen werden vom Nagios-Prozess veranlasst
- aktive Prüfungen laufen auf einer regelmäßig geplanten Basis



Wie werden aktive Prüfungen durchgeführt?

Aktive Prüfungen werden durch die Prüfungslogik im Nagios-Daemon veranlasst. Wenn Nagios den Status eines Hosts oder Services prüfen muss, wird es ein Plugin ausführen und die Informationen übergeben, was geprüft werden soll. Das Plugin wird dann den Betriebszustand des Hosts oder Service prüfen und die Ergebnisse an den Nagios-Daemon zurückmelden. Nagios wird die Ergebnisse der Host- oder Service-Prüfung verarbeiten und entsprechend notwendige Aktionen ausführen (z.B. Benachrichtigungen versenden, Eventhandler ausfüren, usw.).

Mehr Informationen, wie Plugins arbeiten, finden Sie hier.

Wann werden aktive Prüfungen ausgeführt?

Aktive Prüfungen werden ausgeführt

- in regelmäßigen Intervallen, wie sie in den *check_interval* und *retry_interval*-Optionen in Ihren Host- und Service-Definitionen festgelegt sind
- nach Bedarf

Regelmäßig geplante Prüfungen erfolgen in Intervallen, die den Einstellungen in *check_interval* oder *retry_interval* in Ihren Host- oder Service-Definitionen entsprechen, abhängig davon, in welchem Statustyp sich der Host oder Service befindet.

Prüfungen nach Bedarf werden ausgeführt, wann immer Nagios die Notwendigkeit sieht, die neuesten Statusinformationen über einen bestimmten Host oder Service zu ermitteln. Wenn Nagios beispielsweise die Erreichbarkeit eines Hosts feststellt, wird es oft Prüfungen von Eltern- und Kind-Hosts durchführen, um den genauen Status eines bestimmten Netzwerk-Segments zu ermitteln. Prüfungen nach Bedarf finden sich auch in der vorausschauenden Abhängigkeitsprüfungs-Logik, um sicherzustellen, dass Nagios möglichst genaue Statusinformationen hat.

Passive Prüfungen (Passive Checks)

Hoch zu: Inhalt

Siehe auch: Active Checks, Service Checks, Host Checks

Einführung

In den meisten Fällen werden Sie Nagios nutzen, um Ihre Hosts und Services mit Hilfe von regelmäßig geplanten aktiven Prüfungen zu überwachen. Aktive Prüfungen können genutzt werden, um ein Gerät oder Service gelegentlich "abzufragen". Nagios unterstützt auch einen Weg, Hosts und Services passiv zu überwachen statt aktiv. Die Hauptmerkmale von passiven Prüfungen sind wie folgt:

- passive Prüfungen werden von externen Anwendungen/Prozessen veranlasst und ausgeführt
- Ergebnisse von passiven Prüfungen werden an Nagios zur Verarbeitung übermittelt

Der Hauptunterschied zwischen aktiven und passiven Prüfungen ist, dass aktive Prüfungen von Nagios veranlasst und ausgeführt werden, während passive Prüfungen von externen Applikationen durchgeführt werden.

Einsatzmöglichkeiten für passive Prüfungen

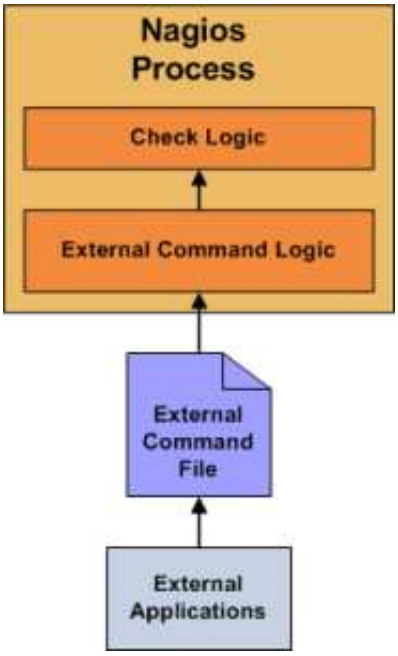
passive Prüfungen sind nützlich, um Services zu überwachen, die

- von Natur aus asynchron sind und nicht effektiv durch Abfrage ihres Zustands auf einer regelmäßig geplanten Basis überwacht werden können
- sich hinter einer Firewall befinden und nicht aktiv vom überwachenden Host aus geprüft werden können

Beispiele für asynchrone Services, bei denen sich eine passive Überwachung lohnt, sind u.a. SNMP-Traps und Sicherheits-Alarme. Sie wissen nie, wie viele (falls überhaupt) Traps oder Alarme Sie innerhalb eines vorgegebenen Zeitfensters erhalten, so dass es nicht sinnvoll ist, ihren Status alle paar Minuten zu überwachen.

Passive Prüfungen werden auch genutzt, um verteilte oder redundante Überwachungsinstallationen zu konfigurieren.

Wie passive Prüfungen arbeiten



Hier nun mehr Details, wie passive Prüfungen arbeiten...

1. eine externe Applikation prüft den Status eines Hosts oder Service.
2. die externe Applikation schreibt die Ergebnisse der Prüfung in das external command file.
3. das nächste Mal, wenn Nagios das "external command file" liest, wird es die Ergebnisse aller passiven Prüfungen zur späteren Verarbeitung in eine Queue stellen. Dieselbe Queue, die für die Speicherung von Ergebnissen von aktiven Prüfungen genutzt wird, wird auch für die Speicherung von Ergebnissen von aktiven Prüfungen verwendet.
4. Nagios wird periodisch ein check result reaper event ausführen und die Ergebnis-Queue abfragen. Jedes Service-Prüfungs-Ergebnis, das in der Queue gefunden wird, wird in der gleichen Weise bearbeitet - unabhängig davon, ob die Prüfung aktiv oder passiv war. Nagios kann abhängig vom Prüfergebnis Benachrichtigungen senden, Alarme protokollieren, usw.

Die Verarbeitung von aktiven und passiven Prüfungsergebnissen ist tatsächlich identisch. Dies erlaubt eine nahtlose Integration von externen Applikationen mit Nagios.

Passive Prüfungen aktivieren

Um passive Prüfungen in Nagios zu aktivieren, müssen Sie folgendes tun:

- setzen Sie die accept_passive_service_checks-Direktive auf 1.
- setzen Sie die *passive_checks_enabled*-Direktive in Ihren Host- und Service-Definitionen auf 1.

Wenn Sie die Verarbeitung von passiven Prüfungen global deaktivieren wollen, setzen Sie die accept_passive_service_checks-Direktive auf 0.

Wenn Sie die Verarbeitung von passiven Prüfungen nur für ein paar Hosts oder Services deaktivieren wollen, nutzen Sie die *passive_checks_enabled*-Direktive in den Host- und/oder Service-Definitionen.

Übermitteln von passiven Service-Prüfungsergebnissen

Externe Applikationen können passive Prüfungsergebisse an Nagios übermitteln, indem sie ein PROCESS_SERVICE_CHECK_RESULT external command in das "external command file" schreiben.

Das Format des Befehls lautet wie folgt:

```
[<Zeitstempel>] PROCESS_SERVICE_CHECK_RESULT;<host_name>;<svc_description>;<return_code>;<plugin_output>
```

wobei...

- *timestamp* ist die Zeit im time_t-Format (Sekunden seit der UNIX-Epoche), zu der die Service-Prüfung durchgeführt (oder übermittelt) wurde. Bitte beachten Sie das einzelne Leerzeichen nach der rechten Klammer.
- *host_name* ist der Kurzname des Hosts, der mit dem Service in der Service-Definition verbunden ist
- *svc_description* ist die Beschreibung des Service wie in der Service-Definition angegeben
- *return_code* ist der Return-Code der Prüfung (0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN)
- *plugin_output* ist die Textausgabe der Service-Prüfung (also die Ausgabe des Plugins)

 Anmerkung: ein Service muss in Nagios definiert sein, bevor Sie passive Prüfungen für ihn abliefern können! Nagios wird alle Prüfergebnisse für Services ignorieren, die nicht konfiguriert waren, bevor es das letzte Mal (neu) gestartet wurde.

 Ein Beispiel-Shell-Script, wie man passive Service-Prüfungsergebnisse an Nagios übermittelt, finden Sie in der Dokumentation zu sprunghaften Services.

Übermitteln von passiven Host-Prüfungsergebnissen

Externe Applikationen können passive Host-Prüfungsergebisse an Nagios übermitteln, indem sie ein PROCESS_HOST_CHECK_RESULT external command in das "external command file" schreiben.

Das Format des Befehls lautet wie folgt:

```
[<timestamp>] PROCESS_HOST_CHECK_RESULT;<host_name>;<host_status>;<plugin_output>
```

wobei...

- *timestamp* ist die Zeit im time_t-Format (Sekunden seit der UNIX-Epoche), zu der die Host-Prüfung durchgeführt (oder übermittelt) wurde. Bitte beachten Sie das einzelne Leerzeichen nach der rechten Klammer.
- *host_name* ist der Kurzname des Hosts (wie in der Host-Definition angegeben)
- *host_status* ist der Status des Hosts (0=UP, 1=DOWN, 2=UNREACHABLE)
- *plugin_output* ist die Textausgabe der Host-Prüfung (also die Ausgabe des Plugins)

 Anmerkung: ein Host muss in Nagios definiert sein, bevor Sie passive Prüfungen für ihn abliefern können! Nagios wird alle Prüfergebnisse für Hosts ignorieren, die nicht konfiguriert waren, bevor es das letzte Mal (neu) gestartet wurde.

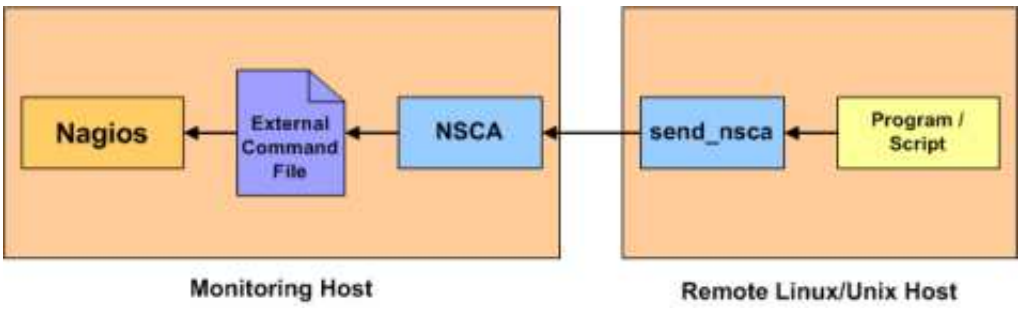
Passive Prüfungen und Host-Zustände

Nagios versucht bei passiven Prüfungen - anders bei aktiven Prüfungen - nicht festzustellen, ob der Host DOWN oder UNREACHABLE ist. Statt dessen nimmt Nagios das passive Prüfergebnis als den wahren Status des Hosts und versucht nicht, den wahren Host-Status mit Hilfe der Erreichbarkeitslogik zu ermitteln. Dies kann Probleme verursachen, wenn Sie passive Prüfungen von einem entfernten Host übermitteln oder Sie ein verteiltes Überwachungs-Setup haben, in dem Eltern/Kind-Verhältnisse unterschiedlich sind.

Sie können Nagios anweisen, die passiven Prüfergebnisse DOWN/UNREACHABLE-Zustände mit Hilfe der translate_passive_host_checks-Variable in ihre "sauberen" Zustände zu übersetzen. Mehr Informationen wie dies funktioniert, finden Sie hier.

Anmerkung: Passive Host-Prüfungen werden normalerweise als HARD-Zustände behandelt, falls nicht die passive_host_checks_are_soft-Option aktiviert ist.

Übermitteln von passiven Prüfungsergebnissen von entfernten Hosts



Wenn eine Applikation, die sich auf dem gleichen Host wie Nagios befindet, passive Host- oder Service-Prüfungsergebnisse sendet, kann es die Ergebisse einfach direkt in das "external command file" schreiben wie oben skizziert. Allerdings können entfernte Hosts das nicht so einfach tun.

Um es entfernten Hosts zu erlauben, passive Prüfungsergebnisse an den überwachenden Host zu senden, habe ich das NSCA-Addon entwickelt. Das NSCA-Addon besteht aus einem Daemon, der auf dem Nagios-Host läuft und einem Client, der auf entfernten Hosts ausgeführt wird. Der Daemon lauscht auf Verbindungen von entfernten Hosts, führt mit den Ergebnissen einige grundlegende Gültigkeitsprüfungen durch und schreibt die Prüfergebnisse direkt in das "external command file" (wie oben beschrieben). Mehr Informationen über das NSCA-Addon finden Sie hier.

Statustypen

Hoch zu: Inhalt

Siehe auch: Host-Prüfungen, Service-Prüfungen, Eventhandler, Benachrichtigungen

Einführung

Der aktuelle Status von überwachten Services und Host wird durch zwei Komponenten ermittelt:

- den Status des Service oder Host (d.h. OK, WARNING, UP, DOWN, etc.)
- den *Typ* des Zustands, in dem der Service oder Host ist

Es gibt zwei Statustypen in Nagios: SOFT- und HARD-Zustände. Diese Statustypen sind ein wichtiger Teil der Überwachungslogik, da sie zur Ermittlung dienen, wann Eventhandler ausgeführt und Benachrichtigungen zuerst versandt werden.

Dieses Dokument beschreibt den Unterschied zwischen SOFT- und HARD-Zuständen, wann sie auftreten und was passiert, wenn sie auftreten.

Service- und Host-Prüfungswiederholungen

Um falsche Alarme bei vorübergehenden Problemen zu verhindern, erlaubt Ihnen Nagios zu definieren, wie oft ein Service oder Host (erneut) geprüft werden soll, bevor es als "echtes" Problem angesehen werden soll. Dies wird durch die *max_check_attempts*-Option in den Host- und Service-Definitionen kontrolliert. Zu verstehen, wie Hosts und Services (erneut) geprüft werden, um festzustellen, ob ein echtes Problem besteht, ist wichtig zum Verstehen, wie Statustypen arbeiten.

Soft-Zustände

Soft-Zustände treten in den folgenden Situationen auf...

- wenn ein Host- oder Service-Prüfungsergebnis in einem nicht-OK oder nicht-UP-Status resultiert und die Service-Prüfung noch nicht so oft (erneut) durchgeführt wurde, wie es in der *max_check_attempts*-Direktive der Service- oder Host-Definition angegeben wurde. Das wird als Soft-Error bezeichnet.
- wenn sich ein Service oder Host von einem Soft-Error erholt. Das wird als Soft-Recovery angesehen.

Die folgenden Dinge passieren, wenn bei Hosts oder Services SOFT-Zustandsänderungen auftreten:

- der SOFT-Status wird protokolliert.
- Eventhandler werden zur Behandlung von SOFT-Zuständen ausgeführt

SOFT-Zustände werden nur protokolliert, wenn Sie die log_service_retries- oder die log_host_retries-Option in Ihrer Hauptkonfigurationsdatei aktiviert haben.

Das einzig Wichtige, was bei einem Soft-Zustand passiert, ist die Ausführung von Eventhandlern. Eventhandler zu benutzen kann insbesondere dann nützlich sein, wenn Sie versuchen wollen, proaktiv ein Problem zu lösen, bevor es sich in einen HARD-Zustand verwandelt. Die $HOSTSTATETYPE$- oder $SERVICESTATETYPE$-Makros werden den Wert "*SOFT*" haben, wenn Eventhandler ausgeführt

werden, was es Ihren Eventhandlern erlaubt zu wissen, wann sie fehlerbehebende Aktionen vornehmen sollen. Mehr Informationen zu Eventhandlern finden Sie hier.

Hard-Zustände

Hard-Zustände treten für Hosts und Services in den folgenden Situationen auf...

- wenn ein Host- oder Service-Prüfungsergebnis in einem nicht-OK oder nicht-UP-Status resultiert und die Prüfung bereits so oft (erneut) durchgeführt wurde, wie es in der *max_check_attempts*-Direktive der Service- oder Host-Definition angegeben wurde. Das wird als Hard-Error bezeichnet.
- wenn ein Host oder Service von einem Hard-Error-Zustand in einen anderen Fehlerzustand wechselt (z.B. von WARNING nach CRITICAL).
- wenn eine Service-Prüfung in einem nicht-OK-Status resultiert und der zugehörige Host entweder DOWN oder UNREACHABLE ist.
- wenn ein Host oder Service sich von einem Hard-Error-Zustand erholt. Dies wird als Hard-Recovery angesehen.
- wenn eine passive Host-Prüfung empfangen wird. Passive Host-Prüfungen werden als HARD angesehen, wenn nicht die passive_host_checks_are_soft-Option aktiviert ist.

Die folgenden Dinge passieren, wenn bei Hosts oder Services HARD-Zustandsänderungen auftreten:

- der HARD-Status wird protokolliert.
- Eventhandler werden zur Behandlung von HARD-Zuständen ausgeführt.
- Kontakte werden über das Host- oder Service-Problem bzw. die Erholung informiert.

Die $HOSTSTATETYPE$ oder $SERVICESTATETYPE$-Makros werden den Wert "*HARD*" haben, wenn Eventhandler ausgeführt werden, was es Ihren Eventhandlern erlaubt zu wissen, wann sie fehlerbehebende Aktionen vornehmen sollen. Mehr Informationen zu Eventhandlern finden Sie hier.

Beispiel

Hier ist ein Beispiel, wie Statustypen ermittelt werden, wenn Statusänderungen auftreten und wann Eventhandler ausgeführt und Benachrichtigungen versandt werden. Die nachfolgende Tabelle zeigt aufeinander folgende Prüfungen eines Service. Der Service hat einen *max_check_attempts*-Wert von 3.

Zeit	Prüfung #	Status	Statustyp	Statuswechsel	Anmerkungen
0	1	OK	HARD	Nein	Initialer Zustand des Service
1	1	CRITICAL	SOFT	Ja	erstes Erkennen eines nicht-OK-Zustandes. Eventhandler wird ausgeführt.
2	2	WARNING	SOFT	Ja	Service bleibt in einem nicht-OK-Zustand. Eventhandler wird ausgeführt.
3	3	CRITICAL	HARD	Ja	"max_check_attempts" wurde erreicht, deshalb geht der Service in einen HARD-Zustand. Eventhandler wird ausgeführt und eine Benachrichtigung versandt. Die Check-Anzahl wird auf 1 zurückgesetzt, sofort nachdem dies passiert.
4	1	WARNING	HARD	Ja	Service wechselt in einen HARD-WARNING-Status. Eventhandler wird ausgeführt und eine Problembenachrichtigung versandt.
5	1	WARNING	HARD	Nein	Service stabilisiert sich zu einem HARD-Problemzustand. Abhängig vom Benachrichtigungsintervall für den Service wird ggf. eine weitere Benachrichtigung verschickt.
6	1	OK	HARD	Ja	Service erfährt eine HARD-Recovery. Eventhandler wird ausgeführt und eine Erholungs-Benachrichtigung wird versandt.
7	1	OK	HARD	Nein	Service ist weiterhin OK.
8	1	UNKNOWN	SOFT	Ja	Für den Service wird ein Wechsel zu einem SOFT nicht-OK-Zustand festgestellt. Eventhandler wird ausgeführt.
9	2	OK	SOFT	Ja	Service erfährt eine SOFT-Recovery. Eventhandler wird ausgeführt, aber keine Benachrichtigung versandt, weil dies kein "echtes" Problem war. Der Statustyp wird auf HARD gesetzt und die Check-Anzahl auf 1 zurückgesetzt, sofort nachdem dies passiert.
10	1	OK	HARD	Nein	Service stabilisiert sind zu einem OK-Status.

Zeitfenster

oder...
"Is This a Good Time?"

Hoch zu: Inhalt

Siehe auch: Rufbereitschaft-Rotation, Host-Prüfungen, Service-Prüfungen, Benachrichtigungen, Benachrichtigungs-Eskalationen, Abhängigkeiten

Einführung

Zeitfenster-Definitionen erlauben Ihnen zu kontrollieren, wann verschiedene Aspekte der Überwachungs- und Alarmierungslogik arbeiten. Zum Beispiel können Sie einschränken

- wann regelmäßig geplante Host- und Service-Prüfungen ausgeführt werden
- wann Benachrichtigungen versandt werden
- wann Benachrichtigungs-Eskalationen benutzt werden können
- wann Abhängigkeiten gültig sind

Vorrang bei Zeitfenstern

Zeitfenster-Definitionen können mehrere Typen von Direktiven enthalten, einschließlich Wochentagen, Monatstagen und Kalenderdaten. Verschiedene Typen von Direktiven haben unterschiedliche Vorrang-Ebenen und können andere Direktiven in Ihren Zeitfenster-Definitionen außer Kraft setzen. Die Rangfolge für verschiedene Typen von Direktiven (in absteigender Reihenfolge) ist wie folgt:

- Kalenderdaten (2008-01-01)
- angegebener Tag des Monats (January 1st)
- generischer Tag des Monats (Day 15)
- Offset Wochentag eines bestimmten Monats (2nd Tuesday in December)
- Offset Wochentag (3rd Monday)
- normaler Wochentag (Tuesday)

Beispiele für verschiedene Zeitfenster-Direktiven finden Sie hier.

Wie Zeitfenster mit Host- und Service-Prüfungen arbeiten

Host- und Service-Definitionen haben eine optionale *check_period*-Direktive, die es Ihnen erlaubt, ein Zeitfenster anzugeben, das zur Einschränkung benutzt werden sollte, wann regelmäßig geplante aktive Prüfungen des Hosts oder Service stattfinden.

Wenn Sie die *check_period*-Direktive nicht nutzen, um ein Zeitfenster anzugeben, wird Nagios in der Lage sein, aktive Prüfungen für den Host oder Service zu jeder Zeit zu planen, wenn es nötig ist. Dies ist in Wirklichkeit ein 24x7-Überwachungsszenario.

Ein Zeitfenster in der *check_period*-Direktive anzugeben erlaubt Ihnen die Einschränkung der Zeit, wann Nagios regelmäßige aktive Host- oder Service-Prüfungen plant. Wenn Nagios versucht, einen Host oder Service neu zu planen, wird es sicherstellen, dass die nächste Prüfung in einen gültigen Zeitbereich im definierten Zeitfenster fällt. Falls das nicht zutreffen sollte, wird Nagios die Zeit der nächsten Prüfung so anpassen, dass sie in die nächste "gültige" Zeit im angegebenen Zeitfenster fällt. Das bedeutet, dass der Host oder Service vielleicht während der nächsten Stunde, des nächsten Tages oder der nächsten Woche, etc. nicht geprüft wird.

 Anmerkung: Prüfungen nach Bedarf und passive Prüfungen sind nicht durch das Zeitfenster beschränkt, das Sie in der *check_period*-Direktive angeben. Nur regelmäßig geplante aktive Prüfungen werden beschränkt.

Außer Sie haben einen guten Grund das zu tun, würde ich raten, dass Sie all Ihre Hosts und Services mit einem Zeitfenster überwachen, das einen 24x7-Zeitbereich abdeckt. Falls Sie das nicht tun, können Sie während der "blackout"-Zeiten in einige Probleme laufen (Zeiten, die nicht gültig sind in der Zeitfenster-Definition):

1. der Status des Hosts oder Service wird in der blackout-Zeit unverändert erscheinen.
2. Kontakte werden während der blackout-Zeit wahrscheinlich nicht erneut über Host- oder Service-Probleme informiert werden.
3. falls sich ein Host oder Service während einer blackout-Zeit erholt, werden Kontakte nicht umgehend über die Erholung informiert.

Wie Zeitfenster mit Kontakt-Benachrichtigungen arbeiten

Durch das Angeben eines Zeitfensters in der *notification_period*-Direktive einer Host- oder Service-Definition kontrollieren Sie, wann Nagios Benachrichtigungen versenden darf, um über Probleme oder Erholungen für den Host oder Service zu informieren. Wenn eine Host-Benachrichtigung versandt werden soll, prüft Nagios, ob die aktuelle Zeit in einem gültigen Bereich der *notification_period* liegt. Wenn eine gültige Zeit vorliegt, wird Nagios versuchen, jeden Kontakt über das Problem oder die Erholung zu informieren.

Sie können Zeitfenster auch nutzen, um zu kontrollieren, wann Benachrichtigungen an einzelne Kontakte versandt werden. Durch die Nutzung der *service_notification_period*- und der *host_notification_period*-Direktiven in den Kontakt-Definitionen sind Sie in der Lage, eine tatsächliche Rufbereitschaft für jeden Kontakt zu definieren. Kontakte werden Host- und Service-Benachrichtigungen nur während der Zeiten erhalten, die Sie in den Benachrichtigungs-Direktiven angegeben haben.

Beispiele, wie Zeitfenster-Definitionen für Rufbereitschafts-Wechsel angelegt werden, finden Sie hier.

Wie Zeitfenster mit Benachrichtigungs-Eskalationen arbeiten

Service- und Host-Benachrichtigungs-Eskalationen haben eine optionale *escalation_period*-Direktive, die es Ihnen erlaubt ein Zeitfenster anzugeben, wann die Eskalation gültig ist und benutzt werden kann. Wenn Sie die *escalation_period*-Direktive nicht in einer Eskalations-Definition benutzen, ist diese Eskalation zu allen Zeiten gültig. Wenn Sie ein Zeitfenster in der *escalation_period*-Direktive angeben, wird Nagios die Eskalations-Definition nur zu Zeiten nutzen, die aufgrund der Zeitfenster-Definition gültig sind.

Wie Zeitfenster mit Abhängigkeiten arbeiten

Service- und Host-Abhängigkeiten haben eine optionale *dependency_period*-Direktive, die es Ihnen erlaubt ein Zeitfenster anzugeben, wann die Abhängigkeit gültig ist und benutzt werden kann. Wenn Sie die *dependency_period*-Direktive nicht in einer Abhängigkeits-Definition benutzen, ist diese Abhängigkeit zu allen Zeiten gültig. Wenn Sie ein Zeitfenster in der *dependency_period*-Direktive angeben, wird Nagios die Abhängigkeits-Definition nur zu Zeiten nutzen, die aufgrund der Zeitfenster-Definition gültig sind.

Ermitteln des Zustands und der Erreichbarkeit von Netzwerk-Hosts

Hoch zu: Inhalt

Siehe auch: Host-Prüfungen, Passive Host-Zustands-Übersetzung

Einführung

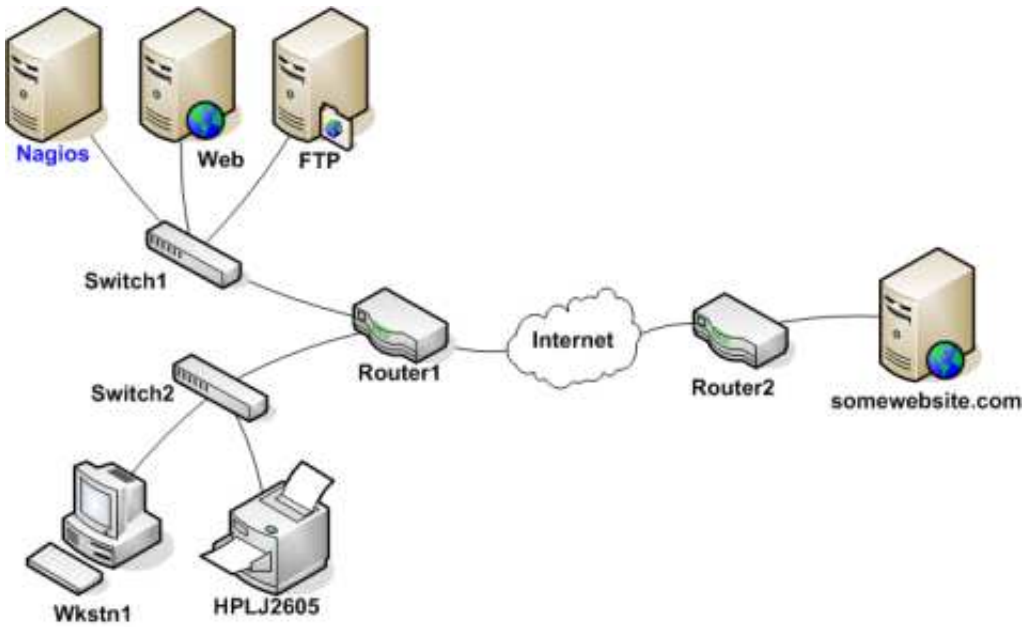
Falls Sie jemals im technischen Support gearbeitet haben, hatten Sie zweifelsohne Benutzer, die Ihnen erzählt haben, "das Internet sei down". Als Techniker waren Sie ziemlich sicher, daß keiner den Stromstecker aus dem Internet gezogen hatte. Irgendetwas muß schiefgehen zwischen dem Stuhl des Benutzers und dem Internet.

Angenommen es ist ein technisches Problem, dann werden Sie nach dem Problem suchen. Vielleicht ist der PC des Benutzers ausgeschaltet oder das Netzwerkkabel ist gezogen oder der zentrale Router Ihres Unternehmen nimmt gerade eine Auszeit. Was immer das Problem sein mag, eines ist sehr sicher - das Internet ist nicht down. Es ist lediglich nicht für den Benutzer erreichbar.

Nagios ist in der Lage festzustellen, ob die Hosts, die Sie überwachen, in einem DOWN- oder UNREACHABLE-Zustand sind. Dies sind sehr unterschiedliche (obwohl durchaus verwandte) Zustände und können Ihnen helfen, schnell die Grundursache für Netzwerkprobleme festzustellen. Hier nun, wie die Netzwerk-Erreichbarkeitslogik arbeitet, um zwischen diesen beiden Zuständen zu unterscheiden...

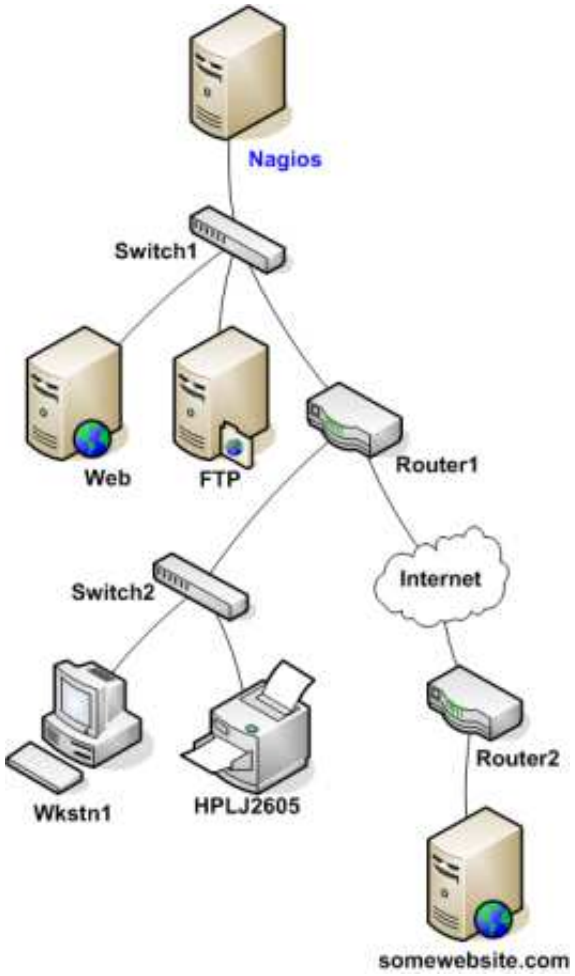
Beispiel-Netzwerk

Werfen Sie einen Blick auf das einfache Netzwerk-Diagramm. Lassen Sie uns annehmen, dass Sie alle Hosts (Server, Router, Switches, etc.) überwachen, die abgebildet sind. Nagios ist installiert und lauffähig auf dem *Nagios*-Host.



Definieren von Eltern/Kind-Beziehungen

Um Nagios in die Lage zu versetzen, zwischen DOWN und UNREACHABLE-Zuständen der überwachten Hosts zu unterscheiden, müssen Sie Nagios mitteilen, wie diese Hosts miteinander verbunden sind - vom Standpunkt des Nagios-Daemons aus gesehen. Um dies zu tun verfolgen Sie den Weg, den ein Datenpaket vom Nagios-Daemon zu jedem einzelnen Host nehmen würde. Jeder Switch, Router und Server, den das Paket trifft oder passiert, wird als "Hop" angesehen und erfordert, dass Sie eine Eltern/Kind-Beziehung in Nagios definieren. Hier nun, wie die Host-Eltern/Kind-Beziehung aus der Sicht von Nagios aussieht:



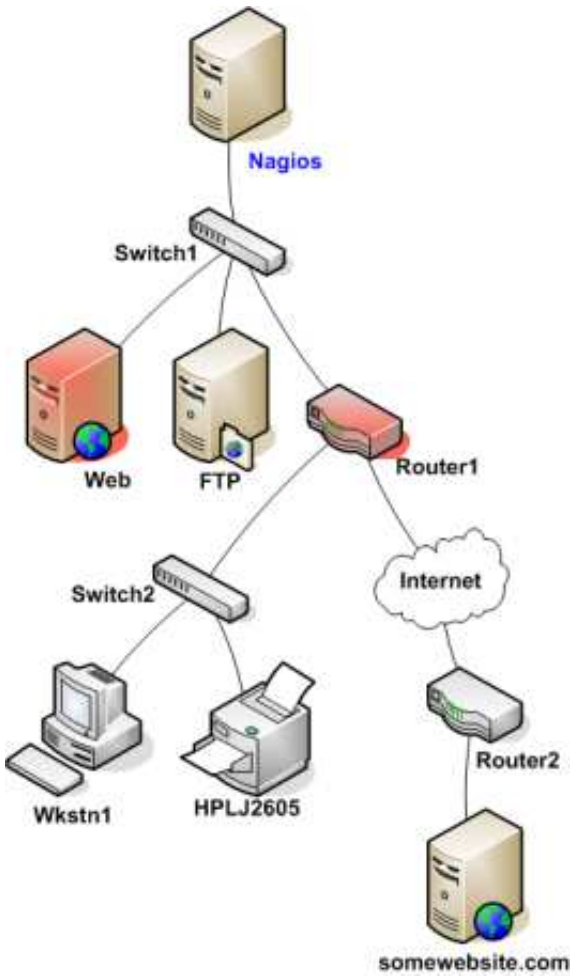
Nun, da Sie wissen, wie die Eltern/Kind-Beziehungen für überwachte Hosts aussehen, wie konfigurieren Sie Nagios, um sie abzubilden? Die *parents*-Direktive in Ihren Host-Definitionen erlaubt Ihnen, das zu tun. Hier nun, wie die (verkürzten) Host-Definitionen mit Eltern/Kind-Beziehung für dieses Beispiel aussehen würden:

```
define host{
        host_name               Nagios   ; <-- der lokale Host hat keine Eltern - es ist der am weitesten oben stehende Host
        }

define host{
        host_name       Switch1
        parents         Nagios
        }

define host{
        host_name       Web
        parents         Switch1
        }

define host{
        host_name       FTP
        parents         Switch1
```

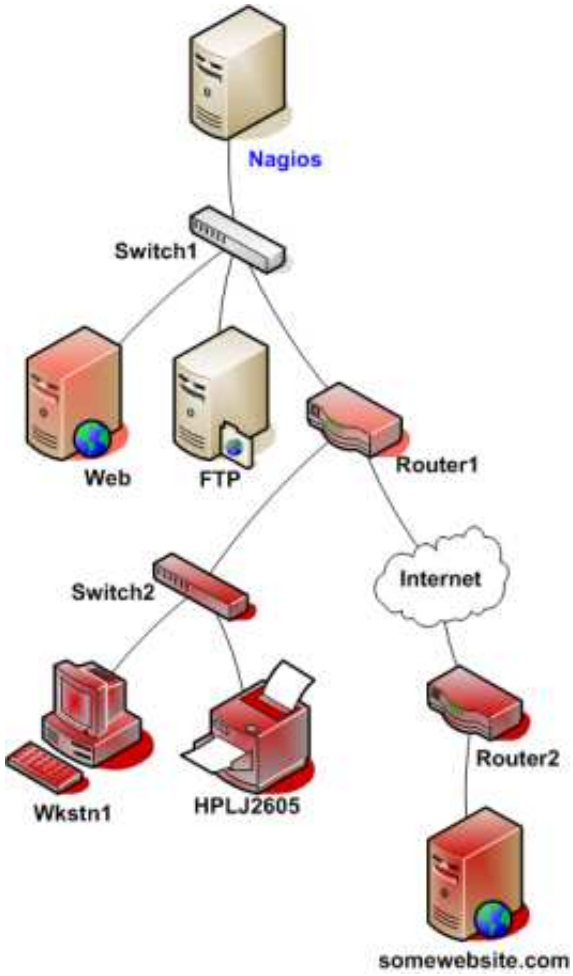
```
        }

define host{
        host_name       Router1
        parents         Switch1
        }

define host{
        host_name       Switch2
        parents         Router1
        }

define host{
        host_name       Wkstn1
        parents         Switch2
        }

define host{
        host_name       HPLJ2605
        parents         Switch2
        }

define host{
        host_name       Router2
        parents         Router1
        }

define host{
        host_name       somewebsite.com
        parents         Router2
        }
```

Erreichbarkeits-Logik in Aktion

Nachdem Sie Nagios mit den passenden Eltern/Kind-Beziehungen konfiguriert haben, lassen Sie uns sehen, was passiert, wenn Probleme auftauchen. Nehmen Sie an, dass zwei Hosts, *Web* und *Router1*, offline gehen...



Wenn Hosts den Status wechseln (d.h. von UP zu DOWN) wird die Host-Erreichbarkeitslogik in Nagios anspringen. Die Erreichbarkeits-Logik wird parallele Prüfungen der Eltern und Kinder aller Hosts veranlassen, deren Status sich ändert. Dies erlaubt es Nagios schnell den aktuellen Status Ihrer Netzwerk-Infrastruktur zu ermitteln, wenn Änderungen auftreten.



In diesem Beispiel wird Nagios feststellen, dass *Web* und *Router1* beide im DOWN-Status sind, weil der "Pfad" zu diesen Hosts nicht blockiert ist.

Nagios wird feststellen, dass alle Hosts "unterhalb" *Router1* alle in einem UNREACHABLE Status sind, weil Nagios sie nicht erreichen kann. *Router1* ist DOWN und blockiert den Weg zu diesen anderen Hosts. Diese Hosts können wunderbar funktionieren oder offline sein - Nagios weiß es nicht, weil es sie nicht erreichen kann. Deshalb wird Nagios sie als UNREACHABLE ansehen anstatt DOWN.

UNREACHABLE Zustände und Benachrichtigungen

Standardmäßig wird Nagios Kontakte über Hosts im DOWN und UNREACHABLE-Status informieren. Als ein Admin/Techniker möchten Sie vielleicht keine Benachrichtigungen über Hosts erhalten, die UNREACHABLE sind. Sie kennen Ihre Netzwerkstruktur und wenn Nagios Sie informiert, dass der Router/die Firewall unten ist, dann wissen Sie, dass alles dahinter nicht erreichbar ist.

Falls Sie sich eine Flut von Benachrichtigungen über UNREACHABLE-Zustände während eines Netzwerkausfalls ersparen möchten, können Sie die unreachable (u)-Option der *notification_options*-Direktive in Ihren Host-Definitionen und/oder die *host_notification_options*-Direktive in Ihren Kontakt-Direktiven ausschließen.

Benachrichtigungen

Hoch zu: Inhalt

Siehe auch: Eskalationen, Zeitfenster, Rufbereitschafts-Wechsel

Einführung

Ich hatte eine Menge Fragen, wie genau Benachrichtigungen arbeiten. Ich werde versuchen, genau zu erklären, wann und wie Host- und Service-Benachrichtigungen versandt werden und ebenso, wer sie bekommt.

Benachrichtigungs-Eskalationen werden hier beschrieben.

Wann erfolgen Benachrichtigungen?

Die Entscheidung, Benachrichtigungen zu senden, wird in der Service- und Host-Prüflogik getroffen. Host- und Service-Benachrichtigungen erfolgen in den folgenden Fällen...

- wenn ein HARD-Statuswechsel erfolgt. Mehr Informationen über Statustypen und Hard-Statuswechsel finden Sie hier.
- wenn ein Host oder Service in einem Hard nicht-OK-Zustand bleibt und die in der <*notification_interval*>-Option der Host- oder Service-Definition angegebene Zeit seit der letzten versandten Benachrichtigung verstrichen ist (für den angegebenen Host oder Service).

Wer wird benachrichtigt?

Jede Host- und Service-Definition hat eine <*contact_groups*>-Option, die angibt, welche Kontaktgruppen Benachrichtigungen für bestimmte Hosts oder Services erhalten. Kontaktgruppen können ein oder mehrere einzelne Kontakte enthalten.

Wenn Nagios eine Host- oder Service-Benachrichtigung versendet, wird es jeden Kontakt informieren, der Mitglied in einer der Kontaktgruppen ist, die in der <*contactgroups*>-Option der Service-Definition angegeben ist. Nagios bemerkt, wenn ein Kontakt Mitglied von mehr als einer Kontaktgruppe ist und entfernt mehrfache Kontaktbenachrichtigungen, bevor es irgendetwas tut.

Welche Filter müssen durchlaufen werden, damit Benachrichtigungen versandt werden?

Nur weil Benachrichtigungen für einen Host- oder Service versandt werden müssen, bedeutet das nicht, dass irgendein Kontakt informiert wird. Es gibt mehrere Filter, die potenzielle Benachrichtungen durchlaufen müssen, bevor sie als würdig genug angesehen werden, um versandt zu werden. Lassen Sie uns einen genaueren Blick auf die Filter werfen, die zu durchlaufen sind...

Programmweite Filter:

Der erste Filter, den Benachrichtigungen durchlaufen müssen, ist ein Test, ob Benachrichtigungen auf einer programmweiten Basis aktiviert sind. Dies wird ursprünglich durch die enable_notifications-Option in der Hauptkonfigurationsdatei festgelegt, kann aber während der Laufzeit über das Web-Interface verändert werden. Falls Benachrichtigungen auf programmweiter Basis deaktiviert sind, werden keine Benachrichtigungen für Hosts oder Services versandt - Punkt. Wenn sie auf programmweiter Basis aktiviert sind, müssen weitere Tests durchlaufen werden...

Service- und Host-Filter:

Der erste Filter für Host- oder Service-Benachrichtigungen ist eine Prüfung, ob sich der Host oder Service in einer geplanten Ausfallzeit (downtime) befindet. Falls es eine geplante Ausfallzeit ist, **wird niemand informiert**. Wenn es keine Ausfallzeit ist, geht es weiter zum nächsten Filter. Als kleine Randnotiz: Service-Benachrichtigungen werden unterdrückt, falls sich der mit ihnen verbundene Host in einer geplanten Ausfallzeit befindet.

Der zweite Filter für Host- oder Service-Benachrichtigungenen ist eine Prüfung, ob der Host oder Service flattert (wenn Sie Flatter-Erkennung aktiviert haben). Falls der Service oder Host gerade flattert, **wird niemand informiert**. Andernfalls geht es weiter zum nächsten Filter.

Der dritte für Hosts oder Services zu durchlaufende Filter sind die Host- oder Service-spezifischen Benachrichtigungsoptionen. Jede Service-Definition enthält Optionen, die festlegen, ob Benachrichtungen für Warnungen, kritische Zustände oder Erholungen versandt werden oder nicht. Ähnlich ist es bei Hosts, wo festgelegt wird, ob Benachrichtigungen versandt werden, wenn der Host down geht, unerreichbar wird oder sich wieder erholt. Falls die Host- oder Service-Benachrichtigungen diese Optionen nicht passieren, **wird niemand informiert**. Wenn sie die Optionen durchlaufen, geht es zum nächsten Filter... Anmerkung: Benachrichtigungen über Host- oder Service-Erholungen werden nur dann versandt, wenn auch eine Benachrichtigung über das ursprüngliche Problem versandt wurde. Es ist nicht sinnvoll, eine Benachrichtigung über eine Erholung zu bekommen, wenn Sie nicht wussten, dass ein Problem existiert.

Der vierte Host- oder Service-Filter, der durchlaufen werden muss, ist der Zeitfenster-Test. Jede Host- und Service-Definition hat eine <*notification_period*>-Option, die angibt, welches Zeitfenster gültige Benachrichtigungszeiten für den Host oder Service enthält. Wenn die Zeit der Benachrichtigung nicht in einen gültigen Bereich des Zeitfensters fällt, **wird niemand informiert**. Wenn sie in einen gültigen Bereich fällt, geht es zum nächsten Filter... Anmerkung: falls der Zeitfenster-Filter nicht erfolgreich durchlaufen wird, plant Nagios die nächste Benachrichtigung für den Host oder Service (falls er sich in einem nicht-OK-Status befindet) für die nächste verfügbare gültige Zeit im Zeitfenster. Dies stellt sicher, dass der Kontakt so früh wie möglich über Probleme informiert wird, wenn die nächste gültige Zeit erreicht wird.

Der letzte Satz von Host- oder Service-Filter ist abhängig von zwei Dingen: (1) zu einem Zeitpunkt in der Vergangenheit wurde bereits eine Benachrichtigung über ein Problem mit dem Host oder Service versandt und (2) blieb der Host oder Service im gleichen nicht-OK-Zustand, der zur Zeit der Benachrichtigung vorlag. Wenn diese beiden Kriterien zutreffen, wird Nagios prüfen und sicherstellen, dass die seit der letzten Benachrichtigung vergangene Zeit den in der Option <*notification_interval*> angegebenen Wert in der Host- oder Service-Definition erreicht oder übertrifft. Falls nicht genug Zeit seit der letzten Benachrichtigung vergangen ist, **wird niemand benachrichtigt**. Wenn entweder genug Zeit seit der letzten Benachrichtigung vergangen ist oder die beiden Kriterien dieses Filters erfüllt wurden, wird die Benachrichtigung versandt. Ob sie tatsächlich an einzelne Kontakte versandt wird, hängt von einem weiteren Satz von Filtern ab...

Kontakt-Filter:

An diesem Punkt hat die Benachrichtigung die programmweiten und alle Host- und Service-Filter durchlaufen und Nagios beginnt, alle betroffenen Leute zu informieren. Bedeutet dies, dass jeder Kontakt die Benachrichtigung erhalten wird? Nein. Jeder Kontakt hat seinen eigenen Satz von Filtern, den die Benachrichtigung passieren muss. Anmerkung: Kontaktfilter sind spezifisch für jeden Kontakt und beeinflussen nicht, ob andere Kontakte Benachrichtigungen erhalten oder nicht.

Der erste zu passierende Filter für jeden Kontakt sind die Benachrichtigungsoptionen. Jede Kontaktdefinition enthält Optionen, die festlegen, ob Service-Benachrichtigungen für Warning- und Critital-Zustände und Erholungen versandt werden können. Jede Kontakt-Definition enthält auch Optionen, die festlegen, ob Host-Benachrichtigungen versandt werden, wenn der Host "down" geht, unerreichbar wird oder sich erholt. Falls die Host- oder Service-Benachrichtigung diese Optionen nicht passieren kann, **wird der Kontakt nicht informiert**. Wenn es diese Optionen passiert, wird die Benachrichtigung an den nächsten Filter weitergereicht... Anmerkung: Benachrichtigungen über die Erholung von Host oder Service werden nur dann versandt, wenn eine Benachrichtigung für das ursprüngliche Problem versandt wurde. Es ist sinnlos, eine Benachrichtigung über eine Erholung zu versenden, wenn Sie nicht wussten, dass ein Problem existiert...

Der letzte zu passierende Filter für jeden Kontakt ist der Zeitfenster-Test. Jede Kontaktdefinition hat eine <*notification_period*>-Option, die angibt, welches Zeitfenster gültige Benachrichtigungszeiten für den Kontakt enthält. Wenn die Zeit, in der die Benachrichtigung erstellt wird, nicht in ein gültiges Zeitfenster fällt, **wird der Kontakt nicht informiert**. Wenn sie in ein gültiges Zeitfenster fällt, wird der Kontakt informiert!

Benachrichtigungs-Methoden

Nagios kann Sie über Probleme und Erholungen auf vielfältige Weise informieren: Pager, Handy, e-Mail, SMS, Audio-Hinweis usw. Wie Benachrichtigungen versandt werden, hängt von den Benachrichtigungs-Befehlen ab, die in Ihren Objekt-Definitionsdateien definiert werden.

Anmerkung: Wenn Sie Nagios nach den Schnellstart-Anleitungen installieren, sollte es zum Versand von e-Mail-Benachrichtigungen konfiguriert sein. Sie können die benutzten e-Mail-Befehle ansehen, indem Sie den Inhalt der Datei */usr/local/nagios/etc/objects/commands.cfg* betrachten.

Spezielle Benachrichtigungs-Methoden (Paging usw.) sind nicht direkt in den Nagios-Code integriert, denn es ist nicht sinnvoll. Der "Kern" von Nagios ist nicht als eierlegene Wollmilchsau gedacht. Wenn Service-Prüfungen im Nagios-Kern enthalten wären, hätten Benutzer große Schwierigkeiten, neue Prüfmethoden hinzuzufügen, bestehende Prüfungen zu modifizieren usw. Benachrichtigungen arbeiten in ähnlicher Weise. Es gibt tausend verschiedene Wege, Benachrichtigungen zu versenden und es gibt bereits viele Pakete, die die schmutzige Arbeit tun, also warum das Rad neu erfinden und sich dann auf einen Fahrrad-Reifen beschränken? Es ist viel einfacher, ein externes Gebilde (das kann ein einfaches Script sein oder ein ausgewachsenes Message-System) die ganze Arbeit tun zu lassen. Einige Message-Pakete, die Benachrichtigungen für Pager und Handys verarbeiten können, sind weiter unten aufgeführt.

Benachrichtigungstyp-Makro

Wenn Sie Benachrichtigungs-Befehle erstellen, müssen Sie beachten, um welchen Typ von Benachrichtigung es sich handelt. Das Makro $NOTIFICATIONTYPE$ enthält eine Zeichenkette, die genau das angibt. Die nachfolgende Tabelle zeigt die möglichen Werte und deren entsprechende Beschreibungen:

Wert	Beschreibung
PROBLEM	Ein Host oder Service hat gerade einen Problemzustand erreicht (oder ist noch in einem). Wenn dies eine Service-Benachrichtigung ist, bedeutet das, dass der Service in einem WARNING-, UNKNOWN- oder CRITICAL-Zustand ist. Wenn dies eine Host-Benachrichtigung ist, bedeutet das, dass der Host in einem DOWN- oder UNREACHABLE-Zustand ist.
RECOVERY	Ein Service oder Host hat sich erholt. Wenn dies eine Service-Benachrichtigung ist, bedeutet es, dass der Service gerade wieder in einen OK-Zustand zurückgekehrt ist. Wenn dies eine Host-Benachrichtigung ist, bedeutet das, dass der Host gerade wieder in einen UP-Zustand zurückgekehrt ist.
ACKNOWLEDGEMENT	Diese Benachrichtigung ist eine Bestätigung für ein Host- oder Service-Problem. Bestätigungen werden von Kontakten für diesen Host oder Service über das Web-Interface ausgelöst.
FLAPPINGSTART	Der Host oder Service hat gerade angefangen zu flattern.
FLAPPINGSTOP	Der Host oder Service hat gerade aufgehört zu flattern.
FLAPPINGDISABLED	Der Host oder Service hat gerade aufgehört zu flattern, weil die Flatter-Erkennung deaktiviert wurde.
DOWNTIMESTART	Der Host oder Service hat gerade ein geplante Downtime begonnen. Weitere Benachrichtigungen werden unterdrückt.
DOWNTIMESTOP	Der Host oder Service hat gerade eine geplante Downtime beendet. Benachrichtungen über Probleme werden wieder versandt.
DOWNTIMECANCELLED	Die Phase der geplanten Downtime für den Host oder Service wurde gerade annulliert. Benachrichtungen über Probleme werden wieder versandt.

Hilfreiche Quellen

Es gibt viele Wege, wie Sie Nagios konfigurieren können, damit Benachrichtigungen versandt werden. Sobald Sie dies tun, müssen Sie notwendige Software installieren und Benachrichtigungs-Befehle konfigurieren, bevor Sie diese benutzen können. Hier sind nur ein paar mögliche Benachrichtigungs-Methoden:

- e-Mail
- Pager
- Telefon (SMS)
- WinPopup-Meldung
- Yahoo-, ICQ- oder MSN-Sofortnachricht
- Audio-Hinweise
- etc...

Im Grunde genommen kann alles, was Sie von einer Kommandozeile aus tun können, so angepasst werden, dass Sie es in einem Benachrichtigungs-Befehl nutzen können.

Wenn Sie nach einer Alternative suchen, um Meldungen per e-Mail an Ihren Pager oder Ihr Handy zu versenden, sollten Sie diese Pakete ausprobieren. Sie können in Verbindung mit Nagios dazu benutzt werden, Benachrichtigungen über ein Modem zu versenden, wenn ein Problem auftritt. Auf diese Weise müssen Sie sich nicht auf e-Mail verlassen, um Benachrichtigungen zu versenden (bedenken Sie, dass e-Mail ggf. *nicht* funktioniert, wenn es ein Netzwerk-Problem gibt). Ich habe diese Pakete nicht selbst ausprobiert, aber andere haben von erfolgreichem Einsatz berichtet...

- Gnokii (SMS-Software, um Nokia-Telefone über das GSM-Netzwerk zu erreichen)
- QuickPage (Alphanumerische Pager-Software)
- Sendpage (Paging-Software)
- SMS Client (Kommandozeilen-Utility, um Meldungen auf Pager und Mobiltelefone zu senden)

Wenn Sie eine nicht-traditionelle Methode für Benachrichtigungen ausprobieren möchten, können Sie ggf. Audio-Hinweise nutzen. Wenn Sie Audio-Hinweise auf dem Überwachungs-Rechner (mit synthetischer Stimme) abspielen möchten, probieren Sie Festival. Wenn Sie den Überwachungs-Rechner lieber in Ruhe lassen und Audio-Hinweise auf einem anderen Rechner abspielen möchten, dann sehen Sie sich die Projekte Network Audio System (NAS) und rplay an.

Informationen über die CGIs

Hoch zu: Inhalt

Siehe auch: Optionen der CGI-Konfigurationsdatei, Authentifizierung und Autorisierung in den CGIs, CGI-Fuß- und Kopfzeilen

Einführung

Die verschiedenen mit Nagios gelieferten CGIs werden hier beschrieben, zusammen mit den Autorisierungsanforderungen für den Zugriff und den Gebrauch jedes CGIs. Im Grundzustand erwarten die CGIs, dass Sie sich dem Web-Server gegenüber authentifiziert haben und autorisiert sind, jede Information zu sehen, die Sie anfordern. Mehr Informationen über die Konfiguration der Autorisierung finden Sie hier.

Index

Status CGI



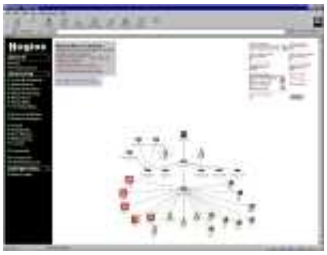

Dateiname: **status.cgi**

Beschreibung:
Dies ist das wichtigste mit Nagios gelieferte CGI. Es erlaubt Ihnen den aktuellen Status aller überwachten Hosts und Services zu sehen. Das Status-CGI kann zwei Arten von Ausgaben liefern - einen Status-Überblick aller Hostgruppen (oder einer bestimmten Hostgruppe) und eine detaillierte Anzeige aller Services (oder diese bezogen auf einen bestimmten Host).

Autorisierungsanforderungen:

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie alle Hosts **und** alle Services ansehen.
- Wenn Sie *für alle Services autorisiert* sind, können Sie alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie alle Hosts und Services ansehen, deren Kontakt Sie sind.

Status Map CGI



Dateiname: **statusmap.cgi**

Beschreibung:
Dieses CGI erstellt eine Karte aller Hosts, die Sie in Ihrem Netzwerk definiert haben. Das CGI nutzt Thomas Boutells gd-Library (Version 1.6.3 oder höher), um ein PNG-Bild Ihrer Netzwerk-Struktur zu erstellen. Die verwendeten Koordinaten (zusammen mit den optionalen Icons) werden aus den Host-Definitionen genommen. Wenn Sie es vorziehen, dass das CGI automatisch für Sie Koordinaten generiert, nutzen Sie die default_statusmap_layout-Direktive, um einen Layout-Algorithmus zu definieren.

Autorisierungsanforderungen:

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie alle Hosts **und** alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie alle Services ansehen, deren Kontakt Sie sind.

Anmerkung: Benutzer, die nicht autorisiert sind, bestimmte Hosts zu sehen, werden *unbekannte* Knoten an diesen Stellen sehen. Mir ist klar, dass sie eigentlich *überhaupt nichts* dort sehen sollten, aber es ist nicht sinnvoll, eine Karte zu generieren, wenn man nicht die ganzen Host-Abhängigkeiten sehen kann.

WAP Interface CGI

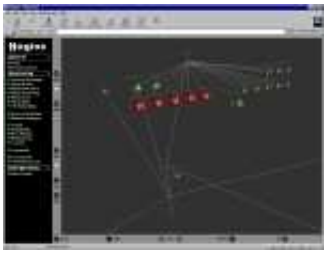
Dateiname: **statuswml.cgi**

Beschreibung:
Dieses CGI dient als WAP-Interface für Netzwerk-Status-Informationen. Wenn Sie ein Gerät mit WAP-Unterstützung haben (z.B. ein Internet-fähiges Handy), können Sie Statusinformationen ansehen, während Sie unterwegs sind. Verschiedene Status-Anzeigen enthalten Hostgruppen-Zusammenfassung, Hostgruppen-Übersicht, Host-Details, Service-Details, alle Probleme und alle unbehandelten Probleme. Zusätzlich zur Ansicht von Statusinformationen können Sie mit Ihrem Handy auch Benachrichtigungen und Prüfungen deaktivieren und Probleme bestätigen. Ziemlich cool, oder?

Autorisierungsanforderungen:

- Wenn Sie *für Systeminformationen autorisiert* sind, können Sie Nagios-Prozess-Informationen ansehen.
- Wenn Sie *für alle Hosts autorisiert* sind, können Sie Zustandsdaten für alle Hosts **und** alle Services ansehen.
- Wenn Sie *für alle Services autorisiert* sind, können Sie Zustandsdaten für alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie Zustandsdaten für alle Hosts und Services ansehen, deren Kontakt Sie sind.

Status World CGI (VRML)



Dateiname: **statuswrl.cgi**

Beschreibung:
Dieses CGI erstellt ein 3-D VRML-Modell aller Hosts, die in Ihrem Netzwerk definiert sind. Die verwendeten Koordination (sowie schöne Textur-Karten) werden aus den Host-Definitionen genommen. Wenn Sie es vorziehen, dass das CGI automatisch für Sie Koordinaten generiert, nutzen Sie die default_statuswrl_layout-Direktive, um einen Layout-Algorithmus zu definieren. Sie benötigen einen VRML-Browser (wie Cortona, Cosmo Player oder WorldView) auf Ihrem System, bevor Sie das generierte Modell betrachten können.

Autorisierungsanforderungen:

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie alle Hosts ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie alle Hosts ansehen, deren Kontakt Sie sind.

Anmerkung: Benutzer, die nicht autorisiert sind, bestimmte Hosts zu sehen, werden *unbekannte* Knoten an diesen Stellen sehen. Mir ist klar, dass sie eigentlich *überhaupt nichts* dort sehen sollten, aber es ist nicht sinnvoll, eine Karte zu generieren, wenn man nicht die ganzen Host-Abhängigkeiten sehen kann.

Tactical Overview CGI

Dateiname: **tac.cgi**

Beschreibung:
Dieses CGI dient als Sicht aus der "Vogelperspektive" auf alle Netzwerk-Überwachungs-Aktivitäten. Es erlaubt Ihnen schnell Netzwerkausfälle sowie Host- und Service-Zustände zu erkennen. Es unterscheidet zwischen Problemen, die auf irgendeine Weise "behandelt" wurden (z.B. bestätigt oder Benachrichtigungen deaktiviert) und solchen, die nicht behandelt wurden und die deshalb Beachtung erfordern. Das ist sehr hilfreich, wenn Sie viele zu überwachende Hosts und Services haben und einen einzelnen Bildschirm zur Alarmierung über Probleme einsetzen möchten.

Autorisierungsanforderungen:

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie alle Hosts **und** alle Services ansehen.
- Wenn Sie *für alle Services autorisiert* sind, können Sie alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie alle Hosts und Services ansehen, deren Kontakt Sie sind.

Network Outages CGI

Dateiname: **outages.cgi**

Beschreibung:
Dieses CGI zeigt eine Liste von "Problem"-Hosts, die Netzwerkausfälle hervorrufen. Dies kann besonders dann hilfreich sein, wenn Sie ein großes Netzwerk haben und schnell die Quelle des Problems identifizieren möchten. Hosts werden sortiert nach der Schwere des Ausfalls, den sie bewirken.

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie alle Hosts ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie alle Hosts ansehen, deren Kontakt Sie sind.

Configuration CGI

Dateiname: **config.cgi**

Beschreibung:
Dieses CGI erlaubt es Ihnen, Objekte (z.B. Hosts, Hostgruppen, Kontakte, Kontaktgruppen, Zeitfenster, Services, etc.) anzusehen, die Sie in Ihrer/Ihren Objekt-Konfigurationsdatei(en) definiert haben.

Autorisierungsanforderungen:

- Sie müssen *für Konfigurationsinformationen autorisiert* sein, um jegliche Konfigurationsinformationen ansehen zu können.

Command CGI

Dateiname: **cmd.cgi**

Beschreibung:
Dieses CGI erlaubt es Ihnen, Befehle an den Nagios-Prozess zu senden. Obwohl dieses CGI mehrere Argumente hat, sollten Sie besser darauf verzichten. Die meisten wechseln zwischen verschiedenen Revisionen von Nagios. Nutzen Sie das extended information CGI als Startpunkt, um Befehle zu erteilen.

Autorisierungsanforderungen:

- Sie müssen *für System-Befehle autorisiert* sein, um Befehle zu erteilen, die den Nagios-Prozess beeinflussen (Start, Stop, Modus-Wechsel, etc.).
- Wenn Sie *für alle Hosts-Befehle autorisiert* sind, können Sie Befehle für alle Hosts **und** alle Services erteilen.
- Wenn Sie *für alle Service-Befehle autorisiert* sind, können Sie Befehle für alle Services erteilen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie Befehle für alle Hosts und Services erteilen, deren Kontakt Sie sind.

Notes:

- Wenn Sie sich entschieden haben, die Option use_authentication für die CGIs nicht zu nutzen, wird dieses CGI *jedem* die Möglichkeit verweigern, Befehle zu erteilen. Dies geschieht zu Ihrem eigenen Schutz. Ich würde empfehlen, dieses CGI komplett zu entfernen, wenn Sie die Authentifizierung für CGI nicht nutzen.

Extended Information CGI




Dateiname: **extinfo.cgi**

Beschreibung:
Dieses CGI erlaubt es Ihnen, Nagios-Prozess-Informationen, Host- und Service-Zustandsstatistiken, Host- und Service-Kommentare und mehr anzusehen. Es dient auch als Startpunkt, um über das command CGI Befehle an Nagios zu erteilen. Obwohl dieses CGI mehrere Argumente hat, sollten Sie besser darauf verzichten. Die meisten wechseln zwischen verschiedenen Revisionen von Nagios. Sie können dieses CGI erreichen, indem Sie auf 'Network Health' bzw. 'Process Information' in der setilichen Navigationsleiste klicken oder auf einen Host- oder Service-Link in der Ausgabe des status CGI.

Autorisierungsanforderungen:

- Sie müssen *für Systeminformationen autorisiert* sein, um Nagios-Prozess-Informationen ansehen zu können.
- Wenn Sie *für lle Hosts autorisiert* sind, können Sie erweiterte Informationen für alle Hosts **und** alle Services ansehen.
- Wenn Sie *für alle Services autorisiert* sind, können Sie erweiterte Informationen für alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie erweiterte Informationen für alle Hosts und Services ansehen, deren Kontakt Sie sind.

Event Log CGI

Dateiname: **showlog.cgi**

Beschreibung:
Dieses CGI zeigt das log file. Wenn Sie die log rotation aktiviert haben, können Sie in archivierten Log-Dateien blättern, indem Sie die Navigations-Links oben auf der Seite benutzen.

Autorisierungsanforderungen:

- Sie müssen *für Systeminformationen autorisiert* sein, um das Logfile ansehen zu können.

Alert History CGI

Dateiname: **history.cgi**

Beschreibung:
Dieses CGI wird benutzt, um die Problem-Historie für einen oder alle Hosts anzuzeigen. Die Ausgabe ist grundsätzlich ein Auszug der Informationen, die über das log file CGI angezeigt werden. Sie haben die Möglichkeit, die Ausgabe zu filtern, um nur die Problemtypen anzuzeigen, die Sie sehen wollen (z.B. Hard- und/oder Soft-Alarme, verschiedene Typen von Service- und Host-Alarmen, alle Arten von Alarmen, usw.). Wenn Sie die log rotation aktiviert haben, können Sie in archivierten Log-Dateien blättern, indem Sie die Navigations-Links oben auf der Seite benutzen.

Autorisierungsanforderungen:

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie historische Informationen für alle Hosts **und** alle Services ansehen.
- Wenn Sie *für alle Services autorisiert* sind, können Sie historische Informationen für alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie historische Informationen für alle Hosts und Services ansehen, deren Kontakt Sie sind.

Notifications CGI

Dateiname: **notifications.cgi**

Beschreibung:
Dieses CGI wird genutzt, um Host- und Service-Benachrichtigungen anzuzeigen, die an verschiedene Kontakte versandt wurden. Die Ausgabe ist grundsätzlich ein Auszug der Informationen, die über das log file CGI angezeigt werden. Sie haben die Möglichkeit, die Ausgabe zu filtern, um nur die Benachrichtigungen anzuzeigen, die Sie sehen wollen (z.B. Service-Benachrichtigungen, Host-Benachrichtigungen, Benachrichtigungen, die an bestimmte Kontakte versandt wurden, usw.). Wenn Sie die log rotation aktiviert haben, können Sie in archivierten Log-Dateien blättern, indem Sie die Navigations-Links oben auf der Seite benutzen.

Autorisierungsanforderungen:

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie Benachrichtigungen für alle Hosts **und** alle Services ansehen.
- Wenn Sie *für alle Services autorisiert* sind, können Sie Benachrichtigungen für alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie Benachrichtigungen für alle Hosts und Services ansehen, deren Kontakt Sie sind.

Trends CGI

Dateiname: **trends.cgi**

Beschreibung:
Dieses CGI wird genutzt, um einen Graphen über Host- oder Service-Zustände für einen beliebigen Zeitraum zu erstellen. Damit dieses CGI von Wert ist, sollten Sie log rotation aktivieren und archivierte Logs in dem Verzeichnis lagern, das durch die log_archive_path-Direktive angegeben wird. Das CGI nutzt Thomas Boutells gd-Library (Version 1.6.3 oder höher), um die Trend-Grafiken zu erstellen.

Autorisierungsanforderungen:

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie Trends für alle Hosts **und** alle Services ansehen.
- Wenn Sie *für alle Services autorisiert* sind, können Sie Trends für alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie Trends für alle Hosts und Services ansehen, deren Kontakt Sie sind.

Availability Reporting CGI

Dateiname: **avail.cgi**

Beschreibung:
Dieses CGI wird genutzt, um einen Bericht über die Verfügbarkeit von Hosts oder Service für einen benutzerdefinierten Zeitraum zu erstellen. Damit dieses CGI von Wert ist, sollten Sie log rotation aktivieren und archivierte Logs in dem Verzeichnis lagern, das durch die log_archive_path-Direktive angegeben wird.

Autorisierungsanforderungen:

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie Verfügbarkeitsdaten für alle Hosts **und** alle Services ansehen.
- Wenn Sie *für alle Services autorisiert* sind, können Sie Verfügbarkeitsdaten für alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie Verfügbarkeitsdaten für alle Hosts und Services ansehen, deren Kontakt Sie sind.

Alert Histogram CGI

Dateiname: **histogram.cgi**

Beschreibung:
Dieses CGI wird genutzt, um einen Bericht über die Verfügbarkeit von Hosts oder Service für einen benutzerdefinierten Zeitraum zu erstellen. Damit dieses CGI von Wert ist, sollten Sie log rotation aktivieren und archivierte Logs in dem Verzeichnis lagern, das durch die log_archive_path-Direktive angegeben wird. Das CGI nutzt Thomas Boutells gd-Library (Version 1.6.3 oder höher), um die Histogramm-Grafiken zu erstellen.

Autorisierungsanforderungen:

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie Histogramme für alle Hosts **und** alle Services ansehen.
- Wenn Sie *für alle Services autorisiert* sind, können Sie Histogramme für alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie Histogramme für alle Hosts und Services ansehen, deren Kontakt Sie sind.

Alert Summary CGI

Dateiname: **summary.cgi**

Beschreibung:
Dieses CGI stellt einige generische Berichte über Host- und Service-Alarmdaten zur Verfügung, darunter Gesamtzahl Alarme, Alarm-Spitzenreiter, etc.

Autorisierungsanforderungen:

- Wenn Sie *für alle Hosts autorisiert* sind, können Sie Summary-Informationen für alle Hosts **und** alle Services ansehen.
- Wenn Sie *für alle Services autorisiert* sind, können Sie Summary-Informationen für alle Services ansehen.
- Wenn Sie ein *authentifizierter Kontakt* sind, können Sie Summary-Informationen für alle Hosts und Services ansehen, deren Kontakt Sie sind.

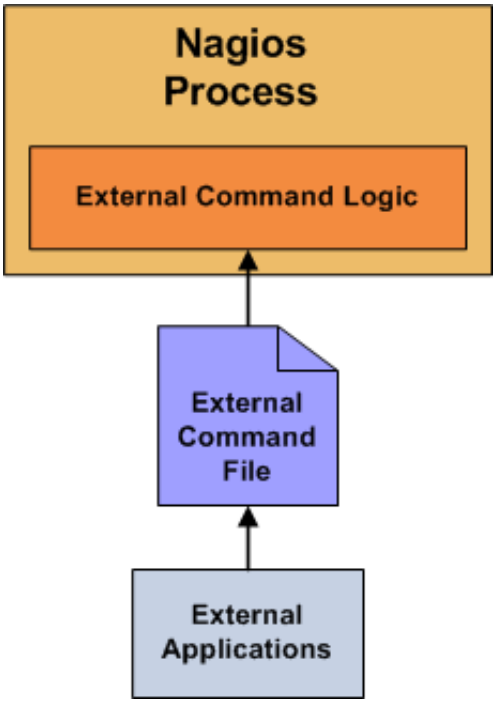
Externe Befehle

Hoch zu: Inhalt

Siehe auch: Passive Prüfungen, Adaptive Überwachung

Einführung

Nagios kann Befehle aus externen Applikationen verarbeiten (einschließlich der CGIs) und verschiedene Aspekte seiner Überwachungsfunktionen aufgrund der Befehle verändern, die es erhält. Externe Applikationen können Befehle "einreichen", indem sie in das command file schreiben, das regelmäßig vom Nagios-Daemon verarbeitet wird.



Externe Befehle aktivieren

Damit Nagios externe Befehle verarbeitet, müssen Sie folgendes tun:

- aktivieren Sie die Prüfung auf externe Befehle mit der check_external_commands-Option.
- setzen Sie die Wiederholrate von Befehlsprüfungen mit der command_check_interval-Option.
- definieren Sie den Ort des Command-File mit der command_file-Option.
- setzen Sie korrekte Berechtigungen für das Verzeichnis, welches das External-Command-File enthält, wie in der Schnellstartanleitung beschrieben.

Wann prüft Nagios auf externe Befehle?

- in regelmäßigen Intervallen, wie sie durch die Option command_check_interval in der Hauptkonfigurationsdatei angegeben sind
- direkt nachdem Eventhandler ausgeführt werden. Das passiert zusätzlich zum regelmäßigen

Zyklus von externen Befehlsprüfungen und wird getan, um unverzügliche Aktivitäten zu ermöglichen, falls ein Eventhandler Befehle an Nagios schickt.

Externe Befehle benutzen

Externe Befehle können benutzt werden, um eine Reihe von Dingen zu erreichen, während Nagios läuft. Beispiele dafür, was getan werden kann, umfassen u.a. vorübergehend Benachrichtigungen für Services und Hosts zu deaktivieren, vorübergehend Service-Prüfungen zu deaktivieren, sofortige Service-Prüfungen zu erzwingen, Kommentare für Hosts und Services hinzuzufügen usw.

Befehlsformat

Externe Befehle, die in das command file geschrieben werden, haben das folgende Format...

```
[Zeit] Befehls-ID;Befehlsargumente
```

...wobei *Zeit* die Zeit (im *time_t*-Format) ist, zu der die externe Applikation den externen Befehl an das Command-File geschickt hat. Die Werte für die *Befehls-ID* und die *Befehlsargumente* hängen davon ab, welcher Befehl an Nagios geschickt wird.

Eine komplette (englischsprachige) Liste der Befehle, die eingesetzt werden können (zusammen mit Beispielen, wie sie benutzt werden), finden Sie online unter der Adresse

http://www.nagios.org/developerinfo/externalcommands/

Eventhandler

Hoch zu: Inhalt

Siehe auch: Statustypen, Host-Prüfungen, Service-Prüfungen

Einführung

Eventhandler sind optionale Systemkommandos (Scripts oder Programme), die gestartet werden, wenn ein Host- oder Service-Zustandswechsel stattfindet.

Ein einleuchtender Einsatz von Eventhandlern ist die Möglichkeit von Nagios, proaktiv Probleme zu beheben, bevor jemand benachrichtigt wird. Einige andere Anwendungsmöglichkeiten für Eventhandler umfassen:

- neustarten eines ausgefallenen Service
- anlegen eines Trouble-Tickets in einem Helpdesk-Systems
- eintragen von Ereignisinformationen in eine Datenbank
- Strom aus- und einschalten bei einem Host*
- etc.

* Strom durch ein automatisiertes Script bei einem Host aus- und einzuschalten, der Probleme hat, sollte wohlüberlegt sein. Betrachten Sie sorgfältig die möglichen Konsequenzen, bevor Sie automatische Reboots implementieren. :-)

Wann werden Eventhandler ausgeführt?

Eventhandler werden ausgeführt, wenn ein Service oder Host

- in einem SOFT-Problemzustand ist
- in einen HARD-Problemzustand wechselt
- aus einem SOFT- oder HARD-Problemzustand zurückkehrt

SOFT- und HARD-Zustände sind ausführlich hier beschrieben.

Eventhandler-Typen

Es gibt unterschiedliche Typen von optionalen Eventhandlern, die Sie definieren können, um Host- und Statuswechsel zu behandeln:

- Globale Host-Eventhandler
- Globale Service-Eventhandler
- Host-spezifische Eventhandler
- Service-spezifische Eventhandler

Globale Host- und Service-Eventhandler werden für *jeden* auftretenden Host- oder Service-Zustandswechsel durchgeführt, direkt vor einem möglichen Host- oder Service-spezifischen Eventhandler. Sie können globale Host- oder Service-spezifische Eventhandler durch die global_host_event_handler und global_service_event_handler-Optionen in der Hauptkonfigurationsdatei angeben.

Einzelne Hosts und Service können ihre eigenen Eventhandler haben, die ausgeführt werden, um Statuswechsel zu behandeln. Sie können einen auszuführenden Eventhandler durch die *event_handler*-Direktive in Ihren Host- oder Service-Definitionen angeben. Diese Host- und Service-spezifischen Eventhandler werden direkt nach dem (optionalen) globalen Host- oder Service-Eventhandler ausgeführt.

Eventhandler aktivieren

Eventhandler können durch die enable_event_handlers-Direktive in Ihrer Hauptkonfigurationsdatei programmweit aktiviert oder deaktiviert werden.

Host- und Service-spezifische Eventhandler werden durch die *event_handler_enabled*-Direktive in Ihrer Host- oder Service-Definition aktiviert oder deaktiviert. Host- und Service-spezifische Eventhandler werden nicht ausgeführt, wenn die globale enable_event_handlers-Option deaktiviert ist.

Eventhandler-Ausführungsreihenfolge

Wie bereits erwähnt werden globale Host- und Service-Eventhandler direkt vor Host- oder Service-spezifischen Eventhandlern ausgeführt.

Eventhandler werden bei HARD-Problemen und Erholungszuständen direkt nach dem Versand von Benachrichtigungen ausgeführt.

Eventhandler-Kommandos schreiben

Eventhandler werden wahrscheinlich Shell- oder Perl-Scripte sein, aber es ist jede Art von ausführbarer Datei denkbar, die von der Kommandozeile aus lauffähig ist. Die Scripte sollten mindestens die folgenden Makros als Argumente nutzen:

Für Services: **$SERVICESTATE$**, **$SERVICESTATETYPE$**, **$SERVICEATTEMPT$**
Für Hosts: **$HOSTSTATE$**, **$HOSTSTATETYPE$**, **$HOSTATTEMPT$**

Die Scripte sollten die Werte der übergebenen Parameter untersuchen und darauf basierend notwendige Aktionen ausführen. Der beste Weg, die Funktionsweise von Eventhandlern zu verstehen, ist der Blick auf ein Beispiel. Glücklicherweise finden Sie eins hier.

 Hinweis: Zusätzliche Eventhandler-Scripte finden Sie im *contrib/eventhandlers/*-Unterverzeichnis der Nagios-Distribution. Einige dieser Beispiel-Scripts demonstrieren die Benutzung von externen Befehlen, um redundante und verteilte Überwachungsumgebungen zu implementieren.

Berechtigungen für Eventhandler-Befehle

Eventhandler werden normalerweise mit den gleichen Berechtigungen ausgeführt wie der Benutzer, der Nagios auf Ihrer Maschine ausführt. Dies kann ein Problem darstellen, wenn Sie einen Eventhandler schreiben möchten, der Systemdienste neu startet, da generell root-Rechte benötigt werden, um diese Aufgaben zu erledigen.

Idealerweise sollten Sie den Typ von Eventhandler einschätzen und dem Nagios-Benutzer gerade genug Berechtigungen gewähren, damit er die notwendigen Systembefehle ausführen kann. Vielleicht möchten Sie sudo ausprobieren, um das zu erreichen.

Service-Eventhandler-Beispiel

Das folgende Beispiel geht davon aus, dass Sie den HTTP-Server auf der lokalen Maschine überwachen und *restart-httpd* als den Eventhandler-Befehl für die HTTP-Service-Definition angegeben haben. Außerdem nehme ich an, dass Sie die Option *max_check_attempts* für den Service auf einen Wert von 4 oder höher gesetzt haben (d.h., der Service wird viermal geprüft, bevor angenommen wird, dass es ein richtiges Problem gibt). Eine gekürzte Service-Definition könnte wie folgt aussehen...

```
define service{
        host_name               somehost
        service_description     HTTP
        max_check_attempts      4
        event_handler           restart-httpd
        ...
        }
```

Sobald der Service mit einem Eventhandler definiert wird, müssen wir diesen Eventhandler als Befehlsfolge definieren. Eine Beispieldefinition für *restart-httpd* sehen Sie nachfolgend. Beachten Sie die Makros in der Kommandozeile, die an das Eventhandler-Script übergeben werden - sie sind wichtig!

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

Lassen Sie uns nun das Eventhandler-Script schreiben (das ist das */usr/local/nagios/libexec/eventhandlers/restart-httpd*-Script).

```
#!/bin/sh
#
# Eventhandler-Script für den Restart des Web-Servers auf der lokalen Maschine
#
# Anmerkung: Dieses Script wird den Web-Server nur dann restarten, wenn der Service
#       dreimal erneut geprüft wurde (sich in einem "soft"-Zustand befindet)
#       oder der Web-Service aus irgendeinem Grund in einen "hard"-Zustand fällt


# In welchem Status befindet sich der Service?
case "$1" in
OK)
        # Der Service hat sich gerade erholt, also tun wir nichts...
        ;;
WARNING)
        # Wir kümmern uns nicht um WARNING-Zustände, denn der Dienst läuft wahrscheinlich noch...
        ;;
UNKNOWN)
        # Wir wissen nicht, was einen UNKNOWN-Fehler auslösen könnte, also tun wir nichts...
        ;;
CRITICAL)
        # Aha!  Der HTTP-Service scheint ein Problem zu haben - vielleicht sollten wir den Server neu starten...

        # Ist dies ein "Soft"- oder ein "Hard"-Zustand?
        case "$2" in

        # Wir sind in einem "Soft"-Zustand, also ist Nagios mitten in erneuten Prüfungen, bevor es in einen
        # "Hard"-Zustand wechselt und Kontakte informiert werden...
        SOFT)
```

```
                # Bei welchem Versuch sind wir? Wir wollen den Web-Server nicht gleich beim ersten Mal restarten,
                # denn es könnte ein Ausrutscher sein!
                case "$3" in

                # Warte, bis die Prüfung dreimal wiederholt wurde, bevor der Web-Server restartet wird.
                # Falls der Check ein viertes Mal fehlschlägt (nachdem wir den Web-Server restartet haben),
                # wird der Zustandstyp auf "Hard" wechseln und Kontakte werden über das Problem informiert.
                # Hoffentlich wird der Web-Server erfolgreich restartet, so dass der vierte Check zu einer
                # "Soft"-Erholung führt. Wenn das passiert, wird niemand informiert, weil wir das Problem gelöst haben.
                3)
                        echo -n "Restart des HTTP-Service (dritter kritischer "Soft"-Zustand)..."
                        # Aufrufen des Init-Scripts, um den HTTPD-Server zu restarten
                        /etc/rc.d/init.d/httpd restart
                        ;;
                        esac
                ;;

        # Der HTTP-Service hat es irgendwie geschafft, in einen "Hard"-Zustand zu wechseln, ohne dass das Problem
        # behoben wurde. Er hätte durch den Code restartet werden sollen, aber aus irgendeinem Grund hat es nicht
        # funktioniert. Wir probieren es ein letztes Mal, okay?
        # Anmerkung: Kontakte wurden bereits darüber informiert, dass es ein Problem mit dem Service gibt (solange
        # Sie nicht Benachrichtungen für diesen Service deaktiviert haben.
        HARD)
                echo -n "Restart des HTTP-Service..."
                # Aufrufen des Init-Scripts, um den HTTPD-Server zu restarten
                /etc/rc.d/init.d/httpd restart
                ;;
        esac
        ;;
esac
exit 0
```

Das mitgelieferte Beispiel-Script wird versuchen, den Web-Server auf der lokalen Maschine in zwei Fällen zu restarten:

- nachdem der Service das dritte Mal erneut geprüft wurde und sich in einem kritischen "Soft"-Zustand befindet
- nachdem der Service das erste Mal in einen kritischen "Hard"-Zustand wechselt

Das Script sollte theoretisch den Web-Server restarten und das Problem beheben, bevor der Service in einen "Hard"-Problemzustand wechselt, aber wir stellen eine Absicherung bereit, falls es nicht das erste Mal funktioniert. Es ist anzumerken, dass der Eventhandler nur einmal ausgeführt wird, wenn der Service in einen HARD-Zustand wechselt. Das hält Nagios davon ab, das Script zum Restart des Web-Servers wiederholt auszuführen, wenn der Service in einem HARD-Problemzustand bleibt. Das wollen Sie nicht. :-)

Das ist alles! Eventhandler sind ziemlich einfach zu schreiben und zu implementieren, also versuchen Sie es und sehen, was Sie tun können.

Nagios®

sprunghafte Services

Hoch zu: Inhalt

Siehe auch: Statusverfolgung

Einführung

Nagios hat die Möglichkeit, zwischen "normalen" und "flüchtigen" Services zu unterscheiden. Die *is_volatile*-Option in jeder Service-Definition erlaubt Ihnen festzulegen, ob ein bestimmter Service flüchtig ist oder nicht. Für die meisten Leute wird die Mehrzahl der überwachten Services nicht-flüchtig (d.h. "normal") sein. Trotzdem können flüchtige Services sehr nützlich sein, wenn sie richtig eingesetzt werden...

Wofür sind sie nützlich?

Flüchtige Services sind nützlich zur Überwachung von...

- Dingen, die sich jedes Mal automatisch in einen "OK"-Zustand zurücksetzen, wenn sie geprüft werden
- Ereignisse wie Sicherheits-Alarme, die jedes Mal Beachtung erfordern, wenn ein Problem vorliegt (und nicht nur beim ersten Mal)

Was ist so besonders an flüchtigen Services?

Flüchtige Services unterscheiden sich von "normalen" Services in drei wichtigen Punkten. *Jedes Mal* wenn sie in einem harten nicht-OK-Zustand sind und die Prüfung einen nicht-OK-Zustand ergibt (also keine Statusänderung eintritt)...

- wird der nicht-OK-Zustand des Service protokolliert
- werden Kontakte über das Problem informiert (falls es das ist, was zu tun ist). Anmerkung: Benachrichtigungsintervalle werden bei flüchtigen Services ignoriert.
- Der Eventhandler für den Service wird ausgeführt (falls einer definiert ist)

Diese Ereignisse finden normalerweise nur für Services statt, wenn sie in einem nicht-OK-Zustand sind und gerade ein Hard-Zustandswechsel erfolgte. In anderen Worten, sie passieren nur das erste Mal, wenn ein Service in einen nicht-OK-Zustand geht. Wenn weitere Prüfungen des Service den gleichen nicht-OK-Zustand ergeben, erfolgt kein harter Zustandswechsel und keines der genannten Ereignisse wird stattfinden.

Hinweis: Wenn Sie nur an der Protokollierung interessiert sind, dann sehen Sie sich die Stalking-Option an.

Die Macht der Zwei

Wenn Sie die Möglichkeiten von flüchtigen Services und passiven Service-Prüfungen kombinieren, können Sie einige sehr nützliche Dinge tun. Beispiele hierfür umfassen u.a. die Behandlung von SNMP-Traps, Sicherheits-Alarme, usw.

Wie wäre es mit einem Beispiel... Nehmen wir an, Sie nutzen PortSentry, um Portscans auf Ihrer Maschine zu erkennen und automatisch potenzielle Eindringlinge auszusperren. Wenn Sie wollen, dass Nagios über Portscans erfährt, können Sie das Folgende tun...

Nagios Konfiguration:

- Legen Sie eine Service-Definition namens *Port Scans* an und verbinden Sie diese mit dem Host, auf dem PortSentry läuft.
- Setzen Sie die *max_check_attempts*-Direktive in der Service-Definition auf 1. Dies teilt Nagios mit, sofort einen Hard-Zustand für den Service zu erzwingen, wenn ein nicht-OK-Zustand ermittelt wird.
- Setzen Sie die *active_checks_enabled*-Direktive in der Service-Definition auf 0. Dies hält Nagios davon ab, den Service aktiv zu prüfen.
- Setzen Sie die *passive_checks_enabled*-Direktive in der Service-Definition auf 1. Das erlaubt passive Prüfungen für den Service.
- Setzen Sie die *is_volatile*-Direktive in der Service-Definition auf 1.

PortSentry Konfiguration:

Editieren Sie die PortSentry-Konfigurationsdatei (portsentry.conf) und definieren Sie einen Befehl für die *KILL_RUN_CMD*-Direktive wie folgt:

```
KILL_RUN_CMD="/usr/local/Nagios/libexec/eventhandlers/submit_check_result host_name 'Port Scans' 2 'Port scan from host $TARGET$ on port $PORT$.  Host has been firewalled.'"
```

Stellen Sie sicher, *host_name* durch den Kurznamen des Hosts zu ersetzen, mit dem der Service verbunden ist.

Portscan-Script:

Erstellen Sie ein Shell-Script im */usr/local/nagios/libexec/eventhandlers*-Verzeichnis namens *submit_check_result*. Der Inhalt des Shell-Scripts sollte ähnlich dem Folgenden sein...

```
#!/bin/sh

# Write a command to the Nagios command file to cause
# it to process a service check result

echocmd="/bin/echo"

CommandFile="/usr/local/nagios/var/rw/nagios.cmd"

# get the current date/time in seconds since UNIX epoch
datetime=`date +%s`

# create the command line to add to the command file
cmdline="[$datetime] PROCESS_SERVICE_CHECK_RESULT;$1;$2;$3;$4"

# append the command to the end of the command file
`$echocmd $cmdline >> $CommandFile`
```

Was passiert, wenn PortSentry in der Zukunft einen Portscan auf der Maschine entdeckt?

- PortSentry wird den Host ausschließen ("firewall", das ist eine Funktion der PortSentry-Software)
- PortSentry wird das *submit_check_result*-Shell-Script ausführen und ein passives Prüfergebnis an Nagios senden
- Nagios wird das external command file lesen und das passive Service-Prüfergebnis von PortSentry verarbeiten
- Nagios wird den *Port Scans*-Service in einen harten CRITICAL-Zustand versetzen und Benachrichtigungen an die Kontakte senden

Ziemlich hübsch, oder?

Service- und Host-Frische-Prüfungen

Hoch zu: Inhalt

Siehe auch: Passive Prüfungen, Verteilte Überwachung, Redundante und Failover-Überwachung

Einführung

Nagios unterstützt ein Feature, das die "Frische" (Freshness) der Host- und Service-Prüfungen überprüft. Der Zweck der Frische-Prüfung ist es, bei passiven Host- und Service-Prüfungen sicherzustellen, dass diese regelmäßig von externen Applikationen zur Verfügung gestellt werden.

Frische-Prüfungen sind sinnvoll, wenn Sie sicherstellen wollen, dass passive Prüfungen so regelmäßig empfangen werden wie Sie das erwarten. Das kann in verteilten und Failover Überwachungsumgebungen sehr sinnvoll sein.

Wie funktioniert die Frische-Prüfung?

Nagios prüft periodisch die Frische der Ergebnisse für alle Hosts und Services, bei denen Frische-Prüfungen aktiviert sind.

- ein Frische-Schwellwert wird für jeden Host oder Service berechnet.
- für jeden Host/Service wird das Alter des letzten Prüfungsergebnisses mit dem Frische-Schwellwert verglichen.
- wenn das Alter des letzten Prüfungsergebnisses größer als der Frisch-Schwellwert ist, wird das Prüfergebnis als "abgestanden" (stale) betrachtet.
- wenn das Prüfergebnis als abgestanden angesehen wird, wird Nagios eine aktive Prüfung für den Host oder Service mit dem Kommando ausführen, das in der Host- oder Service-Definition angegeben ist.

Hinweis: Eine aktive Prüfung wird ausgeführt, selbst wenn aktive Prüfungen programmweit oder auf Host- bzw. Service-spezifischer Basis deaktiviert sind.

Wenn Sie beispielsweise einen Frische-Schwellwert von 60 für einen Ihrer Services haben, wird Nagios diesen Service als abgestanden ansehen, wenn das letzte Prüfergebnis älter als 60 Sekunden ist.

Frische-Prüfungen aktivieren

Was Sie tun müssen, um Frische-Prüfungen zu aktivieren...

- aktivieren Sie Frische-Prüfungen auf programmweiter Basis mit den check_service_freshness und check_host_freshness-Direktiven.
- benutzen Sie die service_freshness_check_interval- und host_freshness_check_interval-Optionen, um Nagios mitzuteilen, wie oft es die Frische von Host- und Service-Ergebnissen prüfen soll.
- aktivieren Sie Frische-Prüfungen auf Host- und Service-spezifischer Basis, indem Sie die *check_freshness*-Option in Ihrer Host- und Service-Definitionen auf 1 setzen.
- konfigurieren Sie Frische-Schwellwerte, indem Sie die *freshness_threshold*-Option in Ihren Host- und Service-Definitionen setzen.
- konfigurieren Sie die *check_command*-Option in Ihren Host- oder Service-Definitionen, so dass sie ein gültiges Script enthalten, das benutzt werden kann, um den Host oder Service aktiv zu prüfen, wenn er als abgestanden angesehen wird.
- Die *check_period*-Option in Ihren Host- und Service-Definitionen wird benutzt, wenn Nagios festlegt, wann ein Host oder Service auf Frische geprüft werden soll, um sicherzustellen, dass es sich um ein gültiges Zeitfenster handelt.

 Hinweis: Wenn Sie keinen Host- oder Service-spezifischen *freshness_threshold*-Wert angeben (oder ihn auf Null setzen), wird Nagios automatisch einen Schwellwert berechnen, der darauf basiert, wie oft Sie den jeweiligen Host- oder Service überwachen. Ich würde empfehlen, dass Sie explizit einen Frische-Schwellwert angeben, statt dass Nagios einen für Sie auswählt.

Beispiel

Ein Beispiel für einen Service, der eine Frische-Prüfung benötigen könnte, wäre einer, der den Status Ihrer nächtlichen Backups meldet. Vielleicht haben Sie ein externes Script, welches das Ergebnis des Backup-Jobs an Nagios meldet, sobald das Backup beendet ist. In diesem Fall werden alle Prüfungen/Ergebnisse für diesen Service durch eine externe Applikation mit Hilfe von passiven Prüfungen zur Verfügung gestellt. Um sicherzustellen, dass der Status des Backup-Jobs täglich gemeldet wird, können Sie die Frische-Prüfung für diesen Service aktivieren. Falls das externe Script das Ergebnis des Backup-Jobs nicht meldet, kann Nagios ein kritisches Ergebnis imitieren, indem man folgendes tut...

Nachfolgend, wie die Definition für den Service aussehen könnte (einige benötigte Optionen fehlen...)

```
define service{
        host_name               backup-server
        service_description     ArcServe Backup Job
        active_checks_enabled   0               ; aktive Prüfungen sind NICHT aktiviert
        passive_checks_enabled  1               ; passive Prüfungen sind aktiviert (dadurch werden Ergebnisse gemeldet)
        check_freshness         1
        freshness_threshold     93600           ; 26 Stunden Schwellwert, nachdem Backups nicht immer zur gleichen Zeit beendet sind
        check_command           no-backup-report        ; dieses Kommando wird nur ausgeführt, wenn der Service als "abgestanden" angesehen wird
        ...andere Optionen...
        }
```

Beachten Sie, dass aktive Prüfungen für den Service deaktiviert sind. Das ist so, weil die Ergebnisse für den Service nur durch eine externe Applikation geliefert werden. Die Frische-Prüfung ist aktiviert und der Frische-Schwellwert ist auf 26 Stunden gesetzt. Das ist ein bisschen mehr als 24 Stunden, weil Backup-Jobs ab und zu länger dauern (abhängig davon, wie viele Daten zu sichern sind, wie viel Netzwerkverkehr herrscht, usw.). Das *no-backup-report*-Kommando wird nur ausgeführt, wenn die Ergebnisse des Service als abgestanden angesehen werden. Die Definition des *no-backup-report*-Kommandos könnte wie folgt aussehen...

```
define command{
        command_name    no-backup-report
        command_line    /usr/local/nagios/libexec/check_dummy 2 "CRITICAL: Results of backup job were not reported!"
        }
```

Falls Nagios erkennt, dass das Service-Ergebnis abgestanden ist, wird es das *no-backup-report*-Kommando als eine aktive Service-Prüfung ausführen. Das führt dazu, dass das *check_dummy*-Plugin ausgeführt wird, das einen kritischen Status an Nagios meldet. Der Service wird dann in einen kritischen Zustand gehen (falls das nicht bereits der Fall ist) und wahrscheinlich wird jemand über das Problem informiert.

Verteilte Überwachung

Einführung

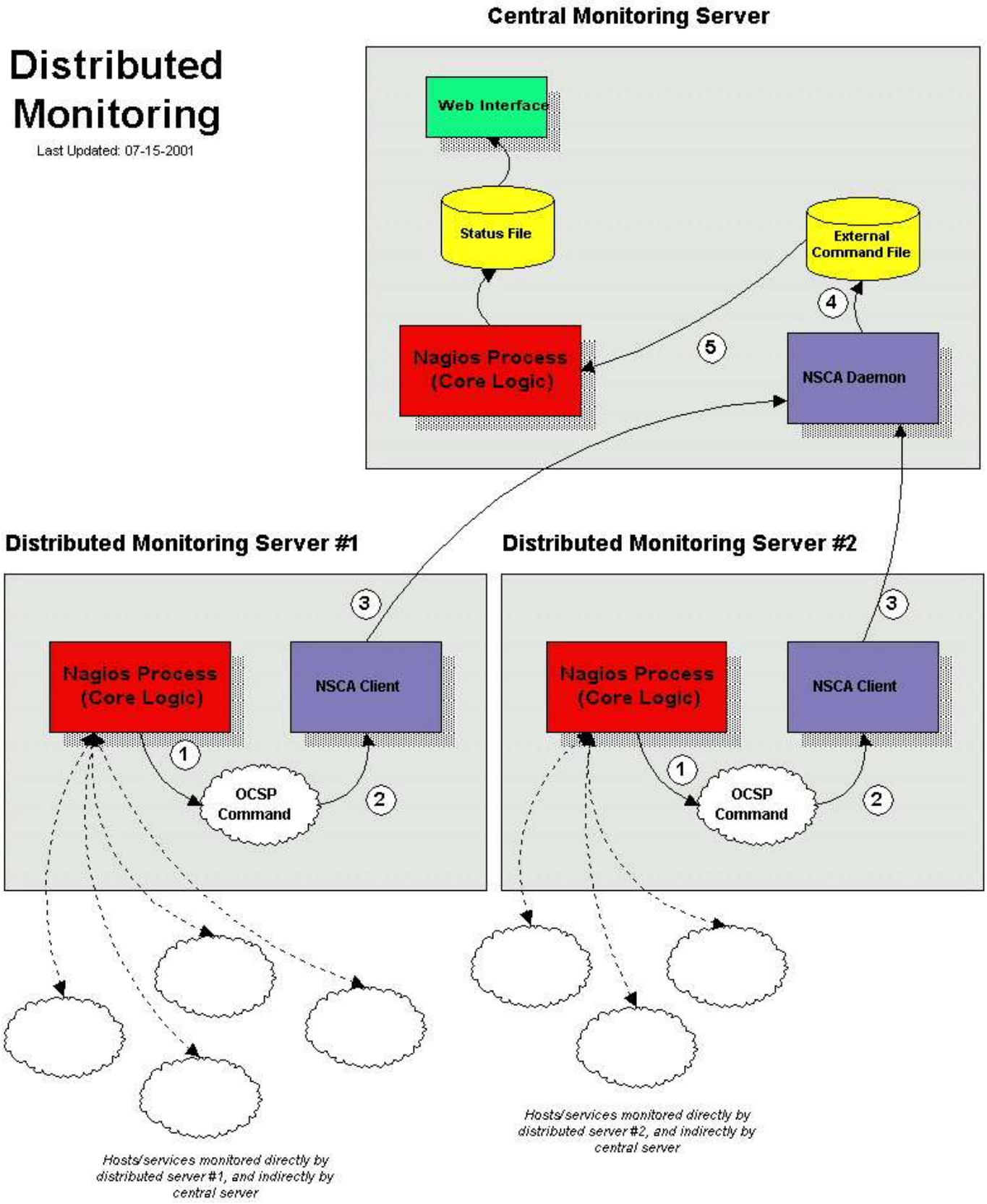
Nagios kann konfiguriert werden, so dass es verteilte Überwachung von Netzwerk-Services und Ressourcen unterstützt. Ich werde versuchen, kurz zu beschreiben, wie das erreicht werden kann...

Ziele

Das Ziel der verteilten Überwachungsumgebung, das ich beschreiben will, ist die Reduzierung des Overheads (CPU-Belastung, etc.) bei der Service-Prüfung von einem "zentralen" Server auf ein oder mehrere "verteilte" Server. Die meisten kleinen bis mittleren Unternehmen werden keinen wirklichen Bedarf für das Aufsetzen solch einer Umgebung haben. Wenn Sie allerdings hunderte oder sogar tausende von *Hosts* (und ein Mehrfaches an Services) mit Nagios überwachen wollen, dann kann das ziemlich wichtig werden.

Referenzdiagramm

Das folgende Diagramm soll Ihnen eine generelle Idee davon geben, wie verteilte Überwachung mit Nagios arbeitet. Ich werde mich auf die Elemente im Diagramm beziehen, während ich die Dinge erkläre...



Zentraler Server vs. Verteilte Server

Beim Einrichten einer verteilten Überwachungsumgebung mit Nagios gibt es Unterschiede in der Art, wie zentrale und verteilte Server konfiguriert sind. Ich werde Ihnen zeigen, wie beide Arten von Servern konfiguriert werden und erklären, welche Auswirkungen die gemachten Änderungen auf die gesamte

Überwachung haben. Für den Anfang beschreibe ich den Zweck der verschiedenen Server-Typen...

Die Funktion eines *verteilten Servers* ist es, aktiv Prüfungen für alle Services durchzuführen, die Sie für eine "Gruppe" (Cluster) von Hosts definieren. Ich benutze den Begriff "Gruppe" locker - er meint lediglich eine willkürliche Gruppe von Hosts in Ihrem Netzwerk. Abhängig von Ihrem Netzwerk-Layout können Sie mehrere Gruppen in einem physischen Standort haben oder jede Gruppe kann durch ein WAN voneinander getrennt sein, mit einer eigenen Firewall, usw. Wichtig anzumerken ist, dass es für jede Gruppe von Hosts (wie immer Sie diese definieren mögen) einen verteilten Server gibt, auf dem Nagios läuft, und der die Services der Hosts dieser Gruppe überwacht. Ein verteilter Server enthält meistens eine simple Installation von Nagios. Es muss kein Web-Interface installiert sein, keine Benachrichtigungen versenden, keine Eventhandler-Scripts ausführen, noch etwas anderes tun außer Service-Prüfungen ausführen, wenn Sie das nicht wollen. Detaillierte Informationen zur Konfiguration eines verteilten Services gibt es später...

Der Zweck des *zentralen* Servers ist es lediglich, auf Service-Prüfungsergebnisse von einem oder mehreren verteilten Servern zu horchen. Obwohl Services ab und zu aktiv durch den zentralen Server geprüft werden, werden diese aktiven Prüfungen nur unter schlimmen Umständen ausgeführt, also lassen Sie uns im Moment sagen, dass der zentrale Server lediglich passive Prüfungen annimmt. Da der zentrale Server Ergebnisse von passiven Service-Prüfungen von einem oder mehreren verteilten Servern erhält, dient er als Mittelpunkt der gesamten Überwachungslogik (d.h., er versendet Benachrichtigungen, startet Eventhandler-Scripts, legt den Zustand von Hosts fest, enthält das Web-Interface, usw.).

Service-Prüfungsinformationen von verteilten Servern erhalten

Bevor wir näher auf Konfigurationsdetails eingehen, müssen wir wissen, wie die Service-Prüfungsergebnisse von den verteilten Servern zum zentralen Server geschickt werden. Ich habe bereits erwähnt, wie man passive Prüfungsergebnisse an den gleichen Host schickt, auf dem Nagios läuft (wie in der Dokumentation zu passive Prüfungen beschrieben), aber ich habe keinerlei Informationen darüber gegeben, wie man passive Prüfergebnisse von anderen Hosts verschickt.

Um den Versand von passiven Prüfergebnissen an einen anderen Host zu erleichtern, habe ich das nsca-Addon geschrieben. Das Addon besteht aus zwei Teilen. Das erste ist ein Client-Programm (send_nsca), das auf einem entfernten Host läuft und benutzt wird, um die Service-Prüfergebnisse an einen anderen Server zu senden. Das zweite Teil ist der nsca-Daemon (nsca), der entweder als eigenständiger Daemon oder unter inetd läuft und auf Verbindungen von Client-Programmen horcht. Nach dem Empfang von Service-Prüfinformationen von einem Client wird der Daemon die Prüfinformationen an Nagios (auf dem zentralen Server) weiterleiten, indem ein *PROCESS_SVC_CHECK_RESULT* zusammen mit den Prüfergebnissen in das external command file eingefügt wird. Das nächste Mal, wenn Nagios auf externe Befehle prüft, wird es die passiven Prüfergebnisse finden, die von den verteilten Servern geschickt wurden und sie verarbeiten. Einfach, oder?

Verteilte Server-Konfiguration

Also wie genau wird Nagios auf einem verteilten Server konfiguriert? Grundsätzlich ist es eine einfache Installation. Sie müssen weder ein Web-Interface installieren noch Benachrichtigungen versenden, weil dies alles vom zentralen Server aus erledigt wird.

Haupt-Konfigurationsanpassungen:

- Nur die direkt durch den verteilten Server zu überwachenden Services werden in der Objekt-Konfigurationsdatei definiert.
- Die enable_notifications-Direktive auf dem verteilten Server wird auf 0 gesetzt. Das verhindert das Versenden von Benachrichtigungen.

- Die obsess over services-Direktive auf dem verteilten Server wird aktiviert.
- Auf dem verteilten Server ist ein ocsp command definiert (wie unten beschrieben).

Damit alles zusammenkommt und ordentlich arbeitet, wollen wir, dass der verteilte Server die Ergebnisse *aller* Service-Prüfungen an Nagios meldet. Wir können Eventhandler benutzen, um *Änderungen* am Zustand eines Service mitzuteilen, aber das bringt's nicht. Um den verteilten Server zu zwingen, alle Prüfergebnisse zu melden, müssen Sie die obsess_over_services-Option in der Hauptkonfigurationsdatei aktivieren und ein ocsp_command bereitstellen, was nach jeder Service-Prüfung ausgeführt wird. Wir werden das ocsp-Kommando benutzen, um die Ergebnisse aller Service-Prüfungen an den zentralen Server zu senden und den send_nsca-Client sowie den nsca-Daemon benutzen (wie oben beschrieben), um die Übertragung zu erledigen.

Um dies zu erreichen, müssen Sie ein ocsp-Kommando wie folgt definieren:

```
ocsp_command=submit_check_result
```

Die Definition für den *submit_check_result*-Befehl sieht ungefähr so aus:

```
define command{
        command_name    submit_check_result
        command_line    /usr/local/nagios/libexec/eventhandlers/submit_check_result $HOSTNAME$ '$SERVI
        }
```

Die *submit_check_result* Shell-Scripte sehen ungefähr so aus (ersetzen Sie *central_server* durch die IP-Adresse des zentralen Servers):

```
#!/bin/sh

# Arguments:
#  $1 = host_name (Short name of host that the service is
#       associated with)
#  $2 = svc_description (Description of the service)
#  $3 = state_string (A string representing the status of
#       the given service - "OK", "WARNING", "CRITICAL"
#       or "UNKNOWN")
#  $4 = plugin_output (A text string that should be used
#       as the plugin output for the service checks)
#

# Convert the state string to the corresponding return code
return_code=-1

case "$3" in
    OK)
                return_code=0
            ;;
        WARNING)
            return_code=1
                ;;
        CRITICAL)
            return_code=2
                ;;
        UNKNOWN)
            return_code=-1
                ;;
esac

# pipe the service check info into the send_nsca program, which
# in turn transmits the data to the nsca daemon on the central
# monitoring server

/bin/printf "%s\t%s\t%s\t%s\n" "$1" "$2" "$return_code" "$4" | /usr/local/nagios/bin/send_nsca -H central_server -c /usr/local/nagios/etc/send_nsca.cfg
```

Das Script oben geht davon aus, dass das send_nsca-Programm und die Konfigurationsdatei (send_nsca.cfg) in den Verzeichnissen */usr/local/nagios/bin/* und */usr/local/nagios/etc/* zu finden sind.

Das ist alles! Wir haben erfolgreich einen entfernten Host konfiguriert, auf dem Nagios als ein verteilter Überwachungs-Server läuft. Lassen Sie uns genau betrachten, was mit dem verteilten Server passiert und wie er Service-Prüfungsergebnisse an Nagios schickt (die unten skizzierten Schritte entsprechen den Zahlen im obigen Referenzdiagramm):

1. Nachdem der verteilte Server eine Service-Prüfung beendet hat, führt er den Befehl aus, den Sie mit der Variable ocsp_command definiert haben. In unserem Beispiel ist dies das */usr/local/nagios/libexec/eventhandlers/submit_check_result*-Script. Beachten Sie, dass die Definition für

den *submit_check_result*-Befehl vier Parameter für das Script übergibt: den Namen des Hosts, der mit dem Service verbunden ist, die Service-Beschreibung, den Rückgabewert der Service-Prüfung und die Plugin-Ausgabe der Service-Prüfung.
2. das *submit_check_result*-Script übergibt die Informationen der Service-Prüfung (Host-Name, Beschreibung, Rückgabewert und Ausgabe) an das *send_nsca*-Client-Programm.
3. das *send_nsca*-Programm überträgt die Informationen der Service-Prüfung an den *nsca*-Daemon auf dem zentralen Überwachungs-Server.
4. der *nsca*-Daemon auf dem zentralen Server nimmt die Informationen der Service-Prüfung und schreibt sie in das external command file, damit Nagios sie später dort aufsammeln kann.
5. der Nagios-Prozess auf dem zentralen Server liest das external command file und verarbeitet die passiven Service-Prüfungsergebnisse, die vom verteilten Überwachungs-Server stammen.

zentrale Server-Konfiguration

Wir haben betrachtet, wie verteilte Überwachungs-Server konfiguriert werden sollten, daher wenden wir uns nun dem zentralen Server zu. Für alle wichtigen Dinge wird der zentrale so konfiguriert wie ein einzelner Server. Dessen Setup ist wie folgt:

- auf dem zentralen Server ist das Web-Interface installiert (optional, aber empfohlen)
- auf dem zentralen Server ist die enable_notifications-Direktive auf 1 gesetzt. Das aktiviert Benachrichtungen (optional, aber empfohlen)
- auf dem zentralen Server sind aktive Service-Prüfungen deaktiviert (optional, aber empfohlen - beachten Sie die folgenden Anmerkungen)
- auf dem zentralen Server sind external command checks aktiviert (erforderlich)
- auf dem zentralen Server sind passive Service-Prüfungen aktiviert (erforderlich)

Es gibt drei andere sehr wichtige Dinge, die Sie beachten sollten, wenn Sie den zentralen Server konfigurieren:

- Der zentrale Server muss Service-Definitionen für *alle Services* haben, die auf allen verteilten Servern überwacht werden. Nagios wird passive Prüfungsergebnisse ignorieren, wenn sie nicht zu einem Service passen, den Sie definiert haben.
- Wenn Sie den zentralen Server nur benutzen, um Services zu verarbeiten, deren Ergebnisse von verteilten Hosts stammen, können Sie alle aktiven Service-Prüfungen auf programmweiter Basis durch das Setzen der execute_service_checks-Direktive auf 0 deaktivieren. Wenn Sie den zentralen Server nutzen, um selbst einige Services aktiv zu überwachen (ohne die Hilfe von verteilten Servern), dann sollten Sie die *enable_active_checks*-Option der Service-Definitionen auf 0 setzen, die von den verteilten Servern überwacht werden. Das hindert Nagios daran, diese Services aktiv zu prüfen.

Es ist wichtig, dass Sie entweder alle Service-Prüfungen auf einer programmweiten Basis deaktivieren oder die *enable_active_checks*-Option in jeder Service-Definition deaktivieren, die von einem verteilten Server überwacht werden. Das stellt sicher, dass aktive Service-Prüfungen unter normalen Umständen niemals ausgeführt werden. Die Services werden weiterhin im normalen Prüfintervall geplant (3 Min., 5 Min., usw.), aber nicht ausgeführt. Ich werde bald erklären, warum das so ist...

Das war's! Einfach, oder?

Probleme bei passiven Prüfungen

Für alle wichtigen Dinge können wir sagen, dass sich der zentrale Server bei Überwachungen allein auf passive Prüfungen verlässt. Das Hauptproblem daran, sich komplett auf passive Prüfungen zu verlassen besteht darin, dass Nagios darauf vertrauen muss, dass jemand anders die Daten liefert. Was passiert, wenn der entfernte Host, der passive Prüfergebnisse sendet, herunterfährt oder unerreichbar wird?

Wenn Nagios nicht aktiv die Services auf dem Host prüft, wie soll es wissen, wann es ein Problem gibt?

Glücklicherweise gibt es einen Weg, diese Art von Problemen zu behandeln...

Frische-Prüfung (Freshness Checking)

Nagios unterstützt ein Feature, das eine "Frische"-Prüfung für die Ergebnisse von Service-Prüfungen durchführt. Mehr Informationen über Frische-Prüfung finden Sie hier. Dieses Feature sorgt für etwas Schutz gegen Situationen, in denen entfernte Hosts keine passiven Service-Prüfungen mehr an den zentralen Überwachungs-Server schicken. Der Zweck der "Frische"-Prüfung besteht darin, sicherzustellen, dass Service-Prüfungen entweder regelmäßig passiv durch verteilte Server oder aktiv durch den zentralen Server durchgeführt werden, falls dies notwendig sein sollte. Wenn die Service-Prüfergebnisse von verteilten Servern als "abgestanden" angesehen werden, kann Nagios so konfiguriert werden, um aktive Prüfungen des Service vom zentralen Überwachungs-Server aus zu erzwingen.

Wie machen Sie das? Auf dem zentralen Überwachungs-Server müssen Sie Services konfigurieren, die von verteilten Server wie folgt überwacht werden:

- Die *check_freshness*-Option in der Service-Definition ist auf 1 zu setzen. Das aktiviert "Frische"-Prüfungen für den Service.
- Die *freshness_threshold*-Option in den Service-Definitionen sollte auf einen Wert (in Sekunden) gesetzt werden, der widerspiegelt, wie "frisch" die (von den entfernten Servern gelieferten) Ergebnisse der Service-Prüfungen sein sollten.
- Die *check_command*-Option in den Service-Definitionen sollte gültige Befehle enthalten, die genutzt werden können, um den Service aktiv vom zentralen Server aus zu prüfen.

Nagios prüft periodisch die "Frische" der Ergebnisse aller Services, für die Frische-Prüfungen aktiviert sind. Die *freshness_threshold*-Option in jeder Service-Definition wird benutzt, um festzulegen, wie "frisch" die Ergebnisse für jeden Service sein sollen. Wenn Sie z.B. diesen Wert für einen Ihrer Services auf 300 setzen, wird Nagios das Service-Ergebnis als "abgestanden" betrachten, wenn es älter als 5 Minuten (300 Sekunden) ist. Falls Sie keinen Wert für die *freshness_threshold*-Option angeben, wird Nagios automatisch einen "Frische"-Schwellwert berechnen, indem es die Werte der *normal_check_interval*- oder der *retry_check_interval*-Option betrachtet (abhängig vom Statustyp, in dem sich der Service befindet). Wenn die Service-Ergebnisse als "abgestanden" angesehen werden, wird Nagios den Service-Prüf-Befehl ausführen, der in der *check_command*-Option der Service-Definition angegeben ist, und dadurch den Service aktiv prüfen.

Denken Sie daran, dass Sie eine *check_command*-Option in den Service-Definitionen angeben müssen, die genutzt werden kann, um den Status des Service aktiv vom zentralen Server aus zu prüfen. Unter normalen Umständen wird dieser Prüfbefehl niemals ausgeführt (weil aktive Prüfungen auf programmweiter Ebene bzw. für den einzelnen Service deaktiviert wurden). Wenn Frische-Prüfungen aktiviert sind, wird Nagios diesen Befehl ausführen, um den Zustand des Service aktiv zu prüfen, *auch wenn aktive Prüfungen auf einer programmweiten Ebene oder Service-spezifischen Basis deaktiviert sind*.

Falls Sie es nicht schaffen, Befehle zu definieren, um aktiv einen Service vom zentralen Überwachungs-Host aus zu prüfen (oder wenn es zu einer großen Qual wird), können Sie ganz einfach bei all Ihren Services in der *check_command*-Option ein Dummy-Script angeben, das einen kritischen Status zurückliefert. Hier ein Beispiel... Lassen Sie uns annehmen, Sie definieren einen Befehl namens 'service-is-stale' und benutzen den Befehlsnamen in der *check_command*-Option Ihrer Services. Hier nun, wie die Definition aussehen könnte...

```
define command{
        command_name    service-is-stale
        command_line    /usr/local/nagios/libexec/check_dummy 2 "CRITICAL: Service results are stale"
        }
```

Wenn Nagios feststellt, dass das Service-Ergebnis abgestanden ist und das **service-is-stale**-Kommando aufruft, wird das */usr/local/nagios/libexec/check_dummy*-Plugin ausgeführt und der Service geht in einen kritischen Zustand. Das wird wahrscheinlich dazu führen, dass Benachrichtigungen versandt werden, so dass Sie wissen, dass es ein Problem gibt.

Host-Prüfungen durchführen

An diesem Punkt wissen Sie, wie man Service-Ergebnisse von verteilten Servern auf passive Weise erhält. Das bedeutet, der zentrale Server nicht aktiv Service-Prüfungen ausführt. Aber was ist mit Host-Prüfungen? Sie müssen sie trotzdem erledigen, aber wie?

Nachdem Host-Prüfungen normalerweise einen kleinen Teil der Überwachungsaktivität verbrauchen (sie werden nur ausgeführt, wenn es dringend notwendig ist), rate ich dazu, dass Sie die Host-Prüfungen aktiv vom zentralen Server aus durchführen. Das bedeutet, dass Sie Host-Prüfungen auf dem zentralen Server genau wie auf den verteilten Servern definieren (und auf die gleiche Weise, wie Sie das in einer normalen, nicht-verteilten Umgebung tun würden).

Passive Host-Prüfungen sind verfügbar (lesen Sie hier), so dass Sie diese in Ihrer verteilten Umgebung nutzen können, allerdings gibt es dabei ein paar Probleme. Das größte Problem besteht darin, dass Nagios Ergebnisse von passiven Host-Prüfungen (die Problemzustände DOWN und UNREACHABLE) nicht übersetzt, wenn sie verarbeitet werden. Das bedeutet, falls Ihre Überwachungs-Server eine unterschiedliche Eltern-/Kind-Host-Struktur haben (und das werden sie, wenn Ihre Überwachungs-Server an unterschiedlichen Standorten stehen), wird der zentrale Überwachungs-Server eine ungenaue Sicht Ihrer Host-Zustände haben.

Falls Sie in Ihrer verteilten Überwachungs-Umgebung passive Host-Prüfungen an einen zentralen Server senden möchten, dann stellen Sie sicher:

- dass auf dem zentralen Server passive Host-Prüfungen aktiviert sind (notwendig)
- dass auf dem verteilten Server obsess over hosts aktiviert ist.
- dass auf dem verteilten Server ein ochp command definiert ist.

Der ochp-Befehl, der zur Verarbeitung von Host-Prüfergebnissen genutzt wird, arbeitet ähnlich wie der ocsp-Befehl, der für die Verarbeitung von Service-Prüfergebnissen benutzt wird (siehe oben). Um sicherzustellen, dass passive Host-Prüfergebnisse aktuell sind, sollten Sie Frische-Prüfungen für Hosts aktivieren (ähnlich zu dem, was weiter oben für Services beschrieben wird).

Redundante und Failover-Netzwerk-Überwachung

Einführung

Dieser Abschnitt beschreibt einige Szenarien zum Implementieren von redundanten Überwachungs-Hosts auf verschiedenen Arten von Netzwerk-Layouts. Mit redundanten Hosts können Sie die Überwachung Ihres Netzwerkes aufrecht erhalten, wenn der primäre Host, auf dem Nagios läuft, ausfällt oder wenn Teile Ihres Netzwerkes unerreichbar werden.

Anmerkung: Wenn Sie gerade lernen, wie Nagios zu nutzen ist, würde ich empfehlen, Redundanz so lange nicht zu implementieren, bis Sie mit den Voraussetzungen vertraut sind. Redundanz ist ein relativ komplexes Thema und es ist noch schwieriger, es zu implementieren.

Index

Voraussetzungen

Bevor Sie überhaupt daran denken können, Redundanz mit Nagios zu implementieren, müssen Sie mit folgenden Dingen vertraut werden...

- Implementieren von Eventhandlern für Hosts und Services
- Erteilen von externen Befehlen an Nagios über Shell-Scripts
- Ausführen von Plugins auf entfernten Hosts mit Hilfe des NRPE Addons oder einer anderen Methode
- Überprüfen des Zustands des Nagios-Prozesses mit dem *check_nagios* Plugin

Beispiel-Scripte

Jedes dieser Beispiel-Scripte, die ich in dieser Dokumentation benutze, finden Sie im *eventhandlers/*-Unterverzeichnis der Nagios-Distribution. Vielleicht müssen Sie sie modifizieren, damit sie auf Ihrem System funktionieren...

Szenario 1 - Redundante Überwachung

Einführung

Dies ist eine einfache (und harmlose) Methode, redundante Überwachungs-Hosts zu implementieren, und es wird nur gegen eine begrenzte Anzahl von Ausfällen schützen. Komplexere Setups werden benötigt, um intelligentere Redundanz, bessere Redundanz über verschiedene Netzwerk-Segmente hinweg zu bieten.

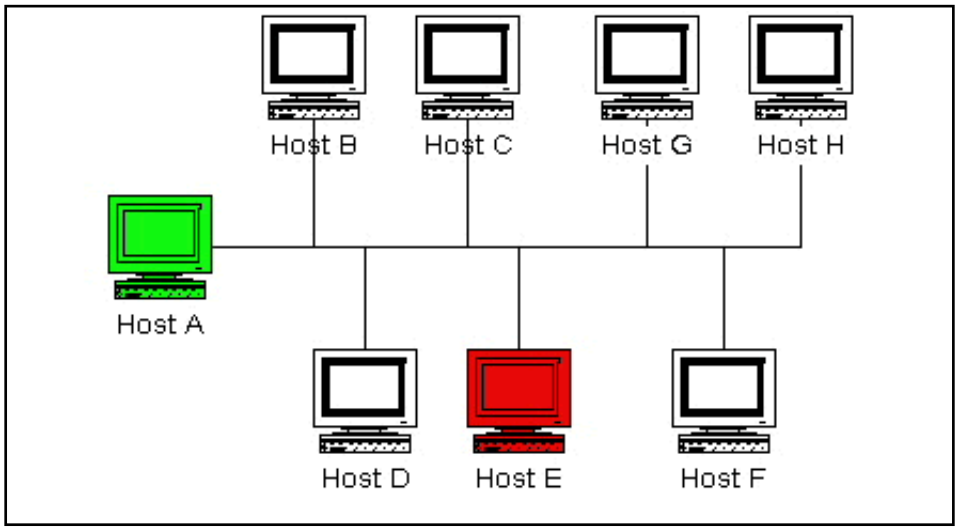
Ziele

Das Ziel dieser Art von Redundanz-Implementierung ist einfach. Sowohl der "Master"- als auch der "Slave"-Host überwachen die gleichen Hosts und Services auf dem Netzwerk. Unter normalen Umständen wird nur der "Master"-Host Benachrichtigungen an Kontakte versenden. Wir wollen, dass der "Slave"-Host die Benachrichtigung von Kontakten übernimmt, wenn:

1. der "Master"-Host, auf dem Nagios läuft, "down" ist oder...
2. der Nagios-Prozess auf dem "Master"-Host aus irgendeinem Grund stoppt

Netzwork-Layout-Diagramm

Das untenstehende Diagramm zeigt ein sehr simples Netzwerk-Setup. Bei diesem Szenario nehme ich an, dass auf den Hosts A und E Nagios läuft und alle gezeigten Hosts überwacht werden. Host A ist der "Master"-Host und Host E der "Slave"-Host.



anfängliche Programmeinstellungen

Auf dem Slave-Host (Host E) wird die ursprüngliche enable_notifications-Direktive deaktiviert, so dass dadurch der Versand von Host- oder Service-Benachrichtigungen verhindert wird. Sie sollten auch sicherstellen, dass die check_external_commands-Direktive deaktiviert ist. Das war einfach genug...

anfängliche Konfiguration

Als nächstes sollten wir die Unterschiede zwischen den Objekt-Konfigurationsdatei von Master- und Slave-Host(s) betrachten...

Ich gehe davon aus, dass Sie den Master-Host (Host A) so konfiguriert haben, dass er alle Services auf den gezeigten Hosts des Diagramms überwacht. Der Slave-Host (Host E) sollte die gleichen Hosts und Services überwachen, mit folgenden Zusätzen in der Konfigurationsdatei...

- Die Host-Definition für Host A (in der Host-Konfigurationsdatei von Host E) sollte einen Host-Eventhandler enthalten. Der Name für den Host-Eventhandler lautet handle-master-host-event.
- Die Konfigurationsdatei auf Host E enthält einen Service, der den Status des Nagios-Prozesses auf Host A prüft. Lassen Sie uns annehmen, dass diese Prüfung das *check_nagios*-Plugin auf Host A aufruft. Das kann durch eine der in den **FAQ** beschriebenen Methoden erfolgen.
- Die Service-Definition für den Nagios-Prozess auf Host A sollte einen Eventhandler-Eintrag enthalten. Als Namen für diese Service-Eventhandler wählen wir handle-master-proc-event.

Es ist wichtig anzumerken, dass Host A (der Master-Host) keine Ahnung von Host E (dem Slave-Host) hat. In diesem Szenario besteht ganz einfach keine Notwendigkeit dazu. Natürlich können Sie von Host A Services auf Host E überwachen, aber das hat nichts mit der Implementierung von Redundanz zu tun...

Eventhandler-Befehlsdefinitionen

Wir müssen kurz innehalten und beschreiben, wie die Befehlsdefinitionen für die Eventhandler auf dem Slave-Host aussehen. Hier ist ein Beispiel...

```
define command{
        command_name    handle-master-host-event
        command_line    /usr/local/nagios/libexec/eventhandlers/handle-master-host-event $HOSTSTATE$ $
        }

define command{
        command_name    handle-master-proc-event
        command_line    /usr/local/nagios/libexec/eventhandlers/handle-master-proc-event $SERVICESTATE
        }
```

Dies setzt voraus, dass Sie die Eventhandler-Scripte im Verzeichnis */usr/local/nagios/libexec/eventhandlers* abgelegt haben. Sie können sie ablegen, wohin Sie wollen, aber dann müssen Sie die beigefügten Beispiele anpassen.

Eventhandler-Scripte

Okay, lassen Sie uns nun einen Blick darauf werden, wie die Eventhandler-Scripte aussehen...

Host-Eventhandler (handle-master-host-event):

```
#!/bin/sh

# Only take action on hard host states...
case "$2" in
HARD)
        case "$1" in
        DOWN)
                # The master host has gone down!
                # We should now become the master host and take
                # over the responsibilities of monitoring the
                # network, so enable notifications...
                /usr/local/nagios/libexec/eventhandlers/enable_notifications
                ;;
        UP)
                # The master host has recovered!
                # We should go back to being the slave host and
                # let the master host do the monitoring, so
                # disable notifications...
                /usr/local/nagios/libexec/eventhandlers/disable_notifications
                ;;
        esac
        ;;
esac
exit 0
```

Service-Eventhandler (handle-master-proc-event):

```
#!/bin/sh

# Only take action on hard service states...
case "$2" in
HARD)
        case "$1" in
        CRITICAL)
                # The master Nagios process is not running!
                # We should now become the master host and
                # take over the responsibility of monitoring
                # the network, so enable notifications...
```

```
                /usr/local/nagios/libexec/eventhandlers/enable_notifications
                ;;
        WARNING)
        UNKNOWN)
                # The master Nagios process may or may not
                # be running.. We won't do anything here, but
                # to be on the safe side you may decide you
                # want the slave host to become the master in
                # these situations...
                ;;
        OK)
                # The master Nagios process running again!
                # We should go back to being the slave host,
                # so disable notifications...
                /usr/local/nagios/libexec/eventhandlers/disable_notifications
                ;;
        esac
        ;;
esac
exit 0
```

Was tun sie für uns

Auf dem Slave-Host (Host E) sind anfänglich die Benachrichtigungen deaktiviert, so dass er keine Host- oder Service-Benachrichtigungen versendet, solange der Nagios-Prozess auf dem Master-Host (Host A) noch läuft.

Der Nagios-Prozess auf dem Slave-host (Host E) wird zum Master-Host, wenn...

- der Master-Host (Host A) "down" geht und der *handle-master-host-event*-Host-Eventhandler ausgeführt wird.
- der Nagios-Prozess auf dem Master-Host (Host A) aufhört zu arbeiten und der *handle-master-proc-event*-Service-Eventhandler ausgeführt wird.

Wenn bei dem Nagios-Prozess auf dem Slave-Host (Host E) Benachrichtigungen aktiviert sind, kann er Benachrichtigungen über jegliche Host- und Service-Probleme und Erholungen versenden. An diesem Punkt hat Host E die Verantwortlichkeiten über die Benachrichtigung von Kontakten über Host- und Service-Probleme übernommen!

Der Nagios-Prozess auf Host E wird wieder zum Host-Slave, wenn...

- sich Host A wieder erholt und der *handle-master-host-event*-Host-Eventhandler ausgeführt wird.
- sich der Nagios-Prozess auf Host A wieder erholt und den *handle-master-proc-event*-Service-Eventhandler ausführt.

Wenn bei dem Nagios-Prozess auf dem Slave-Host (Host E) Benachrichtigungen deaktiviert sind, wird er keine Benachrichtigungen mehr über Host- und Service-Probleme und Erholungen versenden. An diesem Punkt hat Host E die Verantwortlichkeiten über die Benachrichtigung von Kontakten über Host- und Service-Probleme an Host A übergeben. Alles ist wieder so, als wir angefangen haben!

Zeitverzögerungen

Redundanz bei Nagios ist in keinster Weise perfekt. Eins der offenkundigeren Probleme ist die Verzögerung zwischen dem Ausfall von Host A und der Übernahme durch Host E. Das ist bedingt durch folgende Dinge...

- die Zeit zwischen dem Ausfall des Master-Host und dem ersten Mal, dass der Slave-Host ein Problem entdeckt
- die Zeit, die benötigt wird, um festzustellen, dass der Master-Host wirklich ein Problem hat (unter

Verwendung von Host- oder Service-Prüfwiederholungen auf dem Slave-Host)
- die Zeit zwischen der Ausführung des Eventhandlers und der Zeit, zu der Nagios das nächste Mal auf externe Befehle prüft

Sie können diese Verzögerung minimieren durch...

- eine hohe Frequenz von (Wiederholungs-) Prüfungen für Services auf Host E. Das kann durch die *check_interval*- und *retry_interval*-Optionen in jeder Service-Definition erreicht werden.
- eine Zahl der Host-Wiederholungsprüfungen für Host A (auf Host E), die eine schnelle Erkennung von Host-Problemen erlaubt. Das wird erreicht durch das *max_check_attempts*-Argument in der Host-Definition.
- erhöhen der Frequenz der external command-Prüfungen auf Host E. Dies wird erreicht durch die Anpassung der command_check_interval-Option in der Hauptkonfigurationsdatei.

Wenn sich Nagios auf Host A erholt, gibt es ebenfalls eine Verzögerung, bevor Host E wieder zu einem Slave-Host wird. Das wird durch folgende Dinge beeinflusst...

- die Zeit zwischen der Erholung des Master-Hosts und der Zeit, zu der der Nagios-Prozess auf Host E die Erholung erkennt
- die Zeit zwischen der Ausführung des Eventhandlers auf Host A und der Zeit, zu der Nagios-Prozess auf Host E das nächste Mal auf externe Befehle prüft

Die genaue Verzögerung zwischen dem Übergang der Verantwortlichkeiten hängt davon ab, wieviele Services Sie definiert haben, dem Intervall, in dem Services geprüft werden, und einer Menge pures Glück. Auf jeden Falls ist es besser als nichts.

Spezialfälle

Eins sollten Sie beachten: Wenn Host A "down" geht, werden bei Host E die Benachrichtigungen aktiviert und er übernimmt die Verantwortung für das Informieren der Kontakte bei Problemen. Wenn sich Host A wieder erholt, werden bei Host E die Benachrichtigungen deaktiviert. Falls der Nagios-Prozess - wenn sich Host A erholt - auf Host A nicht sauber startet, gibt es eine Zeitspanne, während der keiner der beiden Hosts die Kontakte über Probleme informiert! Glücklicherweise berücksichtigt die Service-Prüflogik in Nagios diesen Umstand. Das nächste Mal, wenn der Nagios-Prozess auf Host E den Status des Nagios-Prozesses auf Host A prüft, wird er feststellen, dass dieser nicht läuft. Auf Host E werden dann wieder die Benachrichtigungen aktiviert und er wird erneut die Verantwortung für die Benachrichtigung der Kontakte übernehmen.

Der exakte Wert für die Zeit, während der keiner der Hosts das Netzwerk überwacht, ist schwer zu ermitteln. Offensichtlich kann diese Zeit durch die Erhöhung der Frequenz von Service-Prüfungen (auf Host E) für Host A minimiert werden. Der Rest ist purer Zufall, aber die gesamte "Blackout"-Zeit sollte nicht allzu hoch sein.

Szenario 2 - Failover-Überwachung

Einführung

Failover-Überwachung ist ähnlich wie die redundante Überwachung (wie beschrieben in Szenario 1).

Ziele

Das grundlegende Ziel der Failover-Überwachung besteht darin, dass der Nagios-Prozess auf dem Slave-Host untätig ist, während der Nagios-Prozess auf dem Master-Host läuft. Wenn der Prozess auf dem Master-Host stoppt (oder der Host "down" geht), übernimmt der Nagios-Prozess auf dem Slave-Host die gesamte Überwachung.

Während es Ihnen die in Szenario 1 beschriebene Methode erlaubt, weiterhin Benachrichtigungen zu erhalten, wenn der Master-Host "down" geht, gibt es einige Fallen. Das größte Problem besteht darin, dass der Slave-Host die gleichen Hosts und Services wie der Master *zur gleichen Zeit wie der Master* überwacht! Dies kann Probleme durch übermäßigen Traffic und Load auf den überwachten Maschinen verursachen, wenn Sie viele Services definiert haben. Hier nun, wie Sie das Problem umgehen können.

Initiale Programm-Einstellungen

Deaktivieren Sie aktive Service-Prüfungen und Benachrichtigungen auf dem Slave-Host durch die execute_service_checks- und die enable_notifications-Direktiven. Dies wird den Slave-Host davon abhalten, Services und Hosts zu überwachen und Benachrichtigungen zu versenden, während der Nagios-Prozess auf dem Master-Host noch läuft. Stellen Sie außerdem sicher, dass die check_external_commands-Direktive auf dem Slave-Host aktiviert ist.

Master-Prozess-Prüfungen

Setzen Sie einen cron-Job auf dem Slave-Host auf, der periodisch (sagen wir jede Minute) läuft und den Status des Nagios-Prozesses auf dem Master-Host (mit dem *check_nrpe* auf dem Slave-Host und den nrpe daemon und *check_nagios*-Plugins auf dem Master-Host) prüft. Das Script sollte den Return-Code des *check_nrpe-Plugins* prüfen. Falls es einen nicht-OK-Status zurückliefert, sollte das Script den entsprechenden Befehl an das external command file senden, um sowohl die Benachrichtigungen als auch die aktiven Service-Prüfungen zu aktivieren. Falls das Plugin einen OK-Status zurückliefert, sollte das Script Befehle an das external command file senden, um sowohl Benachrichtigungen als auch aktive Prüfungen zu deaktivieren.

Auf diese Weise läuft jeweils nur ein Prozess, der Hosts und Services prüft, was wesentlich effizienter ist als alles doppelt zu überwachen.

Auch von Interesse: Sie müssen *nicht* wie in Szenario 1 beschrieben die Host- und Service-Handler definieren, weil die Dinge anders behandelt werden.

Zusätzliche Themen

An diesem Punkt haben Sie ein sehr einfaches Failover-Überwachungs-Setup implementiert. Trotzdem gibt es einen weiteren Punkt, den Sie berücksichtigen sollten, damit die Dinge besser laufen.

Das große Problem dabei, wie die Dinge bisher konfiguriert sind, besteht darin, dass der Slave-Host nicht den aktuellen Status von Hosts und Services kennt, wenn er die Überwachung übernimmt. Ein Weg, dieses Problem zu lösen, ist es, die ocsp command-Option auf dem Master-Host zu aktivieren und alle Service-Prüfergebnisse mit dem nsca Addon an den Slave-Host zu schicken. Der Slave-Host wird dann aktuelle Status-Informationen für alle Services haben, wenn er die Überwachung übernimmt. Weil aktive Service-Prüfungen auf dem Slave-Host nicht aktiviert sind, werden sie nicht ausgeführt. Host-Prüfungen hingegen werden nach Bedarf ausgeführt. Das bedeutet, dass sowohl Master- als auch Slave-Host Host-Prüfungen ausführen, wenn sie benötigt werden, was kein Problem darstellen sollte, weil die Mehrzahl der Überwachung Service-Prüfungen betrifft.

Das ist eigentlich alles, was das Setup betrifft.

Nagios®

Erkennung und Behandlung von Status-Flattern

Hoch zu: Inhalt

Siehe auch: Status-Typen

Einführung

Nagios unterstützt die Erkennung von Hosts und Services, die "flattern". Flattern tritt auf, wenn Hosts oder Services zu oft den Zustand wechseln und dadurch einen Sturm von Problemen und Erholungsbenachrichtigungen erzeugen. Flattern kann auf Konfigurationsprobleme hinweisen (z.B. Schwellwerte, die zu niedrig gesetzt sind), störende Services oder wirkliche Netzwerkprobleme.

Wie Flatter-Erkennung arbeitet

Bevor ich darauf eingehe, lassen Sie mich sagen, dass es etwas schwierig war, Flatter-Erkennung zu implementieren. Wie genau legt man fest, was "zu häufig" in Bezug auf Statusänderungen für einen Host oder Service ist? Als ich zuerst an die Implementierung der Flatter-Erkennung gedacht habe, versuchte ich Informationen zu finden, wie Flattern erkannt werden könnte/sollte. Ich konnte keinerlei Informationen darüber finden, was andere benutzten (benutzen andere so etwas?), also entschied ich mich für das, was ich für eine sinnvolle Lösung hielt...

Sobald Nagios den Zustand eines Hosts oder Services prüft, wird es prüfen, ob dafür Flattern begonnen oder geendet hat. Es tut dies durch:

- speichern der Ergebnisse der letzten 21 Prüfungen des Hosts oder Service
- analysieren der historischen Prüfergebnisse und feststellen, wo Statusänderungen/-übergänge auftreten
- benutzen der Statusübergänge, um einen Statuswechsel-Prozentsatz (ein Maß für die Änderung) für den Statuswechsel des Hosts oder Service festzulegen
- vergleichen des Statuswechsel-Prozentwertes gegen die Flatter-Schwellwerte (hoch und niedrig)

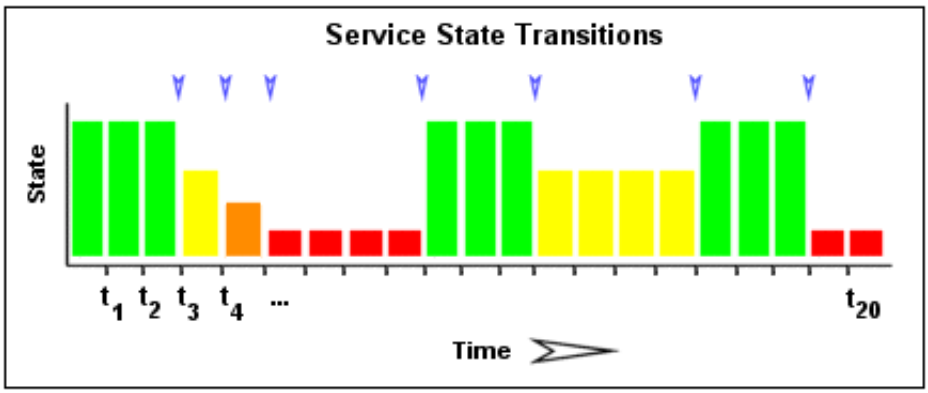
Ein Host oder Service wird angesehen, mit dem Flatter *begonnen* zu haben, wenn der Prozentsatz das erste Mal einen *hohen* Flatter-Schwellwert überschritten hat.

Ein Host oder Service wird angesehen, das Flattern *beendet* zu haben, wenn der Prozentsatz unter einen *niedrigen* Flatter-Schwellwert sinkt (vorausgesetzt, dass er vorher geflattert hat).

Beispiel

Lassen Sie uns etwas detaillierter beschreiben, wie Flatter-Erkennung bei Services arbeitet...

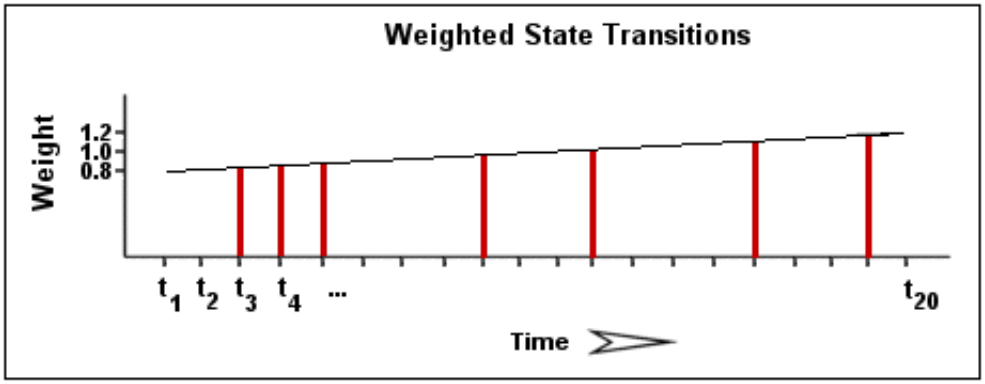
Das Bild unten zeigt eine chronologische Historie von Service-Zuständen der letzten 21 Service-Prüfungen. OK-Zustände sind in grün dargestellt, WARNING-Zustände in gelb, CRITICAL-Zustände in rot und UNKNOWN-Zustände in orange.



Die historischen Service-Prüfergebnisse werden untersucht, um festzustellen, wo Statusänderungen/-übergänge auftreten. Statusänderungen treten auf, wenn ein archivierter Status sich von den archivierten Zuständen unterscheidet, die ihm direkt vorausgehen. Da wir die Ergebnisse der letzten 21 Status-Prüfungen in dem Array ablegen, können wir bis zu 20 Statusänderungen haben. In diesem Beispiel gibt es sieben Statusänderungen, die im Bild durch blaue Pfeile gekennzeichnet sind.

Die Flatter-Erkennungslogik nutzt die Statusänderungen, um einen Gesamtprozentsatz für den Service festzulegen. Dies ist ein Maßstab für die Sprunghaftigkeit/Änderung des Service. Services, die nie den Status wechseln, haben einen Statusänderungswert von 0%, während Services, die ihren Status bei jeder Prüfung wechseln, einen Wert von 100% haben. Die meisten Services werden einen Prozentwert irgendwo dazwischen haben.

Während der Berechnung des Prozentsatzes für den Service wird der Flatter-Erkennungsalgorithmus mehr Gewicht auf neuere Statusänderungen legen als auf alte. Genauer gesagt sind die Flatter-Erkennungsroutinen im Moment so ausgelegt, dass der neueste Statuswechsel 50% mehr Gewicht hat als der älteste. Das Bild unten zeigt, wie neuere Statuswechsel mehr Gewicht erhalten als ältere, während der Gesamtprozentwert für einen bestimmten Service berechnet wird.



Lassen Sie uns mit dem obigen Bild eine Berechnung der prozentualen Statusänderungen für den Service durchführen. Sie werden bemerken, dass es insgesamt sieben Statuswechsel gibt (bei t_3, t_4, t_5, t_9, t_{12}, t_{16} und t_{19}). Ohne Gewichtung der Statuswechsel über die Zeit würde dies einen Gesamtwert von 35% ergeben:

(7 beobachtete Statuswechsel / 20 mögliche Statuswechsel) * 100 = 35 %

Nachdem die Flatter-Erkennungslogik neueren Statuswechseln mehr Gewicht gibt als älteren, wird der eigentliche Wert in diesem Beispiel geringfügig kleiner sein als 35%. Lassen Sie uns annehmen, dass der gewichtete Prozentwert 31% ist...

Der errechnete Prozentwert für den Service (31%) wird dann gegen die Flatter-Schwellwerte verglichen, um zu sehen, was passiert:

- wenn der Service bisher *nicht* flatterte und 31% *gleich oder größer* als der hohe Flatter-Schwellwert ist, nimmt Nagios an, dass der Service gerade angefangen hat zu flattern.
- wenn der Service *bereits* flatterte und 31% *unter* dem niedrigen Flatter-Schwellwert liegt, nimmt Nagios an, dass der Service gerade aufgehört hat zu flattern.

wenn keine der beiden Bedingungen zutrifft, dann macht die Flatter-Erkennungslogik nichts weiteres mit dem Service, da er entweder (noch) nicht flattert oder bereits flattert.

Flatter-Erkennung für Services

Nagios prüft jedes Mal, wenn der Service geprüft wird (egal ob aktiv oder passiv), ob ein Service flattert.

Die Flatter-Erkennungslogik für Services arbeitet wie in dem obigen Beispiel beschrieben.

Flatter-Erkennung für Hosts

Host-Flatter-Erkennung arbeitet in einer ähnlichen Weise wie die Service-Flatter-Erkennung, mit einem wichtigen Unterschied: Nagios wird versuchen zu prüfen, ob ein Host flattert, wenn:

- der Host geprüft wird (aktiv oder passiv)
- manchmal, wenn ein Service geprüft wird, der mit dem Host verbunden ist. Genauer gesagt, wenn wenigstens *x* der Zeit vergangen ist, seit die letzte Flatter-Erkennung durchgeführt wurde, wobei *x* dem Durchschnittsintervall aller Services entspricht, die mit dem Host verbunden sind.

Warum wird das gemacht? Bei Services wissen wir, dass die minimale Zeit zwischen zwei aufeinander folgenden Flatter-Erkennungsroutinen gleich dem Service-Prüfintervall sein wird. Allerdings werden Sie Hosts wahrscheinlich nicht auf einer regelmäßigen Basis überwachen, so dass es kein Prüfintervall gibt, das in der Flatter-Erkennungslogik benutzt werden kann. Außerdem ist es sinnvoll, dass die Prüfung eines Service der Erkennung eines Host-Flatterns dienen sollte. Services sind Attribute eines Hosts bzw. bezogen auf Dinge, die mit dem Host verbunden sind. Auf jeden Fall ist es die beste Methode, die ich gefunden habe, um festzulegen, wie oft die Flatter-Erkennung auf einem Host ausgeführt werden kann.

Flatter-Erkennungsschwellwerte

Nagios benutzt verschiedene Variablen, um die Schwellwert-Prozentsätze der Statusänderungen festzulegen, die es für die Flatter-Erkennung nutzt. Für Hosts und Services gibt es hohe und niedrige *globale* und *Host-* und *Service-spezifische* Schwellwerte, die Sie konfigurieren können. Nagios wird die globalen Schwellwerte für die Flatter-Erkennung nutzen, wenn Sie keine Host- oder Service-spezifischen Schwellwerte angegeben haben.

Die Tabelle unten zeigt die globalen und die Host- oder Service-spezifischen Variablen, die die verschiedenen Schwellwerte kontrollieren, die bei der Flatter-Erkennung benutzt werden.

Objekt-Typ	Globale Variable	Objekt-spezifische Variablen
Host	low_host_flap_threshold high_host_flap_threshold	low_flap_threshold high_flap_threshold
Service	low_service_flap_threshold high_service_flap_threshold	low_flap_threshold high_flap_threshold

Zustände, die für die Flatter-Erkennung benutzt werden

Normalerweise wird Nagios die Ergebnisse der letzten 21 Präfungen eines Hosts oder Service verfolgen, unabhängig vom Prüfergebnis (Host-/Service-Zustand), um sie für die Flatter-Erkennungslogik zu benutzen.

 Hinweis: Sie können durch die *flap_detection_options*-Direktive in Ihren Host- oder Service-Definitonen verschiedene Host- oder Service-Zustände von der Nutzung in der Flatter-Erkennungslogik ausschließen. Diese Direktive erlaubt Ihnen die Angabe, welche Host- oder Service-Zustände (z.B. "UP", "DOWN", "OK", "CRITICAL") Sie für die Flatter-Erkennung benutzen wollen. Wenn Sie diese Direktive nicht nutzen wollen, werden alle Host- und Service-Zustände in der Flatter-Erkennung benutzt.

Flatter-Behandlung

Wenn bei einem Service- oder Host das erste Mal Flattern erkannt wird, wird Nagios:

1. eine Meldung protokollieren, dass der Service oder Host flattert
2. einen nicht-permanenten Kommentar zum Host oder Service hinzufügen, dass er flattert
3. eine "flapping start"-Benachrichtigung für den Host oder Service an die betreffenden Kontakte versenden
4. andere Benachrichtigungen für den Service oder Host unterdrücken (das ist einer der Filter in der Benachrichtigungslogik)

Wenn ein Service oder Host aufhört zu flattern, wird Nagios:

1. eine Meldung protokollieren, dass der Service oder Host nicht mehr flattert
2. den Kommentar löschen, der zum Service oder Host hinzugefügt wurde, als dieser anfing zu flattern
3. eine "flapping stop"-Benachrichtigung für den Host oder Service an die betreffenden Kontakte versenden
4. die Blockade von Benachrichtigungen für den Service oder Host entfernen (Benachrichtigungen sind nach wie vor an die normale Benachrichtigungslogik gebunden)

Aktivieren der Flatter-Erkennung

Um die Flatter-Erkennungsmöglichkeiten in Nagios zu aktivieren, müssen Sie folgendes tun:

- setzen Sie die enable_flap_detection-Direktive auf 1.
- setzen Sie die *flap_detection_enabled*-Direktive in Ihren Host- und Service-Definitionen auf 1.

Wenn Sie die Flatter-Erkennung auf einer globalen Ebene deaktivieren wollen, setzen Sie die enable_flap_detection-Direktive auf 0.

Wenn Sie die Flatter-Erkennung nur für einige Hosts oder Services deaktivieren wollen, nutzen Sie die *flap_detection_enabled*-Direktive in den Host- oder Service-Definitionen, um das zu tun.

Benachrichtigungseskalationen

Hoch zu: Inhalt

Siehe auch: Benachrichtigungen, Zeitfenster

Einführung

Nagios unterstützt optionale Eskalation von Kontakt-Benachrichtigungen für Hosts und Services. Eskalationen von Host- oder Service-Benachrichtigungen werden erreichen durch das Definieren von Host-Eskalationen bzw. Service-Eskalationen in Ihrer/Ihren Objekt-Konfigurationsdatei(en).

Anmerkung: Das Beispiel, das ich unten zeige, benutzt Service-Eskalationsdefinitionen, aber Host-Eskalationen arbeiten genau so. Außer, dass sie für Hosts sind statt für Services. :-)

Wann werden Benachrichtigungen eskaliert?

Benachrichtigungen werden eskaliert, *wenn, und nur wenn* eine oder mehrere Eskalationsdefinitionen mit der aktuellen Benachrichtigung übereinstimmen, die gerade versandt wird. Wenn eine Host- oder Service-Benachrichtigung *keine* gültige Eskalationsdefinition hat, die auf sie zutrifft, dann wird die Benachrichtigung an die Kontaktgruppe(n) verschickt, die in der Hostgroup- oder Service-Definition angegeben wurde(n). Lassen Sie uns das untenstehende Beispiel betrachten:

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   90
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      6
        last_notification       10
        notification_interval   60
        contact_groups          nt-admins,managers,everyone
        }
```

Beachten Sie, dass es "Lücken" in den Benachrichtigungs-Eskalationsdefinitionen gibt. Im Besonderen werden weder die Benachrichtigungen 1 und 2 von den Eskalationen behandelt noch die Benachrichtigungen über 10. Für die ersten beiden und die Benachrichtigungen über 10 werden die

Default-Kontaktgruppen aus der Service-Definition benutzt. Bei allen Beispielen, die ich benutze, nehme ich an, dass die Default-Kontaktgruppe für die Service-Definition *nt-admins* lautet.

Kontaktgruppen

Beim Definieren von Benachrichtigungs-Eskalationen ist es wichtig zu wissen, dass alle Kontaktgruppen, die Mitglieder von "niedrigeren" Eskalationen (d.h. mit niedrigeren Benachrichtigungsnummern-Bereichen) sind, auch in den "höheren" Eskalationsdefinitionen enthalten sein sollen. Das sollte passieren, um sicherzustellen, dass jeder, der über ein Problem informiert wird, *weiterhin* informiert wird, wenn ein Problem eskaliert. Beispiel:

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   90
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      6
        last_notification       0
        notification_interval   60
        contact_groups          nt-admins,managers,everyone
        }
```

Die erste (oder "niedrigste") Eskalationsstufe umfasst die *nt-admins* und die *managers*-Kontaktgruppe. Die letzte (oder "höchste") umfasst die *nt-admins*, *managers* und *everyone*-Kontaktgruppen. Beachten Sie, dass die *nt-admins*-Kontaktgruppe in beiden Eskalationsdefinitionen enthalten ist. Dies passiert, damit sie weiterhin per Pager informiert werden, falls noch Probleme existieren, nachdem die ersten beiden Service-Benachrichtigungen versandt wurden. Die *managers*-Kontaktgruppe erscheint zuerst in der "niedrigen" Eskalationsdefinition - sie wird das erste Mal benachrichtigt, wenn die dritte Benachrichtigung versandt wird. Wir möchten, dass die *managers*-Gruppe weiterhin informiert wird, wenn das Problem nach der fünften Benachrichtigung noch existiert, also sind sie in der "höheren" Eskalationsdefinition enthalten.

Überlappende Eskalationsbereiche

Benachrichtigungs-Eskalationsdefinitionen können Benachrichtigungs-Bereiche haben, die überlappen. Nehmen Sie das folgende Beispiel:

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   20
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      4
        last_notification       0
        notification_interval   30
        contact_groups          on-call-support
        }
```

Im obigen Beispiel:

- die *nt-admins*- und *managers*-Kontaktgruppen werden bei der dritten Benachrichtung informiert
- alle drei Kontaktgruppen werden bei der vierten und fünften Benachrichtigung informiert
- nur die *on-call-support*-Kontaktgruppe wird bei der sechsten (und höheren) Benachrichtigung informiert

Erholungsbenachrichtigungen

Erholungsbenachrichtigungen unterscheiden sich geringfügig von Problembenachrichtigungen, wenn es um Eskalationen geht. Nehmen Sie das folgende Beispiel:

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   20
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      4
        last_notification       0
        notification_interval   30
        contact_groups          on-call-support
        }
```

Falls nach drei Problembenachrichtigungen eine Erholungsbenachrichtigung für den Service versandt wird: wer wird informiert? Die Erholung ist eigentlich die vierte Benachrichtigung, die versandt wird. Allerdings ist der Eskalationscode intelligent genug zu erkennen, dass nur die Leute, die die dritte Problembenachrichtigung erhalten haben, auch über die Erholung informiert werden. In diesem Fall würden die *nt-admins*- und *managers*-Kontaktgruppen über die Erholung informiert werden.

Benachrichtigungsintervalle

Sie können die Häufigkeit, mit der eskalierte Benachrichtigungen für einen bestimmten Host oder Service versandt werden, mit der *notification_interval*-Option in der Hostgroup- oder Service-Eskalations-Definition ändern. Beispiel:

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   45
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      6
        last_notification       0
        notification_interval   60
        contact_groups          nt-admins,managers,everyone
        }
```

In diesem Beispiel sehen wir, dass das Default-Benachrichtigungsintervall für den Service auf 240 Minuten eingestellt ist (das ist der Wert in der Service-Definition). Wenn die Service-Benachrichtigung bei der dritten, vierten und fünften Benachrichtigung eskaliert, wird ein Intervall von 45 Minuten zwischen den Benachrichtigungen genutzt. Bei der sechsten und folgenden Benachrichtigungen ist das Benachrichtigungsintervall 60 Minuten, wie in der zweiten Eskalationsdefinition angegeben.

Nachdem es möglich ist, überlappende Eskalationsdefinitonen für eine bestimmte Hostgruppe oder einen Service zu haben, und der Tatsache, dass ein Host Mitglied von mehreren Hostgruppen sein kann, muss Nagios eine Entscheidung treffen, was zu tun ist, wenn die Benachrichtigungs-Intervalle von Eskalationsdefinitionen überlappen. In jedem Fall, wenn es mehrere gültige Eskalationsdefinitionen für eine bestimmte Benachrichtigung gibt, wird Nagios das kleinste Benachrichtigungs-Intervall wählen. Nehmen Sie das folgende Beispiel:

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   45
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      4
        last_notification       0
        notification_interval   60
        contact_groups          nt-admins,managers,everyone
        }
```

Wir sehen, dass die beiden Eskalationsdefinitionen bei der vierten und fünften Benachrichtigung überlappen. Bei diesen Benachrichtigungen wird Nagios ein Benachrichtigungsintervall von 45 Minuten benutzen, weil dies das kleinste Intervall aller vorhandenen gültigen Eskalationsdefinitionen für diese Benachrichtigungen ist.

Eine letzte Anmerkung zu Benachrichtigungsintervallen, die Intervalle von 0 behandelt. Ein Intervall von 0 bedeutet, dass Nagios lediglich eine Benachrichtigung für die erste gültige Benachrichtigung während der Eskalationsdefinition versendet. Alle folgenden Benachrichtigungen für die Hostgruppe oder den Service werden unterdrückt. Nehmen Sie dieses Beispiel:

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   45
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      4
        last_notification       6
        notification_interval   0
        contact_groups          nt-admins,managers,everyone
        }

define serviceescalation{
        host_name               webserver
```

```
service_description     HTTP
first_notification      7
last_notification       0
notification_interval   30
contact_groups          nt-admins,managers
}
```

In dem obigen Beispiel werden maximal vier Problembenachrichtigungen zu diesem Service versandt. Das ist so, weil das Benachrichtigungsintervall 0 in der zweiten Eskalationsdefinition angibt, dass nur eine Benachrichtigung versandt werden soll (beginnend mit der vierten und diese einschließend) und folgende Benachrichtigungen unterdrückt werden sollen. Deshalb hat die dritte Eskalationsdefinition keinerlei Auswirkungen, denn es wird nie mehr als vier Benachrichtigungen geben.

Zeitfenster-Beschränkungen

Unter normalen Umständen können Eskalationen zu jeder Zeit benutzt werden, zu der Benachrichtigungen für einen Host oder Service versandt werden. Dieses "Benachrichtigungs-Zeitfenster" ist festgelegt durch die *notification_period*-Direktive in der Host- oder Service-Definition.

Sie können optional Eskalationen durch die *escalation_period*-Direktive in der Host- oder Service-Eskalationsdefinition beschränken, so dass sie lediglich während bestimmter Zeitspannen benutzt werden. Wenn Sie die *escalation_period*-Direktive benutzen, um eine Zeitspanne zu definieren, während der die Eskalation benutzt werden kann, wird sie nur zu dieser Zeit benutzt. Wenn Sie keine *escalation_period*-Direktive angeben, kann die Eskalation zu jeder Zeit innerhalb des "Benachrichtigungs-Zeitfensters" des Hosts oder Service benutzt werden.

Anmerkung: eskalierte Benachrichtigungen unterliegen weiterhin den normalen Zeitbeschränkungen, die durch die *notification_period*-Direktive in einer Host- oder Service-Definition festgelegt wurden, so dass die Zeitspanne, die Sie in einer Eskalationsdefinition angeben, ein Teil des größeren "Benachrichtigungs-Zeitfensters" sein sollte.

Status-Beschränkungen

Wenn Sie die Eskalationsdefinition beschränken wollen, damit sie nur benutzt wird, während sich der Host oder Service in einem bestimmten Zustand befindet, so können Sie die *escalation_options*-Direktive in der Host- oder Service-Eskalationsdefinition benutzen. Wenn Sie die *escalation_options*-Direktive nicht verwenden, werden die Eskalationen in jedem Status der Hosts oder Services benutzt.

Bereitschafts-Rotationen

Hoch zu: Inhalt

Siehe auch: Zeitfenster, Benachrichtigungen

Einführung

Admins müssen oft genug Pager, Mobiltelefonanrufe usw. beantworten, wenn sie es am wenigsten gebrauchen können. Keiner mag es, morgens um 4 Uhr geweckt zu werden. Allerdings ist es oft besser, das Problem mitten in der Nacht zu lösen als den Zorn eines unglücklichen Chefs zu spüren, wenn Sie am nächsten Morgen um 9 Uhr ins Büro kommen.

Für die glücklichen Admins, die ein Team von Gurus haben, die die Verantwortlichkeiten bei der Beantwortung von Alarmen teilen können, gibt es oft Bereitschaftspläne. Mehrere Admins werden oft abwechselnd Benachrichtigungen an Wochenenden, Nächten, Urlauben usw. entgegennehmen.

Ich werde Ihnen zeigen, wie Sie Zeitfenster-Definitionen erstellen können, die die meisten Bereitschafts-Benachrichtigungen behandeln werden. Diese Definitionen werden keine menschlichen Dinge berücksichtigen, die unweigerlich auftreten werden (Admins, die sich krank melden, Tausch von Schichten, oder Pager, die ins Wasser fallen), aber sie werden es Ihnen erlauben, eine grundlegende Struktur in Ihre Aufteilung zu bringen, die für die meiste Zeit funktionieren wird.

Szenario 1: Urlaub und Wochenenden

Zwei Admins - John und Bob - sind verantwortlich für die Bearbeitung von Nagios-Alarmen. John erhält alle Benachrichtigungen an Wochentagen (und Nächten) - außer im Urlaub - und Bob erhält Benachrichtigungen während der Wochenenden und Urlaube. Glücklicher Bob. Hier nun, wie Sie diese Art der Rotation mit Zeitfenstern definieren...

Definieren Sie zuerst ein Zeitfenster, das Bereiche für Urlaube enthält:

```
define timeperiod{
        name                    holidays
        timeperiod_name         holidays
        january 1               00:00-24:00     ; New Year's Day
        2008-03-23              00:00-24:00     ; Easter (2008)
        2009-04-12              00:00-24:00     ; Easter (2009)
        monday -1 may           00:00-24:00     ; Memorial Day (Last Monday in May)
        july 4                  00:00-24:00     ; Independence Day
        monday 1 september      00:00-24:00     ; Labor Day (1st Monday in September)
        thursday 4 november     00:00-24:00     ; Thanksgiving (4th Thursday in November)
        december 25             00:00-24:00     ; Christmas
        december 31             17:00-24:00     ; New Year's Eve (5pm onwards)
        }
```

Als nächstes definieren Sie ein Zeitfenster für Johns Bereitschaftszeiten, das die Wochentage und Nächte während der Woche enthält, aber die Daten/Zeiten im Urlaubs-Zeitfenster ausschließt:

```
define timeperiod{
        timeperiod_name john-oncall
        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
        exclude         holidays                ; Exclude holiday dates/times defined elsewhere
        }
```

Sie können nun dieses Zeitfenster in Johns Kontaktdefinition referenzieren:

```
define contact{
        contact_name                    john
        ...
        host_notification_period        john-oncall
        service_notification_period     john-oncall
        }
```

Definieren Sie ein neues Zeitfenster für Bobs Bereitschaftszeiten, das die Wochenenden und die Daten/Zeiten der o.g. holiday-Zeitfenster enthält:

```
define timeperiod{
        timeperiod_name bob-oncall
        friday          00:00-24:00
        saturday        00:00-24:00
        use             holidays        ; Also include holiday date/times defined elsewhere
        }
```

Sie können nun auf dieses Zeitfenster in Bobs Kontaktdefinition referenzieren:

```
define contact{
        contact_name                    bob
        ...
        host_notification_period        bob-oncall
        service_notification_period     bob-oncall
        }
```

Szenario 2: Abwechselnde Tage

In diesem Szenario wechseln sich John und Bob täglich mit der Bearbeitung von Alarmen ab - unabhängig davon, ob es sich um Wochenenden, Wochentage oder Urlaub handelt.

Definieren Sie ein Zeitfenster, wann John Benachrichtigungen erhalten soll. Angenommen, der heutige Tage ist der 1. August 2007 und John beginnt heute mit der Bearbeitung von Benachrichtigungen, dann würde die Definition wie folgt aussehen:

```
define timeperiod{
        timeperiod_name                 john-oncall
        2007-08-01 / 2                  00:00-24:00     ; Every two days, starting August 1st, 2007
        }
```

Nun definieren Sie ein Zeitfenster, wann Bob Benachrichtigungen erhalten soll. Bob erhält Benachrichtigungen an den Tagen, an denen John keine erhält, also beginnt seine erste Bereitschaft morgen (2. August 2007).

```
define timeperiod{
        timeperiod_name                 bob-oncall
        2007-08-02 / 2                  00:00-24:00     ; Every two days, starting August 2nd, 2007
        }
```

Nun müssen Sie diese Zeitfenster-Definitionen in den Kontaktdefinitionen von John und Bob referenzieren.

```
define contact{
        contact_name                    john
        ...
        host_notification_period        john-oncall
        service_notification_period     john-oncall
        }

define contact{
        contact_name                    bob
        ...
        host_notification_period        bob-oncall
        service_notification_period     bob-oncall
        }
```

Szenario 3: Abwechselnde Wochen

In diesem Szenario wechseln sich John und Bob jede Woche mit der Bearbeitung von Alarmen ab. John bearbeitet Alarme von Sonntag bis Samstag in der einen Woche und Bob bearbeitet Alarme in den nächsten sieben Tagen. Dies wiederholt sich immer wieder.

Definieren Sie ein Zeitfenster, wann John Benachrichtigungen erhalten soll. Angenommen, heute ist Sonntag, der 29. Juli 2007 und John bearbeitet Benachrichtigungen in dieser Woche (beginnend mit heute), würde die Definition wie folgt aussehen:

```
define timeperiod{
        timeperiod_name john-oncall
        2007-07-29 / 14 00:00-24:00     ; Every 14 days (two weeks), starting Sunday, July 29th, 2007
        2007-07-30 / 14 00:00-24:00     ; Every other Monday starting July 30th, 2007
        2007-07-31 / 14 00:00-24:00     ; Every other Tuesday starting July 31st, 2007
        2007-08-01 / 14 00:00-24:00     ; Every other Wednesday starting August 1st, 2007
        2007-08-02 / 14 00:00-24:00     ; Every other Thursday starting August 2nd, 2007
        2007-08-03 / 14 00:00-24:00     ; Every other Friday starting August 3rd, 2007
        2007-08-04 / 14 00:00-24:00     ; Every other Saturday starting August 4th, 2007
        }
```

Nun definieren Sie ein Zeitfenster, wann Bob Benachrichtigungen erhalten soll. Bob erhält Benachrichtigungen in den Wochen, in denen John keine bekommt, also startet seine erste Bereitschaft am nächsten Sonntag (5. August 2007).

```
define timeperiod{
        timeperiod_name bob-oncall
        2007-08-05 / 14 00:00-24:00     ; Every 14 days (two weeks), starting Sunday, August 5th, 2007
        2007-08-06 / 14 00:00-24:00     ; Every other Monday starting August 6th, 2007
        2007-08-07 / 14 00:00-24:00     ; Every other Tuesday starting August 7th, 2007
        2007-08-08 / 14 00:00-24:00     ; Every other Wednesday starting August 8th, 2007
        2007-08-09 / 14 00:00-24:00     ; Every other Thursday starting August 9th, 2007
        2007-08-10 / 14 00:00-24:00     ; Every other Friday starting August 10th, 2007
        2007-08-11 / 14 00:00-24:00     ; Every other Saturday starting August 11th, 2007
        }
```

Nun müssen Sie diese Zeitfenster-Definitionen in den Kontaktdefinitionen von John und Bob referenzieren.

```
define contact{
        contact_name                    john
        ...
        host_notification_period        john-oncall
        service_notification_period     john-oncall
        }

define contact{
        contact_name                    bob
        ...
        host_notification_period        bob-oncall
        service_notification_period     bob-oncall
        }
```

Szenario 4: Urlaubstage

In diesem Szenario bearbeitet John Benachrichtigungen an allen Tagen außer an denen, an denen er frei hat. Er hat frei an einigen festen Tagen im Monat ebenso wie an einigen geplanten Urlaubszeiten. Bob bearbeitet Benachrichtigungen, wenn John Urlaub hat oder nicht im Büro ist.

Definieren Sie zuerst ein Zeitfenster, das die Bereiche für Johns Urlaubstage und freie Tage enthält:

```
define timeperiod{
       name                    john-out-of-office
       timeperiod_name         john-out-of-office
       day 15                  00:00-24:00             ; 15th day of each month
       day -1                  00:00-24:00             ; Last day of each month (28th, 29th, 30th, or 31st)
       day -2                  00:00-24:00             ; 2nd to last day of each month (27th, 28th, 29th, or 30th)
       january 2               00:00-24:00             ; January 2nd each year
       june 1 - july 5         00:00-24:00             ; Yearly camping trip (June 1st - July 5th)
       2007-11-01 - 2007-11-10 00:00-24:00             ; Vacation to the US Virgin Islands (November 1st-10th, 2007)
       }
```

Als nächstes definieren Sie ein Zeitfenster für Johns Bereitschaftszeiten, das die Daten/Zeiten im o.g. Zeitfenster ausschließt:

```
define timeperiod{
        timeperiod_name         john-oncall
        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
        exclude                 john-out-of-office      ; Exclude dates/times John is out
        }
```

Sie können nun dieses Zeitfenster in Johns Kontaktdefinition referenzieren:

```
define contact{
        contact_name                    john
        ...
        host_notification_period        john-oncall
        service_notification_period     john-oncall
        }
```

Definieren Sie ein neues Zeitfenster für Bobs Bereitschaftszeiten, das die Zeiten von Johns Abwesenheiten enthält:

```
define timeperiod{
       timeperod_name                bob-oncall
       use                           john-out-of-office      ; Include holiday date/times that John is out
       }
```

Sie können nun dieses Zeitfenster in Bobs Kontaktdefinition referenzieren:

```
define contact{
        contact_name                    bob
        ...
        host_notification_period        bob-oncall
        service_notification_period     bob-oncall
        }
```

Andere Szenarien

Es gibt eine Menge von anderen Bereitschafts-Benachrichtigungs-Szenarien, die Sie haben könnten. Die Datumsausschluss-Direktive in den Zeitfenster-Definitionen ist in der Lage, die meisten Datums- und Zeitbereiche abzubilden, die Sie brauchen könnten, also betrachten Sie die verschiedenen Formate, die Sie benutzen können. Wenn Sie einen Fehler bei der Erstellung von Zeitfenster-Definitionen machen, dann sollten Sie darauf achten, jemand anderem mehr Bereitschaftszeit zu geben. :-)

Nagios®

Service- und Host-Gruppen überwachen

Hoch zu: Inhalt

Siehe auch: Aktive Prüfungen, Makros

Einführung

Einige Leute haben gefragt, wie man Gruppen (Cluster) von Hosts und Services überwacht, also habe ich entschieden, eine kleine Dokumentation zu schreiben, wie man das macht. Es ist ziemlich geradeaus, also hoffentlich sind die Dinge einfach zu verstehen...

Zuerst benötigen wir eine Definition, was wir mit "Cluster" meinen. Der einfachste Weg, dies zu verstehen, ist mit einem Beispiel. Lassen Sie uns annehmen, Ihr Unternehmen hat fünf Hosts, die redundante DNS-Services für Ihr Unternehmen zur Verfügung stellt. Wenn einer ausfällt, ist das keine große Katastrophe, weil die verbleibenden Server weiterhin die Namensauflösung bereitstellen. Wenn Sie mit der Überwachung der Verfügbarkeit der DNS-Server betraut sind, werden Sie fünf Server überwachen wollen. Das ist das, was ich als *Service*-Cluster ansehen würde. Der Service-Cluster besteht aus fünf einzelnen DNS-Services, die Sie überwachen wollen. Obwohl Sie jeden einzelnen Service überwachen wollen, wird Ihr Hauptaugenmerk eher auf dem Gesamtstatus des DNS-Service-Clusters liegen als auf der Verfügbarkeit eines einzelnen Service.

Wenn Ihre Organisation eine Gruppe von Hosts hat, die eine Hochverfügbarkeitslösung darstellt, würde ich dies als *Host*-Cluster bezeichnen. Wenn ein bestimmter Host ausfällt, wird ein anderer einspringen, um die Aufgaben des ausgefallenen zu übernehmen. Als eine Randbemerkung: sehen Sie sich das High-Availability Linux Project für Informationen zur Redundanz von Hosts und Services mit Linux an.

Angriffsplan

Es gibt mehrere Wege, wie Sie eventuell Service- oder Host-Gruppen überwachen können. Ich werde die Methode beschreiben, von der ich glaube, dass sie die Einfachste ist. Service- oder Host-Cluster überwachen umfasst zwei Dinge:

- überwachen einzelner Cluster-Elemente
- überwachen des Clusters als eine gesamte Einheit

Das Überwachen von einzelnen Host- oder Service-Cluster-Elementen ist einfacher als Sie denken. Eigentlich tun Sie es wahrscheinlich schon. Bei Service-Clustern sollten Sie sicherstellen, dass Sie jedes Service-Element des Clusters überwachen. Wenn Sie ein Cluster aus fünf DNS-Servern haben, dann stellen Sie sicher, dass Sie fünf einzelne Service-Definitionen haben (z.B. mit dem *check_dns*-Plugin). Bei Host-Clustern stellen Sie sicher, dass Sie entsprechende Host-Definitionen für jedes Mitglied des Clusters haben (Sie müssen auch mindestens einen Service auf jedem Host überwachen). **Wichtig:** Sie können die Benachrichtigungen für die einzelnen Cluster-Elemente deaktivieren (Host- oder Service-Definitionen). Obwohl keine Benachrichtigungen für die einzelnen Elemente versandt werden, bekommen Sie trotzdem eine visuelle Anzeige des einzelnen Host- oder Service-Zustands in der Status CGI. Das ist nützlich bei der genauen Erkennung der Quelle von Problemen im Cluster in der Zukunft.

Die Überwachung des gesamten Clusters kann mit Hilfe der bereits im Cache verfügbaren Ergebnisse der Cluster-Elemente erfolgen. Auch wenn Sie alle Elemente des Clusters erneut prüfen könnten, um den Cluster-Status zu ermitteln: warum sollten Sie Bandbreite und Ressourcen vergeuden, wenn bereits die Ergebnisse im Cache vorliegen? Wo werden die Ergebnisse abgelegt? Ergebnisse für Cluster-Elemente sind im Status-File zu finden (vorausgesetzt, dass Sie jedes Element überwachen). Das *check_cluster*-Plugin ist genau für den Zweck ausgelegt, um Host- und Service-Zustände im Status-File zu prüfen. **Wichtig:** Auch wenn Sie Benachrichtigungen für einzelne Elemente des Clusters nicht aktiviert haben, möchten Sie sie vielleicht für den Gesamtstatus des Clusters aktivieren.

Das *check_cluster*-Plugin benutzen

Das check_cluster-Plugin ist dafür ausgelegt, den Gesamtstatus eines Host- oder Service-Clusters durch die Prüfung der Statusinformationen jedes einzelnen Host- oder Service-Cluster-Elements zu ermitteln.

noch mehr... Das *check_cluster*-Plugin finden Sie im contrib-Verzeichnis der Nagios-Plugins unter http://sourceforge.net/projects/nagiosplug/.

Service-Cluster überwachen

Nehmen wir an, dass Sie drei DNS-Server haben, die redundante Dienste in Ihrem Netzwerk bereitstellen. Zuerst müssen Sie jeden einzelnen DNS-Server überwachen, bevor Sie sie als Cluster überwachen können. Ich nehme an, dass Sie bereits drei einzelne Services haben (die alle "DNS Service" heißen), die mit Ihren DNS-Hosts verbunden sind ("host1", "host2" und "host3" genannt).

Um die Services als einen Cluster zu überwachen, müssen Sie einen neuen "Cluster"-Service erstellen. Bevor Sie das tun, sollten Sie ein Service-Cluster-Prüfbefehl konfigurieren. Lassen Sie uns annehmen, dass Sie einen Befehl namens *check_service_cluster* wie folgt definieren:

```
define command{
        command_name    check_service_cluster
        command_line    /usr/local/nagios/libexec/check_cluster --service -l $ARG1$ -w $ARG2$ -c $ARG3$ -d $ARG4$
        }
```

Nun müssen Sie den "Cluster"-Service erstellen und den *check_service_cluster*-Befehl benutzen, den Sie gerade als Cluster-Prüfbefehl erstellt haben. Das folgende Beispiel gibt einen Hinweis, wie das zu tun ist. Es generiert einen CRITICAL-Alarm, wenn zwei oder mehr Services im Cluster in einem nicht-OK-Zustand sind und einen WARNING-Alarm, wenn nur einer der Services in einem nicht-OK-Zustand ist. Wenn jedes der einzelnen Service-Mitglieder des Clusters OK sind, wird auch die Cluster-Prüfung einen OK-Status zurückliefern.

```
define service{
       ...
       check_command   check_service_cluster!"DNS Cluster"!1!2!$SERVICESTATEID:host1:DNS Service$,$SERVICESTATEID:host2:DNS Service$,$SERVICESTATEID:host3:DNS Service$
       ...
       }
```

Es ist wichtig anzumerken, dass wir eine Komma-separierte Liste von *on-demand* Service-Zustands-Makros an das $ARG4$-Makro des Cluster-Prüfbefehls übergeben. Das ist wichtig! Nagios wird diese On-Demand-Makros mit den aktuellen Service-Status-IDs (numerischen Werten statt Zeichenketten) der einzelnen Mitglieder des Clusters füllen.

Host-Cluster überwachen

Host-Cluster zu überwachen ist ziemlich ähnlich zur Überwachung von Service-Clustern. Offenkundig besteht der Hauptunterschied darin, dass Hosts überwacht werden und nicht Services. Um den Status eines Host-Clusters zu überwachen, müssen Sie einen Service definieren, der das *check_cluster*-Plugin benutzt. Der Service sollte *nicht* mit einem der Hosts im Cluster verbunden werden, weil dies Probleme mit Benachrichtigungen für den Cluster erzeugt, wenn der Host "down" geht. Eine gute Idee könnte es sein, den Service mit dem Host zu verbinden, auf dem Nagios läuft. Wenn der Host, auf dem Nagios läuft, "down" geht, dann funktioniert auch Nagios nicht mehr und dann können Sie auch nichts mehr

tun (es sei denn, Sie hätten eine redundante Host-Überwachung eingerichtet)...

Wie auch immer, lassen Sie uns annehmen, dass Sie einen *check_host_cluster*-Befehl wie folgt definiert haben:

```
define command{
        command_name    check_host_cluster
        command_line    /usr/local/nagios/libexec/check_cluster --host -l $ARG1$ -w $ARG2$ -c $ARG3$ -d $ARG4$
        }
```

Sagen wir, dass Sie drei Hosts ("host1", "host2" und "host3" genannt) in Ihrem Host-Cluster haben. Wenn Nagios einen WARNING-Alarm generieren soll, wenn einer der Host im Cluster nicht UP ist bzw. einen CRITICAL-Alarm, wenn zwei oder mehr Hosts nicht UP sind, dann sollte der Service, um das Host-Cluster zu überwachen, ungefähr so aussehen:

```
define service{
       ...
       check_command   check_host_cluster!"Super Host Cluster"!1!2!$HOSTSTATEID:host1$,$HOSTSTATEID:host2$,$HOSTSTATEID:host3$
       ...
       }
```

Es ist wichtig anzumerken, dass wir eine Komma-separierte Liste von *on-demand* Host-Zustands-Makros an das $ARG4$-Makro des Cluster-Prüfbefehls übergeben. Das ist wichtig! Nagios wird diese On-Demand-Makros mit den aktuellen Host-Status-IDs (numerischen Werten statt Zeichenketten) der einzelnen Mitglieder des Clusters füllen.

Das ist es! Nagios wird regelmäßig den Status des Host-Clusters prüfen und Benachrichtigungen an Sie versenden, wenn der Status nicht OK ist (vorausgesetzt, dass Sie Benachrichtigungen für den Service aktiviert haben). Beachten Sie, dass Sie höchstwahrscheinlich die Benachrichtigungen in den einzelnen Host-Definitionen deaktivieren werden, wenn der Host "down" geht. Denken Sie daran, dass Sie der Status der einzelnen Hosts weniger interessiert als der Gesamtstatus des Clusters. Abhängig von Ihrem Netzwerk-Layout und von dem, was Sie erreichen wollen, möchten Sie vielleicht die Benachrichtigungen für UNREACHABLE-Zustände bei den Host-Definitionen aktiviert lassen.

Host- und Service-Abhängigkeiten

Hoch zu: Inhalt

Siehe auch: Vorausschauende Abhängigkeitsprüfungen, Service-Prüfungen, Host-Prüfungen

Einführung

Service- und Hostabhängigkeiten sind ein fortgeschrittenes Feature von Nagios, dass Ihnen die Kontrolle über Hosts und Services erlaubt basierend auf dem Status von einem oder mehreren anderen Hosts oder Services. Ich werde erklären, wie Abhängigkeiten arbeiten, zusammen mit den Unterschieden zwischen Host- und Service-Abhängigkeiten.

Überblick Service-Abhängigkeiten

Es gibt ein paar Dinge, die Sie über Service-Abhängigkeiten wissen sollten:

1. ein Service kann von einem oder mehreren Services abhängig sein
2. ein Service kann von Services abhängig sein, die nicht mit dem gleichen Host verbunden sind
3. Service-Abhängigkeiten werden nicht vererbt (solange es nicht explizit konfiguriert wurde)
4. Service-Abhängigkeiten können benutzt werden, um Service-Prüfungen auszuführen und Service-Benachrichtigungen können unter verschiedenen Umständen unterdrückt werden (OK, WARNING, UNKNOWN und/oder CRITICAL-Zustände)
5. Service-Abhängigkeiten sind ggf. nur während bestimmter Zeitfenster gültig

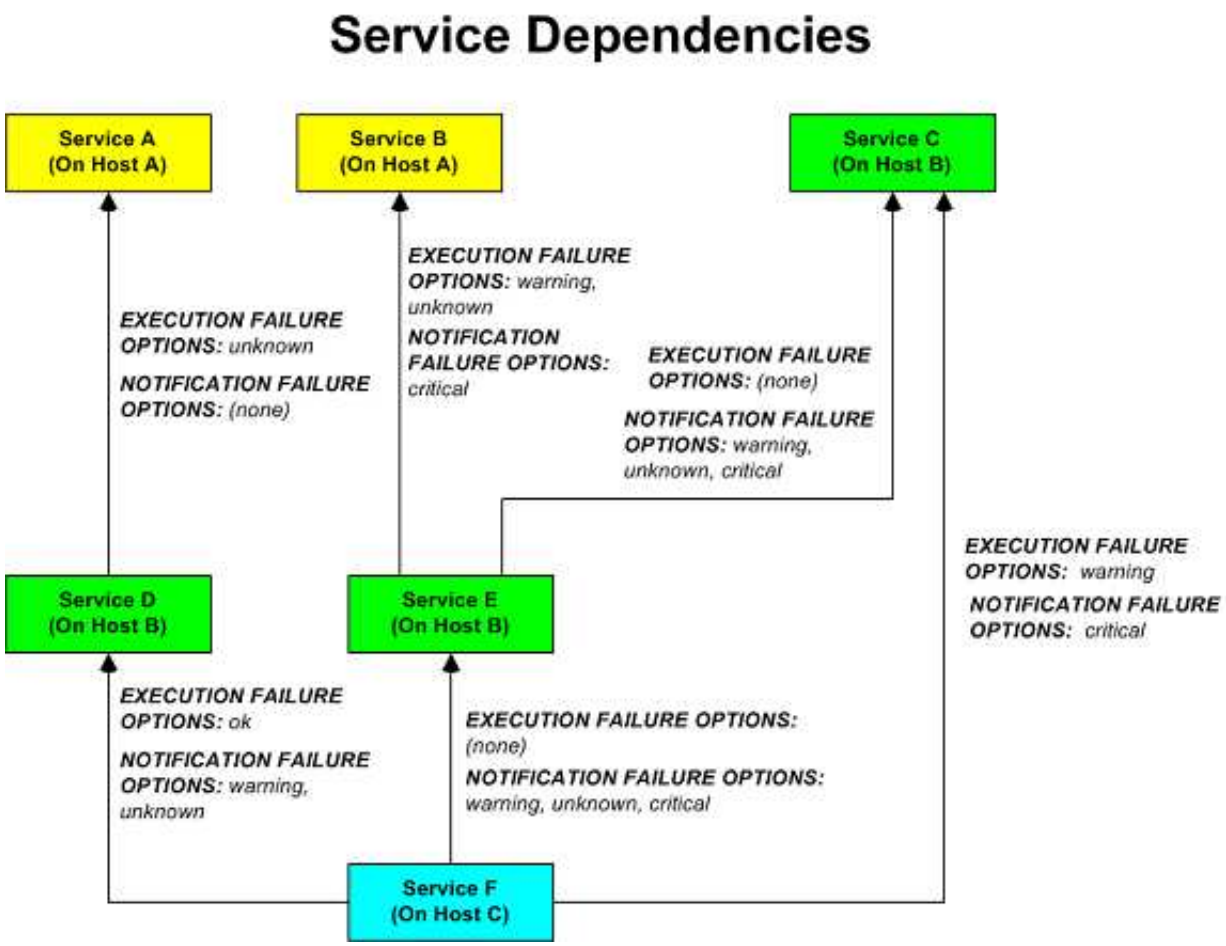
Service-Abhängigkeiten definieren

Zuerst die Grundlagen. Sie erstellen Service-Abhängigkeiten durch Hinzufügen von Service-Abhängigkeitsdefinitionen in der/n Objekt-Konfigurationsdatei(en). In jeder Definition geben Sie den *abhängigen* Service an, den Service, von dem sie *abhängen* und die Kriterien (falls vorhanden), die die Ausführung und Benachrichtungsabhängigkeiten fehlschlagen lassen (diese werden später beschrieben).

Sie können mehrere Abhängigkeiten für einen bestimmten Service erstellen, aber Sie müssen eine eigene Service-Abhängigkeitsdefinition anlegen für jede Abhängigkeit, die Sie erstellen.

Beispiel Service-Abhängigkeiten

Das folgende Bild zeigt ein beispielhaftes Logik-Layout von Service-Benachrichtigungen und Ausführungsabhängigkeiten. Verschiedene Services sind abhängig von anderen Services bzgl. Benachrichtigungen und Prüfausführung.



In diesem Beispiel würde die Abhängigkeitsdefinition für *Service F* auf *Host C* wie folgt aussehen:

```
define servicedependency{
        host_name                       Host B
        service_description             Service D
        dependent_host_name             Host C
        dependent_service_description   Service F
        execution_failure_criteria      o
        notification_failure_criteria   w,u
        }

define servicedependency{
        host_name                       Host B
        service_description             Service E
        dependent_host_name             Host C
        dependent_service_description   Service F
        execution_failure_criteria      n
        notification_failure_criteria   w,u,c
        }

define servicedependency{
        host_name                       Host B
        service_description             Service C
        dependent_host_name             Host C
        dependent_service_description   Service F
        execution_failure_criteria      w
        notification_failure_criteria   c
        }
```

Die anderen im obigen Bild gezeigten Abhängigkeitsdefinitionen würden wie folgt definiert:

```
define servicedependency{
        host_name                       Host A
        service_description             Service A
        dependent_host_name             Host B
        dependent_service_description   Service D
        execution_failure_criteria      u
        notification_failure_criteria   n
        }

define servicedependency{
        host_name                       Host A
        service_description             Service B
        dependent_host_name             Host B
        dependent_service_description   Service E
        execution_failure_criteria      w,u
        notification_failure_criteria   c
        }

define servicedependency{
        host_name                       Host B
        service_description             Service C
        dependent_host_name             Host B
        dependent_service_description   Service E
        execution_failure_criteria      n
        notification_failure_criteria   w,u,c
        }
```

Wie Service-Abhängigkeiten getestet werden

Bevor Nagios eine Service-Prüfung ausführt oder Benachrichtigungen für einen Service versendet, wird es prüfen, ob der Service irgendwelche Abhängigkeiten hat. Wenn es keine Abhängigkeiten gibt, wird die Prüfung ausgeführt oder die Benachrichtigung versandt, wie es sein sollte. Falls der Service ein oder mehrere Abhängigkeiten *hat*, wird Nagios jeden Abhängigkeitseintrag wie folgt prüfen:

1. Nagios erhält den aktuellen Status* des Services, von dem der Eintrag *abhängig* ist.
2. Nagios vergleicht den Status des Services, von dem der Eintrag *abhängig* ist, gegen die Ausführungs- oder Benachrichtigungsfehleroptionen in der Abhängigkeitsdefinition (je nachdem, welche zu dieser Zeit relevant ist).
3. wenn der aktuelle Status des Services, von dem der Eintrag *abhängig* ist, mit einer der Fehleroptionen übereinstimmt, dann wird die Abhängigkeit als fehlerhaft angesehen und Nagios verlässt die Abhängigkeits-Prüfschleife.
4. wenn der aktuelle Status des Services, von dem der Eintrag *abhängig* ist, nicht mit einer der Fehleroptionen übereinstimmt, dann wird die Abhängigkeit als korrekt durchlaufen angesehen und Nagios wird fortfahren und den nächsten Abhängigkeitseintrag prüfen.

Dieser Ablauf wird ausgeführt, bis entweder alle Abhängigkeiten für diesen Service geprüft wurden oder eine Abhängigkeitsprüfung fehlschlägt.

 Anmerkung: *Bitte beachten Sie, dass Nagios per Default den aktuellsten Hard-Status des/r Services benutzt, von dem der Eintrag abhängig ist, wenn es die Abhängigkeitsprüfungen durchführt. Wenn Nagios den aktuellsten Status des/r Services benutzen soll (egal, ob es sich um einen Hard- oder Soft-Zustand handelt), dann aktivieren Sie die soft_state_dependencies-Option.

Ausführungsabhängigkeiten

Ausführungsabhängigkeiten werden benutzt, um einzuschränken, wann aktive Prüfungen eines Service ausgeführt werden können. Passive Prüfungen werden durch Ausführungsabhängigkeiten nicht eingeschränkt.

Wenn *alle* der Ausführungsabhängigkeitstests für den Service *erfolgreich* durchlaufen wurden, wird Nagios die Prüfung für den Service durchführen, wie es das normal tun würde. Wenn auch nur einer der Ausführungsabhängigkeiten für einen Service fehlschlägt, wird Nagios vorübergehend die Ausführung von Prüfungen für diesen (abhängigen) Service verhindern. Irgendwann in der Zukunft können die Ausführungsabhängigkeitstests für den Service erfolgreich durchlaufen werden. Wenn dies geschieht, wird Nagios mit der Prüfung des Service beginnen, wie es das normal tun würde. Mehr Informationen über die Logik der Prüfungsplanung finden Sie hier.

Im obigen Beispiel wären die Tests der Ausführungsabhängigkeiten für **Service E** fehlgeschlagen, wenn **Service B** in einem WARNING- oder UNKNOWN-Zustand ist. Falls dies der Fall ist, würde die Service-Prüfung nicht ausgeführt und die Prüfung würde für eine (mögliche) Ausführung zu einem späteren Zeitpunkt geplant.

Benachrichtigungsabhängigkeiten

Wenn *alle* der Benachrichtigungsabhängigkeitstests für den Service *erfolgreich* durchlaufen wurden, wird Nagios Benachrichtigungen für den Service versenden, wie es das normal tun würde. Wenn auch nur einer der Benachrichtigungsabhängigkeiten für einen Service fehlschlägt, wird Nagios vorübergehend die Benachrichtigungen für diesen (abhängigen) Service unterdrücken. Irgendwann in der Zukunft können die Benachrichtigungsabhängigkeitstests für den Service erfolgreich durchlaufen werden. Wenn dies geschieht, wird Nagios mit dem Versand von Benachrichtigungen für diesen Service beginnen, wie es das normal tun würde. Mehr Informationen über die Benachrichtigungslogik finden Sie hier.

Im obigen Beispiel wären die Tests der Benachrichtigungsabhängigkeiten für **Service F** fehlgeschlagen, wenn **Service C** in einem CRITICAL-Zustand *und/oder* **Service D** in einem WARNING- oder UNKNOWN-Zustand *und/oder* **Service E** in einem WARNING-, UNKNOWN- oder CRITICAL-Zustand ist. Falls dies der Fall ist, würden keine Benachrichtigungen versandt werden.

Abhängigkeitsvererbung

Wie bereits erwähnt werden Service-Abhängigkeiten *nicht* per Default vererbt. Im obigen Beispiel sehen Sie, dass Service F von Service E abhängig ist. Trotzdem erbt er nicht automatisch die Abhängigkeiten von Service E zu Service B und Service C. Um Service F von Service C abhängig zu machen, müssen wir eine weitere Service-Abhängigkeitsdefinition hinzufügen. Es gibt keine Abhängigkeitsdefinition für Service B, also ist Service F *nicht* abhängig von Service B.

Wenn Sie Service-Abhängigkeiten vererbbar machen *wollen*, müssen Sie die *inherits_parent*-Direktive in der Service-Abhängigkeits-Definition benutzen. Wenn diese Direktive aktiviert ist, bedeutet das, dass der Abhängige die Abhängigkeiten des Service erbt, von dem er abhängt (auch als Master-Service bezeichnet). Mit anderen Worten, wenn der Master-Service von anderen Services abhängt und eine von diesen Abhängigkeiten fehlschlägt, wird auch dieser Service fehlschlagen.

Stellen Sie sich für das obige Beispiel vor, Sie möchten eine neue Abhängigkeit für Service F hinzufügen, um ihn von Service A abhängig zu machen. Sie können eine neue Abhängigkeitsdefinition erzeugen, die Service F als den *abhängigen* Service und Service A als den *Master*-Service angibt (d.h. der Service, auf den F *angewiesen* ist). Sie können alternativ die Abhängigkeitsdefinition der Services D und F verändern, die dann wie folgt aussehen:

```
define servicedependency{
        host_name                       Host B
        service_description             Service D
        dependent_host_name             Host C
        dependent_service_description   Service F
        execution_failure_criteria      o
        notification_failure_criteria   n
        inherits_parent                 1
        }
```

Weil die *inherits_parent*-Direktive aktiviert ist, werden die Abhängigkeiten zwischen den Services A und D getestet, wenn die Abhängigkeiten zwischen den Services F und D getestet werden.

Abhängigkeiten können mehrere Vererbungsebenen haben. Wenn bei der Abhängigkeitsdefinition zwischen A und D die *inherits_parent*-Direktive aktiviert ist und Service A von einem anderen Service abhängig ist (sagen wir Service G), dann wäre Service F von den Services D, A und G abhängig (jeder mit möglicherweise unterschiedlichen Kriterien).

Host-Abhängigkeiten

Wie Sie vielleicht erwarten, arbeiten Host-Abhängigkeiten in einer ähnlichen Weise wie Service-Abhängigkeiten. Der Unterschied ist, dass sie für Hosts gelten und nicht für Services.

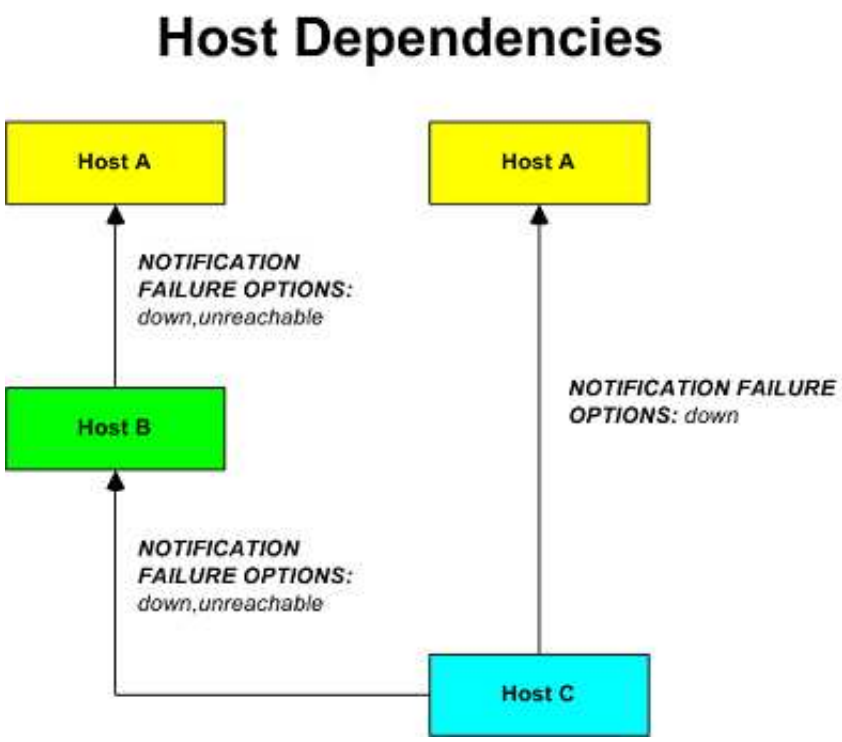
 Hinweis: Verwechseln Sie Host-Abhängigkeiten nicht mit Eltern/Kind-Beziehungen. Sie sollten in den meisten Fällen Eltern/Kind-Beziehungen (mit Hilfe der *parents*-Direktive in den Host-Definitionen) benutzen statt Host-Abhängigkeiten. Eine Beschreibung, wie Eltern/Kind-Beziehungen arbeiten, finden Sie in der Dokumentation zur Netzwerkerreichbarkeit.

Hier die Grundlagen zu Host-Abhängigkeiten:

1. ein Host kann von einem oder mehreren Hosts abhängig sein
2. Host-Abhängigkeiten werden nicht vererbt (solange es nicht explizit konfiguriert wurde)
3. Host-Abhängigkeiten können genutzt werden, um Host-Prüfungen auszuführen und Host-Benachrichtigungen unter bestimmten Umständen zu unterdrücken (UP, DOWN- und/oder UNREACHABLE-Zustände)
4. Host-Abhängigkeiten sind ggf. nur während bestimmter Zeitfenster gültig

Beispiel Host-Abhängigkeiten

Das folgende Bild zeigt ein Beispiel für das logische Layout von Benachrichtigungsabhängigkeiten. Verschiedene Hosts sind bzgl. Benachrichtigungen abhängig von anderen Hosts.



Im obigen Beispiel würden die Abhängigkeitsdefinitionen für *Host C* wie folgt aussehen:

```
define hostdependency{
        host_name                       Host A
        dependent_host_name             Host C
        notification_failure_criteria   d
        }

define hostdependency{
        host_name                       Host B
        dependent_host_name             Host C
        notification_failure_criteria   d,u
        }
```

Wie bei Service-Abhängigkeiten werden Host-Abhängigkeiten nicht vererbt. Im Beispielbild sehen Sie, dass Host C nicht die Host-Abhängigkeiten von Host B erbt. Um Host C von Host A abhängig zu machen, muss eine neue Host-Abhängigkeitsdefinition erstellt werden.

Host-Benachrichtigungsabhängigkeiten arbeiten in einer ähnlichen Weise wie Service-Benachrichtigungsabhängigkeiten. Wenn *alle* der Benachrichtigungsabhängigkeitstests für den Host *erfolgreich* durchlaufen wurden, wird Nagios Benachrichtigungen für den Host versenden, wie es das normal tun würde. Wenn auch nur einer der Benachrichtigungsabhängigkeiten für einen Host fehlschlägt, wird Nagios vorübergehend die Benachrichtigungen für diesen (abhängigen) Host unterdrücken. Irgendwann in der Zukunft können die Benachrichtigungsabhängigkeitstests für den Host erfolgreich durchlaufen werden. Wenn dies geschieht, wird Nagios mit dem Versand von Benachrichtigungen für diesen Host beginnen, wie es das normal tun würde. Mehr Informationen über die Benachrichtigungslogik finden Sie hier.

Nagios®

Statusverfolgung

Hoch zu: Inhalt

Siehe auch: Flüchtige Services

Einführung

Statusverfolgung ("state stalking") ist ein Feature, welches wahrscheinlich von den meisten Benutzern nicht eingesetzt wird. Wenn es aktiviert ist, erlaubt es Ihnen die Protokollierung von Änderungen bei Service- und Host-Prüfungen, selbst wenn sich der Zustand von Host oder Service nicht ändert. Wenn die Verfolgung für einen bestimmten Host oder Service aktiviert ist, wird Nagios den Host oder Service sehr genau beobachten und jede Änderung protokollieren, die es in der Ausgabe der Prüfergebnisse sieht. Wie Sie sehen werden, kann es sehr hilfreich bei der späteren Analyse der Log-Files sein.

Wie funktioniert es?

Unter normalen Umständen wird das Ergebnis einer Host- oder Service-Prüfung nur protokolliert, wenn der Host oder Service seit der letzten Prüfung seinen Zustand geändert hat. Es gibt wenige Ausnahmen, aber normalerweise ist das die Regel.

Wenn Sie die Verfolgung für einen oder mehrere Zustände eines bestimmten Hosts oder Service aktivieren, wird Nagios die Ergebnisse der Host- oder Service-Prüfung protokollieren, wenn sich die Ausgaben der Prüfung von den Ausgaben der letzten Prüfung unterscheiden. Nehmen Sie das folgende Beispiel von acht aufeinander folgenden Prüfungen eines Service:

Service Check #:	Service State:	Service Check Output:	Logged Normally	Logged With Stalking
x	OK	RAID array optimal	-	-
x+1	OK	RAID array optimal	-	-
x+2	WARNING	RAID array degraded (1 drive bad, 1 hot spare rebuilding)	✓	✓
x+3	CRITICAL	RAID array degraded (2 drives bad, 1 host spare online, 1 hot spare rebuilding)	✓	✓
x+4	CRITICAL	RAID array degraded (3 drives bad, 2 hot spares online)	-	✓
x+5	CRITICAL	RAID array failed	-	✓
x+6	CRITICAL	RAID array failed	-	-
x+7	CRITICAL	RAID array failed	-	-

Bei dieser Reihenfolge von Prüfungen würden Sie normalerweise nur zwei Einträge dieser Katastrophe sehen. Der erste würde bei Prüfung x+2 auftreten, wenn der Service von einem OK- in einen WARNING-Zustand wechselt. Der zweite Log-Eintrag würde bei Service-Prüfung x+3 auftreten, wenn

der Service von einem WARNING- in einen CRITICAL-Zustand wechselt.

Aus welchem Grund auch immer möchten Sie die komplette Geschichte dieser Katastrophe in Ihren Log-Dateien sehen. Vielleicht, um Ihrem Manager zu zeigen, wie schnell die Situation außer Kontrolle geriet, vielleicht nur, um bei ein paar Bier in der Kneipe darüber zu lachen...

Wenn Sie die Verfolgung dieses Services für CRITICAL-Zustände aktiviert haben, hätten Sie zusätzlich zu den Ereignissen x+2 und x+3 auch noch x+4 und x+5 protokolliert. Warum ist das so? Mit aktivierter Verfolgung hätte Nagios die Ausgaben jeder Service-Prüfung untersucht, um Veränderungen zur Ausgabe der vorherigen Prüfung festzustellen. Wenn sich die Ausgaben unterscheiden und sich der Zustand des Service zwischen den beiden Prüfungen nicht verändert hat, würde die Ausgabe der neueren Prüfung protokolliert.

Ein ähnliches Beispiel für die Verfolgung könnte eines Service sein, der Ihren Web-Server prüft. Wenn das check_http-Plugin das erste Mal einen WARNING-Zustand wegen eines 404-Fehlers zurückliefert und bei folgenden Prüfungen einen WARNING-Zustand wegen eines nicht gefundenen Musters zurückliefert, dann möchten Sie das vielleicht wissen. Wenn Sie die Statusverfolgung für WARNING-Zustände nicht aktiviert haben, würde nur das erste WARNING-Ereignis (der 404-Fehler) protokolliert und Sie hätten keine Ahnung (beim Blick auf archivierte Protokolle), dass weitere WARNING-Zustände nicht auf dem 404-Fehler zurückzuführen sind, sondern dass bestimmte Textmuster nicht in der untersuchten Web-Seite zu finden waren.

Sollte ich die Verfolgung aktivieren?

Zuerst müssen Sie entscheiden, ob Sie wirklich Bedarf zur Untersuchung archivierter Protokolldaten haben, um die genaue Ursache eines Problems zu finden. Sie können entscheiden, dass Sie dieses Feature für ein paar Hosts oder Services brauchen, aber nicht für alle. Sie können auch feststellen, dass Sie die Verfolgung nur für einige Host- oder Service-Zustände brauchen, statt für alle. Sie könnten z.B. entscheiden, dass Sie die Verfolgung nur für die WARNING- und CRITICAL-Zustände eines Service benötigen, aber nicht OK- und UNKNOWN-Zustände.

Die Entscheidung, die Verfolgung für einen bestimmten Host oder Service zu aktivieren, hängt auch vom Plugin ab, das Sie zur Prüfung des Hosts oder Service einsetzen.

Wie aktivieren ich die Verfolgung?

Sie können die Verfolgung für Hosts und Services durch die *stalking_options*-Direktive in den Host- und Service-Definitionen aktivieren.

Wie unterschieden sich Verfolgung und "flüchtige Services"?

Flüchtige Services (volatile services) sind ähnlich, aber sie verursachen Benachrichtigungen und führen Eventhandler aus. Die Verfolgung dient lediglich der Protokollierung.

Risiken

Sie sollten beachten, dass es einige potenzielle Fallstricke bei der Aktivierung von Verfolgungen gibt. Sie beziehen sich alle auf die Berichtsfunktionen, die in verschiedenen CGIs zu finden sind (Histogramm, Alarmübersicht, usw.). Weil die Statusverfolgung zusätzliche Alarmeinträge schreibt, werden die Berichte eine erhöhte Anzahl von Alarmen anzeigen.

Als generelle Regel würde ich empfehlen, dass Sie die Verfolgung für Hosts und Services *nicht* aktivieren, ohne gründlich darüber nachgedacht zu haben. Auf jeden Fall ist sie da, wenn Sie sie brauchen.

Performance-Daten

Hoch zu: Inhalt

Siehe auch: Plugins, Plugin API

Einführung

Nagios ist ausgelegt, dass Plugins neben den normalen Statusinformationen optional auch Performance-Daten zurückliefern können, die Sie außerdem zur weiteren Verarbeitung an externe Applikationen übergeben können. Eine Beschreibung der verschiedenen Typen von Performance-Daten wie auch Informationen darüber, wie diese Daten verarbeitet werden, finden Sie im Folgenden...

Typen von Performance-Daten

Es gibt zwei grundlegende Kategorien von Performance-Daten, die von Nagios erhalten werden können:

1. Prüf-Performance-Daten
2. Plugin-Performance-Daten

Prüf-Performance-Daten sind interne Daten, die sich auf die aktuelle Ausführung einer Host- oder Service-Prüfung beziehen. Dies kann Dinge wie die Service-Prüfverzögerung enthalten (service check latency, d.h., wie viel Zeit von der geplanten Ausführung bis zu eigentlichen Ausführung verging) oder die Anzahl der Sekunden, die die Ausführung einer Host- oder Service-Prüfung dauerte. Dieser Typ von Performance-Daten ist für alle ausgeführten Prüfungen verfügbar. Die $HOSTEXECUTIONTIME$- und $SERVICEEXECUTIONTIME$-Makros können benutzt werden, um die Anzahl der Sekunden zu ermitteln, die eine Host- oder Service-Prüfung dauerte und die $HOSTLATENCY$- und $SERVICELATENCY$-Makros können zur Ermittlung der "Verspätung" einer regulär geplanten Host- oder Service-Prüfung genutzt werden.

Plugin-Performance-Daten sind externe Daten, die spezifisch für das Plugin sind, das die Host- oder Service-Prüfung ausführt. Plugin-spezifische Daten können Dinge wie Prozentsatz des Paketverlustes, freie Plattenplatz, Prozessor-Load, Anzahl der gegenwärtigen Benutzer usw. umfassen - generell jede Art von Metrik, die das Plugin misst, wenn es ausgeführt wird. Plugin-spezifische Performance-Daten sind optional und werden ggf. nicht von allen Plugins unterstützt. Plugin-spezifische Performance-Daten (falls verfügbar) werden durch die $HOSTPERFDATA$- und $SERVICEPERFDATA$-Makros bereit gestellt. Lesen Sie weiter, um mehr Informationen darüber zu erhalten, wie Plugins Performance-Daten an Nagios zur Bereitstellung durch die $HOSTPERFDATA$- und $SERVICEPERFDATA$-Makros zurückliefern können.

Plugin-Performance-Daten

Als Minimum müssen Nagios-Plugins eine einzelne Zeile mit menschlich lesbarem Text zurückliefern, die den Status eines Typs von Messdaten enthält. Zum Beispiel könnte das check_ping-Plugin eine Textzeile wie die folgende zurückliefern:

```
PING ok - Packet loss = 0%, RTA = 0.80 ms
```

Bei dieser einfachen Art von Ausgabe ist die gesamte Textzeile in den $HOSTOUTPUT$- oder $SERVICEOUTPUT$-Makros verfügbar (abhängig davon, ob dieses Plugin als Host- oder Service-Prüfung benutzt wurde).

Plugins können in ihrer Ausgabe optionale Performance-Daten zurückliefern, indem nach dem normalen, menschlich lesbaren Text ein Pipe-Symbol (|) folgt und danach eine Zeichenkette, die ein oder mehrere Performance-Daten-Metriken enthält. Lassen Sie uns das check_ping-Plugin als Beispiel nehmen und annehmen, dass es um die Ausgabe von Performance-Daten-Metriken für den Prozentsatz von Paketverlusten (percent paket loss) und durchschnittlicher Umlaufzeit (average round trip time) erweitert wurde. Die Beispielausgabe des Plugins könnte wie folgt aussehen:

```
PING ok - Packet loss = 0%, RTA = 0.80 ms | percent_packet_loss=0, rta=0.80
```

wenn Nagios dieses Plugin-Ausgabeformat sieht, wird es die Ausgabe in zwei Teile aufteilen:

1. alles vor dem Pipe-Symbol wird als "normale" Ausgabe des Plugins angesehen und im $HOSTOUTPUT$- oder $SERVICEOUTPUT$-Makro gespeichert
2. alles nach dem Pipe-Symbol wird als Plugin-spezifische Ausgabe angesehen und in den $HOSTPERFDATA$- oder $SERVICEPERFDATA$-Makros gespeichert.

Im obigen Beispiel würde das $HOSTOUTPUT$- oder das $SERVICEOUTPUT$-Makro "*PING ok - Packet loss = 0%, RTA = 0.80 ms*" enthalten (ohne Anführungszeichen) und das $HOSTPERFDATA$- oder das $SERVICEPERFDATA$-Makro würde "*percent_packet_loss=0, rta=0.80*" enthalten (ohne Anführungszeichen).

Nagios kann mehrere Zeilen Performance-Daten (ebenso wie normale Textausgaben) von Plugins entgegennehmen, wie in der plugin API documentation beschrieben.

 Anmerkung: der Nagios-Daemon verarbeitet Plugin-Performance-Daten nicht direkt, so dass es ihm egal ist, wie die Performance-Daten aussehen. Es gibt daher eigentlich keine Beschränkungen des Formats oder des Inhalts der Performance-Daten. Wenn Sie allerdings ein externes Addon benutzen, um die Performance-Daten zu verarbeiten (z.B. PerfParse), erwartet das Addon die Performance-Daten möglicher Weise in einem bestimmten Format. Prüfen Sie die Dokumentation des Addon auf weitere Informationen.

Performance-Daten verarbeiten

Wenn Sie die Performance-Daten, die von den Plugins und in Nagios verfügbar sind, müssen Sie folgendes tun:

1. aktivieren Sie die process_performance_data-Option.
2. konfigurieren Sie Nagios so, dass Performance-Daten in Dateien geschrieben und/oder durch Befehle verarbeitet wird.

Lesen Sie weiter, um Informationen darüber zu erhalten, wie Performance-Daten durch das Schreiben in Dateien oder die Ausführung von Befehlen verarbeitet werden.

Performance-Daten verarbeiten durch Befehle

Der flexibelste Weg, um Performance-Daten zu verarbeiten, besteht darin, Nagios Befehle ausführen zu lassen (die Sie angeben), um die Daten zu verarbeiten oder sie umzulenken, damit sie später von externen Applikationen verarbeiten werden. Die Befehle, die Nagios ausführt, um Host- und Service-Performance-Daten zu verarbeiten, werden durch die host_perfdata_command- und service_perfdata_command-Optionen festgelegt.

Eine Beispiel-Befehlsdefinition, die Service-Prüf-Performance-Daten zur späteren Verarbeitung durch eine andere Applikation in eine Textdatei umleitet, finden Sie nachfolgend:

```
define command{
        command_name    store-service-perfdata
        command_line    /bin/echo -e "$LASTSERVICECHECK$\t$HOSTNAME$\t$SERVICEDESC$\t$SERVICESTATE$\t$SERVICEATTEMPT$\t$SERVICESTATETYPE$\t$SERVICEEXECUTIONTIME$\t$SERVICELATENCY$\t$SERVICEOUTPUT$\t$S...
        }
```

Hinweis: Diese Methode, obwohl flexibel, erzeugt einen relativ hohen CPU-Overhead. Wenn Sie Performance-Daten für viele Hosts und Services verarbeiten, dann ist es vielleicht besser, diese Daten in eine Datei zu schreiben. Diese Methode wird im nächsten Abschnitt beschrieben.

Performance-Daten in Dateien schreiben

Sie können Nagios mit Hilfe der host_perfdata_file- und service_perfdata_file-Optionen anweisen, die Host- und Service-Performance-Daten direkt in Textdateien auszugeben. Das Format, in dem Host- und Service-Performance-Daten in diese Dateien geschrieben wird, wird durch die host_perfdata_file_template- und service_perfdata_file_template-Optionen festgelegt.

Eine Beispiel-Dateiformatvorlage für Performance-Daten könnte wie folgt aussehen:

```
service_perfdata_file_template=[SERVICEPERFDATA]\t$TIMET$\t$HOSTNAME$\t$SERVICEDESC$\t$SERVICEEXECUTIONTIME$\t$SERVICELATENCY$\t$SERVICEOUTPUT$\t$SERVICEPERFDATA$
```

Per Default werden die Textdateien im "append"-Modus ("anhängen") eröffnet. Wenn Sie den Modus auf "write" ("schreiben") oder "non-blocking read/write" ("nicht-blockierendes Lesen/Schreiben", nützlich beim Schreiben in Pipes) ändern, können Sie die host_perfdata_file_mode- und service_perfdata_file_mode-Optionen nutzen.

Zusätzlich können Sie Nagios mit den host_perfdata_file_processing_command- und service_perfdata_file_processing_command-Optionen anweisen, periodisch Befehle auszuführen, um regelmäßig die Performance-Daten-Dateien zu verarbeiten (z.B., um sie zu rotieren). Das Intervall, in dem diese Befehle ausgeführt werden, ist durch die host_perfdata_file_processing_interval- und service_perfdata_file_processing_interval-Optionen festgelegt.

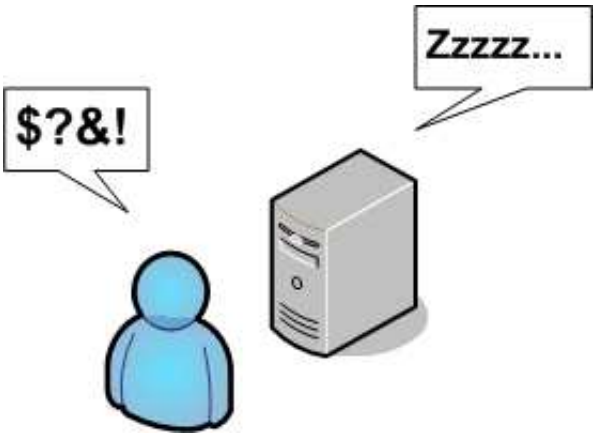
Geplante Ausfallzeit

Hoch zu: Inhalt

Siehe auch: Benachrichtigungen

Einführung

Nagios erlaubt Ihnen, Termine für geplante Ausfallzeiten (downtime) von Hosts und Services zu vergeben, die Sie überwachen. Das ist nützlich, wenn Sie bereits wissen, dass Sie einen Server für einen Upgrade oder etwas Ähnliches herunterfahren müssen.



Ausfallzeit einplanen

Sie können eine Ausfallzeit für Hosts und Services über das extinfo CGI einplanen (wenn Sie Host- oder Service-Informationen ansehen). Klicken Sie auf den Link "Schedule downtime for this host/service", um die Ausfallzeit zu planen.

Sobald Sie die Ausfallzeit für einen Host oder Service einplanen, wird Nagios für diesen Host oder Service einen Kommentar hinzufügen, der anzeigt, dass für diese Periode eine Ausfallzeit geplant ist. Wenn die Zeit vorüber ist, wird Nagios diesen Kommentar automatisch löschen. Cool, oder?

Feste und flexible Ausfallzeiten

Wenn Sie über das Web-Interface eine Ausfallzeit einplanen, werden Sie gefragt, ob sie fest oder flexibel sein soll. Hier eine Erklärung, wie sich "fest" und "flexibel" unterscheiden:

"Feste" Ausfallzeiten starten und stoppen genau zu den Zeiten, die Sie bei der Planung festgelegt haben. Okay, das war einfach genug...

"Flexible" Ausfallzeiten sind gedacht für Zeiten, wenn Sie wissen, dass ein Host oder Service für X Minuten (oder Stunden) nicht verfügbar sein wird, aber Sie nicht genau wissen, wann das sein wird. Wenn Sie flexible Ausfallzeiten planen, wird Nagios die geplante Ausfallzeit irgendwann zwischen den Start- und Endzeiten beginnen, die Sie angegeben haben. Die Ausfallzeit wird solange dauern, wie Sie das bei der Planung angegeben haben. Dabei wird angenommen, dass der Host oder Service, für den Sie eine flexible Ausfallzeit geplant haben, ausfällt (oder unerreichbar wird) oder zwischen der angegebenen Start- und Endezeit in einen nicht-OK-Zustand wechselt. Die Zeit, zu der der Host oder

Service in einen Problemzustand wechselt, legt die Zeit fest, zu der Nagios tatsächlich die Ausfallzeit startet. Die Ausfallzeit wird die angegebene Zeitspanne dauern, auch wenn sich der Host oder Service vor der definierten Zeit erholt. Das wird aus gutem Grund getan. Wie wir alle wissen, denken Sie vielleicht, dass Sie ein Problem gelöst haben, aber müssen den Server doch noch zehnmal neu starten, bevor es wirklich funktioniert. Geschickt, oder?

ausgelöste Ausfallzeiten

Während des Planens von Host- oder Service-Ausfallzeiten haben Sie die Möglichkeit, sie zu "ausgelösten" Ausfallzeiten (triggered downtime) zu machen. Was ist eine ausgelöste Ausfallzeit, fragen Sie? Bei ausgelösten Ausfallzeiten wird der Start der Ausfallzeit durch den Start einer anderen geplanten Host- oder Service-Ausfallzeit ausgelöst. Dies ist sehr nützlich, wenn Sie Ausfallzeiten für eine große Zahl von Hosts oder Services planen und die Startzeit der Auszeit von der Startzeit eines anderen Ausfallzeiteintrags abhängt. Wenn Sie zum Beispiel eine flexible Ausfallzeit für einen bestimmten Host planen (weil er zur Wartung heruntergefahren wird), könnten Sie ausgelöste Ausfallzeiten für alle "Kinder" des Hosts planen.

Wie geplante Ausfallzeiten Benachrichtigungen beeinflussen

Wenn sich ein Host oder Service in einer Phase geplanter Ausfallzeit befindet, wird Nagios keine normalen Benachrichtigungen für den Host oder Service versenden. Allerdings wird es eine "DOWNTIMESTART"-Benachrichtigung für den Host oder Service versenden, die jeden Admin darüber informiert, dass sie nachfolgend keine Problemalarme erhalten werden.

Wenn die geplante Ausfallzeit vorbei ist, wird Nagios wieder normale Benachrichtigungen für den Host oder Service versenden. Eine "DOWNTIMEEND"-Benachrichtigung wird an die Admins versandt, dass die geplante Ausfallzeit vorüber ist und dass sie wieder normale Alarme erhalten werden.

Wenn die geplante Auszeit vorzeitig abgebrochen wird (bevor sie endet), wird eine "DOWNTIMECANCELLED"-Benachrichtigung an die betroffenen Admins versandt.

Überlappende geplante Ausfallzeiten

Ich mag es, dieses als das "Oh Mist, es funktioniert nicht"-Syndrom zu bezeichnen. Sie wissen, wovon ich spreche. Sie fahren einen Server herunter, um einen "Routine"-Hardware-Upgrade zu machen, nur um später festzustellen, dass die OS-Treiber nicht funktionieren, das RAID-Array hochgegangen ist oder Laufwerkskopien fehlgeschlagen und Ihre Original-Platten jetzt nutzlos sind. Moral der Geschichte ist, dass jede Routinearbeit an einem Server durchaus drei- oder viermal länger dauern kann, als Sie ursprünglich geplant haben...

Nehmen wir das folgende Szenario:

1. Sie planen eine Auszeit für Host A an einem Montag von 19:30 Uhr bis 21:30 Uhr
2. Sie fahren den Server am Montag gegen 19:45 Uhr herunter, um einen Platten-Upgrade durchzuführen
3. nachdem Sie eineinhalb Stunden mit SCSI-Fehlern und Treiberinkompatibilitäten verschwendet haben, können Sie endlich den Server starten
4. um 21:15 Uhr stellen Sie fest, dass eine Ihrer Partitions nirgends auf der Platte zu finden ist
5. da Sie wissen, dass es eine lange Nacht wird, gehen Sie zurück und planen eine zusätzliche Auszeit für Host A von Montag 21:20 Uhr bis Dienstagmorgen 1:30 Uhr

Wenn Sie überlappende Ausfallzeiten für einen Host oder Service planen (in diesem Fall waren die Zeiten von 19:40 Uhr bis 21:30 Uhr und 21:20 bis 1:30 Uhr), wird Nagios warten, bis die letzte Periode geplanter Ausfallzeiten vorüber ist, bevor Benachrichtigungen zu diesem Host oder Service versandt werden. In diesem Beispiel werden Benachrichtigungen für Host A bis Dienstagmorgen 1:30 Uhr

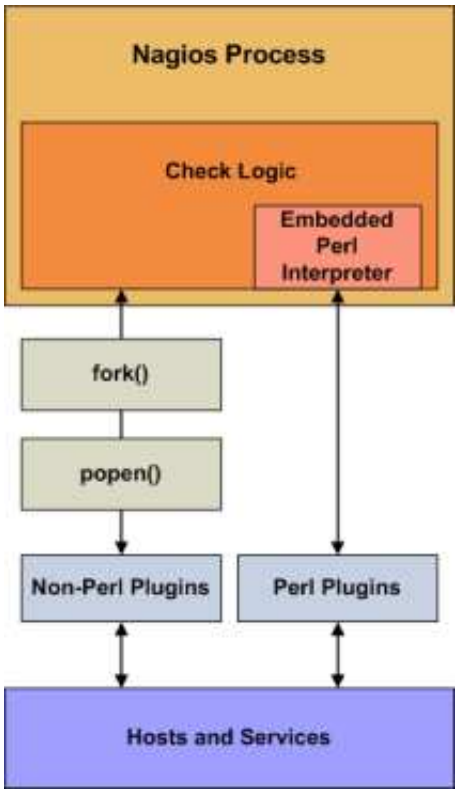
unterdrückt.

Benutzen des Embedded Perl Interpreters

Hoch zu: Inhalt

Siehe auch: Entwicklung von Plugins für die Nutzung mit Embedded Perl

Einführung



Nagios kann für die Unterstützung eines eingebetteten Perl-Interpreters (embedded perl interpreter) kompiliert werden. Dies erlaubt es Nagios, Perl-Plugins effizienter als sonst auszuführen, also mag es interessant sein, wenn Sie sich viel auf Plugins verlassen, die in Perl geschrieben sind.

Ohne den eingebetteten Perl-Interpreter führt Nagios Perl- (und andere) Plugins durch "forking" und ausführen als einen externen Befehl aus. Wenn der eingebettete Perl-Interpreter benutzt wird, kann Nagios Perl-Plugins durch einen einfachen Library-Call ausführen.

Hinweis: Der Perl-Interpreter arbeitet mit allen Perl-Scripten, die Nagios ausführt - nicht nur Plugins. Diese Dokumentation behandelt den eingebetteten Perl-Interpreter in Verbindung mit Plugins für Host- und Service-Prüfungen, aber sie trifft genauso auf andere Arten von Perl-Scripten zu, die Sie vielleicht für andere Arten von Befehlen benutzen (z.B. Benachrichtigungs-Scripte, Eventhandler-Scripte usw.).

Stephen Davis hat den originalen eingebetteten Perl-Interpreter-Code vor einigen Jahren beigetragen. Stanley Hopcroft war die erste Person, die geholfen hat, den eingebetteten Perl-Interpreter-Code zu verbessern und die die Vor- und Nachteile bei der Benutzung kommentiert hat. Er hat auch verschiedene hilfreiche Hinweise zur Erstellung von Perl-Plugins gegeben, die sauber mit dem eingebetteten Interpreter arbeiten.

Es sollte angemerkt werden, dass sich "ePN" - wie in dieser Dokumentation benutzt - auf "embedded Perl Nagios" bezieht, oder wenn Sie das bevorzugen, Nagios mit eingebettetem Perl-Interpreter.

Vorteile

Einige Vorteile von ePN (embedded Perl Nagios) umfassen:

- Nagios wird viel weniger Zeit bei der Ausführung Ihrer Perl-Plugins verbringen, weil es nicht länger "fork"t, um das Plugin auszuführen (das Laden des Perl-Interpreters entfällt). Statt dessen führt es das Plugin durch einen Library-Call durch.
- es reduziert die Systembelastung durch Perl-Plugins und/oder erlaubt es Ihnen, mehr Prüfungen mit Perl-Plugins durchzuführen, als Ihnen das sonst möglich wäre. Mit anderen Worten haben Sie weniger Anreiz, Plugins in anderen Sprachen wie z.B. C/C++, oder Expect/TCL zu schreiben, die bei den Entwicklungszeiten eine Zehnerpotenz langsamer angesehen werden als Perl (wobei sie auch zehnmal schneller ablaufen, von TCL mal abgesehen).
- Wenn Sie kein C-Programmierer sind, dann können Sie trotzdem eine Menge mit Perl erledigen, ohne dass es Nagios viel langsamer macht. Beachten Sie, dass ePN Ihre Plugins nicht schneller macht (außer, dass es die Ladezeit eliminiert). Wenn Sie schnelle Plugins wollen, dann berücksichtigen Sie Perl XSUBs (XS) oder C, *nachdem* Sie sicher sind, dass Sie Ihr Perl getuned haben und dass Sie einen angemessenen Algorithmus haben (Benchmark.pm is *unbezahlbar* für den Vergleich von Perl-Sprachelementen).
- ePN zu benutzen ist eine exzellente Gelegenheit, mehr über Perl zu lernen.

Nachteile

Die Nachteile von ePN (embedded Perl Nagios) sind ziemlich die gleichen wie bei Apache mod_perl (d.h. Apache mit einem eingebetteten Interpreter) verglichen mit einem schlichten Apache:

- Ein Perl-Programm, das *wunderbar* mit schlichtem Nagios arbeitet, muss *nicht* mit ePN funktionieren. Möglicherweise müssen Sie Ihre Plugins modifizieren, damit sie funktionieren.
- Perl-Plugins sind unter ePN schwieriger zu debuggen als unter schlichtem Nagios.
- Ihr ePN wird eine größere SIZE (Speichernutzung) haben als ein schlichtes Nagios.
- Einige Perl-Konstrukte können nicht genutzt werden oder mögen sich anders verhalten als Sie das erwarten würden.
- Sie sollte sich bewusst sein, dass es 'mehr als einen Weg gibt, es zu tun' und ggf. einen Weg wählen, der weniger attraktiv oder offensichtlich ist.
- Sie werden mehr Perl-Know-How benötigen (aber nichts sehr esoterisches oder Zeug über Perl-Interna - außer, wenn Ihre Plugins XSUBS benutzen).

Benutzung des eingebetteten Perl-Interpreters

Wenn Sie den eingebetteten Perl-Interpreter benutzen wollen, um Ihre Perl-Plugins und Scripts auszuführen, dann lesen Sie hier, was Sie tun müssen:

1. Kompilieren Sie Nagios mit Unterstützung für den eingebetteten Perl-Interpreter (Anweisungen s.u.).
2. aktivieren Sie die enable_embedded_perl-Option in der Hauptkonfigurationsdatei.
3. setzen Sie die use_embedded_perl_implicitly-Option entsprechend Ihren Anforderungen. Diese Option legt fest, ob der Perl-Interpreter per Default für einzelne Perl-Plugins und Scripts benutzt werden sollte.
4. Optional sollten Sie bei verschiedenen Perl-Plugins und Scripts die Ausführung durch den eingebetteten Perl-Interpreter aktivieren oder deaktivieren. Das kann nützlich sein, wenn bestimmte Perl-Scripte Probleme bei der Ausführung mit dem Perl-Interpreter haben. Beachten Sie die Anweisungen weiter unten für mehr Informationen, wie das zu tun ist.

Nagios mit eingebettetem Perl kompilieren

Wenn Sie den eingebetteten Perl-Interpreter benutzen möchten, müssen Sie zuerst Nagios mit der Unterstützung dafür kompilieren. Um dies zu tun, starten Sie einfach das configure-Script zusätzlich mit der *--enable-embedded-perl*-Option. Wenn Sie aktivieren wollen, dass der Perl-Interpreter intern kompilierte Scripts in einem Cache ablegen soll, dann nutzen Sie die *--with-perlcache*-Option. Beispiel:

```
./configure --enable-embedded-perl --with-perlcache otheroptions...
```

Sobald Sie das configure-Script mit den neuen Optionen ausgeführt haben, müssen Sie Nagios erneut kompilieren.

Plugin-spezifische Benutzung des Perl-Interpreters

Beginnend mit Nagios 3 könenn Sie angeben, welche Perl-Plugins oder Scripts mit dem eingebetteten Perl-Interpreter ablaufen sollen und welche nicht. Das ist besonders dann nützlich, wenn Sie Perl-Scripte haben, die nicht sauber mit dem Perl-Interpreter laufen.

Um Nagios *explizit* mitzuteilen, ob der Perl-Interpreter benutzt werden soll oder nicht, fügen Sie Ihrem Perl-Script/Plugin einen der folgenden Einträge hinzu...

Um Nagios mitzuteilen, den Perl-Interpreter für ein bestimmtes Script zu nutzen, fügen Sie dem Perl-Script diese Zeile hinzu:

```
# nagios: +epn
```

Um Nagios mitzuteilen, den Perl-Interpreter für ein bestimmtes Script NICHT zu nutzen, fügen Sie dem Perl-Script diese Zeile hinzu:

```
# nagios: -epn
```

Eine der beiden Zeilen muss innerhalb der ersten zehn Zeilen stehen, damit sie von Nagios erkannt wird.

 Hinweis: Wenn Sie nicht *explizit* die oben genannte Methode nutzen, um Nagios mitzuteilen, den Perl-Interpreter für ein einzelnes Plugin zu nutzen, wird Nagios eine Entscheidung für Sie treffen. Dieser Entscheidungsprozess wird von der use_embedded_perl_implicitly-Variable kontrolliert. Wenn der Wert auf 1 gesetzt ist, werden alle Perl-Plugins/Scripts (bei denen nicht explizit der ePN aktiviert/deaktiviert ist) mit dem Perl-Interpreter ausgeführt. Wenn der Wert auf 0 gesetzt ist, werden sie NICHT mit dem Perl-Interpreter ausgeführt.

Plugins für die Nutzung mit Embedded Perl entwickeln

Informationen über die Entwicklung von Plugins zur Nutzung mit dem eingebetteten Perl-Interpreter finden Sie hier.

Adaptive Überwachung

Hoch zu: Inhalt

Siehe auch: Externe Befehle

Einführung

Nagios erlaubt Ihnen, verschiedene Befehle und Host- und Service-Prüfattribute während der Laufzeit zu verändern. Ich bezeichne diese Möglichkeit als "adaptive Überwachung". Bitte beachten Sie, dass diese adaptiven Überwachungs-Features in Nagios wahrscheinlich von 99% aller Benutzer nicht genutzt werden, aber sie erlauben Ihnen einige nette Dinge.

Was kann verändert werden?

Die folgenden Service-Prüfattribute können während der Laufzeit verändert werden:

- Prüfbefehl (und Befehlsparameter)
- Prüfintervall
- max. Prüfversuche
- Prüfzeitfenster
- Eventhandler-Befehl (und Befehlsparameter)

Die folgenden Host-Prüfattribute können während der Laufzeit verändert werden:

- Prüfbefehl (und Befehlsparameter)
- Prüfintervall
- max. Prüfversuche
- Prüfzeitfenster
- Eventhandler-Befehl (und Befehlsparameter)

Die folgenden globalen Attribute können während der Laufzeit verändert werden:

- Globaler Host-Eventhandler-Befehl (und Befehlsparameter)
- Globaler Service-Eventhandler-Befehl (und Befehlsparameter)

Externe Befehle für adaptive Überwachung

Um globale oder Host- bzw. Service-spezifische Attribute während der Laufzeit zu ändern, müssen Sie über das external command file den entsprechenden externen Befehl an Nagios senden. Die folgende Tabelle zeigt die verschiedenen Attribute, die während der Laufzeit verändert werden könen, zusammen mit dem externen Befehl, um das zu erreichen.

Eine vollständige Liste von externen Befehlen, die zur adaptiven Überwachung benutzt werden können (zusammen mit Beispielen, wie sie genutzt werden können), finden Sie online unter: http://www.nagios.org/developerinfo/externalcommands/.

 Anmerkungen:

- Bei der Änderung von Prüfbefehlen, Prüfzeitfenstern oder Eventhandler-Befehlen ist es wichtig anzumerken, dass die neuen Werte für diese Optionen vor dem Neustart von Nagios definiert werden müssen. Jede Anfrage, die einen Befehl oder ein Zeitfenster auf einen Wert ändert, der beim Start nicht definiert war, wird ignoriert.
- Sie können Befehlsparameter zusammen mit dem tatsächlichen Befehlsnamen angeben - trennen Sie einfach die einzelnen Parameter vom Befehlsnamen (und voneinander) durch Ausrufezeichen (!). Mehr Informationen, wie Parameter in Befehlsdefinitionen während der Laufzeit verarbeitet werden, finden Sie in der Dokumentation zu Makros.

Vorausschauende Abhängigkeitsprüfungen

Hoch zu: Inhalt

Siehe auch: Abhängigkeiten, Service-Prüfungen, Host-Prüfungen, Cached Checks

Einführung

Host- und Service-Abhängigkeiten können definiert werden, um Ihnen größere Kontrolle darüber zu geben, wann Prüfungen ausgeführt und wann Benachrichtigungen versandt werden. Da Abhängigkeiten benutzt werden, um grundlegende Aspekte des Überwachungsprozesses zu kontrollieren, ist es wichtig sicherzustellen, dass die Status-Informationen in der Abhängigkeitslogik so aktuell wie möglich sind.

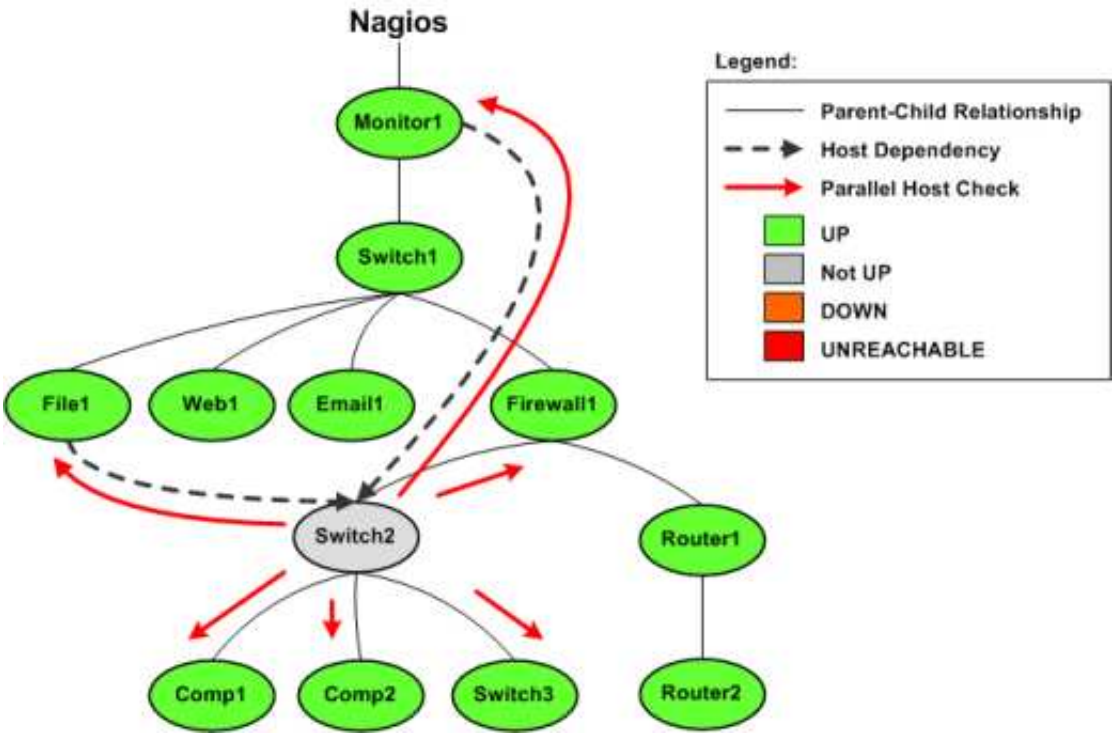
Nagios erlaubt Ihnen, vorausschauende Abhängigkeitsprüfungen für Hosts und Services zu aktivieren, um sicherzustellen, dass die Abhängigkeitslogik die aktuellsten Status-Informationen hat, wenn Entscheidungen darüber getroffen werden müssen, ob Benachrichtigungen verschickt werden oder um aktive Prüfungen für einen Host oder Service zu erlauben.

Wie arbeiten vorausschauende Prüfungen?

Das nachfolgende Bild zeigt ein einfaches Diagramm von Hosts, die von Nagios überwacht werden, zusammen mit ihren Eltern/Kindbeziehungen und Abhängigkeiten.

Der *Switch2*-Host in diesem Beispiel hat gerade den Status von UP in einen Problemzustand gewechselt. Nagios muss feststellen, ob der Host DOWN oder UNREACHABLE ist, also wird es parallele Prüfungen für die direkten Eltern (*Firewall1*) und Kinder (*Comp1*, *Comp2*, und *Switch3*) auslösen. Das ist eine normale Funktion der Host-Erreichbarkeits-Logik.

Sie werden auch feststellen, dass *Switch2* von *Monitor1* und *File1* in Bezug auf Benachrichtigungen oder Prüfausführung abhängt (welches davon ist unwichtig für dieses Beispiel). Wenn vorausschauende Host-Abhängigkeitsprüfungen aktiviert sind, wird Nagios parallele Prüfungen von *Monitor1* und *File1* sowie gleichzeitig für die direkten Eltern und Kinder von *Switch2* auslösen. Nagios tut dies, weil es weiß, dass es die Abhängigkeitslogik in der nahen Zukunft prüfen muss (z.B. für Zwecke der Benachrichtigung) und es will sicherstellen, dass es die aktuellsten Statusinformationen für die Hosts hat, die an der Abhängigkeit beteiligt sind.



So arbeiten vorausschauende Abhängigkeitsprüfungen. Einfach, oder?

Anmerkung: Vorausschauende Service-Abhängigkeitsprüfungen arbeiten in einer ähnlichen Weise wie oben beschrieben. Außer natürlich, dass sie mit Services arbeiten statt mit Hosts.

Vorausschauende Prüfungen aktivieren

Vorausschauende Abhängigkeitsprüfungen verursachen ziemlich wenig Overhead, also würde ich empfehlen, dass Sie diese aktivieren. In den meisten Fällen werden die Vorteile, aktuelle Informationen für die Abhängigkeitslogik zu haben, den zusätzlichen Overhead durch diese Prüfungen mehr als ausgleichen.

Vorausschauende Abhängigkeitsprüfungen zu aktivieren ist einfach:

- Vorausschauende Host-Abhängigkeitsprüfungen werden durch die enable_predictive_host_dependency_checks-Option kontrolliert.
- Vorausschauende Service-Abhängigkeitsprüfungen werden durch die enable_predictive_service_dependency_checks-Option kontrolliert.

Cached Checks

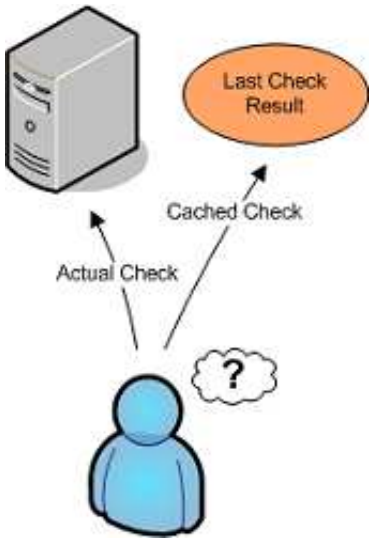
Vorausschauende Abhängigkeitsprüfungen sind Prüfungen nach Bedarf und daher den Regeln von cached checks unterworfen. Cached checks können Ihnen Performance-Verbesserungen liefern, wenn Nagios darauf verzichtet, eine Host- oder Serviceprüfung durchzuführen, wenn es statt dessen ein relativ aktuelles Prüfungsergebnis nutzen kann. Mehr Informationen über cached checks finden Sie hier.

Zwischengespeicherte Prüfungen

Hoch zu: Inhalt

Siehe auch: Host-Prüfungen, Service-Prüfungen, Vorausschauende Abhängigkeitsprüfungen

Einführung



Die Leistung der Überwachungslogik von Nagios kann mit Hilfe von zwischengespeicherten Prüfungen (cached checks) nennenswert gesteigert werden. Zwischengespeicherte Prüfungen erlauben es Nagios, auf die Ausführung einer Host- oder Service-Prüfung zu verzichten, wenn es feststellt, dass ein recht aktuelles Prüfergebnis ausreicht.

nur für Prüfungen nach Bedarf

Regelmäßig eingeplante Host- und Service-Prüfungen werden keine Leistungssteigerung durch zwischengespeicherte Prüfungen erfahren. Zwischengespeicherte Prüfungen sind nur sinnvoll zur Steigerung von Host- und Service-Prüfungen nach Bedarf. Geplante Prüfungen sorgen dafür, dass Host- und Service-Zustände regelmäßig aktualisiert werden, was in der Zukunft dazu führen kann, dass die Ergebnisse als zwischengespeicherte Prüfungen genutzt werden können.

Zur Erinnerung: Host-Prüfungen nach Bedarf treten auf...

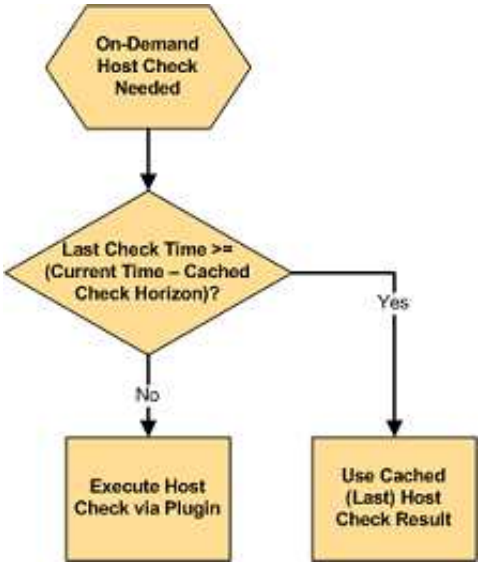
- wenn ein mit einem Host verbundener Service den Status wechselt
- wenn nötig als Teil der Host-Erreichbarkeits-Logik
- wenn nötig für vorausschauende Host-Abhängigkeitsprüfungen

und Service-Prüfungen nach Bedarf treten auf...

- wenn nötig für vorausschauende Service-Abhängigkeitsprüfungen

Hinweis: Solange Sie keinen Gebrauch von Service-Abhängigkeiten machen, wird Nagios nicht in der Lage sein, zwischengespeicherte Prüfungen zur Leistungssteigerung von Service-Prüfungen zu nutzen. Keine Bange - das ist normal. Zwischengespeicherte Host-Prüfungen sorgen für große Leistungssteigerungen und jeder sollte dort einen Vorteil sehen.

Wie Zwischenspeicherung arbeitet



Wenn Nagios eine Host- oder Service-Prüfung nach Bedarf durchführen muss, wird es eine Festlegung treffen, ob es ein zwischengespeichertes Ergebnis benutzen kann oder ob es wirklich eine Prüfung durchführen muss. Es tut dies, indem es schaut, ob die letzte Prüfung für den Host oder Service innerhalb der letzten X Minuten erfolgte, wobei X der zwischengespeicherte Host- oder Service-Horizont ist.

Wenn die letzte Prüfung innerhalb des Zeitfensters erfolgte, das durch die cached-check-horizon-Variable angegeben ist, wird Nagios das Ergebnis der letzen Host- oder Service-Prüfung nutzen und *nicht* eine neue Prüfung ausführen. Wenn der Host oder Service noch nicht geprüft wurde oder die letzte Prüfung außerhalb des cached-check-horizon-Zeitfensters liegt, wird Nagios durch ein Plugin eine neue Host- oder Service-Prüfung durchführen.

Was dies wirklich bedeutet

Nagios führt Prüfungen nach Bedarf durch, weil es den aktuellen Status eines Hosts oder Service *in diesem Moment* wissen muss. Durch die Nutzung von zwischengespeicherten Prüfungen lassen Sie Nagios glauben, dass die kürzlichen Prüfungsergebnisse für die Ermittlung des aktuellen Zustands von Hosts "gut genug" sind und dass es nicht hergehen muss und erneut den Zustand für den Host oder Service prüfen muss.

Die cached-check-horizon-Variable teilt Nagios mit, wie aktuell Prüfergebnisse sein müssen, um zuverlässig den jetzigen Status eines Hosts oder Services darzustellen. Bei einem cached-check-horizon-Wert von 30 Sekunden sagen Sie Nagios, dass die Prüfung des Zustands eines Host innerhalb der letzten 30 Sekunden erfolgt sein muss, um noch als aktueller Zustand dieses Hosts angesehen zu werden.

Die Anzahl von zwischengespeicherten Prüfergebnissen, die Nagios nutzen kann, im Verhältnis zu der Anzahl von Prüfungen nach Bedarf, kann als die cached-check "Treffer"-Rate bezeichnet werden. Durch die Erhöhung des cached-check-horizon-Wertes bis zum regulären Prüfintervall des Hosts können Sie theoretisch eine Trefferrate von 100% erreichen. In diesem Fall würden alle Prüfungen nach Bedarf

zwischengespeicherte Prüfergebnisse benutzen. Was für eine Leistungssteigerung! Aber ist es das wirklich? Wahrscheinlich nicht.

Die Zuverlässigkeit von zwischengespeicherten Prüfergebnissen nimmt mit der Zeit ab. Höhere Trefferraten erfordern, dass vorherige Prüfergebnisse für längere Zeit als "gültig" angesehen werden. Dinge können sich schnell in jedem Netzwerk-Szenario ändern, und es gibt keine Garantie dafür, dass es bei einem Server auf einmal brennt, der vor 30 Sekunden fehlerfrei funktionierte. Das ist der Kompromiss: Zuverlässigkeit gegen Geschwindigkeit. Wenn der cached-check-horizon-Wert groß ist, riskieren Sie, dass Sie unzuverlässige Prüfergebnisse in der Überwachungslogik haben.

Nagios wird letztendlich den korrekten Status aller Hosts und Services ermitteln, so dass es lediglich für eine kurze Zeit mit inkorrekten Informationen arbeitet, selbst wenn sich die zwischengespeicherten Prüfergebnisse als unzuverlässig herausstellen sollten. Selbst kurze Zeiten von unzuverlässigen Statusinformationen können sich für Admins als Ärgernis erweisen, wenn sie Benachrichtigungen über Probleme bekommen, die nicht länger existieren.

Es gibt keinen Standard-cached-check-horizon-Wert oder keine Trefferrate, die für jeden Nagios-Benutzer akzeptierbar wäre. Einige Leute möchten einen kleines horizon-Zeitfenster und eine niedrige Trefferrate während andere ein größeres Zeitfenster und eine höhrere Trefferrate bevorzugen (mit einer kleineren Zuverlässigkeitsrate). Einige Leute möchten vielleicht ganz auf zwischengespeicherte Prüfungen verzichten, um eine hundertprozentige Zuverlässigkeitsrate zu erhalten. Verschiedene horizon-Zeitfenster auszuprobieren und ihren Einfluss auf die Zuverlässigkeit von Statusinformationen zu sehen ist vielleicht das einzige Bedürfnis, das ein einzelner Benutzer hat, um den "richtigen" Wert für seine Situation zu finden.

Konfigurationsvariablen

Die folgenden Variablen legen die Zeitfenster fest, in denen ein vorangegangenes Prüfergebnis als ein zwischengespeichertes Prüfergebnis genutzt werden kann:

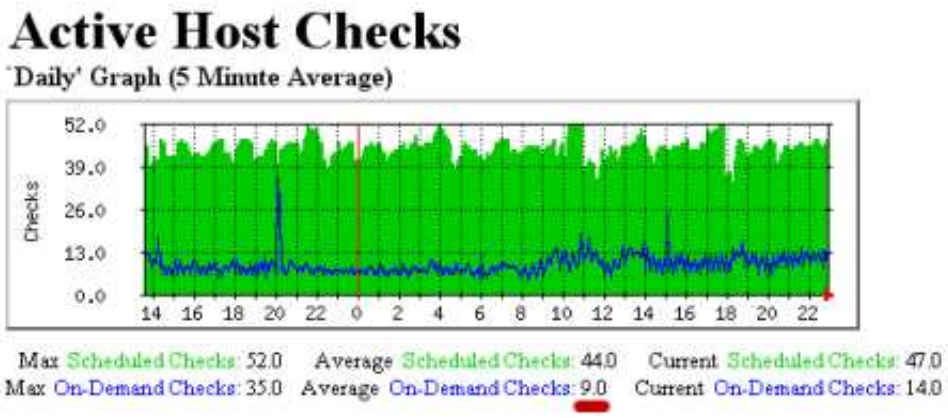
- Die cached_host_check_horizon-Variable kontrolliert zwischengespeicherte Host-Prüfungen.
- Die cached_service_check_horizon-Variable kontrolliert zwischengespeicherte Service-Prüfungen.

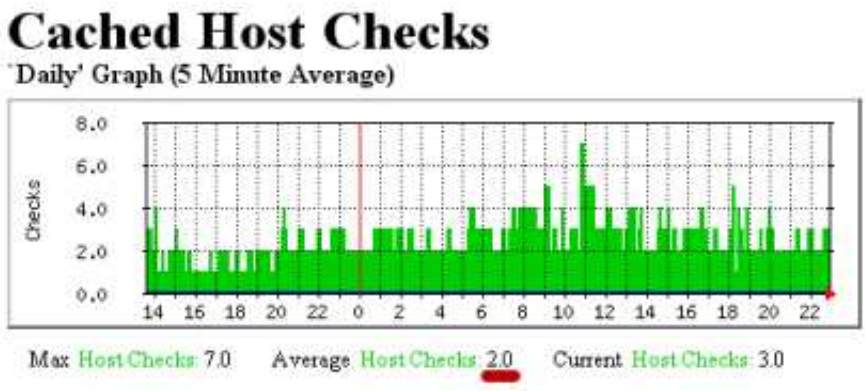
Zwischenspeichereffektivität optimieren

Um den größten Nutzen aus zwischengespeicherten Prüfungen zu ziehen, sollten Sie:

- regelmäßige Host-Prüfungen einplanen
- MRTG (oder PNP ;-)) benutzen, um grafische Auswertungen von 1) Prüfungen nach Bedarf und 2) zwischengespeicherten Prüfungen zu erstellen
- die cached-check-horizon-Variablen Ihren Anforderungen entsprechend anpassen

Sie können regelmäßige Prüfungen für Ihre Hosts durch einen größeren Wert als 0 in der *check_interval*-Option in Ihren Host-Definitionen einplanen. Wenn Sie das tun, sollten Sie die *max_check_attempts*-Option auf einen Wert größer als 1 setzen, oder es wird ein Performance-Problem geben. Das potenzielle Performance-Problem ist hier genauer beschrieben.





Ein guter Weg, um den richtigen Wert für die cached-check-horizon-Optionen zu ermitteln, besteht im Vergleich der Anzahl von Prüfungen nach Bedarf gegen die Anzahl, in denen zwischengespeicherte Ergebnisse benutzt werden. Das nagiostats-Dienstprogramm kann Informationen über zwischengespeicherte Prüfungen erzeugen, die dann mit MRTG dargestellt werden können. MRTG-Beispiel-Diagramme, die zwischengespeicherte Prüfungen gegen solche nach Bedarf darstellen, werden rechts gezeigt.

Bei der überwachten Installation, die diese Graphen erzeugte, gab es...

- insgesamt 44 Hosts, die alle in regelmäßigen Abständen geprüft wurden
- ein durchschnittliches (regelmäßig geplantes) Host-Prüfintervall von fünf Minuten
- ein cached_host_check_horizon von 15 Sekunden

Das erste MRTG-Diagramm zeigt, wie viele regelmäßig geplante Host-Prüfungen im Vergleich zu zwischengespeicherten Host-Prüfungen erfolgt sind. In diesem Beispiel wurden alle fünf Minuten ein Durchschnitt von 53 Host-Prüfungen durchgeführt. Neun von diesen (17%) sind Prüfungen nach Bedarf.

Das zweite MRTG-Diagramm zeigt, wie viele zwischengespeicherte Host-Prüfungen während der Zeit aufgetreten sind. In diesem Beispiel waren es im Durchschnitt zwei Host-Prüfungen alle fünf Minuten.

Erinnern Sie sich, dass zwischengespeicherte Prüfungen nur für Prüfungen nach Bedarf verfügbar sind. Basierend auf den 5-Minuten- Durchschnitten der Graphen sehen wir, dass Nagios in zwei von neun Fällen ein zwischengespeichertes Ergebnis benutzen kann, wenn Prüfungen nach Bedarf auszuführen sind. Das scheint nicht viel zu sein, aber diese Graphen stellen eine kleine Überwachungsumgebung dar. Bedenken Sie, dass zwei von neun 22% sind und Sie können sich vorstellen, wie dies die Host-Prüf-Performance in großen Umgebungen steigern kann. Der Prozentsatz könnte größer sein, wenn der Wert der cached_host_check_horizon-Variablen erhöht wird, aber das würde die Zuverlässigkeit der zwischengespeicherten Host-Statusinformation verringern.

Sobald Sie ein paar Stunden oder Tage mit MRTG-Graphen haben, sollten Sie sehen, wie viele Host- und Service-Prüfungen mit Hilfe von Plugins ausgeführt werden gegen die, die zwischengespeicherte Prüfergebnisse benutzen. Nutzen Sie diese Informationen, um die cached-check-horizon-Variablen entsprechend für Ihre Situation anzupassen. Überwachen Sie weiterhin die MRTG-Graphen, um zu sehen, wie die Änderung der horizon-Variablen die zwischengespeicherten Prüf-Statistiken beeinflusst. Ändern und wiederholen Sie, falls erforderlich.

Passive Host-Zustandsübersetzung

Hoch zu: Inhalt

Siehe auch: Host-Prüfungen, Netzwerk-Erreichbarkeit, Passive Prüfungen, Verteilte Überwachung, Redundante/Failover-Überwachung

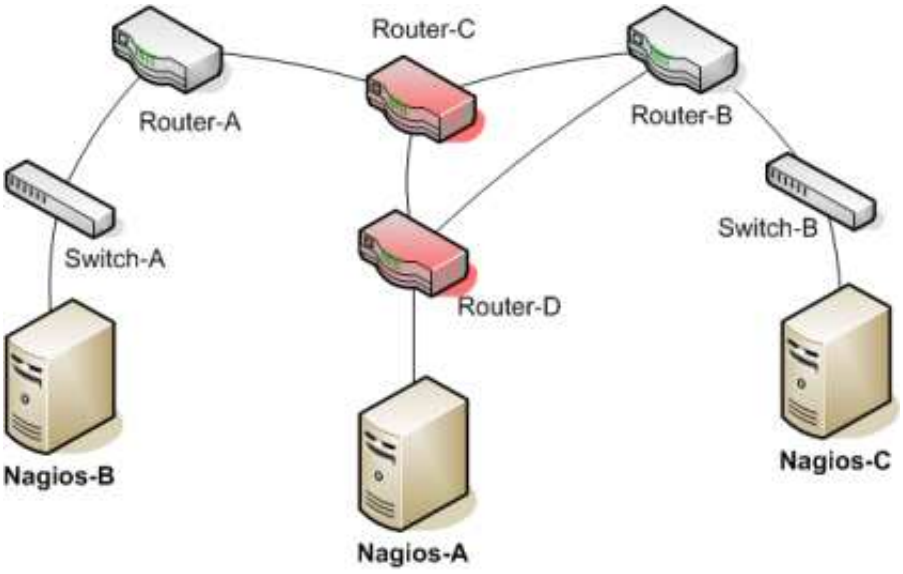
Einführung

Wenn Nagios passive Host-Prüfungen von entfernten Quellen erhält (d.h. andere Nagios-Instanzen in verteilten oder Failover-Umgebungen), gibt der von der entfernten Quelle gelieferte Host-Status ggf. nicht genau den aus Nagios' Sicht zutreffenden Zustand wieder. Weil verteilte und Failover-Überwachungs-Installationen ziemlich identisch sind, ist es wichtig einen Mechanismus anzubieten, um exakte Host-Zustände zwischen verschiedenen Nagios-Instanzen sicherzustellen.

Verschiedene Sichten

Das folgende Bild zeigt eine vereinfachte Sicht für ein Failover-Überwachungsaufbau.

- *Nagios-A* ist der primäre Überwachungsserver, der aktiv alle Switches und Router überwacht.
- *Nagios-B* und *Nagios-C* sind Backup-Überwachungsserver, die passive Prüfergebnisse von *Nagios-A* erhalten.
- Sowohl *Router-C* als auch *Router-D* sind fehlerhaft und daher offline.



In welchem Status sind *Router-C* und *Router-D* gerade? Die Antwort hängt davon ab, welche Nagios-Instanz Sie fragen.

- *Nagios-A* sieht *Router-D* als DOWN und *Router-C* als UNREACHABLE
- *Nagios-B* sollte *Router-C* als DOWN und *Router-D* als UNREACHABLE sehen
- *Nagios-C* sollte beide Router als DOWN sehen.

Jede Nagios-Instanz hat eine unterschiedliche Sicht des Netzwerks. Die Backup-Überwachungsserver sollten nicht blind passive Host-Zustände vom primären Überwachungsserver akzeptieren oder sie werden inkorrekte Informationen über den aktuellen Zustand des Netzwerks haben.

Ohne die Übersetzung von passiven Host-Prüfergebnissen vom primären Überwachungsserver (*Nagios-A*) würde *Nagios-C* den *Router-D* als UNREACHABLE sehen, obwohl dieser vom eigenen Standpunkt eigentlich DOWN ist. Ähnliches gilt für die DOWN/UNREACHABLE-Zustände von *Router-C* und *Router-D* (vom Standpunkt von *Nagios-A* aus), die aus Sicht von *Nagios-B* umgedreht werden sollten.

 Anmerkung: Es kann einige Situationen geben, in denen Sie nicht möchten, dass Nagios die DOWN/UNREACHABLE-Zustände von entfernten Quellen in ihre "korrekten" Zustände vom Standpunkt der lokalen Nagios-Instanz aus umsetzt. Zum Beispiel möchten Sie vielleicht in verteilten Überwachungsumgebungen, dass die zentrale Nagios-Instanz weiß, wie verteilte Instanzen ihre jeweiligen Teile des Netzwerks sehen.

Status-Übersetzung aktivieren

Per Default wird Nagios *nicht* automatisch die DOWN/UNREACHABLE-Zustände von passiven Prüfergebnissen übersetzen. Sie müssen dieses Feature aktivieren, wenn Sie es benötigen und nutzen wollen.

Die automatische Übersetzung von passiven Host-Prüfzuständen wird durch die translate_passive_host_checks-Variable kontrolliert. Durch die Aktivierung wird Nagios automatisch DOWN- und UNREACHABLE-Zustände von entfernten Quellen in die korrekten Zustände für die lokale Instanz übersetzen.

Service- und Host-Prüfungsplanung

Hoch zu: Inhalt

Siehe auch: Aktive Prüfungen

TODO

Diese Dokumentation wird für Nagios 3 umgeschrieben. Bleiben Sie dran...

Angepasste CGI-Kopf- und Fußzeilen

Hoch zu: Inhalt

Siehe auch: Information zu den CGIs

Einführung

Wenn Sie Nagios-Installationen für Kunden machen, dann möchten Sie vielleicht, dass in den CGIs angepasste Kopf- und Fußzeilen angezeigt werden. Dies ist besonders dann nützlich, wenn Sie Support-Kontaktinformationen u.ä. für den Endbenutzer anzeigen möchten.

Es ist wichtig anzumerken, dass angepasste Dateien mit Kopf- und Fußzeilen nicht in irgendeiner Form vorverarbeitet werden (solange es sich nicht um ausführbare Dateien handelt), bevor sie angezeigt werden. Der Inhalt der Kopf- und Fußzeilen wird ganz einfach gelesen und in der CGI-Ausgabe angezeigt. Das bedeutet, dass diese Dateien lediglich Informationen enthalten können, die ein Web-Browser versteht (HTML, JavaScript, usw.).

Wenn die angepassten Kopf- und Fußzeilendateien ausführbar sind, dann werden sie ausgeführt und die Ausgaben an den Benutzer zurückgeliefert, so dass die Dateien gültigen HTML-Code enthalten sollten. Auf diese Weise können Sie Ihre eigenen CGIs nutzen, um Daten in der Nagios-Anzeige auszugeben. Dies kann genutzt werden, um mit ddraw Grafiken aus rrdtool einzufügen und Befehlsmenüs im Nagios-Fenster anzuzeigen. Die ausführbaren angepassten Kopf- und Fußzeilendateien werden mit der gleichen CGI-Umgebung ausgeführt wie das Nagios-Haupt-CGI, so dass Ihre Dateien die Abfrageinformationen, Benutzerauthentifizierungsinformationen usw. analysieren können, um entsprechende Ausgaben zu erzeugen.

Wie funktioniert es?

Sie können angepasste Kopf- und Fußzeilen in die Ausgaben der CGIs einschließen, indem Sie entsprechend benannte HTML-Dateien im *ssi*-Unterverzeichnis des Nagios-HTML-Verzeichnisses (z.B. */usr/local/nagios/share/ssi*) ablegen.

Angepasste Kopfzeilen werden direkt hinter dem <BODY>-Tag in der CGI-Ausgabe eingefügt, während angepasste Fußzeilen direkt vor dem schließenden </BODY>-Tag eingefügt werden.

Es gibt zwei Arten von angepassten Kopf- und Fußzeilen:

- Globale Kopf-/Fußzeilen: diese Dateien sollten *common-header.ssi* und *common-footer.ssi* benannt werden. Wenn diese Dateien existieren, werden sie in die Ausgaben aller CGIs eingefügt.
- CGI-spezifische Kopf-/Fußzeilen: diese Dateinamen sollten im Format *CGINAME-header.ssi* und *CGINAME-footer.ssi* benannt werden, wobei *CGINAME* der (Datei-) Name des CGIs ohne die .cgi-Erweiterung ist. Die Kopf- und Fußzeilendateien des alert summary CGI (summary.cgi) würden beispielsweise *summary-header.ssi* und *summary-footer.ssi* heißen.

Sie sind nicht gezwungen, irgendwelche angepassten Kopf- und Fußzeilen zu benutzen. Sie können nur eine globale Kopfzeile benutzen, wenn Sie möchten. Sie können nur CGI-spezifische Kopfzeilen und eine globale Fußzeile benutzen, wenn Sie möchten. Ganz wie Sie wollen. Wirklich.

Objektvererbung

Hoch zu: Inhalt

Siehe auch: Objektkonfiguration, Objekttricks, eigene Objektvariablen, Fast Startup Options

Einführung

Dieses Dokument versucht Objektvererbung zu erklären und wie sie in Ihren Objektdefinitionen genutzt werden kann.

Wenn Sie nach dem Lesen verwirrt sind, wie Rekursion und Vererbung arbeiten, sollten Sie einen Blick in die Beispielobjektkonfigurationsdateien in der Nagios-Distribution werfen. Wenn das immer noch nicht hilft, dann senden Sie eine (englischsprachige) e-Mail mit einer *detaillierten* Beschreibung Ihres Problems an die *nagios-users*-Mailing-List.

Grundlagen

Es gibt drei Variablen in allen Objektdefinitionen, die Rekursion und Vererbung beeinflussen. Sie sind in rot wie folgt dargestellt:

```
define someobjecttype{
        object-specific variables ...
        name            template_name
        use             name_of_template_to_use
        register        [0/1]
        }
```

Die erste Variable heißt *name*. Das ist lediglich ein "Vorlagen"-Name (template name), auf den in anderen Objektdefinitonen verwiesen wird, so dass diese die Objekteigenschaften/Variablen erben. Vorlagennamen müssen innerhalb der Objekte des gleichen Typs eindeutig sein, so dass Sie nicht zwei oder mehr Host-Definitionen mit "hosttemplate" als Namen haben können.

Die zweite Variable heißt *use*. Hier geben Sie den Namen der Vorlage an, deren Eigenschaften/Variablen Sie erben möchten. Der Name, den Sie für diese Variable angeben, muss als Vorlage definiert sein (mit Hilfe der *name*-Variable).

Die dritte Variable heißt *register*. Diese Variable wird benutzt, um anzuzeigen, ob die Objektdefinition "registriert" werden soll. Per Default werden alle Objektdefinitionen registriert. Wenn Sie eine partielle Objektdefinition als Vorlage nutzen, möchten Sie verhindern, dass sie registriert wird (ein Beispiel dazu folgt). Die Werte sind wie folgt: 0 = die Objektdefinition NICHT registrieren, 1 = die Objektdefinition registrieren (das ist der Default). Diese Variable wird NICHT vererbt, bei jeder als Vorlage genutzten (Teil-) Objektdefinition muss explizit die *register*-Direktive auf *0* gesetzt werden. Dies verhindert die Notwendigkeit, eine vererbte *register*-Direktive für jedes zu registrierende Objekt mit einem Wert von *1* zu übersteuern.

Lokale Variablen gegenüber vererbten Variablen

Bei der Vererbung ist es wichtig zu wissen, dass "lokale" Objektvariablen immer Vorrang vor Variablen aus der Vorlage haben. Werfen Sie einen Blick auf das folgende Beispiel mit zwei Host-Definitionen (nicht alle notwendigen Variablen sind dargestellt):

```
define host{
        host_name               bighost1
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
        name                    hosttemplate1
        }

define host{
        host_name               bighost2
        max_check_attempts      3
        use                     hosttemplate1
        }
```

Sie werden bemerken, dass die Definiton für den Host *bighost1* mit Hilfe der Vorlage *hosttemplate1* definiert wurde. Die Definition für Host *bighost2* nutzt die Definition von *bighost1* als Vorlagenobjekt. Sobald Nagios diese Daten verarbeitet hat, wäre die resultierende Definition von *bighost2* äquivalent zu dieser Definition:

```
define host{
        host_name               bighost2
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      3
        }
```

Sie sehen, dass die *check_command*- und *notification_options*-Variablen vom Vorlagenobjekt geerbt wurden (wo Host *bighost1* definiert wird). Trotzdem wurden die *host_name*- und *check_attempts*-Variablen nicht vom Vorlagenobjekt geerbt, weil sie lokal definiert wurden. Erinnern Sie sich, dass von einem Vorlagenobjekt geerbte Variablen von lokal definierten Variablen überschrieben werden. Das sollte ein ziemlich einfach zu verstehendes Konzept sein.

 Hinweis: wenn Sie möchten, dass lokale Zeichenketten-Variablen an geerbte Zeichenkettenwerte angehängt werden, können Sie das tun. Lesen Sie weiter unten mehr darüber, wie das erreicht werden kann.

Vererbungsverkettung

Objekte können Eigenschaften/Variablen aus mehreren Ebenen von Vorlagenobjekten erben. Nehmen Sie das folgende Beispiel:

```
define host{
        host_name               bighost1
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
        name                    hosttemplate1
        }

define host{
        host_name               bighost2
        max_check_attempts      3
        use                     hosttemplate1
        name                    hosttemplate2
        }

define host{
        host_name               bighost3
        use                     hosttemplate2
        }
```

Sie werden bemerken, dass die Definition von Host *bighost3* Variablen von der Definition von *bighost2* erbt, die wiederum Variablen von der Definition von Host *bighost1* erbt. Sobald Nagios diese Konfigurationsdaten verarbeitet, sind die resultierenden Host-Definition äquivalent zu den folgenden:

```
define host{
        host_name               bighost1
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
        }

define host{
        host_name               bighost2
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      3
        }

define host{
        host_name               bighost3
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      3
        }
```

Es gibt keine eingebaute Beschränkung, wie "tief" Vererbung gehen kann, aber Sie sollten sich vielleicht selbst auf ein paar Ebenen beschränken, um die Übersicht zu behalten.

Unvollständige Objektdefinitionen als Vorlagen nutzen

Es ist möglich, unvollständige Objektdefinitionen als Vorlage für andere Objektdefinitionen zu nutzen. Mit "unvollständiger" Definition meine ich, dass nicht alle benötigten Variablen in der Objektdefinition angegeben wurden. Es mag komisch klingen, unvollständige Definitionen als Vorlagen zu nutzen, aber es ist tatsächlich empfohlen, dies zu tun. Warum? Nun, sie können als ein Satz von Defaults für alle anderen Objektdefinitionen dienen. Nehmen Sie das folgende Beispiel:

```
define host{
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
        name                    generichosttemplate
        register                0
        }

define host{
        host_name               bighost1
        address                 192.168.1.3
        use                     generichosttemplate
        }

define host{
        host_name               bighost2
        address                 192.168.1.4
        use                     generichosttemplate
        }
```

Beachten Sie, dass die erste Host-Definition unvollständig ist, weil die erforderliche *host_name*-Variable fehlt. Wir müssen keinen Host-Namen angeben, weil wir diese Definition als Vorlage nutzen wollen. Um Nagios daran zu hindern, diese Definition als einen normalen Host anzusehen, setzen wir die *register*-Variable auf 0.

Die Definitionen von *bighost1* und *bighost2* erben ihre Werte von der generischen Host-Definition. Die einzige Variable, die überschrieben wird, ist die *address*-Variable. Das bedeutet, dass beide Hosts exakt die gleichen Eigenschaften haben, bis auf die *host_name*- und *address*-Variablen. Sobald Nagios die Konfigurationsdaten im Beispiel verarbeitet, wären die resultierenden Host-Definitionen äquivalent zu folgenden:

```
define host{
        host_name               bighost1
        address                 192.168.1.3
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
        }

define host{
        host_name               bighost2
        address                 192.168.1.4
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
        }
```

Die Nutzung einer Vorlagendefinition für Default-Werte erspart Ihnen mindestens eine Menge Tipparbeit. Es spart Ihnen auch eine Menge Kopfschmerzen, wenn Sie später die Default-Werte von Variablen für eine große Zahl von Hosts wollen.

eigene Objektvariablen(custom object variables)

Jede eigene Objektvariable, die Sie in Ihren Host-, Service- oder Kontaktdefinitionen definieren, wird wie jede andere Standardvariable vererbt. Nehmen Sie das folgende Beispiel:

```
define host{
        _customvar1             somevalue  ; <-- Custom host variable
        _snmp_community         public  ; <-- Custom host variable
        name                    generichosttemplate
        register                        0
        }

define host{
        host_name               bighost1
        address                 192.168.1.3
        use                     generichosttemplate
        }
```

Der Host *bighost1* wird die eigenen Host-Variablen *_customvar1* und *_snmp_commnity* von der *generichosttemplate*-Definition erben, zusammen mit den entsprechenden Werten. Die daraus resultierende Definition für *bighost1* sieht wie folgt aus:

```
define host{
        host_name               bighost1
        address                 192.168.1.3
        _customvar1             somevalue
        _snmp_community         public
        }
```

Vererbung für Zeichenketten-Werte aufheben

In einigen Fällen möchten Sie vielleicht nicht, dass Ihre Host-, Service- oder Kontakt-Definitionen Werte von Zeichenketten-Variablen aus Vorlagen erben. Wenn das der Fall ist, können Sie "**null**" (ohne Anführungszeichen) als den Wert der Variable, die Sie nicht erben möchten. Nehmen Sie das folgende Beispiel:

```
define host{
        event_handler           my-event-handler-command
        name                    generichosttemplate
        register                0
        }

define host{
        host_name               bighost1
        address                 192.168.1.3
        event_handler           null
        use                     generichosttemplate
        }
```

In diesem Fall wird der Host *bighost1* nicht den Wert der *event_handler*-Variable erben, die in der *generichosttemplate*-Vorlage definiert ist. Die resultierende Definition von *bighost1* sieht wie folgt aus:

```
define host{
        host_name               bighost1
        address                 192.168.1.3
        }
```

additive Vererbung von Zeichenketten-Werten

Nagios gibt lokalen Variablen Vorrang vor Werten, die von Vorlagen vererbt werden. In den meisten Fällen überschreiben lokale Variablenwerte jene, die in Vorlagen definiert sind. In einigen Fällen ist es sinnvoll, dass Nagios die Werte von geerbten *und* lokalen Variablen gemeinsam nutzt.

Diese "additive Vererbung" kann durch Voranstellen eines Pluszeichens (**+**) vor den lokalen Variablenwert erreicht werden. Dieses Feature ist nur für Standard-Variablen verfügbar, die Zeichenketten-Werte enthalten. Nehmen Sie das folgende Beispiel:

```
define host{
        hostgroups              all-servers
        name                    generichosttemplate
        register                0
        }

define host{
        host_name               linuxserver1
        hostgroups              +linux-servers,web-servers
        use                     generichosttemplate
        }
```

In diesem Fall wird der *linuxserver1* den Wert der lokalen *hostgroups*-Variablen dem der *generichosttemplate*-Vorlage hinzufügen. Die resultierende Definition von *linuxserver1* sieht wie folgt aus:

```
define host{
        host_name               linuxserver1
        hostgroups              all-servers,linux-servers,web-servers
        }
```

Implizite Vererbung

Normalerweise müssen Sie entweder explizit den Wert einer erforderlichen Variable in einer Objektdefinition angeben oder sie von einer Vorlage erben. Es gibt ein paar Ausnahmen zu dieser Regel, in denen Nagios annimmt, dass Sie einen Wert benutzen wollen, der statt dessen von einem verbundenen Objekt kommt. Die Werte einiger Service-Variablen werden zum Beispiel vom Host kopiert, mit dem der Service verbunden ist, wenn Sie diese nicht anderweitig angeben.

Die folgende Tabelle führt die Objektvariablen auf, die implizit von verbundenen Objekten vererbt werden, wenn Sie deren Werte nicht explizit angeben oder sie von einer Vorlage erben.

Objekttyp	Objektvariable	implizite Quelle
Services	*contact_groups*	*contact_groups* in der verbundenen Host-Definition
	notification_interval	*notification_interval* in der verbundenen Host-Definition
	notification_period	*notification_period* in der verbundenen Host-Definition
Host Escalations	*contact_groups*	*contact_groups* in der verbundenen Host-Definition
	notification_interval	*notification_interval* in der verbundenen Host-Definition
	escalation_period	*notification_period* in der verbundenen Host-Definition
Service Escalations	*contact_groups*	*contact_groups* in der verbundenen Service-Definition
	notification_interval	*notification_interval* in der verbundenen Service-Definition
	escalation_period	*notification_period* in der verbundenen Service-Definition

implizite/additive Vererbung bei Eskalationen

Service- und Host-Eskalationsdefinitionen können eine spezielle Regel benutzen, die die Möglichkeiten von impliziter und additiver Vererbung kombiniert. Wenn Eskalationen 1) nicht die Werte ihrer *contact_groups*- oder *contacts*-Direktiven von anderen Eskalationsvorlagen erben und 2) ihre *contact_groups*- oder *contacts*-Direktiven mit einen Plus-Zeichen (+) beginnen, dann werden die Werte der *contact_groups* oder *contacts*-Direktiven der entsprechenden Host- oder Service-Definitionen in der additiven Vererbungslogik benutzt.

Verwirrt? Hier ein Beispiel:

```
define host{
        name            linux-server
        contact_groups  linux-admins
        ...
        }

define hostescalation{
        host_name               linux-server
        contact_groups  +management
        ...
        }
```

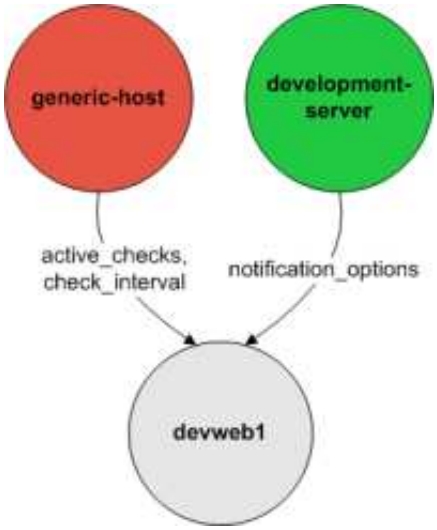
Das ist ein viel einfacheres Äquivalent zu:

```
define hostescalation{
        host_name               linux-server
        contact_groups  linux-admins,management
        ...
        }
```

Mehrere Vererbungsquellen

Bisher haben alle Beispiele Objektdefinitionen gezeigt, die Variablen/Werte von einer einzelnen Quelle erben. Sie können für komplexere Konfigurationen auch Variablen/Werte von mehreren Quellen erben, wie unten gezeigt.

```
# Generic host template
define host{
 name    generic-host
 active_checks_enabled 1
 check_interval  10
 ...
 register   0
 }

# Development web server template
define host{
 name    development-server
 check_interval  15
 notification_options d,u,r
 ...
 register   0
 }

# Development web server
define host{
 use   generic-host,development-server
 host_name   devweb1
 ...
 }
```



Im obigen Beispiel erbt *devweb1* Variablen/Werte von zwei Quellen: *generic-host* und *development-server*. Sie werden bemerken, dass in beiden Quellen eine *check_interval*-Variable definiert ist. Weil *generic-host* die erste in *devweb1* durch die *use*-Direktive angegebene Vorlage ist, wird der Wert für die *check_interval*-Variable durch den *devweb1*-Host vererbt. Nach der Vererbung sieht die Definition von *devweb1* wie folgt aus:

```
# Development web server
define host{
        host_name               devweb1
        active_checks_enabled   1
        check_interval          10
        notification_options    d,u,r
        ...
        }
```

Vorrang bei mehreren Vererbungsquellen

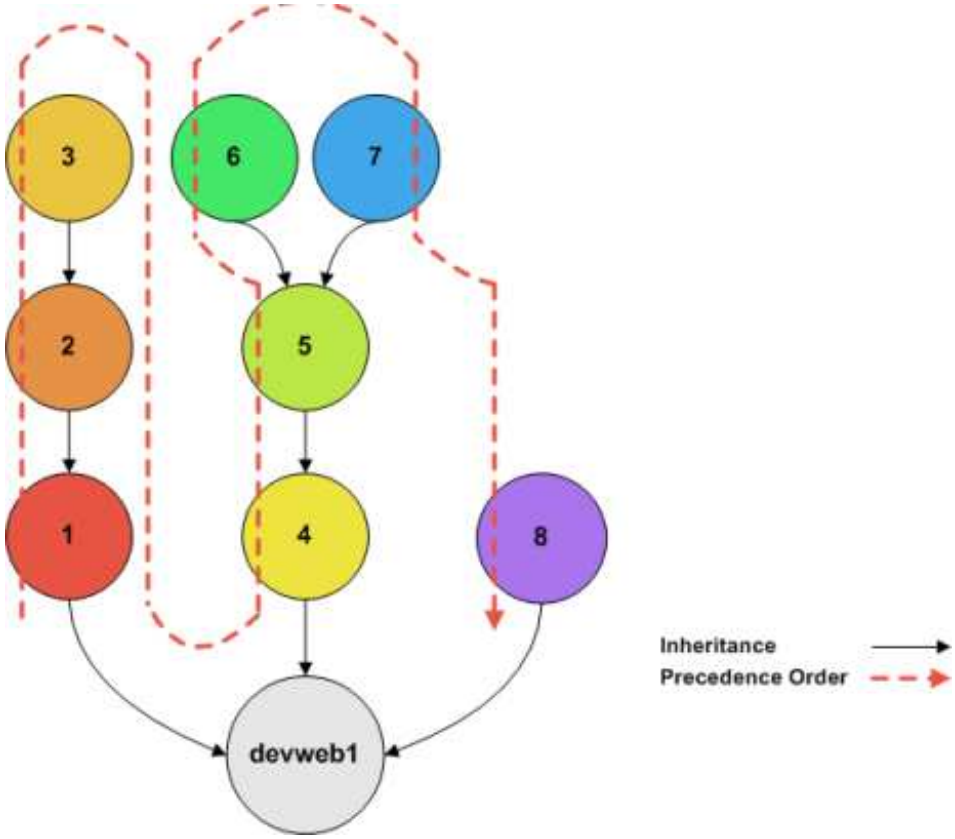
Wenn Sie mehrere Vererbungsquellen nutzen, ist es wichtig zu wissen, wie Nagios Variablen behandelt, die in mehreren Quellen definiert sind. In diesen Fällen wird Nagios die Variable/den Wert aus der ersten Quelle benutzen, die in der *use*-Direktive angegeben ist. Weil Vererbungsquellen ebenfalls Variablen/Werte aus ein oder mehreren Quellen erben können, kann es kompliziert werden herauszufinden, welche Variablen/Werte-Paare Vorrang haben.

Betrachten Sie die folgende Host-Definition, die drei Vorlagen referenziert:

```
# Development web server
define host{
 use    1, 4, 8
 host_name  devweb1
 ...
 }
```

Wenn einige dieser referenzierten Vorlagen selbst Variablen/Werte von ein oder mehreren Vorlagen erben, werden die Vorrangregeln auf der rechten Seite gezeigt.

Test, Versuch und Irrtum werden Ihnen helfen, besser zu verstehen, wie die Dinge in komplexen Vererbungssituationen wie dieser funktionieren. :-)



Zeitsparende Tricks für Objektdefinitionen

or...
"Wie Sie Ihren Verstand bewahren"

Hoch zu: Inhalt

Siehe auch: Objektkonfiguration, Objektvererbung

Einführung

Dieses Dokument versucht zu erklären, wie Sie die (etwas) versteckten Möglichkeiten von vorlagenbasierenden Objektdefinitionen ausnutzen können, um Ihren Verstand zu bewahren. Sie fragen sich Wie? Verschiedene Objekttypen erlauben es Ihnen, mehrere Host-Namen und/oder Hostgruppen-Namen in Definitionen anzugeben und die Objektdefinitionen in mehrere Hosts oder Services zu "kopieren". Ich werde jeden Objekttyp, der diese Möglichkeiten unterstützt, separat behandeln. Für den Anfang sind die Objekttypen, die diese zeitsparende Möglichkeit unterstützen, wie folgt:

- Services
- Service-Eskalationen
- Service-Abhängigkeiten
- Host-Eskalationen
- Host-Abhängigkeiten
- Hostgruppen

Objekttypen, die nicht oben aufgeführt sind (z.B. Zeitfenster, Befehle usw.), unterstützen nicht die Möglichkeiten, die ich beschreiben werde.

Übereinstimmung von regulären Ausdrücken (Regular Expression Matching)

Die Beispiele, die ich unten zeige, benutzen "Standard"-Übereinstimmung (Matching) von Objektnamen und ***erfordern***, dass die Option use_regexp_matching ***deaktiviert***ist.

Wenn Sie wollen, können Sie die Übereinstimmung von regulären Ausdrücken mit Hilfe der use_regexp_matching-Konfigurationsoption aktivieren. Reguläre Ausdrücke können in jedem der Felder benutzt werden, die in den Beispielen unten benutzt werden (Hostnamen, Hostgruppen-Namen, Service-Namen und Servicegruppen-Namen).

Anmerkung: Seien Sie vorsichtig bei der Aktivierung der Übereinstimmung von regulären Ausdrücken - es kann sein, dass Sie Ihre Konfigurationsdatei ändern müssen, weil vielleicht einige der Direktiven als reguläre Ausdrücke interpretiert werden, bei denen Sie das nicht möchten! Probleme sollten offensichtlich werden, sobald Sie Ihre Konfiguration überprüfen.

Service-Definitionen

Mehrere Hosts:
Wenn Sie identische Services erzeugen möchten, die mehreren Hosts zugeordnet sind, können Sie mehrere Hosts in der *host_name*-Direktive angeben. Die folgende Definition würde einen Service namens *SOMESERVICE* auf den Hosts *HOST1* bis *HOSTN* erzeugen. Jede Instanz des *SOMESERVICE*-Service wäre identisch (d.h. hätte den gleichen Prüfbefehl, Benachrichtigungsperiode, usw.).

```
define service{
        host_name               HOST1,HOST2,HOST3,...,HOSTN
        service_description     SOMESERVICE
        weitere Service-Direktiven ...
        }
```

Alle Hosts in mehreren Hostgruppen:
Wenn Sie identische Services erzeugen wollen, die allen Hosts in einer oder mehreren Hostgruppen zugeordnet sind, können Sie das mit einer einzigen Service-Definition erreichen. Wie? Die *hostgroup_name*-Direktive erlaubt es Ihnen, den Namen von einer oder mehreren Hostgruppen anzugeben, für den dieser Service erzeugt werden soll. Die folgende Definition würde einen Service namens *SOMESERVICE* auf allen Hosts anlegen, die Mitglied von Hostgruppe *HOSTGROUP1* bis *HOSTGROUPN* sind. Alle Instanzen des *SOMESERVICE*-Service wären identisch (d.h. hätten den gleichen Prüfbefehl, Benachrichtigungsperiode, usw.).

```
define service{
        hostgroup_name          HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
        service_description     SOMESERVICE
        weitere Service-Direktiven ...
        }
```

Alle Hosts:
Wenn Sie identische Services erzeugen wollen, die allen Hosts in Ihren Konfigurationsdateien zugeordnet sind, können Sie einen Platzhalter (wildcard) in der *host_name*-Direktive benutzen. Die folgende Definition würde einen Service namens *SOMESERVICE* auf **allen Hosts** erzeugen, die in Ihren Konfigurationsdateien definiert sind. Alle Instanzen des *SOMESERVICE*-Service wären identisch (d.h. hätten den gleichen Prüfbefehl, Benachrichtigungsperiode, usw.).

```
define service{
        host_name               *
        service_description     SOMESERVICE
        weitere Service-Direktiven ...
        }
```

Hosts ausschließen:
Wenn Sie identische Services auf zahlreichen Hosts anlegen, aber einige Hosts von dieser Definition ausnehmen möchten, kann dies durch das Voranstellen eines Ausrufezeichens (!) vor dem Host oder der Hostgruppe geschehen.

```
define service{
        host_name               HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
        hostgroup_name          HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
        service_description     SOMESERVICE
        weitere Service-Direktiven ...
        }
```

Service-Eskalationsdefinitionen

Mehrere Hosts:
Wenn Sie identische Service-Eskalationen für Services mit dem gleichen Namen/der gleichen Beschreibung erzeugen möchten, die mehreren Hosts zugeordnet sind, können Sie mehrere Hosts in der *host_name*-Direktive angeben. Die folgende Definition würde eine Service-Eskalation für Services namens *SOMESERVICE* auf den Hosts *HOST1* bis *HOSTN* erzeugen. Alle Instanzen des

SOMESERVICE-Service wären identisch (d.h. hätten den gleichen Prüfbefehl, Benachrichtigungsperiode, usw.).

```
define serviceescalation{
        host_name               HOST1,HOST2,HOST3,...,HOSTN
        service_description     SOMESERVICE
        weitere Eskalations-Direktiven ...
        }
```

Alle Hosts in mehreren Hostgruppen:
Wenn Sie identische Service-Eskalationen für Services mit dem gleichen Namen/der gleichen Beschreibung erzeugen wollen, die allen Hosts in einer oder mehreren Hostgruppen zugeordnet sind, können Sie das mit der *hostgroup_name*-Direktive tun. Die folgende Definition würde eine Service-Eskalation für Services namens *SOMESERVICE* auf allen Hosts anlegen, die Mitglied von Hostgruppe *HOSTGROUP1* bis *HOSTGROUPN* sind. Alle Instanzen des *SOMESERVICE*-Service wären identisch (d.h. hätten den gleichen Prüfbefehl, Benachrichtigungsperiode, usw.).

```
define serviceescalation{
        hostgroup_name          HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
        service_description     SOMESERVICE
        weitere Eskalations-Direktiven ...
        }
```

Alle Hosts:
Wenn Sie identische Service-Eskalationen für Services mit dem gleichen Namen/der gleichen Beschreibung erzeugen wollen, die allen Hosts in Ihren Konfigurationsdateien zugeordnet sind, können Sie einen Platzhalter (wildcard) in der *host_name*-Direktive benutzen. Die folgende Definition würde eine Service-Eskalation für alle Service namens *SOMESERVICE* auf **allen Hosts** erzeugen, die in Ihren Konfigurationsdateien definiert sind. Alle Instanzen des *SOMESERVICE*-Service wären identisch (d.h. hätten den gleichen Prüfbefehl, Benachrichtigungsperiode, usw.).

```
define serviceescalation{
        host_name               *
        service_description     SOMESERVICE
        weitere Eskalations-Direktiven ...
        }
```

Hosts ausschließen:
Wenn Sie identische Service-Eskalationen für Services auf zahlreichen Hosts anlegen, aber einige Hosts von dieser Definition ausnehmen möchten, kann dies durch das Voranstellen eines Ausrufezeichens (!) vor dem Host oder der Hostgruppe geschehen.

```
define serviceescalation{
        host_name               HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
        hostgroup_name          HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
        service_description     SOMESERVICE
        weitere Eskalations-Direktiven ...
        }
```

Alle Services auf dem gleichen Host:
Wenn Sie Service-Eskalationen für alle Services eines bestimmten Hosts anlegen möchten, können Sie einen Platzhalter in der *service_description*-Direktive benutzen. Die folgende Definition würde eine Service-Eskalation für *alle* Services auf Host *HOST1* erzeugen. Alle Instanzen der Service-Eskalation wären identisch (d.h. hätten die gleichen Kontaktgruppe, Benachrichtigungsintervall, usw.).

Wenn Sie sich abenteuerlustig fühlen, dann können Sie einen Platzhalter sowohl bei der *host_name*- als auch bei der *service_description*-Direktive angeben. Dadurch würden Sie eine Service-Eskalation für **alle Services** anlegen, die Sie in Ihren Konfigurationsdateien definiert haben.

```
define serviceescalation{
        host_name               HOST1
        service_description     *
        weitere Eskalations-Direktiven ...
        }
```

Mehrere Services auf dem gleichen Host:
Wenn Sie Service-Eskalationen für mehrere Services eines bestimmten Hosts anlegen möchten, können Sie mehr als eine Service-Beschreibung in der *service_description*-Direktive benutzen. Die folgende Definition würde eine Service-Eskalation für die Services *SERVICE1* bis *SERVICEN* auf Host *HOST1* erzeugen. Alle Instanzen der Service-Eskalation wären identisch (d.h. hätten die gleichen Kontaktgruppe, Benachrichtigungsintervall, usw.).

```
define serviceescalation{
        host_name               HOST1
        service_description     SERVICE1,SERVICE2,...,SERVICEN
        weitere Eskalations-Direktiven ...
        }
```

Alle Services in mehreren Servicegruppen:
Wenn Sie Service-Eskalationen für alle Services anlegen möchten, die zu einer oder mehreren Servicegruppen gehören, können Sie die *servicegroup_name*-Direktive benutzen. Die folgende Definition würde Service-Eskalationen für alle Services anlegen, die Mitglied der Servicegruppen *SERVICEGROUP1* bis *SERVICEGROUPN* sind. Alle Instanzen der Service-Eskalation wären identisch (d.h. hätten die gleichen Kontaktgruppe, Benachrichtigungsintervall, usw.).

```
define serviceescalation{
        servicegroup_name       SERVICEGROUP1,SERVICEGROUP2,...,SERVICEGROUPN
        weitere Eskalations-Direktiven ...
        }
```

Service-Abhängigkeitsdefinitionen

Mehrere Hosts:
Wenn Sie Service-Abhängigkeiten für Services mit dem gleichen Namen/der gleichen Beschreibung erstellen möchten, die mehreren Hosts zugeordnet sind, können Sie mehrere Hosts in den *host_name*- und/oder *dependent_host_name*-Direktiven benutzen. Im folgenden Beispiel wäre Service *SERVICE2* auf den Hosts *HOST3* und *HOST4* abhängig von *SERVICE1* auf den Hosts *HOST1* und *HOST2*. Alle Instanzen der Service-Abhängigkeiten wären identisch bis auf die Host-Namen (d.h. hätten die gleichen Fehlerbenachrichtigungs-Kriterien usw.).

```
define servicedependency{
        host_name                       HOST1,HOST2
        service_description             SERVICE1
        dependent_host_name             HOST3,HOST4
        dependent_service_description   SERVICE2
        weitere Abhängigkeits-Direktiven ...
        }
```

Alle Hosts in mehreren Hostgruppen:
Wenn Sie Service-Abhängigkeiten für Services mit dem gleichen Namen/der gleichen Beschreibung erstellen möchten, die allen Hosts in einer oder mehreren Hostgruppen zugeordnet sind, können Sie die *hostgroup_name*- und/oder *dependent_hostgroup_name*-Direktiven benutzen. Im folgenden Beispiel wäre Service *SERVICE2* auf allen Hosts in den Hostgruppen *HOSTGROUP3* und *HOSTGROUP4* abhängig von *SERVICE1* auf allen Hosts in den Hostgruppen *HOSTGROUP1* und *HOSTGROUP2*. Angenommen, es gibt fünf Hosts in jeder der Hostgruppen, dann wäre diese Definition äquivalent zur Definition von 100 einzelnen Service-Abhängigkeitsdefinitionen! Alle Instanzen der Service-Abhängigkeiten wären identisch bis auf die Host-Namen (d.h. hätten die gleichen Fehlerbenachrichtigungs-Kriterien usw.).

```
define servicedependency{
        hostgroup_name                  HOSTGROUP1,HOSTGROUP2
        service_description             SERVICE1
        dependent_hostgroup_name        HOSTGROUP3,HOSTGROUP4
        dependent_service_description   SERVICE2
        weitere Abhängigkeits-Direktiven ...
        }
```

Alle Services auf einem Host:
Wenn Sie Service-Abhängigkeiten für alle Services eines bestimmten Hosts erstellen möchten, können Sie einen Platzhalter in den *service_description*- und/oder *dependent_service_description*-Direktiven benutzen. Im folgenden Beispiel wären **alle Services** auf Host *HOST2* abhängig von **allen Services** auf Host *HOST1*. Alle Instanzen der Service-Abhängigkeiten wären identisch (d.h. hätten die gleichen Fehlerbenachrichtigungs-Kriterien usw.).

```
define servicedependency{
        host_name                       HOST1
        service_description             *
        dependent_host_name             HOST2
        dependent_service_description   *
        weitere Abhängigkeits-Direktiven ...
        }
```

Mehrere Services auf einem Host:
Wenn Sie Service-Abhängigkeiten für mehrere Services eines bestimmten Hosts erstellen möchten, können Sie mehr als eine Service-Beschreibung in den *service_description*- und/oder *dependent_service_description*-Direktiven wie folgt angeben:

```
define servicedependency{
        host_name                       HOST1
        service_description             SERVICE1,SERVICE2,...,SERVICEN
        dependent_host_name             HOST2
        dependent_service_description   SERVICE1,SERVICE2,...,SERVICEN
        weitere Abhängigkeits-Direktiven ...
        }
```

Alle Services in mehreren Servicegruppen:
Wenn Sie Service-Abhängigkeiten für alle Services erstellen möchten, die einer oder mehreren Servicegruppen zugeordnet sind, können Sie die *servicegroup_name*- und/oder *dependent_servicegroup_name*-Direktiven wie folgt benutzen:

```
define servicedependency{
        servicegroup_name               SERVICEGROUP1,SERVICEGROUP2,...,SERVICEGROUPN
        dependent_servicegroup_name     SERVICEGROUP3,SERVICEGROUP4,...SERVICEGROUPN
        weitere Abhängigkeits-Direktiven ...
        }
```

Abhängigkeiten des gleichen Hosts:
Wenn Sie Service-Abhängigkeiten für mehrere Services erstellen möchten, die von Services auf dem gleichen Host abhängig sind, lassen Sie die *dependent_host_name*- und *dependent_hostgroup_name*-Direktiven leer. Im folgenden Beispiel wird angenommen, dass den Hosts *HOST1* und *HOST2* mindestens die folgenden vier Services zugeordnet sind: *SERVICE1*, *SERVICE2*, *SERVICE3* und *SERVICE4*. In diesem Beispiel sind *SERVICE3* und *SERVICE4* auf *HOST1* abhängig von *SERVICE1* und *SERVICE2* auf *HOST1*. Ähnlich sind *SERVICE3* und *SERVICE4* auf *HOST2* abhängig von *SERVICE1* und *SERVICE2* auf *HOST2*.

```
define servicedependency{
        host_name                       HOST1,HOST2
        service_description             SERVICE1,SERVICE2
        dependent_service_description   SERVICE3,SERVICE4
        weitere Abhängigkeits-Direktiven ...
        }
```

Host-Eskalationsdefinitionen

Mehrere Hosts:
Wenn Sie Host-Eskalationen für mehrere Hosts erstellen möchten, können Sie mehrere Hosts in der *host_name*-Direktive angeben. Die folgende Definitione würde eine Host-Eskalation für die Hosts *HOST1* bis *HOSTN* anlegen. Alle Instanzen der Host-Eskalation wären identisch (d.h. hätten die gleichen Kontaktgruppen, Benachrichtigungsintervalle usw.).

```
define hostescalation{
        host_name               HOST1,HOST2,HOST3,...,HOSTN
        weitere Eskalations-Direktiven ...
        }
```

Alle Hosts in mehreren Hostgruppen:
Wenn Sie Host-Eskalationen für alle Hosts in einer oder mehreren Hostgruppen erstellen möchten, können Sie die *hostgroup_name*-Direktive benutzen. Die folgende Definition würde eine Host-Eskalation für alle Hosts anlegen, die Mitglieder der Hostgruppen *HOSTGROUP1* bis *HOSTGROUPN* sind. Alle Instanzen der Host-Eskalation wären identisch (d.h. hätten die gleichen Kontaktgruppen, Benachrichtigungsintervalle usw.).

```
define hostescalation{
        hostgroup_name          HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
        weitere Eskalations-Direktiven ...
        }
```

Alle Hosts:
Wenn Sie identische Host-Eskalationen für alle Hosts erstellen wollen, die in Ihren Konfigurationsdateien definiert sind, können Sie einen Platzhalter in der *host_name*-Direktive benutzen. Die folgende Definition würde eine Host-Eskalation für alle Hosts anlegen, die in Ihren Konfigurationsdateien definiert sind. Alle Instanzen der Host-Eskalation wären identisch (d.h. hätten die gleichen Kontaktgruppen, Benachrichtigungsintervalle usw.).

```
define hostescalation{
        host_name               *
        weitere Eskalations-Direktiven ...
        }
```

Hosts ausschließen:
Wenn Sie identische Host-Eskalationen auf zahlreichen Hosts oder Hostgruppen erstellen, aber einige Hosts von der Definition ausschließen möchten, kann dies durch das Voranstellen eines Ausrufezeichens (!) vor dem Host oder der Hostgruppe geschehen.

```
define hostescalation{
        host_name               HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
        hostgroup_name          HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
        weitere Eskalations-Direktiven ...
        }
```

Host-Abhängigkeitsdefinitionen

Mehrere Hosts:
Wenn Sie Host-Abhängigkeiten für mehrere Hosts erstellen möchten, können Sie mehrere Hosts in den *host_name*- und/oder *dependent_host_name*-Direktiven angeben. Die folgende Definition wäre äquivalent mit der Erstellung von sechs einzelnen Host-Abhängigkeiten. Im obigen Beispiel wären die Hosts *HOST3*, *HOST4* und *HOST5* abhängig von den Hosts *HOST1* und *HOST2*. Alle Instanzen der Host-Abhängigkeiten wären identisch bis auf die Host-Namen (d.h. sie hätten die gleichen Fehlerbenachrichtigungs-Kriterien, usw.).

```
define hostdependency{
        host_name               HOST1,HOST2
        dependent_host_name     HOST3,HOST4,HOST5
        weitere Abhängigkeits-Direktiven ...
        }
```

Alle Hosts in mehreren Hostgruppen:
Wenn Sie Host-Abhängigkeiten für alle Hosts in einer oder mehreren Hostgruppen erstellen möchten, können Sie die *hostgroup_name*- und/oder *dependent_hostgroup_name*-Direktiven benutzen. Im folgenden Beispiel wären alle Hosts in den Hostgruppen *HOSTGROUP3* und *HOSTGROUP4* abhängig von allen Hosts in den Hostgruppen *HOSTGROUP1* und *HOSTGROUP2*. Alle Instanzen der Host-Abhängigkeiten wären identisch bis auf die Host-Namen (d.h. sie hätten die gleichen Fehlerbenachrichtigungs-Kriterien, usw.).

```
define hostdependency{
        hostgroup_name                  HOSTGROUP1,HOSTGROUP2
        dependent_hostgroup_name        HOSTGROUP3,HOSTGROUP4
        weitere Abhängigkeits-Direktiven ...
        }
```

Hostgruppen

Alle Hosts:
Wenn Sie eine Hostgruppe anlegen möchten, die alle Hosts aus Ihren Konfigurationsdateien als Mitglieder enthält, können Sie einen Platzhalter in der *members*-Direktive benutzen. Die folgende Definition würde eine Hostgruppe namens *HOSTGROUP1* erstellen, die **alle Hosts** aus Ihren Konfigurationsdateien als Mitglieder enthält.

```
define hostgroup{
        hostgroup_name          HOSTGROUP1
        members                 *
        weitere Hostgruppen-Direktiven ...
        }
```

Sicherheitsüberlegungen

Hoch zu: Inhalt

Einführung

Dies ist als ein kurzer Überblick einiger Dinge gedacht, die Sie bei der Installation von Nagios im Hinterkopf behalten sollten, um es in einer sicheren Weise aufzusetzen.

Ihr Überwachungsrechner sollte als eine Hintertür in Ihre anderen System betrachtet werden. In vielen Fällen wird dem Nagios-Rechner der Zugriff auf Firewalls gewährt, um entfernte Server zu überwachen. In den meisten Fällen ist die Abfrage von verschiedenen Informationen der entfernten Server erlaubt. Überwachenden Servern wird ein gewisses Maß an Vertrauen entgegen gebracht, damit sie entfernte Systeme abfragen können. Das bietet einem potenziellen Angreifer eine attraktive Hintertür zu Ihren Systemen. Ein Angreifer könnte es einfacher haben, in Ihre Systeme einzudringen, wenn er zuerst den Überwachungsserver kompromittiert. Das trifft besonders dann zu, wenn Sie gemeinsame SSH-Schlüssel nutzen, um entfernte Systeme zu überwachen.

Wenn ein Eindringling in der Lage ist, Prüfergebnisse oder externe Befehle an den Nagios-Daemon zu erteilen, hat er die Möglichkeit, falsche Überwachungsdaten zu übertragen, Sie mit falschen Benachrichtigungen auf die Palme bringen oder Eventhandler-Scripte auszulösen. Wenn Sie Eventhandler-Scripte haben, die Services neu starten, Strom unterbrechen usw., dann kann das ziemlich problematisch sein.

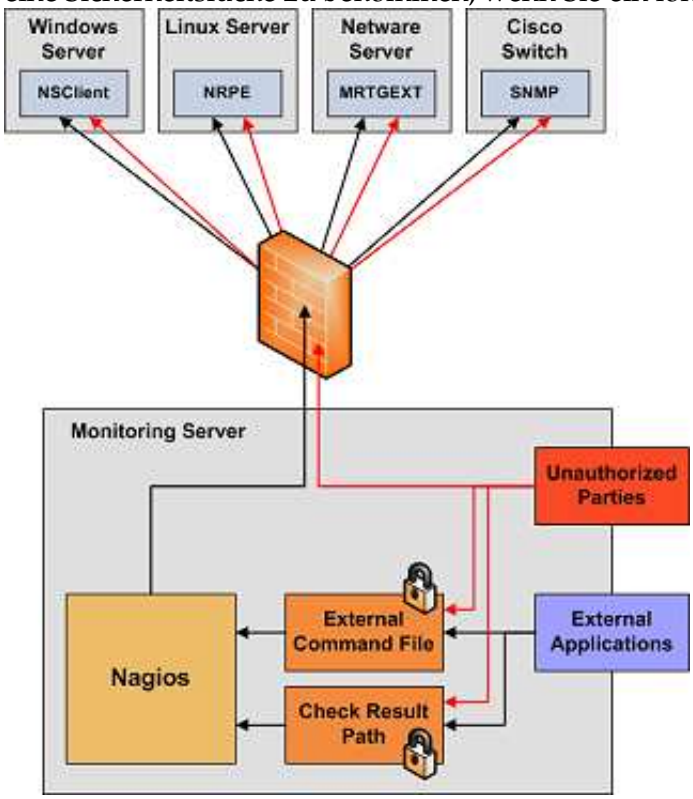
Ein weiterer zu beachtender Bereich ist die Möglichkeit von Eindringlingen, Überwachungsdaten (Statusinformationen) zu belauschen, während sie über den Draht gehen. Wenn Übertragungskanäle nicht verschlüsselt sind, können Angreifer durch Beobachtung Ihrer Überwachungsdaten wertvolle Informationen gewinnen. Nehmen Sie als Beispiel die folgende Situation: ein Angreifer belauscht für eine gewisse Zeit die Überwachungsdaten und analysiert die typische CPU- und Plattenauslastung Ihrer Systeme zusammen mit der Zahl der Benutzer, die typischerweise angemeldet sind. Der Angreifer ist dann in der Lage, die beste Zeit für die Kompromittierung eines Systems und dessen Ressourcen (CPU usw.) zu ermitteln, ohne bemerkt zu werden.

Hier sind einige Hinweise, wie Sie Ihre Systeme sichern können, wenn Sie eine Nagios-basierte Überwachungslösung implementieren...

Optimale Verfahren

1. **Benutzen Sie eine eigene Überwachungs-Box**. Ich würde empfehlen, dass Sie einen Server benutzen, der nur für die Überwachung (und ggf. andere administrative Aufgaben) vorgesehen ist. Schützen Sie Ihren Überwachungsserver, als wäre es einer der wichtigsten Server Ihres Netzwerks. Halten Sie die laufenden Services auf einem Minimum und beschränken Sie den Zugang durch

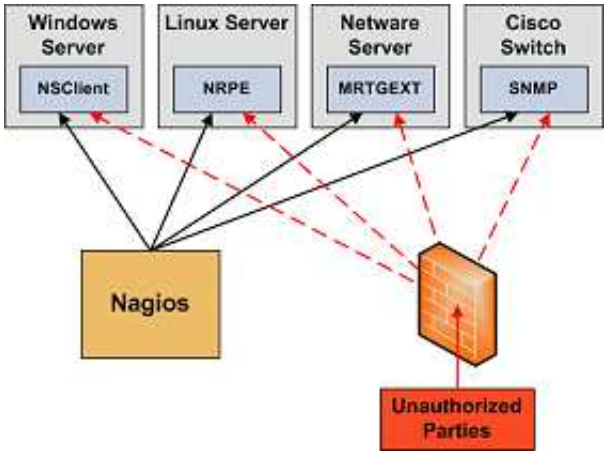
TCP-Wrapper, Firewalls usw. Weil der Nagios-Rechner berechtigt ist, mit Ihren Servern zu reden und vielleicht durch Ihre Firewalls zu gehen, kann es ein Sicherheitsrisiko sein, wenn Sie Benutzern Zugang zu Ihrem Überwachungsserver gewähren. Bedenken Sie, dass es einfacher ist, root-Zugang über eine Sicherheitslücke zu bekommen, wenn Sie ein lokales Benutzerkonto auf dem System haben.



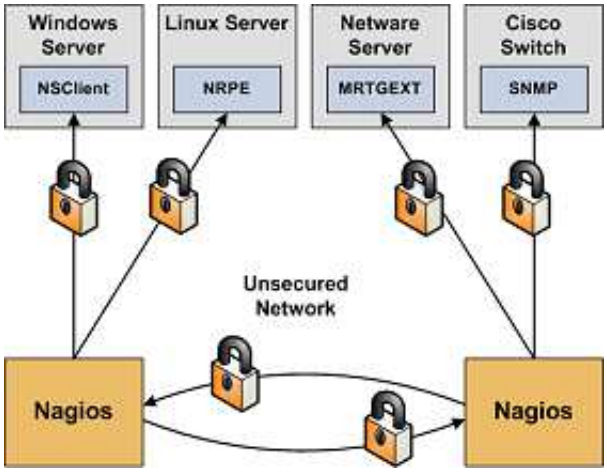
2. **Lassen Sie Nagios nicht als root laufen**. Nagios muss nicht als root laufen, also tun Sie es nicht. Sie können Nagios anweisen, die Berechtigungen nach dem Start zu "droppen" und mit Hilfe der nagios_user- und nagios_group-Direktiven in der Hauptkonfigurationsdatei unter anderen Benutzer- und/oder Gruppenberechtigungen zu laufen. Wenn Sie Eventhandler oder Plugins ausführen müssen, die Root-Berechtigungen benötigen, möchten Sie vielleicht sudo nutzen.
3. **Verriegeln Sie das Prüfergebnis-Verzeichnis**. Stellen Sie sicher, dass nur der *nagios*-Benutzer im check result path lesen und schreiben darf. Wenn andere Benutzer außer *nagios* (oder *root*) in diesem Verzeichnis schreiben dürfen, dann können sie falsche Host-/Service-Prüfergebnisse an den Nagios-Daemon senden. Dies kann zu Ärger (falschen Benachrichtigungen) oder Sicherheitsproblemen (ausgelösten Eventhandlern) führen.
4. **Verriegeln Sie das External Command File**. Wenn Sie externe Befehle aktivieren, dann stellen Sie sicher, dass Sie passende Berechtigungen für das */usr/local/nagios/var/rw*-Verzeichnis setzen. Nur der Nagios-Benutzer (normalerweise *nagios*) und der Web-Server-Benutzer (normalerweise *nobody*, *httpd*, *apache2* oder *www-data*) sollten Schreibberechtigung für das Commmand-File besitzen. Wenn Sie Nagios auf einer Maschine installiert haben, die der Überwachung und administrativen Aufgaben dient, dann sollte das ausreichen. Wenn Sie es auf einer allgemeinen- oder Multi-User-Maschine installiert haben (nicht empfohlen) und dem Web-Server-Benutzer Schreibberechtigung auf das Command-File geben, kann das ein Sicherheitsproblem sein. Sie wollen schließlich nicht, dass jeder Benutzer auf Ihrem System Nagios über das External-Command-File kontrollieren kann. In diesem Fall würde ich raten, nur dem *nagios*-Benutzer Schreibberechtigung zu erlauben und etwas wie CGIWrap zu benutzen, um die CGIs als *nagios* statt als *nobody* laufen zu lassen.
5. **Fordern Sie Authentifizierung bei den CGIs**. Ich empfehle dringend Authentifizierung für den Zugriff auf die CGIs. Sobald Sie das tun, lesen Sie die Dokumentation zu Standardberechtigungen von authentifizierten Kontakten und autorisieren Sie bestimmte Kontakte für zusätzliche Rechte nur, wenn es nötig ist. Eine Anleitung zur Einrichtung von Authentifizierung und Autorisierung finden Sie hier. Wenn Sie mit der use_authentication-Direktive die Authentifizierung in der CGI-Konfigurationsdatei deaktivieren, wird das command CGI das Schreiben jeglicher Befehle in

das external command file verweigern. Sie wollen schließlich nicht, dass alle Welt in der Lage ist, Nagios zu kontrollieren, oder?

6. **Benutzen Sie absolute Pfade in Befehlsdefinitionen**. Wenn Sie Befehle definieren, benutzen Sie den *absoluten Pfad* (keinen relativen) für Scripte oder Programm, die Sie ausführen.
7. **Verstecken Sie sensible Daten mit $USERn$-Makros**. Die CGIs lesen die Hauptkonfigurationsdatei und die Objekt-Konfigurationsdatei(en), so dass Sie dort keine sensiblen Informationen (Benutzernamen, Passwörter, usw.) ablegen sollten. Wenn Sie Benutzernamen und/oder Passwörter in einer Befehlsdefinition angeben müssen, dann nutzen Sie ein $USERn$-Makro, um sie zu verstecken. $USERn$-Makros werden in einer oder mehreren Ressourcen-Dateien definiert. Die CGIs werden nicht versuchen, den Inhalt von Ressourcen-Dateien zu lesen, so dass Sie restriktivere Berechtigungen (600 oder 660) dafür benutzen können. Betrachten Sie die Beispiel-*resource.cfg*-Datei im Basisverzeichnis der Nagios-Distribution für ein Beispiel, wie $USERn$-Makros zu definieren sind.
8. **Entfernen Sie gefährliche Zeichen aus Makros**. Benutzen Sie die illegal_macro_output_chars-Direktive, um gefährliche Zeichen aus den $HOSTOUTPUT$- , $SERVICEOUTPUT$-, $HOSTPERFDATA$- und $SERVICEPERFDATA$-Makros zu entfernen, bevor sie in Benachrichtigungen usw. benutzt werden. Gefährliche Zeichen kann alles sein, was ggf. durch die Shell interpretiert wird und dadurch eine Sicherheitslücke öffnet. Ein Beispiel dafür sind Backtick-Zeichen (‘) in den $HOSTOUTPUT$, $SERVICEOUTPUT$, $HOSTPERFDATA$ und /oder $SERVICEPERFDATA$-Makros, die es einem Angreifer erlauben, einen beliebigen Befehl als Nagios-Benutzer auszuführen (ein guter Grund, Nagios NICHT als root-Benutzer laufen zu lassen).
9. **Sicherer Zugang zu entfernten Agenten**. Verriegeln Sie den Zugang zu Agenten (NRPE, NSClient, SNMP, usw.) auf entfernten Systemen durch Firewalls, Zugangsliste usw. Sie wollen nicht, dass jeder Ihre Systeme nach Statusinformationen abfragt. Diese Informationen können durch einen Angreifer genutzt werden, um entfernte Eventhandler-Scripte auszuführen oder die beste Zeit zu ermitteln, um nicht beobachtet zu werden.



10. **Sichere Kommunikationskanäle**. Stellen Sie sicher, dass Sie die Kommunikationskanäle zwischen verschiedenen Nagios-Installationen und Ihren Überwachungskanälen verschlüsseln, wann immer möglich. Sie wollen nicht, dass jemand Statusinformationen belauscht, die über Ihr Netzwerk gehen. Diese Informationen können durch einen Angreifer genutzt werden, um die besten Zeit für einen unbeobachteten Zugang zu ermitteln.


Windows Server
NSClient
Linux Server
NRPE
Netware Server
MRTGEXT
Cisco Switch
SNMP
Unsecured Network
Nagios
Nagios

Verbesserte CGI-Sicherheit und Authentifizierung

Hoch zu: Inhalt

Siehe auch: Sicherheitsüberlegungen, Konfigurationsüberblick

Einführung

Dies ist als eine Einführung für die Implementierung stärkerer Authentifizierung und Server-Sicherheit bezogen auf das CGI-Web-Interface gedacht.

Es gibt viele Wege, die Sicherheit Ihres Überwachungs-Servers und des Nagios-Umfeldes zu verbessern. Dies sollte nicht als das Ende aller Bemühungen angesehen werden. Nehmen Sie es statt dessen als eine Einführung für einige der Techniken, die Sie nutzen können, um die Sicherheit Ihres System zu verstärken. Wie immer sollten Sie forschen und die besten Techniken nutzen, die verfügbar sind. Behandeln Sie Ihren Überwachungs-Server, als wäre es der wichtigste Server in Ihrem Netzwerk und Sie werden belohnt werden.

Zusätzliche Techniken

- **Stärkere Authentifizierung durch Digest Authentication**. Wenn Sie den Schnellstartanleitungen gefolgt sind, werden Sie wahrscheinlich Apaches Basic Authentication nutzen. "Basic Authentication" wird Benutzer und Password bei jedem HTTP-Request im Klartext übertragen. Ziehen Sie eine sicherere Authentifierungsmethode wie z.B. Digest Authentication in Betracht, die aus Ihrem Benutzernamen und Passwort einen MD5-Hash erzeugt, der bei jeder Anfrage gesendet wird.
- **Erzwingen von TLS/SSL für jede Web-Kommunikation**. Apache bietet TLS/SSL durch das mod_ssl-Modul. TLS/SSL bietet einen sicheren Tunnel zwischen Client und Server, der Abhören und Verfälschung durch starke publickey/privatekey-Kryptographie verhindert.
- **Beschränken Sie Apache mit Hilfe von Zugangskontrollen**. Überlegen Sie, ob Sie den Zugang zur Nagios-Box auf Ihre IP-Adresse, IP-Adressbereich oder IP-Subnetz beschränken. Wenn Sie Zugang von außen auf Ihr Netzwerk benötigen, können Sie VPN und SSH-Tunnel nutzen. Es ist einfach, den Zugang zu Ihrem System auf HTTP/HTTPS zu begrenzen.

Implementieren der Digest Authentication

Die Implementierung der Digest Authentication ist einfach. Dazu müssen Sie den neuen Typ der Passwort-Datei mit dem 'htdigest'-Tool anlegen, dann die Apache-Konfiguration für Nagios anpassen (typischerweise /etc/httpd/conf.d/nagios.conf).

Legen Sie eine neue Passwort-Datei mit dem 'htdigest'-Tool an. Den Unterschied, den Sie feststellen werden, wenn Sie mit dem 'htpasswd'-Tool vertraut sind, ist die Anforderung, ein 'Realm'-Parameter anzugeben. In diesem Fall bezieht sich 'realm' auf den Wert der 'AuthName'-Direktive in der Apache-Konfiguration.

```
htdigest -c /usr/local/nagios/etc/.digest_pw "Nagios Access" nagiosadmin
```

Als nächstes editieren Sie die Apache-Konfigurationsdatei für Nagios (typischerweise /etc/httpd/conf.d/nagios.conf) mit Hilfe des folgenden Beispiels:

```
## BEGIN APACHE CONFIG SNIPPET - NAGIOS.CONF
ScriptAlias /nagios/cgi-bin "/usr/local/nagios/sbin"
<Directory "/usr/local/nagios/sbin">
   Options ExecCGI
   AllowOverride None
   Order allow,deny
   Allow from all
   AuthType Digest
   AuthName "Nagios Access"
   AuthUserFile /usr/local/nagios/etc/.digest_pw
   Require valid-user
</Directory>

Alias /nagios "/usr/local/nagios/share"
<Directory "/usr/local/nagios/share">
   Options None
   AllowOverride None
   Order allow,deny
   Allow from all
   AuthType Digest
   AuthName "Nagios Access"
   AuthUserFile /usr/local/nagios/etc/.digest_pw
   Require valid-user
</Directory>
## END APACHE CONFIG SNIPPETS
```

Danach starten Sie den Apache-Service, damit die neuen Einstellungen aktiv werden können.

```
/etc/init.d/httpd restart
```

Implementieren erzwungener TLS/SSL-Kommunikation

Stellen Sie sicher, dass Sie Apache und OpenSSL installiert haben. Normalerweise sollten Sie mod_ssl-Unterstützung haben. Falls Sie trotzdem Schwierigkeiten haben, finden Sie ggf. Hilfe durch das Lesen von Apaches TLS/SSL Encryption Documentation.

Als nächstes prüfen Sie durch den Aufruf des Nagios-Web-Interfaces über HTTPS (https://your.domain/nagios), dass die TLS/SSL-Unterstützung funktioniert. Wenn es funktioniert, können Sie mit den nächsten Schritten fortfahren, die die Nutzung von HTTPS erzwingen und alle HTTP-Anfragen an das Nagios-Web-Interface blockiert. Wenn Sie Schwierigkeiten haben, lesen Sie bitte Apaches TLS/SSL Encryption Documentation und nutzen Sie Google für die Suche nach Lösungen zu Ihrer Apache-Installation.

Danach editieren Sie die Apache-Konfigurationsdatei für Nagios (typischerweise /etc/httpd/conf.d/nagios.conf) und fügen Sie den 'sbin'- und 'share'-Verzeichnissen die 'SSLRequireSSL'-Direktive hinzu.

```
## BEGIN APACHE CONFIG SNIPPET - NAGIOS.CONF
ScriptAlias /nagios/cgi-bin "/usr/local/nagios/sbin"
<Directory "/usr/local/nagios/sbin">
   ...
```

```
   SSLRequireSSL
   ...
</Directory>

Alias /nagios "/usr/local/nagios/share"
<Directory "/usr/local/nagios/share">
   ...
   SSLRequireSSL
   ...
</Directory>
## END APACHE CONFIG SNIPPETS
```

Danach starten Sie den Apache-Service, damit die neuen Einstellungen aktiv werden können.

```
/etc/init.d/httpd restart
```

Implementieren von IP-Subnetz-Beschränkung

Das folgende Beispiel zeigt, wie Sie den Zugang auf die Nagios-CGIs auf eine bestimmte IP-Adresse, einen IP-Adressbereich oder ein IP-Subnetz mit Hilfe von Apaches Access Controls beschränken.

Danach editieren Sie die Apache-Konfigurationsdatei für Nagios (typischerweise /etc/httpd/conf.d/nagios.conf) und fügen Sie die 'Allow'-, 'Deny'- und 'Order'-Direktiven hinzu. Dazu folgendes Beispiel:

```
## BEGIN APACHE CONFIG SNIPPET - NAGIOS.CONF
ScriptAlias /nagios/cgi-bin "/usr/local/nagios/sbin"
<Directory "/usr/local/nagios/sbin">
   ...
   AllowOverride None
   Order deny,allow
   Deny from all
   Allow from 127.0.0.1 10.0.0.25              # Allow single IP addresses
   Allow from 10.0.0.0/255.255.255.0           # Allow network/netmask pair
   Allow from 10.0.0.0/24                      # Allow network/nnn CIDR spec
   ...
</Directory>

Alias /nagios "/usr/local/nagios/share"
<Directory "/usr/local/nagios/share">
   ...
   AllowOverride None
   Order deny,allow
   Deny from all
   Allow from 127.0.0.1 10.0.0.25              # Allow single IP addresses
   Allow from 10.0.0.0/255.255.255.0           # Allow network/netmask pair
   Allow from 10.0.0.0/24                      # Allow network/nnn CIDR spec
   ...
</Directory>
## END APACHE CONFIG SNIPPET
```

Wichtige Anmerkungen

- **Digest Authentication sendet Daten im Klartext, aber nicht Ihren Benutzernamen und Passwort**.
- **Digest Authentication ist nicht ganz so gut unterstützt wie Basic Authentication**.
- **TLS/SSL hat das Potential für einen "Man-in-the-middle-Angriff"**. MITM-Angriffe machen verletzbar, wenn ein Angreifer in der Lage ist, sich zwischen Server und Client zu schieben wie bei einem Phishing-Angriff, ISP-Monitoring oder Resignierung von Unternehmens-LAN Firewall-Zertifikaten. Bitte machen Sie sich kundig zu Zertifikats-Verifikation!
- **Apache Access Controls schützen nur die HTTP/HTTPS-Protokolle**. Sehen Sie sich IPtables für eine starke systemweite Firewall-Kontrolle an.

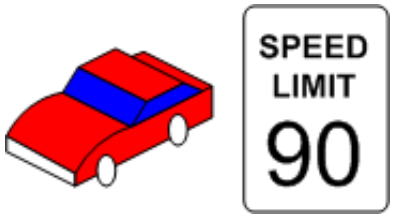
- **Am wichtigsten: Sicherheit ist ein bewegliches Ziel, also bleiben Sie informiert und forschen** Sie! Vielleicht durch das Anhören eines Podcasts wie z.B. "Security Now!".

Nagios für maximale Leistung optimieren

Hoch zu: Inhalt

Siehe auch: Verbesserungen für große Installationen, Schnellstartoptionen, graphische Darstellung von Performance-Informationen

Einführung



Jetzt haben Sie Nagios endlich eingerichtet und lauffähig und nun wollen Sie wissen, wie man ein wenig daran drehen kann. Die Leistung von Nagios zu optimieren kann notwendig sein, wenn Sie eine große Zahl (> 1.000) von Hosts und Services haben. Hier ein paar Dinge, nach denen Sie schauen können, um Nagios zu optimieren...

Optimierungshinweise:

1. **Stellen Sie Performance-Statistiken mit MRTG dar**. Um zu verfolgen, wie die Last Ihrer Nagios-Installation aussieht und welche Auswirkungen Ihre Konfigurationsänderungen darauf haben, sollten Sie verschiedene wichtige Statistiken mit MRTG darstellen. Das ist wirklich sehr, sehr sinnvoll, wenn es um die Leistungsoptimierung einer Nagios-Installation geht. Informationen, wie das zu tun ist, finden Sie hier.
2. **Benutzen Sie "Verbesserungen für große Installationen"** (large installation tweaks). Das Aktivieren der use_large_installation_tweaks-Option kann Ihnen bessere Leistung bringen. Lesen Sie hier mehr darüber, was diese Option tut.
3. **Deaktivieren Sie Umgebungs-Makros**. Makros werden Prüfungen, Benachrichtigungen, Eventhandlern usw. normalerweise über Umgebungsvariablen zur Verfügung gestellt. Das kann in einer großen Nagios-Installation zu einem Problem werden, weil es zusätzlichen Speicher (und wichtiger) mehr CPU verbraucht. Wenn Ihre Scripte nicht über Umgebungsvariablen auf Makros zugreifen (d.h., wenn Sie alle benötigen Makros in der Kommandozeile übergeben), dann brauchen Sie dieses Feature nicht. Sie können über die enable_environment_macros-Option einstellen, ob Makros als Umgebungsvariablen verfügbar sind.
4. **Prüfergebnis-Ernterhythmus** (Check Result Reaper Frequency). Die check_result_reaper_frequency-Variable legt fest, wie oft Nagios prüfen soll, ob Host- und Service-Ergebnisse verarbeitet werden müssen. Die maximale Zeit, die es zur Verarbeitung solcher Ergebnisse benötigen darf, ist durch die maximale Erntezeit (max reaper time) festgelegt (siehe unten). Wenn Ihr Ernterhythmus zu hoch (zu selten) ist, könnten Sie hohe Latenzzeiten für Host- und Service-Prüfungen sehen.
5. **maximale Erntezeit** (Max Reaper Time). Die max_check_result_reaper_time-Variable legt die maximale Zeit fest, die der Nagios-Daemon für die Verarbeitung der Ergebnisse von Host- und Service-Prüfungen verbringen darf, bevor er sich anderen Dingen zuwendet - wie z.B. dem Ausführen von neuen Host- und Service-Prüfungen. Ein zu hoher Wert kann zu hohen Latenzzeiten bei Ihren Host- und Service-Prüfungen führen. Ein zu niedriger Wert kann den gleichen Effekt

haben. Wenn Sie zu hohe Latenzzeiten haben, dann passen Sie diesen Wert an und sehen Sie, welchen Effekt das hat. Graphisch dargestellte Statistiken helfen Ihnen bei der Auswertung der Auswirkungen.

6. **Anpassen der Pufferwerte**. Gegebenenfalls müssen Sie den Wert der external_command_buffer_slots-Option anpassen. Die graphische Analyse mit MRTG (siehe oben) zeigt Ihnen, welche Werte Sie für diese Option nutzen sollten.
7. **Prüfen Sie Service-Latenzzeiten, um den besten Wert für die maximale Anzahl von gleichzeitigen Prüfungen zu ermitteln**. Nagios kann die Anzahl von gleichzeitig ausgeführten Prüfungen durch die max_concurrent_checks-Option begrenzen. Das ist gut, weil es Ihnen etwas Kontrolle darüber gibt, wieviel Last Nagios auf Ihrem Überwachungsrechner erzeugt, aber es kann auch die Dinge verlangsamen. Wenn Sie für die Mehrzahl Ihrer Service-Prüfungen hohe Latenzzeiten sehen (> 10 oder 15 Sekunden), dann enthalten Sie Nagios Prüfungen vor, die es braucht. Das ist nicht der Fehler von Nagios - es ist Ihrer. Unter idealen Bedingungen hätten alle Service-Prüfungen eine Latenzzeit von 0, was bedeutet, dass alle Prüfungen zu der Zeit stattfinden, für die sie geplant sind. Allerdings ist es normal, dass einige Prüfungen kleine Latenzzeiten haben. Ich würde empfehlen, die niedrigste Zahl der meisten gleichzeitigen Prüfungen zu nehmen, wenn Sie Nagios mit der **-s**-Option starten und diesen Wert zu verdoppeln. Erhöhen Sie diesen Wert dann soweit, bis die durchschnittlichen Latenzzeiten für Service-Prüfungen ziemlich niedrig ist. Mehr Informationen zur Planung von Service-Prüfungen finden Sie hier.
8. **Nutzen Sie passive Prüfungen, wenn möglich**. Der nötige Overhead, um die Ergebnisse von passiven Service-Prüfungen zu verarbeiten, ist viel niedriger als bei "normalen" aktiven Prüfungen, also machen Sie Gebrauch von dieser Information, wenn Sie eine Menge von Services überwachen. Es sollte angemerkt werden, dass passive Prüfungen nur dann wirklich sinnvoll sind, wenn Sie irgendeine externe Applikation haben, die überwachen oder berichten kann; wenn also Nagios all die Arbeit machen muss, ist das nicht hilfreich.
9. **Vermeiden Sie interpretierte Plugins**. Etwas, was spürbar die Last Ihres Überwachungs-Hosts senkt, ist die Nutzung von kompilierten (C/C++, usw.) Plugins statt interpretierter Scripts (Perl, usw.). Während Perl und ähnliches einfach zu schreiben ist und gut läuft, kann die Tatsache, dass es bei jeder Ausführung kompiliert/interpretiert werden muss, zu einer spürbaren Steigerung der Last Ihres Überwachungs-Hosts führen, wenn Sie eine Menge von Service-Prüfungen haben. Wenn Sie Perl-Plugins nutzen wollen, dann überlegen Sie, ob Sie diese nicht mit perlcc(1) (einem Utility, das Teil der Standard-Perl-Distribution ist) zu einem richtigen Programm umwandeln oder Nagios mit eingebettetem Perl-Interpreter kompilieren (siehe unten).
10. **Nutzen Sie den eingebetteten Perl-Interpreter**. Wenn Sie eine Menge von Perl-Scripten für Prüfungen benutzen, dann werden Sie vielleicht feststellen, dass das Kompilieren des eingebetteten Perl-Interpreters (embedded Perl interpreter) in das Nagios-Binary die Dinge beschleunigt.
11. **Optimieren Sie Host-Prüfbefehle**. Wenn Sie Host-Zustände mit dem check_ping-Plugin prüfen, dann werden Sie feststellen, dass die Host-Prüfungen viel schneller durchgeführt werden, wenn Sie diese abbrechen. Statt einen *max_attempts*-Wert von 1 anzugeben und mit dem check_ping-Plugins 10 ICMP-Pakete an den Host zu schicken, wäre es viel schneller, den *max_attempts*-Wert auf 10 zu setzen und jedes Mal nur ein ICMP-Paket zu senden. Das liegt daran, dass Nagios den Zustand eines Hosts oft nach der Ausführung eines Plugins feststellen kann, so dass Sie die erste Prüfung so schnell wie möglich machen sollten. Diese Methode hat in einigen Situationen ihre Fallstricke (z.B. Hosts, die langsam reagieren, könnten als "down" angesehen werden), aber ich denke, dass Sie schnellere Host-Prüfungen sehen werden, wenn Sie sie benutzen. Eine weitere Möglichkeit wäre, statt check_ping ein schnelleres Plugin (z.B. check_fping) als *host_check_command* zu benutzen.
12. **Planen Sie regelmäßige Host-Prüfungen**. Regelmäßige Host-Prüfungen zu planen kann tatsächlich die Leistung von Nagios steigern. Das liegt an der Art, wie die Zwischenspeicher-Prüflogik (cached check logic) arbeitet (siehe unten). Vor Nagios 3 führten regelmäßige Host-Prüfungen zu einer großen Leistungseinbuße. Das ist nicht länger der Fall, weil Host-Prüfungen parallel stattfinden - genau wie Service-Prüfungen. Um regelmäßige Prüfungen eines Hosts zu planen, setzen Sie die *check_interval*-Direktive in der Host-Definition auf einen Wert größer als Null.
13. **Aktivieren Sie zwischengespeicherte Host-Prüfungsergebnisse** (cached host checks). Beginnend

mit Nagios 3 können Host-Prüfungen nach Bedarf von der Zwischenspeicherung (caching) profitieren. Host-Prüfungen nach Bedarf werden ausgeführt, wenn Nagios einen Service-Zustandswechsel feststellt. Diese Prüfungen nach Bedarf werden ausgeführt, wenn Nagios wissen will, ob der mit dem Service verbundene Host den Zustand gewechselt hat. Durch die Aktivierung von zwischengespeicherten Host-Prüfungsergebnissen können Sie die Leistung optimieren. In einigen Fällen könnte Nagios in der Lage sein, den alten/zwischengespeicherten Zustand des Hosts zu benutzen, statt eine Host-Prüfung auszuführen. Das kann die Dinge beschleunigen und die Last des Überwachungsservers reduzieren. Damit zwischengespeicherte Prüfungen effektiv sind, müssen Sie regelmäßige Prüfungen für Ihre Hosts planen (siehe oben). Mehr Informationen zu zwischengespeicherten Prüfungen finden Sie hier.

14. **Nutzen Sie keine agressiven Host-Prüfungen**. Solange Sie keine Probleme damit haben, dass Nagios Host-Erholungen nicht korrekt erkennt, würde ich empfehlen, die use_aggressive_host_checking-Option nicht zu aktivieren. Wenn diese Option abgeschaltet ist, werden Host-Prüfungen viel schneller ausgeführt, was zu schnellerer Ausführung von Service-Prüfungen führt. Allerdings können Host-Erholungen unter bestimmten Umständen übersehen werden, wenn sie ausgeschaltet ist. Wenn sich z.B. der Host erholt, aber alle mit ihm verbundenen Services in einem nicht-OK-Zustand bleiben (und nicht zwischen verschiedenen nicht-OK-Zuständen "kippeln"), dann könnte Nagios übersehen, dass sich der Host erholt hat. Einige wenige Leute könnten diese Option aktivieren, aber die Mehrheit nicht und ich würde empfehlen, sie nicht zu aktivieren, solange Sie nicht glauben, dass Sie sie benötigen...
15. **Optimierung externer Befehle**. Wenn Sie eine Menge externer Befehle verarbeiten (d.h. passive Prüfungen in einer verteilten Umgebung, dann wollen Sie vielleicht die command_check_interval-Variable auf **-1** setzen. Das bewirkt, dass Nagios so oft wie möglich auf externe Befehle prüft. Sie sollten außerdem überlegen, die Anzahl verfügbarer externer Befehlspuffer zu erhöhen. Puffer werden benutzt, um externe Befehle zu speichern, die (durch einen separaten Thread) aus dem external command file gelesen werden, bevor sie vom Nagios-Daemon verarbeitet werden. Wenn Ihr Nagios-Daemon eine Menge von passiven Prüfungen oder externen Befehlen empfängt, dann könnten Sie in eine Situation kommen, in der immer alle Puffer voll sind. Das führt zu blockierenden Kind-Prozessen (externe Scripte, NSCA-Daemon usw.), wenn sie versuchen, in das "external command file" zu schreiben. Ich würde sehr empfehlen, dass Sie die Nutzung von externen Befehlspuffern graphisch mit Hilfe von MRTG und dem nagiostats-Utility darstellen, wie es hier beschrieben ist, so dass Sie die typische externe Befehlspuffernutzung Ihrer Nagios-Installation sehen.
16. **Optimieren Sie die Hardware für maximale Leistung**. Hinweis: Hardware-Leistung sollte kein Thema sein, solange Sie nicht 1) Tausende von Services überwachen, 2) eine Menge von Nachverarbeitung von Performance-Daten usw. machen. Ihre Systemkonfiguration und Ihre Hardware-Ausstattung werden direkt beeinflussen, was Ihr Betriebssystem leistet, so dass sie beeinflussen, was Nagios leistet. Die häufigste Hardware-Optimierung betrifft die Festplatte(n). CPU und Speichergeschwindigkeit sind offensichtliche Faktoren, die die Leistung beeinflussen, aber der Plattenzugriff wird Ihr größter Flaschenhals sein. Speichern Sie Plugins, das Status-Log usw. nicht auf langsamen Platten (d.h. alte IDE-Platten oder NFS-Mounts). Wenn Sie sie haben, dann nutzen Sie UltraSCSI- oder schnelle IDE-Platten. Ein wichtiger Hinweis für IDE/Linux-Benutzer ist, dass viele Linux-Installationen nicht versuchen, den Plattenzugriff zu optimieren. Wenn Sie die Plattenzugriffsparameter nicht ändern (z.B. mit einem Utility wie **hdparam**), werden Sie eine **Menge** der schnellen Features der neuen IDE-Platten verlieren.

Schnellstart-Optionen

Hoch zu: Inhalt

Siehe auch: Leistungsoptimierung, Large Installation Tweaks

Einführung

Es gibt einige Dinge, die Sie tun können, um die Zeit zu verringern, die Nagios zum (Neu-)Start benötigt. Diese Beschleunigung umfasst u.a. Änderungen bei der Verarbeitung Ihrer Konfigurationsdateien.

Diese Techniken zu benutzen ist besonders dann sinnvoll, wenn bei Ihnen einer oder mehrere der folgenden Punkte zutreffen:

- große Konfigurationen
- komplexe Konfigurationen (massiver Einsatz von Template-Features)
- Installationen, bei denen häufige Neustarts notwendig sind

Hintergrund

Bei jedem (erneuten) Start von Nagios müssen die Konfigurationsdateien verarbeitet werden, bevor die Überwachung beginnen kann. Dieser Konfigurationsanlaufprozess umfasst eine Reihe von Schritten:

- Lesen der Konfigurationsdateien
- Auflösen von Template-Definitionen
- "Recombobulating" Ihrer Objekte (mein Begriff für die verschiedenen Arten von Arbeiten, die auftreten)
- duplizieren von Objektdefinitionen
- vererben von Objekteigenschaften
- sortieren Ihrer Objektdefinitionen
- überprüfen der Objektbeziehungsintegrität
- prüfen von zirkulären Pfaden
- und mehr...

Einige dieser Schritte können ziemlich zeitintensiv sein, wenn Sie große oder komplexe Konfigurationen haben. Gibt es einen Weg, einen dieser Schritte zu beschleunigen? Ja!

Bewertung von Anlaufzeiten

Bevor wir weitermachen, die Dinge zu beschleunigen, müssen wir sehen was möglich ist und ob wir uns mit der ganzen Sache beschäftigen sollten oder nicht. Das ist einfach - starten Sie Nagios mit der **-s**-Option, um Zeiten und Planungsinformationen zu bekommen.

Ein Beispiel für die Ausgabe (gekürzt, um nur relevante Teile zu zeigen) sehen Sie nachfolgend. In diesem Beispiel nutze ich eine Nagios-Konfigurations mit 25 Host und etwas mehr als 10.000 Services.

```
/usr/local/nagios/bin/nagios -s /usr/local/nagios/etc/nagios.cfg

Nagios 3.0-prealpha
Copyright (c) 1999-2007 Ethan Galstad (http://www.nagios.org)
Last Modified: 01-27-2007
License: GPL

Timing information on object configuration processing is listed
below.  You can use this information to see if precaching your
object configuration would be useful.

Object Config Source: Config files (uncached)

OBJECT CONFIG PROCESSING TIMES      (* = Potential for precache savings with -u option)
---------------------------------
Read:                 0.486780 sec
Resolve:              0.004106 sec  *
Recomb Contactgroups: 0.000077 sec  *
Recomb Hostgroups:    0.000172 sec  *
Dup Services:         0.028801 sec  *
Recomb Servicegroups: 0.010358 sec  *
Duplicate:            5.666932 sec  *
Inherit:              0.003770 sec  *
Recomb Contacts:      0.030085 sec  *
Sort:                 2.648863 sec  *
Register:             2.654628 sec
Free:                 0.021347 sec
                      ============
TOTAL:                11.555925 sec  * = 8.393170 sec (72.63%) estimated savings


Timing information on configuration verification is listed below.

CONFIG VERIFICATION TIMES          (* = Potential for speedup with -x option)
---------------------------------
Object Relationships: 1.400807 sec
Circular Paths:       54.676622 sec  *
Misc:                 0.006924 sec
                      ============
TOTAL:                56.084353 sec  * = 54.676622 sec (97.5%) estimated savings
```

Okay, lassen Sie uns ansehen was passiert ist. Wenn wir die Summen ansehen, dauerte es ungefähr **11,6** Sekunden, die Konfigurationsdateien zu verarbeiten und weitere **56** Sekunden, die Konfigurations zu verifizieren. Das bedeutet, dass es fast **68 Sekunden** dauert, bis die erste Überwachung beginnen kann! Das ist nicht akzeptierbar, wenn ich Nagios ziemlich regelmäßig neu starten muss.

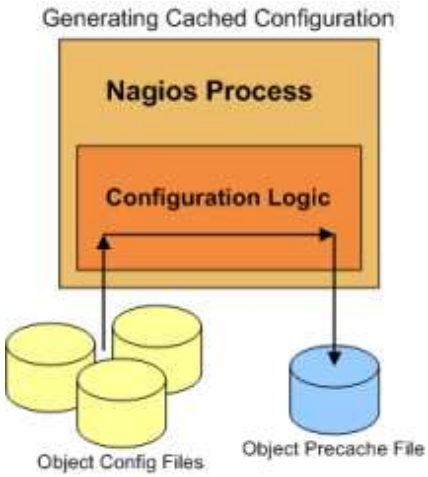
Was kann ich daran ändern? Werfen Sie einen erneuten Blick auf die Ausgabe und Sie sehen, dass Nagios schätzt, dass ich etwa **8,4** Sekunden bei der Verarbeitung der Konfiguration und weitere **54,7** bei der Verifizierung einsparen kann. Nagios denkt, dass ich **63 Sekunden** der normalen Anlaufzeit sparen kann, wenn einige Optimierungen vorgenommen werden.

Whow! Von **68 Sekunden** auf gerade mal **5 Sekunden**? Yep, lesen Sie weiter, um zu sehen, wie das geht.

Pre-Caching der Objektkonfiguration

Nagios kann einige Zeit beim analysieren Ihrer Konfigurationsdateien verbringen, besonders dann, wenn Sie Template-Features wie z.B. Vererbung usw. nutzen. Um die Zeit der Analyse Ihrer Konfiguration zu verringern, können Sie Nagios veranlassen, Ihre Konfigurationsdateien für die Zukunft vorzuverarbeiten (pre-process) und vor-zwischenzuspeichern (pre-cache).

Wenn Sie Nagios mit der **-p**-Kommandozeilenoption starten, wird Nagios Ihre Konfigurationsdateien einlesen, verarbeiten und sie in einer vor-zwischengespeicherten (pre-cached) (durch die precached_object_file-Direktive angegebene) Konfigurationsdatei sichern. Diese Konfigurationsdatei enthält vorverarbeitete Konfigurationseinträge, die Nagios in Zukunft einfacher/schneller verarbeiten kann.

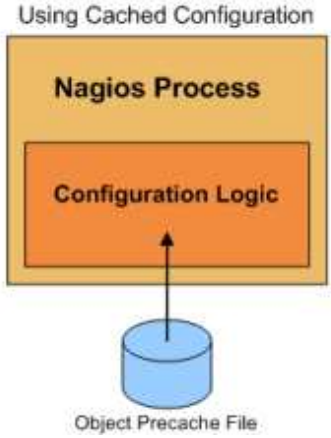


Sie müssen die **-p**-Kommandozeilenoption zusammen mit der **-v** oder **-s**-Kommandozeilenoption benutzen, wie nachfolgend gezeigt. Dies stellt sicher, dass Ihre Konfiguration überprüft wird, bevor die precached-Datei erstellt wird.

```
/usr/local/nagios/bin/nagios -pv /usr/local/nagios/etc/nagios.cfg
```

Die precached-Konfigurationsdatei wird wahrscheinlich um einiges größer sein als die Summe Ihrer Objektkonfigurationsdateien. Das ist normal und beabsichtigt.

Sobald die precached-Objektkonfigurationdatei erstellt wurde, können Sie Nagios starten und mit der **-u**-Kommandozeilenoption angeben, dass diese Datei statt Ihrer Konfigurationsdatei(en) benutzt werden soll.




```
/usr/local/nagios/bin/nagios -ud /usr/local/nagios/etc/nagios.cfg
```

! Wenn Sie Ihre Konfigurationsdateien ändern, müssen Sie diese erneut überprüfen und die precached-Konfigurationsdatei neu erstellen, bevor Sie Nagios erneut starten. Wenn Sie die precached-Konfigurationsdatei nicht neu generieren, wird Nagios Ihre alte Konfiguration benutzen, weil die precached-Konfigurationsdatei gelesen wird und nicht Ihre geänderten Konfigurationsdateien.

Überspringen der Test von zirkulären Pfaden

Der zweite (und zeitintensivste) Teil der Konfigurationsanlaufphase ist die Prüfung auf zirkuläre Pfade. Im obigen Beispiel dauerte es fast eine Minute, um diesen Schritt der Konfigurationsprüfung auszuführen.

Was ist diese zirkuläre-Pfad-Prüfung und warum dauert sie so lange? Die zirkuläre-Pfad-Prüfung soll verhindern, dass Sie zirkuläre Pfade in Ihren Host-, Host-Abhängigkeits- oder Service-Abhängigkeitsdefinitionen haben. Wenn ein zirkulärer Pfad in Ihren Konfigurationsdateien existiert, könnte Nagios in einer Deadlock-Situation enden. Der wahrscheinlichste Grund dafür, dass die Prüfung so lange dauert, dürfte darin liegen, dass ich keinen effizienten Algorithmus benutze. Ein effizienterer Algorithmus wäre daher willkommen. Wink: das bedeutet, dass alle Absolventen der Computerwissenschaften, die mir ihre Thesen zu Nagios gemailt haben, ein wenig Code liefern könnten. :-)

Wenn Sie die Prüfung auf zirkuläre Pfade überspringen möchten, wenn Sie Nagios starten, dann fügen Sie die **-x**-Option wie folgt hinzu:

```
/usr/local/nagios/bin/nagios -xd /usr/local/nagios/etc/nagios.cfg
```

! Es ist von äußerster Wichtigkeit, dass Sie Ihre Konfiguration überprüfen, bevor Sie Nagios (erneut) starten, wenn Sie auf die Prüfung auf zirkuläre Pfade verzichten. Wenn Sie es nicht tun, kann dies zu Deadlocks führen. Sie sind gewarnt worden.

Alles zusammenfassen

Folgen Sie diesen Schritten, wenn Sie mögliche Beschleunigungen durch pre-Caching Ihrer Konfiguration und überspringen der Prüfungen auf zirkuläre Pfade nutzen wollen.

1. Überprüfen Sie Ihre Konfiguration und legen Sie die precache-Datei mit den folgenden Befehlen an:

```
/usr/local/nagios/bin/nagios -vp /usr/local/nagios/etc/nagios.cfg
```

2. Stoppen Sie Nagios, wenn es momentan läuft.

3. Starten Sie Nagios wie folgt, um die precached-Konfigurationsdatei zu nutzen und auf Prüfung auf zirkuläre Pfade zu überspringen:

```
/usr/local/nagios/bin/nagios -uxd /usr/local/nagios/etc/nagios.cfg
```

4. Wenn Sie in Zukunft Ihre Konfigurationsdateien verändern und Nagios erneut starten müssen, damit diese Änderungen aktiv werden, dann wiederholen Sie Schritt 1, um Ihre Konfiguration erneut zu überprüfen und die precached-Konfigurationsdatei zu erstellen. Sobald das getan ist, können Sie Nagios über das Web-Interface oder durch das Senden eines SIGHUP-Signals neustarten. Wenn Sie die precached-Objektdatei nicht neu erstellen, wird Nagios wieder Ihre alte Konfiguration benutzen, weil es die precached-Datei liest statt Ihrer Konfigurationsdateien.

5. Das war's! Erfreuen Sie sich am Geschwindigkeitsgewinn beim Start.

Large Installation Tweaks

Hoch zu: Inhalt

Siehe auch: Leistungsoptimierung, Schnellstartoptionen

Einführung

Benutzer mit großen Nagios-Installation können von der use_large_installation_tweaks-Konfigurationsoption profitieren. Das Aktivieren dieser Option erlaubt es dem Nagios-Daemon, bestimmte Abkürzungen zu nehmen, die in geringerer Systembelastung und besserer Leistung resultieren.

Effekte

Wenn Sie die use_large_installation_tweaks-Option in Ihrer Nagios-Hauptkonfigurationsdatei aktivieren, werden mehrere Anpassungen gemacht, wie der Nagios-Daemon arbeitet:

1. **Keine Zusammenfassungsmakros in Umgebungsvariablen** - Die Zusammenfassungsmakros werden Ihnen nicht als Umgebungsvariablen zur Verfügung stehen. Die Berechnung der Werte dieser Makros kann in großen Konfigurationen ziemlich zeitintensiv sein, so dass sie nicht als Umgebungsvariablen zur Verfügung stehen, wenn Sie diese Option benutzen. Zusammenfassungsmakros sind weiterhin als reguläre Makros verfügbar, wenn Sie diese Ihren Scripts als Parameter übergeben.
2. **Unterschiedliche Speicherbereinigung** - Normalerweise wird Nagios den allokierten Speicher in Kind-Prozessen freigeben, bevor sie enden. Dies ist wahrscheinlich die beste Vorgehensweise, aber vielleicht in großen Installationen unnötig, weil die meisten Betriebssysteme selbst darauf achten, allokierten Speicher freizugeben, wenn Prozesse enden. Das Betriebssystem neigt dazu, belegten Speicher schneller freizugeben, als Nagios das kann, so dass Nagios nicht versucht, Speicher in Kind-Prozessen freizugeben, wenn Sie diese Option aktivieren.
3. **Weniger fork()** - Normalerweise wird Nagios zweimal fork() aufrufen, wenn es Host- und Service-Prüfungen ausführt. Das wird getan, um (1) ein hohes Maß an Resistenz sicherzustellen gegen Plugins, die fehlschlagen und einen SegFault erzeugen und (2) dafür sorgen, dass das Betriebssystem sich um die Bereinigung der Enkel-Prozesse kümmert, sobald sie enden. Der zusätzliche fork() ist nicht wirklich nötig, so dass er übersprungen wird, wenn Sie diese Option aktivieren. Als Ergebnis werden Kind-Prozesse von Nagios selbst bereinigt (anstatt diese Aufgabe dem Betriebssystem zu überlassen). Dieses Feature sollte für spürbare Lasteinsparungen in Ihrer Nagios-Installation sorgen.

Nutzung des Nagiostats-Utilitys

Hoch zu: Inhalt

Siehe auch: Graphing Performance Info, Leistungsoptimierung

Einführung

Ein Utility namens **nagiostats** ist in der Nagios-Distribution enthalten. Es wird zusammen mit dem Nagios-Daemon kompiliert und installiert. Das nagiostats-Utility liefert Ihnen verschiedene Informationen zu einem laufenden Nagios-Prozess, die sehr hilfreich bei der Leistungsoptimierung sein können. Sie können Informationen in einem menschlich-lesbaren oder im MRTG-kompatiblen Format erhalten.

Gebrauchshinweise

SIe können das *nagiostats*-Utility mit der **--help**-Option starten, um Gebrauchshinweise zu bekommen.

menschlich-lesbare Ausgabe

Um menschlich-lesbare Informationen zur Leistung eines laufenden Nagios-Prozesses zu erhalten, starten Sie das *nagiostats*-Utility mit dem **-c**-Kommandozeilenargument, um die Position Ihrer Hauptkonfigurationsdatei wie folgt anzugeben:

```
[nagios@lanman ~]# /usr/local/nagios/bin/nagiostats -c /usr/local/nagios/etc/nagios.cfg


Nagios Stats 3.0prealpha-05202006
Copyright (c) 2003-2007 Ethan Galstad (www.nagios.org)
Last Modified: 05-20-2006
License: GPL

CURRENT STATUS DATA
------------------------------------------------------
Status File:                            /usr/local/nagios/var/status.dat
Status File Age:                        0d 0h 0m 9s
Status File Version:                    3.0prealpha-05202006

Program Running Time:                   0d 5h 20m 39s
Nagios PID:                             10119
Used/High/Total Command Buffers:        0 / 0 / 64
Used/High/Total Check Result Buffers:   0 / 7 / 512

Total Services:                         95
Services Checked:                       94
Services Scheduled:                     91
Services Actively Checked:              94
Services Passively Checked:             1
Total Service State Change:             0.000 / 78.950 / 1.026 %
Active Service Latency:                 0.000 / 4.272 / 0.561 sec
Active Service Execution Time:          0.000 / 60.007 / 2.066 sec
Active Service State Change:            0.000 / 78.950 / 1.037 %
Active Services Last 1/5/15/60 min:     4 / 68 / 91 / 91
Passive Service State Change:           0.000 / 0.000 / 0.000 %
Passive Services Last 1/5/15/60 min:    0 / 0 / 0 / 0
```

```
Services Ok/Warn/Unk/Crit:              58 / 16 / 0 / 21
Services Flapping:                      1
Services In Downtime:                   0

Total Hosts:                            24
Hosts Checked:                          24
Hosts Scheduled:                        24
Hosts Actively Checked:                 24
Host Passively Checked:                 0
Total Host State Change:                0.000 / 9.210 / 0.384 %
Active Host Latency:                    0.000 / 0.446 / 0.219 sec
Active Host Execution Time:             1.019 / 10.034 / 2.764 sec
Active Host State Change:               0.000 / 9.210 / 0.384 %
Active Hosts Last 1/5/15/60 min:        5 / 22 / 24 / 24
Passive Host State Change:              0.000 / 0.000 / 0.000 %
Passive Hosts Last 1/5/15/60 min:       0 / 0 / 0 / 0
Hosts Up/Down/Unreach:                  18 / 4 / 2
Hosts Flapping:                         0
Hosts In Downtime:                      0

Active Host Checks Last 1/5/15 min:     9 / 52 / 164
   Scheduled:                           4 / 23 / 75
   On-demand:                           3 / 23 / 69
   Cached:                              2 / 6 / 20
Passive Host Checks Last 1/5/15 min:    0 / 0 / 0
Active Service Checks Last 1/5/15 min:  9 / 80 / 244
   Scheduled:                           9 / 80 / 244
   On-demand:                           0 / 0 / 0
   Cached:                              0 / 0 / 0
Passive Service Checks Last 1/5/15 min: 0 / 0 / 0

External Commands Last 1/5/15 min:      0 / 0 / 0

[nagios@lanman ~]#
```

Wie Sie sehen können, zeigt das Utility ein Reihe von verschiedenen Metriken zum Nagios-Prozess an. Metriken mit mehreren Werten sind (wenn nicht anders angegeben) Minimum-, Maximum- und Durchschnittswerte für die betreffende Metrik.

MRTG-Integration

Sie können das *nagiostats*-Utility benutzen, um verschiedene Nagios-Metriken mit MRTG (oder anderen kompatiblen Programmen) anzuzeigen. Um das zu tun, starten Sie das *nagiostats*-Utility mit den **--mrtg**- und **--data**-Optionen. Die **--data**-Option wird benutzt, um anzugeben, welche Statistiken dargestellt werden sollen. Mögliche Werte für die **--data**-Option finden Sie durch Start des *nagiostats*-Utilitys mit der **--help**-Option.

Anmerkung: Informationen zum Gebrauch des *nagiostats*-Utilitys zu Generierung von MRTG-Grafiken zu Darstellung von Nagios-Leistungsstatistiken finden Sie hier.

grafische Darstellung von Performance-Informationen mit MRTG

Hoch zu: Inhalt

Siehe auch: Nagiostats-Utility, Leistungsoptimierung

Einführung

Das nagiostats-Utility erlaubt Ihnen zusammen mit MRTG, verschiedene Nagios-Performance-Statistiken über eine bestimmten Zeitraum grafisch darzustellen. Das ist wichtig, weil es Ihnen helfen kann

- dass Nagios effizient arbeitet
- um Problembereiche im Überwachungsprozess zu lokalisieren
- um die Einflüsse von Änderungen in Ihrer Nagios-Konfiguration zu beobachten

MRTG-Beispielkonfiguration

Schnipsel von MRTG-Beispieldateien zur Darstellung von verschiedenen Nagios-Performance-Statistiken finden Sie in der *mrtg.cfg*-Datei im *sample-config/*-Unterverzeichnis in der Nagios-Distribution. Sie können auch Graphen von anderen Performance-Informationen erstellen, wenn Sie wollen - die Beispiele liefern Ihnen dazu einen guten Ausgangspunkt.

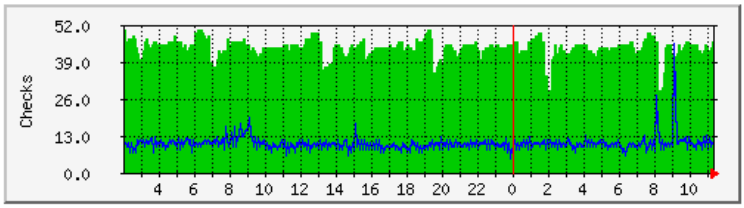
Sobald Sie diese Beispieleinträge in die MRTG-Konfigurationsdatei (/etc/mrtg/mrtg.cfg) kopieren, sollten Sie einige neue Graphen haben, wenn MRTG das nächste Mal läuft.

Beispielgraphen

Ich werde beschreiben, welche Bedeutung einige der Beispielgraphen haben und wofür sie benutzt werden können...

Aktive Host-Prüfungen - Dieser Graph zeigt, wie viele aktive Host-Prüfungen (regelmäßig geplant und nach Bedarf) über die Zeit auftraten. Nützlich zum Verstehen von:

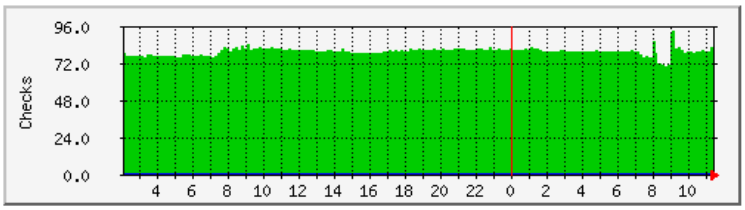
- Host-Prüfungen
- voraussschauende Host-Abhängigkeitsprüfungen
- zwischengespeicherten Prüfungen (cached checks)



Max Scheduled Checks: 50.0 Average Scheduled Checks: 44.0 Current Scheduled Checks: 46.0
Max On-Demand Checks: 45.0 Average On-Demand Checks: 10.0 Current On-Demand Checks: 10.0

Aktive Service-Prüfungen - Dieser Graph zeigt, wie viele aktive Service-Prüfungen (regelmäßig geplant und nach Bedarf) über die Zeit auftraten. Nützlich zum Verstehen von:

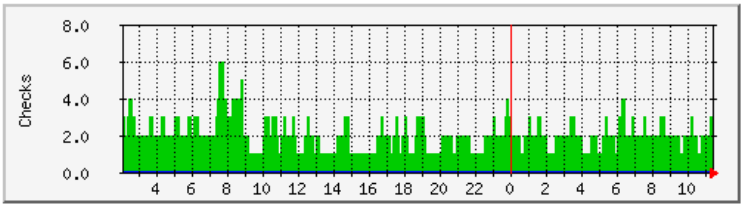
- Service-Prüfungen
- voraussschauende Service-Abhängigkeitsprüfungen
- zwischengespeicherten Prüfungen (cached checks)



Max Scheduled Checks: 93.0 Average Scheduled Checks: 72.0 Current Scheduled Checks: 66.0
Max On-Demand Checks: 0.0 Average On-Demand Checks: 0.0 Current On-Demand Checks: 0.0

Zwischengespeicherte Host- und Service-Prüfungen - Dieser Graph zeigt, wie viele zwischengespeicherte Host- und Service-Prüfungen (regelmäßig geplant und nach Bedarf) über die Zeit auftraten. Nützlich zum Verstehen von:

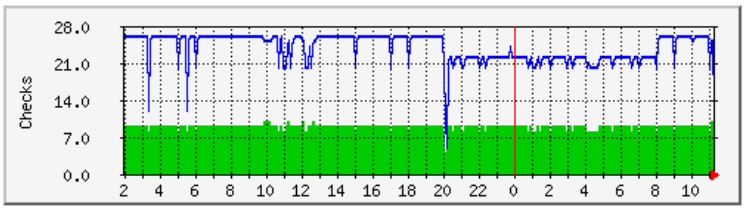
- zwischengespeicherten Prüfungen (cached checks)
- voraussschauende Host- und Service-Abhängigkeitsprüfungen



Max Host Checks: 6.0 Average Host Checks: 2.0 Current Host Checks: 3.0
Max Service Checks: 0.0 Average Service Checks: 0.0 Current Service Checks: 0.0

Passive Host- und Service-Prüfungen - Dieser Graph zeigt, wie viele passive Host- und Service-Prüfungen (regelmäßig geplant und nach Bedarf) über die Zeit auftraten. Nützlich zum Verstehen von:

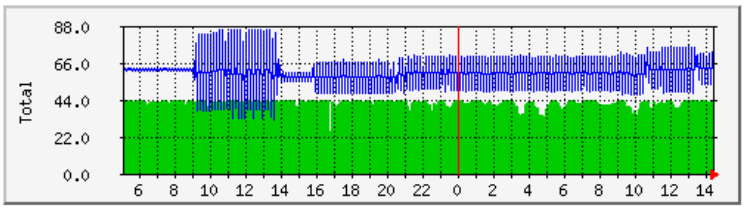
- passiven Prüfungen



Max Host Checks: 10.0 Average Host Checks: 9.0 Current Host Checks: 6.0
Max Service Checks: 26.0 Average Service Checks: 24.0 Current Service Checks: 12.0

aktive Host-/Serviceprüfungen - Dieser Graph zeigt, wie viele (die Gesamtzahl von) Hosts und Service aktiv über die Zeit geprüft wurden. Nützlich zum Verstehen von:

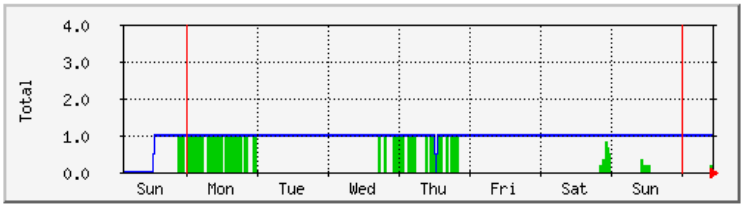
- aktiven Prüfungen



Max Hosts: 44.0 Average Hosts: 42.0 Current Hosts: 43.0
Max Services: 85.0 Average Services: 60.0 Current Services: 53.0

passive Host-/Serviceprüfungen - Dieser Graph zeigt, wie viele (die Gesamtzahl von) Hosts und Service passiv über die Zeit geprüft wurden. Nützlich zum Verstehen von:

- passiven Prüfungen



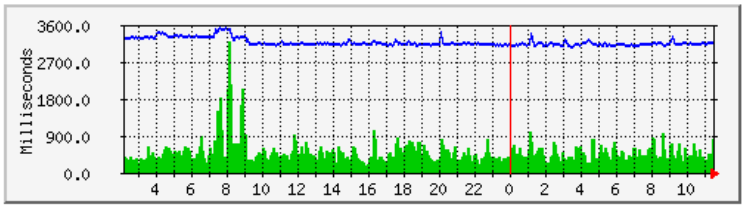
Max Hosts: 1.0 Average Hosts: 0.0 Current Hosts: 0.0
Max Services: 1.0 Average Services: 1.0 Current Services: 1.0

durchschnittliche Service-Prüfungs-Latenz- und Ausführungszeiten - Dieser Graph zeigt die durchschnittlichen Service-Prüfungs-Latenz- und Ausführungszeiten über die Zeit. Nützlich zum Verstehen von:

- Service-Prüfungen
- Leistungsoptimierung

Konstant hohe Latenzzeiten können ein Anzeichen dafür sein, eine oder mehrere der folgenden Variablen zu ändern:

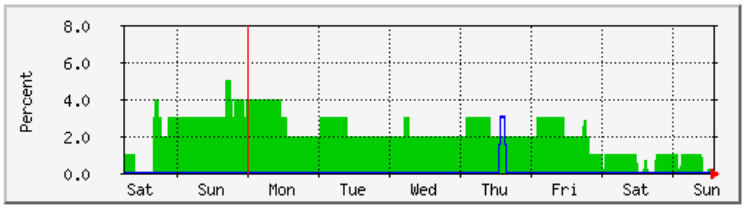
- max_concurrent_checks
- check_result_reaper_frequency
- max_check_result_reaper_time



Max Latency: 3203.0 Average Latency: 463.0 Current Latency: 816.0
Max Execution Time: 3498.0 Average Execution Time: 3163.0 Current Execution Time: 3185.0

durchschnittliche Service-Zustandsänderungen - Dieser Graph zeigt die durchschnittliche prozentuale Service-Zustandsänderung (ein Maßstab für die Sprunghaftigkeit) von Services über die Zeit, heruntergebrochen auf Services, die in der letzten Zeit aktiv oder passiv geprüft wurden. Nützlich zum Verstehen von:

- Flattererkennung



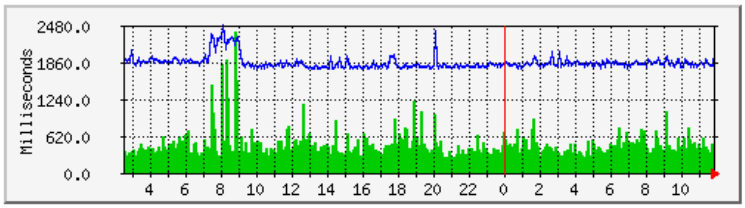
Max Active Check % Change: 5.0 Average Active Check % Change: 2.0 Current Active Check % Change: 0.0
Max Passive Check % Change: 3.0 Average Passive Check % Change: 0.0 Current Passive Check % Change: 0.0

durchschnittliche Host-Prüfungs-Latenz- und Ausführungszeiten - Dieser Graph zeigt die durchschnittlichen Host-Prüfungs-Latenz- und Ausführungszeiten über die Zeit. Nützlich zum Verstehen von:

- Host-Prüfungen
- Leistungsoptimierung

Konstant hohe Latenzzeiten können ein Anzeichen dafür sein, eine oder mehrere der folgenden Variablen zu ändern:

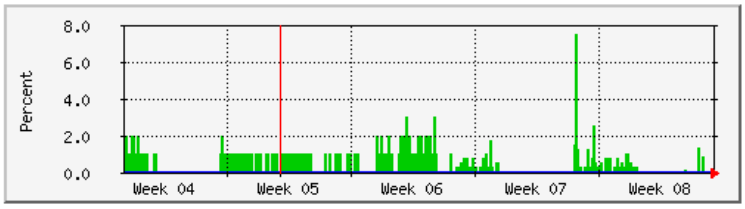
- max_concurrent_checks
- check_result_reaper_frequency
- max_check_result_reaper_time



Max Latency: 2373.0 Average Latency: 433.0 Current Latency: 374.0
Max Execution Time: 2480.0 Average Execution Time: 1839.0 Current Execution Time: 2021.0

durchschnittliche Host-Zustandsänderungen - Dieser Graph zeigt die durchschnittliche prozentuale Host-Zustandsänderung (ein Maßstab für die Sprunghaftigkeit) von Services über die Zeit, heruntergebrochen auf Services, die in der letzten Zeit aktiv oder passiv geprüft wurden. Nützlich zum Verstehen von:

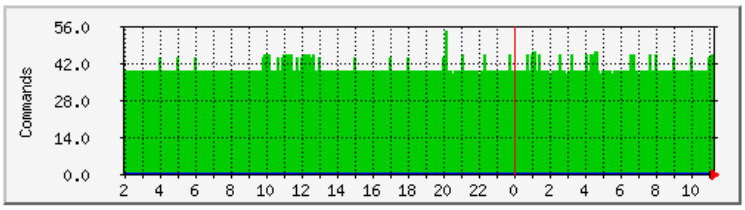
- Flattererkennung



Max Active Check % Change: 7.0 Average Active Check % Change: 1.0 Current Active Check % Change: 0.0
Max Passive Check % Change: 0.0 Average Passive Check % Change: 0.0 Current Passive Check % Change: 0.0

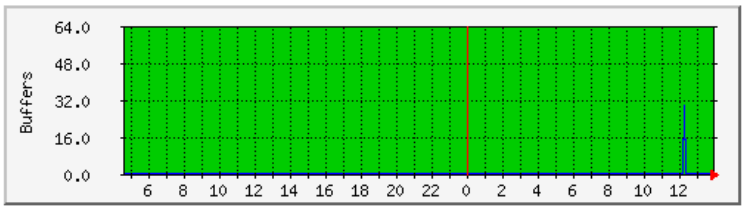
externe Befehle - Dieser Graph zeigt, wie viele externe Befehle von Nagios über die Zeit verarbeitet wurden. Solange Sie nicht eine große Zahl von externen Befehlen verarbeiten (wie im Fall von verteilten Überwachungsumgebungen), sollte dieser Graph nahezu leer sein. Die Überwachung von externen Befehlen kann nützlich sein zum Verstehen von Einflüssen durch:

- passiven Prüfungen
- verteilte Überwachung
- redundante/Failover-Überwachung



Max Commands: 54.0 Average Commands: 40.0 Current Commands: 20.0

externe Befehlspuffer - Dieser Graph zeigt, wie viele externe Befehlspuffer über die Zeit benutzt wurden. Wenn die Zahl von benutzten Befehlspuffern regelmäßig nahe an der Zahl von verfügbaren Puffern ist, dann sollten Sie wahrscheinlich den Wert der external command buffer slots erhöhen. Puffer werden benötigt, um temporär die externen Befehle zu speichern in der Zeit, wo sie vom external command file gelesen werden bis zur Verarbeitung durch den Nagios-Daemon.



Max Avail: 64.0 Average Avail: 64.0 Current Avail: 64.0
Max Used: 29.0 Average Used: 0.0 Current Used: 0.0

Nagios®

Integrationsüberblick

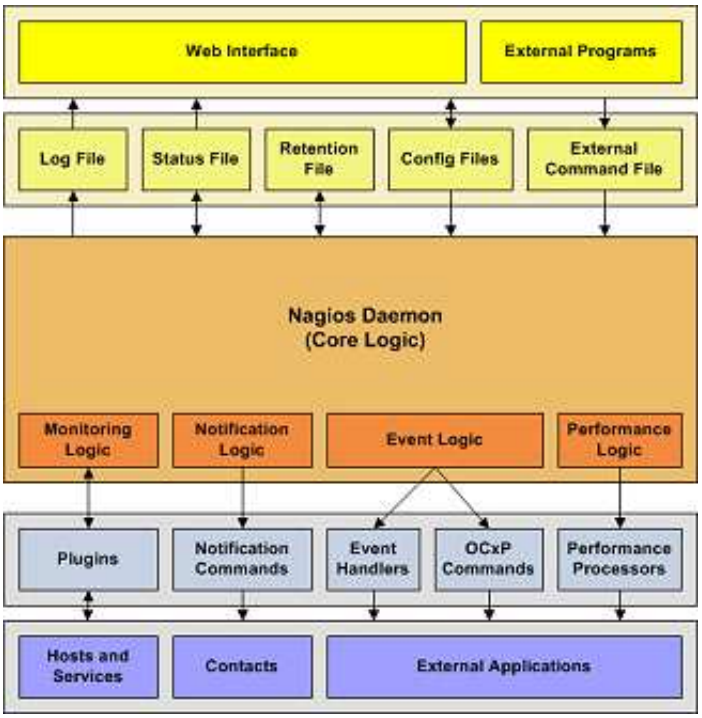
Hoch zu: Inhalt

Siehe auch: Externe Befehle, Passive Prüfungen, Eventhandler, Plugins

Einführung

Einer der Gründe, warum Nagios solch eine populäre Überwachungsapplikation ist, liegt in der Tatsache, dass es einfach in Ihre vorhandene Infrastruktur integriert werden kann. Es gibt mehrere Methoden, um Nagios mit der Management-Software zu integrieren, die Sie bereits nutzen und Sie können fast jede Art von neuer oder angepasster Hardware, Service oder Applikation überwachen, die Sie haben.

Integrationsstellen



Um neue Hardware, Services oder Applikationen zu überwachen, prüfen Sie die Dokumentationen zu:

- Plugins
- Plugin API
- Passive Prüfungen
- Eventhandler

Um Daten aus externen Applikationen in Nagios zu bekommen, prüfen Sie die Dokumentationen zu:

- Passive Prüfungen
- Externe Befehle

Um Zustands-, Leistungs- oder Benachrichtigungsinformationen von Nagios an externe Applikationen zu senden, prüfen Sie die Dokumentationen zu:

- Eventhandlers
- OCSP- und OCHP-Befehlen
- Performance-Daten
- Nenachrichtigungen

Integrationsbeispiele

Ich habe ein paar Beispiele dokumentiert, wie Nagios mit externen Applikationen integriert wird:

- TCP-Wrappers (Sicherheitsalarme)
- SNMP-Traps (Arcserve Backup-Job-Status)

SNMP-Trap-Integration

Hoch zu: Inhalt

Siehe auch: Integrationsüberblick, Externe Befehle, Passive Prüfungen

Einführung

Hinweis: Nagios ist nicht als Ersatz für eine ausgewachsene SNMP-Management-Applikation wie HP-OpenView oder OpenNMS gedacht. Allerdings können Sie die Dinge so einrichten, dass SNMP-Traps, die von einem Host in Ihrem Netzwerk empfangen werden, Alarme in Nagios zu generieren.

Als wenn es dazu gemacht wäre, die Götter der Scheinheiligkeit vor Lachen sterben zu lassen, ist SNMP alles andere als einfach. SNMP-Traps zu übersetzen und sie in Nagios zu bekommen (als passive Prüfresultate) kann ein wenig mühselig sein. Um diese Aufgabe zu erleichtern, empfehle ich, dass Sie sich Alex Burger's SNMP Trap Translator Projekt unter http://www.snmptt.org ansehen. Wenn es mit Net-SNMP kombiniert wird, liefert SNMPTT ein fortgeschrittenes Trap-Behandlungssystem, das mit Nagios integriert werden kann.

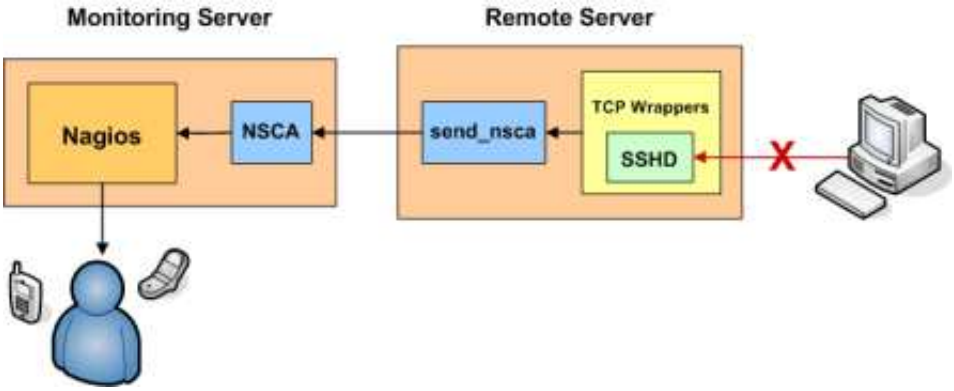
Yep, das ist alles.

TCP-Wrapper-Integration

Hoch zu: Inhalt

Siehe auch: Integrationsüberblick, Externe Befehle, Passive Prüfungen

Einführung



Dieses Dokument erklärt, wie einfach in Nagios Alarme für Verbindungsversuche generiert werden können, die von TCP-Wrappern zurückgewiesen werden. Wenn zum Beispiel ein unautorisierter Host versucht, sich mit Ihrem SSH-Server zu verbinden, können Sie in Nagios einen Alarm empfangen, der den Namen des Hosts enthält, der zurückgewiesen wurde. Wenn Sie das auf Ihren Linux/Unix-Boxen installieren, dann werden Sie erstaunt sein, wie viele Port-Scans Sie in Ihrem Netzwerk entdecken.

Diese Anweisungen gehen davon aus, dass

1. Sie bereits mit passiven Prüfungen vertraut sind und wissen, wie sie arbeiten.
2. Sie bereits mit sprunghaften Services vertraut sind und wissen, wie sie arbeiten.
3. der Host, für den Sie Alarme generieren (d.h. der Host, auf dem Sie TCP-Wrapper benutzen), ein entfernter Host ist (in diesem Beispiel *firestorm* genannt). Wenn Sie Alarme auf dem gleichen Host generieren möchten, müssen Sie ein paar Anpassungen an den Beispielen machen, die ich bereitstelle.
4. Sie den NSCA-Daemon auf Ihrem Überwachungs-Server und den NSCA client (*send_nsca*) auf der entfernten Maschine installiert haben, für die Sie TCP-Wrapper-Alarme generieren möchten.

Einen Service definieren

Wenn Sie es nicht bereits getan haben, erstellen Sie eine Host-Definition für den entfernten Host (*firestorm*).

Als nächstes definieren Sie einen Service in einer Ihrer Objektkonfigurationsdateien für die TCP-Wrapper-Alarme auf dem Host *firestorm*. Die Service-Definition könnte wie folgt aussehen:

```
define service{
        host_name                       firestorm
        service_description             TCP Wrappers
        is_volatile                     1
        active_checks_enabled           0
        passive_checks_enabled          1
        max_check_attempts              1
        check_command                   check_none
        ...
        }
```

Es gibt einige wichtige Dinge zu der obigen Service-Definition anzumerken:

1. Die *volatile*-Option ist aktiviert. Wir wollen, dass diese Option aktiviert ist, weil wir eine Benachrichtigung für jeden Alarm haben wollen, der herein kommt.
2. Aktive Prüfungen für den Service sind deaktiviert, während passive Prüfungen aktiviert sind. Das bedeutet, dass der Service niemals aktiv von Nagios geprüft wird - alle Alarminformationen müssen passiv von einer externen Quelle empfangen werden.
3. Der *max_check_attempts*-Wert wird auf 1 gesetzt. Das gewährleistet, dass Sie eine Benachrichtigung erhalten, sobald der erste Alarm generiert wird.

TCP-Wrapper konfigurieren

Nun müssen Sie die */etc/hosts.deny*-Datei auf *firestorm* editieren. Damit die TCP-Wrapper einen Alarm an den Überwachungs-Host senden, sobald ein Verbindungsversuch verweigert wird, müssen Sie eine Zeile hinzufügen, die der folgenden ähnlich ist.

```
ALL: ALL: RFC931: twist (/usr/local/nagios/libexec/eventhandlers/handle_tcp_wrapper %h %d) &
```

Diese Zeile nimmt an, dass es ein Script namens *handle_tcp_wrapper* im */usr/local/nagios/libexec/eventhandlers/*-Verzeichnis auf *firestorm* gibt. Wir werden dieses Script als nächstes schreiben.

Das Script schreiben

Als letztes müssen Sie das *handle_tcp_wrapper*-Script auf *firestorm* schreiben, das den Alarm zurück an den Nagios-Server schickt. Es könnte ungefähr so aussehen:

```
#!/bin/sh

/usr/local/nagios/libexec/eventhandlers/submit_check_result firestorm "TCP Wrappers" 2 "Denied $2-$1" > /dev/null 2> /dev/null
```

Beachten Sie, dass das *handle_tcp_wrapper*-Script das *submit_check_result*-Script aufruft, um den Alarm zurück an den Überwachungs-Host zu schicken. Angenommen, Ihr Nagios-Server heißt *monitor*, dann könnte das *submit check_result*-Script wie folgt aussehen:

```
#!/bin/sh
# Arguments
#       $1 = name of host in service definition
#       $2 = name/description of service in service definition
#       $3 = return code
#       $4 = output

/bin/echo -e "$1\t$2\t$3\t$4\n" | /usr/local/nagios/bin/send_nsca monitor -c /usr/local/nagios/etc/send_nsca.cfg
```

Aufräumen

Sie haben nun alles konfiguriert, was Sie brauchen, so dass Sie nur noch den *inetd*-Prozess auf *firestorm* und Nagios auf Ihrem Überwachungs-Server neu starten müssen. Das war's! Wenn die TCP-Wrapper auf *firestorm* einen Verbindungsversuch verweigern, dann sollten Sie Alarme in Nagios erhalten. Die Plugin-Ausgabe für den Alarm könnte wie folgt aussehen:

```
Denied sshd2-sdn-ar-002mnminnP321.dialsprint.net
```

Nagios Addons

Hoch zu: Inhalt

Einführung

Es gibt eine Menge von "Addon"-Software-Paketen, die für Nagios verfügbar sind. Addons können genutzt werden, um die Funktionalität von Nagios zu erweitern oder Nagios mit anderen Applikationen zu integrieren.

Addons gibt es für:

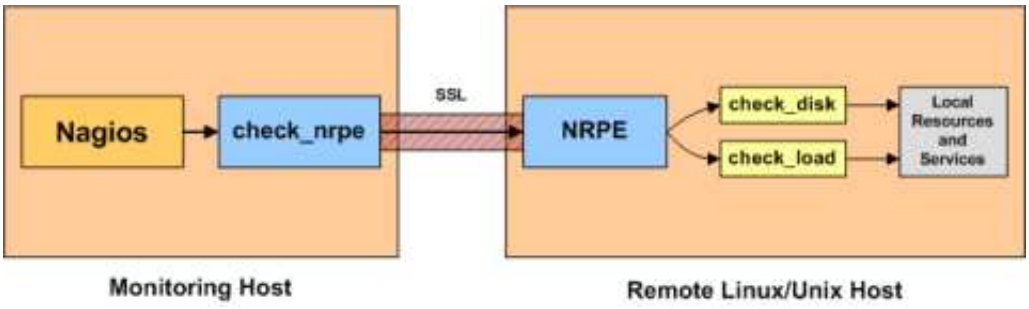
- die Verwaltung der Konfigurationsdateien über ein Web-Interface
- die Überwachung von entfernten Hosts (*NIX, Windows, etc.)
- die Erteilung von passiven Prüfungen von entfernten Hosts
- die Vereinfachung/Erweiterung der Benachrichtigungslogik
- ...und vieles mehr

Sie finden viele Addons für Nagios unter:

- Nagios.org
- SourceForge.net
- NagiosExchange.org

Ich werde eine kurze Einführung für ein paar Addons geben, die ich für Nagios entwickelt habe...

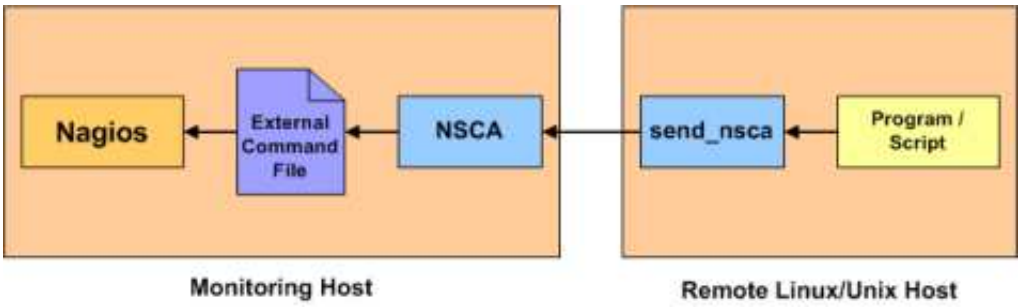
NRPE



NRPE ist ein Addon, das es Ihnen erlaubt, Plugins auf entfernten Linux-/Unix-Hosts auszuführen. Dies ist nützlich, wenn Sie lokale Ressourcen/Attribute wie Plattenbelegung, CPU-Last, Speicherbelegung usw. auf entfernten Hosts überwachen wollen. Ähnliche Funktionalitäten können durch das *check_by_ssh*-Plugin erreicht werden, obwohl es auf dem Überwachungsrechner für eine höhere CPU-Belastung sorgen kann - besonders dann, wenn Sie hunderte oder tausende von Hosts überwachen.

Das NRPE-Addon und die Dokumentation finden Sie unter http://www.nagios.org/.

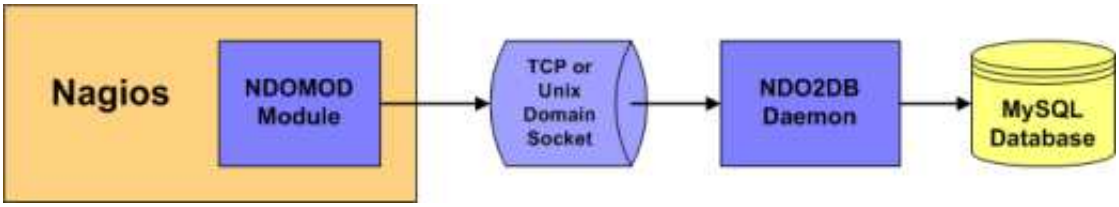
NSCA



NSCA ist ein Addon, das es Ihnen erlaubt, passive Prüf-Resultate von entfernten Linux-/Unix-Hosts an den Nagios-Daemon zu senden, der auf dem Überwachungs-Server läuft. Das ist sehr hilfreich in verteilten und redundanten/Failover-Überwachungs-Umgebungen.

Das NSCA-Addon und die Dokumentation finden Sie unter http://www.nagios.org/.

NDOUtils



NDOUtils ist ein Addon, das es Ihnen erlaubt, alle Nagios-Statusinformationen in einer MySQL-Datenbank zu speichern. Mehrere Instanzen von Nagios können all ihre Informationen in einer zentralen Datenbank für ein zentrales Berichtswesen speichern. Dies wird wahrscheinlich in der Zukunft als Basis für ein neues PHP-basiertes Web-Interface für Nagios dienen.

Das NDOUtils-Addon und die Dokumentation finden Sie unter http://www.nagios.org/.

Nagios Plugin API

Hoch zu: Inhalt

Siehe auch: Plugin-Überblick, Developing Plugins For Use With Embedded Perl, Performance-Daten

Andere Ressourcen

Wenn Sie planen, Ihren eigenen Plugins für Nagios zu schreiben, dann besuchen Sie folgende Ressourcen:

- Die offizielle Nagios-Plugin-Projekt-Website
- Die offiziellen Nagios-Plugin-Entwicklungsrichtlinien

Plugin-Überblick

Scripts und ausführbare Programme müssen (mindestens) zwei Dinge tun, um als Nagios-Plugins zu funktionieren:

- mit einem von verschiedenen möglichen Return-Codes enden
- mindestens eine Zeile Textausgabe an STDOUT zurückliefern

Die inneren Abläufe Ihres Plugins sind für Nagios unwichtig. Ihr Plugin könnte den Zustand eines TCP-Ports prüfen, eine Datenbankabfrage durchführen, den freien Plattenplatz ermitteln oder was immer benötigt wird, um etwas zu prüfen. Die Einzelheiten hängen davon ab, was zu prüfen ist - das liegt an Ihnen.

Return-Code

Nagios ermittelt den Zustand eines Hosts oder Service über die Auswertung des Return-Codes des Plugins. Die folgenden Tabellen zeigen eine Liste von gültigen Return-Codes zusammen mit ihren entsprechenden Service- oder Host-Zuständen.

Plugin Return-Code	Service-Zustand	Host-Zustand
0	OK	UP
1	WARNING	UP oder DOWN/UNREACHABLE*
2	CRITICAL	DOWN/UNREACHABLE
3	UNKNOWN	DOWN/UNREACHABLE

Anmerkung: Wenn die use_aggressive_host_checking-Option aktiviert ist, dann ergibt ein Return-Code von 1 einen Host-Zustand "DOWN" oder "UNREACHABLE". Andernfalls ergibt ein Return-Code von 1 einen Host-Zustand "UP". Der Prozess, durch den Nagios ermittelt, ob ein Host DOWN oder UNREACHABLE ist, wird hier erklärt.

Spezifikation der Plugin-Ausgabe(n)

Als Minimum sollten Plugins mindestens eine Zeile Textausgabe zurückliefern. Beginnend mit Nagios 3 können Plugins mehrere Zeilen Ausgaben erzeugen. Plugins können zusätzlich Performance-Daten zurückliefern, die von externen Applikationen verarbeitet werden können. Das grundlegende Format für Plugin-Ausgaben ist wie folgt:

```
TEXT OUTPUT | OPTIONAL PERFDATA
LONG TEXT LINE 1
LONG TEXT LINE 2
...
LONG TEXT LINE N | PERFDATA LINE 2
PERFDATA LINE 3
...
PERFDATA LINE N
```

Die Performance-Daten (in orange dargestellt) sind optional. Wenn ein Plugin Performance-Daten in der Ausgabe zurückliefert, dann müssen die Performance-Daten von den anderen Textausgaben mit einem Pipe-Symbol (|) getrennt werden. Zusätzliche Zeilen von langen Textausgaben (in blau dargestellt) sind ebenso optional.

Plugin-Beispielausgaben

Nun ein paar Beispiele von möglichen Plugin-Ausgaben...

Fall 1: Eine Zeile Ausgabe (nur Text)
Angenommen, wir haben ein Plugin, das eine Zeile ausgibt, dann sieht das wie folgt aus:

```
DISK OK - free space: / 3326 MB (56%);
```

Wenn dieses Plugin benutzt wurde, um eine Service-Prüfung durchzuführen, wird die gesamte Zeile der Ausgabe im $SERVICEOUTPUT$-Makro gespeichert.

Fall 2: Eine Zeile Ausgabe (Text und Performance-Daten)
Ein Plugin kann optionale Performance-Daten zurückliefern, die von externen Applikationen benutzt werden. Um dies zu tun, müssen die Performance-Daten von der Textausgabe durch ein Pipe-Symbol (|) wie folgt getrennt werden:

```
DISK OK - free space: / 3326 MB (56%);
 |
/=2643MB;5948;5958;0;5968
```

Wenn dieses Plugin benutzt wurde, um eine Service-Prüfung durchzuführen, wird der
rote
Teil der Ausgabe (links vom Pipe-Symbol) im $SERVICEOUTPUT$-Makro und der
orange
Teil der Ausgabe (rechts vom Pipe-Symbol) im $SERVICEPERFDATA$-Makro gespeichert.

Fall 3: Mehrere Zeilen Ausgaben (Text und Performance-Daten)
Ein Plugin kann optional mehrere Zeilen von Text und Perforance-Daten wie folgt zurückliefern:

```
DISK OK - free space: / 3326 MB (56%); | /=2643MB;5948;5958;0;5968
/ 15272 MB (77%);
/boot 68 MB (69%);
/home 69357 MB (27%);
/var/log 819 MB (84%); | /boot=68MB;88;93;0;98
/home=69357MB;253404;253409;0;253414
```

/var/log=818MB;970;975;0;980

Wenn dieses Plugin benutzt wurde, um eine Service-Prüfung durchzuführen, wird der rote Teil der ersten Zeile der Ausgabe (links vom Pipe-Symbol) im $SERVICEOUTPUT$-Makro gespeichert. Der orange Teil der ersten und folgender Zeilen wird (durch Leerzeichen verbunden) im $SERVICEPERFDATA$-Makro gespeichert. Der blaue Teil der zweiten bis fünften Zeile der Ausgabe wird (mit maskierten Newlines) verkettet und im $LONGSERVICEOUTPUT$-Makro gespeichert.

Der endgültige Inhalt jedes Makros ist wie folgt:

Makro	Wert
$SERVICEOUTPUT$	DISK OK - free space: / 3326 MB (56%);
$SERVICEPERFDATA$	/=2643MB;5948;5958;0;5968 /boot=68MB;88;93;0;98 /home=69357MB;253404;253409;0;253414 /var/log=818MB;970;975;0;980
$LONGSERVICEOUTPUT$	/ 15272 MB (77%);\n/boot 68 MB (69%);\n/var/log 819 MB (84%);

Mit Blick auf mehrere Zeilen Ausgaben haben Sie die folgenden Möglichkeiten, Performance-Daten zurückzuliefern:

- Sie können keinerlei Performance-Daten zurückliefern
- Sie können nur in der ersten Zeile Performance-Daten zurückliefern
- Sie können Performance-Daten in nachfolgenden Zeilen zurückliefern (nach der ersten)
- Sie können Performance-Daten in der ersten und folgenden Zeilen zurückliefern (wie oben gezeigt)

Längenbeschränkungen von Plugin-Ausgaben

Nagios wird nur die ersten vier KB an Daten lesen, die ein Plugin zurückliefert. Dies wird getan, um durchgedrehte Plugins davon abzuhalten, Megabyte oder Gigabyte an Daten an Nagios zurückzuliefern. Diese Beschränkung von vier KB kann einfach geändert werden, wenn Sie das brauchen. Ändern Sie einfach den Wert der MAX_PLUGIN_OUTPUT_LENGTH-Definition in der *include/nagios.h.in*-Datei der Source-Code-Distribution und rekompilieren Sie Nagios. Mehr müssen Sie nicht tun!

Beispiele

Wenn Sie nach Beispiel-Plugins suchen, um sie zu studieren, würde ich empfehlen, dass Sie die offiziellen Nagios-Plugins herunterladen und den Code von verschiedenen C-, Perl- und Shell-Script-Plugins ansehen. Informationen, wie Sie die offiziellen Plugins besorgen können, finden Sie hier.

Perl-Plugins

Nagios bietet einen optionalen eingebauten Perl-Interpreter (embedded Perl interpreter), der die Ausführung von Perl-Plugins beschleunigen kann. Mehr Informationen zur Entwicklung von Perl-Plugins zur Nutzung mit dem eingebauten Perl-Interpreter finden Sie hier.

Entwickeln von Plugins für die Nutzung mit Embedded Perl

Hoch zu: Inhalt

Siehe auch: Überblick eingebetteter Perl-Interpreter (Embedded Perl Interpreter), Plugin API

Einführung

Stanley Hopcroft hat ziemlich viel mit dem eingebetten Perl-Interpreter gearbeitet und die Vor- und Nachteile der Nutzung kommentiert. Er hat auch verschiedene Hinweise gegeben, um Perl-Plugins zu erstellen, die sauber mit dem eingebetteten Interpreter laufen. Der überwiegende Teil dieser Dokumentation stammt aus seinen Kommentaren.

Es ist anzumerken, dass sich "ePN", wie in dieser Dokumentation verwendet, auf den eingebetteten Perl-Interpreter, oder wenn Ihnen das lieber ist, auf Nagios kompiliert mit einem eingebetteten Perl-Interpreter bezieht.

Zielgruppe

- Durchschnittliche Perl-Entwickler mit einem Verständnis für die mächtigen Eigenschaften der Sprache ohne Wissen der Interna bzw. einem vertieften Wissen dieser Eigenschaften.
- die mit einem benutzenden Wissen statt einem tiefen Verständnis
- wenn Sie glücklich sind mit Perl-Objekten, sprich Verwaltung, Datenstrukturen und dem Debugger, dann ist das wahrscheinlich ausreichend.

Dinge, die Sie tun sollten, wenn Sie ein Perl-Plugin entwickeln (mit ePN oder ohne)

- generieren Sie immer etwas Output
- Verwenden Sie 'use utils' und importieren Sie die Dinge, die es exportiert ($TIMEOUT %ERRORS &print_revision &support)
- Werfen Sie einen Blick darauf, wie die Standard-Plugins ihren Kram erledigen
  - beenden Sie immer mit $ERRORS{CRITICAL}, $ERRORS{OK}, etc.
  - verwenden Sie getopt, um Kommandozeilenparameter einzulesen
  - denken Sie an Timeout-Verwaltung
  - rufen Sie print_usage auf (das Sie liefern müssen), wenn keine Kommandozeilenparameter übergeben wurden
  - benutzen Sie Standard-Optionen (eg H 'host', V 'version')

Dinge, die Sie tun müssen, um ein Perl-Plugin für ePN zu entwickeln

1. <DATA> kann nicht verwendet werden, benutzen Sie statt dessen here-Dokumente, z.B.

```
my $data = <<DATA;
portmapper 100000
portmap 100000
sunrpc 100000
rpcbind 100000
rstatd 100001
rstat 100001
rup 100001
..
DATA

%prognum = map { my($a, $b) = split; ($a, $b) } split(/\n/, $data) ;
```

2. BEGIN-Blöcke werden nicht so funktionieren, wie Sie das erwarten. Es wird das Beste sein, wenn Sie darauf verzichten.
3. stellen Sie sicher, dass es während des Compile absolut sauber ist, d.h.
   - use strict
   - use perl -w (andere Switches [namentlich T] könnten nicht weiterhelfen)
   - use perl -c
4. Vermeiden Sie lexikalische Variablen (my) mit globalem Geltungsbereich, um damit __variable__ Daten in Unterroutinen zu übergeben. Das ist in der Tat __fatal__, wenn die Unterroutine mehrfach aufgerufen wird, während die Prüfung läuft. Solche Unterroutinen arbeiten als 'closures', die den ersten Wert der globalen lexikalischen Variable bei folgenden Aufrufen der Unterroutine beibehalten. Wenn die globale Variable allerdings read-only ist (bei einer komplizierten Struktur zum Beispiel), dann ist das kein Problem. Was Bekman Ihnen statt dessen rät, ist eines der folgenden Dinge:
   - machen Sie die Unterroutine anonym und rufen Sie sie z.B. über eine Code-Referenz auf

```
ändern Sie dies                 in

my $x = 1 ;                     my $x = 1 ;
sub a { .. Process $x ... }     $a_cr = sub { ... Process $x ... } ;
.                               .
.                               .
a ;                             &$a_cr ;
$x = 2                          $x = 2 ;
a ;                             &$a_cr ;

# anonyme Closures binden __immer__ den aktuellen lexikalischen Wert ein
```

   - packen Sie globale Lexikale und die Unterroutine, die sie benutzt, in ihr eigenes Package (als ein Objekt oder Modul)
   - übergeben Sie Informationen an Unterroutinen als Referenzen oder Aliases (\$lex_var oder $_[n])
   - ersetzen Sie Lexikale durch Package Globals und schließen Sie diese von 'use strict'-Beanstandungen durch 'use vars qw(global1 global2 ..)' aus
5. Seien Sie sich bewusst, woher Sie mehr Informationen bekommen können.

   Nützliche Informationen können Sie von den üblichen Verdächtigen bekommen (die O'Reilly-Bücher, plus Damien Conways "Object Oriented Perl"), aber um den wirklich nützlichen Kram im richtigen Kontext zu bekommen, starten Sie mit Stas Bekman's mod_perl guide unter http://perl.apache.org/guide/.

   Dieses wundervolle Dokument in Buchgröße hat überhaupt nichts mit Nagios zu tun, aber dafür umso mehr mit dem Schreiben von Perl-Programmen für den eingebetteten Perl-Interpreter in Apache (d.h. Doug MacEacherns mod_perl).

Die perlembed-Manpage ist wichtig für den Zusammenhang und die Ermunterung..

Auf der Basis, dass Lincoln Stein und Doug MacEachern ein oder zwei Dinge über Perl und eingebettetes Perl wissen, ist ihr Buch 'Writing Apache Modules with Perl and C' ziemlich sicher einen Blick wert.

6. Achten Sie darauf, dass Ihr Plugin mit ePN vielleicht merkwürdige Werte zurückliefert und dass das wahrscheinlich an dem unter Punkt 4 angesprochenen Problem liegt
7. Seien Sie darauf vorbereitet, dass Sie debuggen über:
   - ein Test-ePN und
   - print-Befehle in Ihr Plugin einfügen, um Variablenwerte auf STDERR auszugeben (da Sie STDOUT nicht verwenden können)
   - print-Befehle in p1.pl einfügen, um anzuzeigen, was ePN glaubt, was Ihr Plugin ist, bevor es versucht, das auszuführen (vi)
   - ePN im Vordergrund-Modus auszuführen (möglicherweise in Verbindung mit den obigen Empfehlungen)
   - das 'Deparse#-Modul in Ihrem Modul zu benutzen, um zu sehen, wie der Parser es optimiert hat und was der Interpreter wirklich bekommt (lesen Sie 'Constants in Perl' von Sean M. Burke, The Perl Journal, Fall 2001)

```
perl -MO::Deparse <your_program>
```

8. Beachten Sie, in was ePN Ihr Plugin transformiert, und falls alles andere fehlschlägt, debuggen Sie die transformierte Version.

   Wie Sie unten sehen können, schreibt p1.pl Ihr Plugin um in eine Unterroutine namens 'hndlr' im Package 'Embed::<something_related_to_your_plugin_file_name>'.

   Ihr Plugin wird ggf. Kommandozeilenparameter in @ARGV erwarten, so dass p1.pl auch @_ an @ARGV zuweist.

   Dies wiederum wird 'eval'-t und falls dieser Test mit einem Fehler fehlschlägt (jeder Parse- oder Laufzeitfehler), wird das Plugin 'rausgeschmissen'.

   Die folgenden Ausgaben zeigen, wie ein Test-ePN das *check_rpc*-Plugin transformiert hat, bevor es versucht, es auszuführen. Der meiste Code des eigentlichen Plugins wird nicht gezeigt, weil wir nur an den Umformungen interessiert sind, die der ePN am Plugin vorgenommen hat). Zur Verdeutlichung sind die Umformungen in rot dargestellt:

```
                package main;
                use subs 'CORE::GLOBAL::exit';
                sub CORE::GLOBAL::exit { die "ExitTrap: $_[0]
(Embed::check_5frpc)"; }
                package Embed::check_5frpc; sub hndlr { shift(@_);
@ARGV=@_;
#! /usr/bin/perl -w
#
# check_rpc plugin for Nagios
#
# usage:
#    check_rpc host service
#
# Check if an rpc service is registered and running
# using rpcinfo - $proto $host $prognum 2>&1 |";
#
# Use these hosts.cfg entries as examples
#
# command[check_nfs]=/some/path/libexec/check_rpc $HOSTADDRESS$ nfs
# service[check_nfs]=NFS;24x7;3;5;5;unix-admin;60;24x7;1;1;1;;check_rpc
```

```
#
# initial version: 3 May 2000 by Truongchinh Nguyen and Karl DeBisschop
# current status: $Revision: 1.18 $
#
# Copyright Notice: GPL
#
... der Rest des Plugin-Codes folgt (und wurde aus Gründen der Kürze entfernt) ...
}
```

9. Nutzen Sie 'use diagnostics' nicht in einem produktiven ePN. Ich glaube, es sorgt dafür, dass __alle__ Perl-Plugins CRITICAL zurückliefern.
10. Überlegen Sie, ob Sie ein Mini-ePN benutzen, um Ihr Plugin zu testen. Das ist nicht ausreichend, um zu garantieren, dass Ihr Plugin mit einem ePN fehlerfrei ausgeführt wird, aber wenn bereits der Plugin-Test fehlschlägt, dann wird er auf jeden Fall mit Ihrem ePN fehlschlagen. [Ein Beispiel-Mini-ePN ist im *contrib*/-Verzeichnis der Nagios-Distribution zu finden. Wechseln Sie in das contrib/-Verzeichnis und tippen Sie 'make mini_epn', um es zu kompilieren. Es muss im gleichen Verzeichnis ausgeführt werden, in dem die p1.pl-Datei steht (diese Datei wird mit Nagios ausgeliefert).]

Schnellstart Fedora

Hoch zu: Inhalt

Siehe auch: Schnellstart-Installationsanleitungen, Sicherheitsüberlegungen

Einführung

Diese Schnellstartanleitung ist dazu gedacht, Ihnen einfache Anweisungen zu liefern, wie Sie Nagios innerhalb von 20 Minuten aus dem Quellcode auf Fedora installieren und Ihren lokalen Rechner damit überwachen. Hier werden keine fortgeschrittenen Installationsoptionen vorgestellt - lediglich die Grundlagen, die für 95% aller Benutzer funktionieren, die anfangen wollen.

Diese Anweisungen basieren auf einer Standard-**Fedora Core 6**-Linux-Distribution.

Was dabei herauskommt

Wenn Sie diesen Anweisungen folgen, werden Sie am Ende folgendes haben:

- Nagios und die Plugins werden unterhalb von /usr/local/nagios installiert sein
- Nagios wird so konfiguriert sein, dass es einige Dinge auf Ihrem lokalen System überwacht (CPU-Auslastung, Plattenbelegung, usw.)
- Das Nagios-Web-Interface ist erreichbar unter http://localhost/nagios/

Voraussetzungen

Während einiger Teile der Installation benötigen Sie **root**-Zugang zu Ihrer Maschine.

Stellen Sie sicher, dass die folgenden Pakete installiert sind, bevor Sie fortfahren.

- Apache
- GCC-Compiler
- GD-Development-Libraries

Sie können *yum* benutzen, um diese Pakete mit Hilfe der folgenden Befehle zu installieren (als root):

```
yum install httpd
yum install gcc
yum install glibc glibc-common
yum install gd gd-devel
```

1) Benutzerinformationen erstellen

Werden Sie zum root-Benutzer.

```
su -l
```

Erstellen Sie ein neues Benutzerkonto *nagios* und vergeben Sie ein Passwort.

```
/usr/sbin/useradd -m nagios
passwd nagios
```

Legen Sie eine neue Gruppe *nagcmd* an, damit über das Web-Interface externe Befehle erteilt werden können. Fügen Sie den Nagios- und den Web-Server-Benutzer zu dieser Gruppe hinzu.

```
/usr/sbin/groupadd nagcmd
/usr/sbin/usermod -a -G nagcmd nagios
/usr/sbin/usermod -a -G nagcmd apache
```

2) Nagios und Plugins herunterladen

Legen Sie ein Verzeichnis an, um die Dateien zu speichern.

```
mkdir ~/downloads
cd ~/downloads
```

Laden Sie die Quellcode-Archive von Nagios und den Nagios-Plugins herunter (besuchen Sie http://www.nagios.org/download/, um Verweise auf die aktuellsten Versionen zu sehen). Die Anweisungen wurden mit Nagios 3.0.6 und Nagios-Plugins 1.4.11 getestet.

```
wget http://osdn.dl.sourceforge.net/sourceforge/nagios/nagios-3.0.6.tar.gz
wget http://osdn.dl.sourceforge.net/sourceforge/nagiosplug/nagios-plugins-1.4.11.tar.gz
```

3) Nagios kompilieren und installieren

Entpacken Sie die Nagios-Quellcode-Archivdatei.

```
cd ~/downloads
tar xzf nagios-3.0.6.tar.gz
cd nagios-3.0.6
```

Führen Sie das Nagios-configure-Script aus und übergeben Sie den Namen der Gruppe, die Sie vorhin angelegt haben:

```
./configure --with-command-group=nagcmd
```

Kompilieren Sie den Nagios-Quellcode.

```
make all
```

Installieren Sie die Programme, Init-Scripte, Beispiel-Konfigurationsdaten und setzen Sie die Berechtigungen auf das Verzeichnis für die externen Befehle.

```
make install
make install-init
make install-config
make install-commandmode
```

Starten Sie Nagios noch nicht - es gibt noch ein paar Dinge zu tun...

4) Anpassen der Konfiguration

Die Beispiel-Konfigurationsdateien wurden nun im */usr/local/nagios/etc*-Verzeichnis installiert. Diese Beispieldateien sollten funktionieren, um mit Nagios zu starten. Nun fehlt nur noch eine Änderung, bevor Sie fortfahren können...

Ändern Sie die */usr/local/nagios/etc/objects/contacts.cfg*-Konfigurationsdatei mit Ihrem bevorzugten Editor und passen die e-Mail-Adresse in der *nagiosadmin*-Kontaktdefinition an, so dass sie die Adresse enthält, die im Falle von Alarmen benachrichtigt werden soll.

```
vi /usr/local/nagios/etc/objects/contacts.cfg
```

5) Konfigurieren des Web-Interface

Installieren Sie die Nagios-Web-Konfigurationsdateien im Apache conf.d-Verzeichnis.

```
make install-webconf
```

Legen Sie ein *nagiosadmin*-Konto an, um sich am Web-Interface anmelden zu können. Merken Sie sich das Passwort, das Sie diesem Konto geben - Sie brauchen es später.

```
htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
```

Starten Sie Apache neu, damit die Änderungen wirksam werden.

```
service httpd restart
```

 Anmerkung: Prüfen Sie die Implementierung der verbesserten CGI-Sicherheitsmaßnahmen wie hier beschrieben, um sicherzustellen, dass Ihre Web-Authentifizierungsinformationen nicht kompromittiert werden.

6) Kompilieren und installieren der Nagios-Plugins

Entpacken Sie die Nagios-Plugins-Quellcode-Archivdatei.

```
cd ~/downloads
tar xzf nagios-plugins-1.4.11.tar.gz
cd nagios-plugins-1.4.11
```

Kompilieren und installieren Sie die Plugins.

```
./configure --with-nagios-user=nagios --with-nagios-group=nagios
make
make install
```

7) Nagios starten

Fügen Sie Nagios zu der Liste der System-Services hinzu und sorgen Sie für einen automatischen Start, wenn das System hochfährt.

```
chkconfig --add nagios
chkconfig nagios on
```

Überprüfen Sie die Nagios-Beispielkonfigurationsdateien.

```
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

Wenn keine Fehler auftreten, starten Sie Nagios.

```
service nagios start
```

8) Anpassen der SELinux-Einstellungen

Fedora wird mit installiertem SELinux (Security Enhanced Linux) ausgeliefert und läuft im "Enforcing"-Modus. Dies kann zu "Internal Server Error"-Fehlern führen, wenn Sie versuchen, die Nagios-CGIs aufzurufen.

Schauen Sie, ob SELinux im Enforcing-Modus läuft.

```
getenforce
```

Setzen Sie SELinux in den "Permissive"-Modus.

```
setenforce 0
```

Damit diese Änderung dauerhaft wird, müssen Sie diese Einstellung in */etc/selinux/config* anpassen und das System neustarten.

Statt SELinux zu deaktivieren oder es in den Permissive-Modus zu versetzen, können Sie den folgenden Befehl benutzen, um die CGIs im Enforcing/Targeted-Modus laufen zu lassen:

```
chcon -R -t httpd_sys_content_t /usr/local/nagios/sbin/
chcon -R -t httpd_sys_content_t /usr/local/nagios/share/
```

Besuchen Sie das NagiosCommunity.org-Wiki unter http://www.nagioscommunity.org/wiki, um Informationen darüber zu erhalten, wie die Nagios-CGIs im Enforcing-Modus mit einer Targeted-Richtlinie ausgeführt werden.

9) Anmelden am Web-Interface

Sie sollten nun auf das Nagios-Web-Interface zugreifen können. Sie werden nach dem Benutzernamen (*nagiosadmin*) und Passwort gefragt, das Sie vorhin angegeben haben.

```
http://localhost/nagios/
```

Klicken Sie auf den "Service Detail"-Verweis in der Navigationsleiste, um Details darüber zu erhalten, was auf Ihrer lokalen Maschine überwacht wird. Es wird ein paar Minuten dauern, bis Nagios alle mit Ihrer Maschine verbundenen Services geprüft hat, weil die Prüfungen über eine gewisse Zeit verteilt werden.

10) Andere Anpassungen

Stellen Sie sicher, dass die Firewall-Einstellungen Ihrer Maschine einen Zugriff auf das Web-Interface ermöglichen, wenn Sie von anderen Rechnern darauf zugreifen wollen.

Die Konfiguration von e-Mail-Benachrichtigungen ist nicht Gegenstand dieser Anleitung. Nagios ist konfiguriert, um e-Mail-Benachrichtigungen zu versenden, aber möglicherweise ist auf Ihrem System noch kein Mail-Programm installiert bzw. konfiguriert. Schauen Sie in Ihre Systemdokumentation, suchen Sie im Web oder gucken Sie im NagiosCommunity.org-Wiki nach genauen Anweisungen, wie Ihr System konfiguriert werden muss, damit es e-Mail-Mitteilungen an externe Adressen versendet. Mehr Informationen zu Benachrichtigungen finden Sie hier.

11) Fertig

Glückwunsch! Sie haben erfolgreich Nagios installiert. Ihre Reise in die Überwachung hat gerade begonnen. Sie werden ohne Zweifel mehr als nur Ihre lokale Maschine überwachen wollen, so dass Sie u.a. die folgenden Abschnitte lesen sollten...

- Windows-Rechner überwachen
- Linux/Unix-Rechner überwachen
- Netware-Server überwachen
- Router/Switches überwachen
- Öffentlich zugängliche Dienste (HTTP, FTP, SSH, etc.) überwachen

Schnellstart openSUSE

Hoch zu: Inhalt

Siehe auch: Schnellstart-Installationsanleitungen, Sicherheitsüberlegungen

Einführung

Diese Schnellstartanleitung ist dazu gedacht, Ihnen einfache Anweisungen zu liefern, wie Sie Nagios innerhalb von 20 Minuten aus dem Quellcode auf openSUSE installieren und Ihren lokalen Rechner damit überwachen. Hier werden keine fortgeschrittenen Installationsoptionen vorgestellt - lediglich die Grundlagen, die für 95% aller Benutzer funktionieren, die anfangen wollen.

Diese Anweisungen basieren auf einer **openSUSE 10.2**-Installation.

Benötigte Pakete

Stellen Sie sicher, dass die folgenden Pakete installiert sind, bevor Sie fortfahren. Sie können mit *yast* Pakete unter openSUSE installieren.

- apache2
- C/C++-Development-Libraries

1) Benutzerinformationen erstellen

Werden Sie zum root-Benutzer.

```
su -l
```

Erstellen Sie ein neues Benutzerkonto *nagios* und vergeben Sie ein Passwort.

```
/usr/sbin/useradd -m nagios
passwd nagios
```

Legen Sie eine neue Gruppe *nagios* an. Fügen Sie den Nagios-Benutzer zu dieser Gruppe hinzu.

```
/usr/sbin/groupadd nagios
/usr/sbin/usermod -G nagios nagios
```

Legen Sie eine neue Gruppe *nagcmd* an, damit über das Web-Interface externe Befehle erteilt werden können. Fügen Sie den Nagios- und den Web-Server-Benutzer zu dieser Gruppe hinzu.

```
/usr/sbin/groupadd nagcmd
/usr/sbin/usermod -G nagcmd nagios
/usr/sbin/usermod -G nagcmd wwwrun
```

2) Nagios und Plugins herunterladen

Legen Sie ein Verzeichnis an, um die Dateien zu speichern.

```
mkdir ~/downloads
cd ~/downloads
```

Laden Sie die Quellcode-Archive von Nagios und den Nagios-Plugins herunter (besuchen Sie http://www.nagios.org/download/, um Verweise auf die aktuellsten Versionen zu sehen). Die Anweisungen wurden mit Nagios 3.0.6 und Nagios-Plugins 1.4.11 getestet.

```
wget http://osdn.dl.sourceforge.net/sourceforge/nagios/nagios-3.0.6.tar.gz
wget http://osdn.dl.sourceforge.net/sourceforge/nagiosplug/nagios-plugins-1.4.11.tar.gz
```

3) Nagios kompilieren und installieren

Entpacken Sie die Nagios-Quellcode-Archivdatei.

```
cd ~/downloads
tar xzf nagios-3.0.6.tar.gz
cd nagios-3.0.6
```

Führen Sie das Nagios-configure-Script aus und übergeben Sie den Namen der Gruppe, die Sie vorhin angelegt haben:

```
./configure --with-command-group=nagcmd
```

Kompilieren Sie den Nagios-Quellcode.

```
make all
```

Installieren Sie die Programme, Init-Scripte, Beispiel-Konfigurationsdaten und setzen Sie die Berechtigungen auf das Verzeichnis für die externen Befehle.

```
make install
make install-init
make install-config
make install-commandmode
```

Starten Sie Nagios noch nicht - es gibt noch ein paar Dinge zu tun...

4) Anpassen der Konfiguration

Die Beispiel-Konfigurationsdateien wurden nun im */usr/local/nagios/etc*-Verzeichnis installiert. Diese Beispieldateien sollten funktionieren, um mit Nagios zu starten. Nun fehlt nur noch eine Änderung, bevor Sie fortfahren können...

Ändern Sie die */usr/local/nagios/etc/objects/contacts.cfg*-Konfigurationsdatei mit Ihrem bevorzugten Editor und passen die e-Mail-Adresse in der *nagiosadmin*-Kontaktdefinition an, so dass sie die Adresse enthält, die im Falle von Alarmen benachrichtigt werden soll.

```
vi /usr/local/nagios/etc/objects/contacts.cfg
```

5) Konfigurieren des Web-Interface

Installieren Sie die Nagios-Web-Konfigurationsdateien im Apache conf.d-Verzeichnis.

```
make install-webconf
```

Legen Sie ein *nagiosadmin*-Konto an, um sich am Web-Interface anmelden zu können. Merken Sie sich das Passwort, das Sie diesem Konto geben - Sie brauchen es später.

```
htpasswd2 -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
```

Starten Sie Apache neu, damit die Änderungen wirksam werden.

```
service apache2 restart
```

 Anmerkung: Prüfen Sie die Implementierung der verbesserten CGI-Sicherheitsmaßnahmen wie hier beschrieben, um sicherzustellen, dass Ihre Web-Authentifizierungsinformationen nicht kompromittiert werden.

6) Kompilieren und installieren der Nagios-Plugins

Entpacken Sie die Nagios-Plugins-Quellcode-Archivdatei.

```
cd ~/downloads
tar xzf nagios-plugins-1.4.11.tar.gz
cd nagios-plugins-1.4.11
```

Kompilieren und installieren Sie die Plugins.

```
./configure --with-nagios-user=nagios --with-nagios-group=nagios
make
make install
```

7) Nagios starten

Fügen Sie Nagios zu der Liste der System-Services hinzu und sorgen Sie für einen automatischen Start, wenn das System hochfährt.

```
chkconfig --add nagios
chkconfig nagios on
```

Überprüfen Sie die Nagios-Beispielkonfigurationsdateien.

```
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

Wenn keine Fehler auftreten, starten Sie Nagios.

```
service nagios start
```

8) Anmelden am Web-Interface

Sie sollten nun auf das Nagios-Web-Interface zugreifen können. Sie werden nach dem Benutzernamen (*nagiosadmin*) und Passwort gefragt, das Sie vorhin angegeben haben.

```
http://localhost/nagios/
```

Klicken Sie auf den "Service Detail"-Verweis in der Navigationsleiste, um Details darüber zu erhalten, was auf Ihrer lokalen Maschine überwacht wird. Es wird ein paar Minuten dauern, bis Nagios alle mit Ihrer Maschine verbundenen Services geprüft hat, weil die Prüfungen über eine gewisse Zeit verteilt werden.

9) Andere Anpassungen

Stellen Sie sicher, dass die Firewall-Einstellungen Ihrer Maschine einen Zugriff auf das Web-Interface ermöglichen, wenn Sie von anderen Rechnern darauf zugreifen wollen.

Sie können das wie folgt tun:

- Öffnen Sie das Kontrollzentrum
- Wählen Sie 'Administrator-Einstellungen', um das YaST-Administrator-Kontrollzentrum zu öffnen
- Wählen Sie 'Firewall' aus der Kategorie 'Sicherheit und Benutzer'
- Klicken Sie die 'erlaubte Dienste'-Option im Firewall-Konfigurationsfenster
- Fügen Sie 'HTTP Server' zu der Liste der zugelassenen Dienste für die 'externe Zone'
- Klicken Sie 'Weiter' und 'Übernehmen' zur Aktivierung der neuen Firewall-Einstellungen

Die Konfiguration von e-Mail-Benachrichtigungen ist nicht Gegenstand dieser Anleitung. Nagios ist konfiguriert, um e-Mail-Benachrichtigungen zu versenden, aber möglicherweise ist auf Ihrem System noch kein Mail-Programm installiert bzw. konfiguriert. Schauen Sie in Ihre Systemdokumentation, suchen Sie im Web oder gucken Sie im NagiosCommunity.org-Wiki nach genauen Anweisungen, wie Ihr System konfiguriert werden muss, damit es e-Mail-Mitteilungen an externe Adressen versendet. Mehr Informationen zu Benachrichtigungen finden Sie hier.

10) Fertig

Glückwunsch! Sie haben erfolgreich Nagios installiert. Ihre Reise in die Überwachung hat gerade begonnen. Sie werden ohne Zweifel mehr als nur Ihre lokale Maschine überwachen wollen, so dass Sie u.a. die folgenden Abschnitte lesen sollten...

- Windows-Rechner überwachen
- Linux/Unix-Rechner überwachen
- Netware-Server überwachen
- Router/Switches überwachen
- Öffentlich zugängliche Dienste (HTTP, FTP, SSH, etc.) überwachen

---

Schnellstart Ubuntu

Hoch zu: Inhalt

Siehe auch: Schnellstart-Installationsanleitungen, Sicherheitsüberlegungen

Einführung

Diese Schnellstartanleitung ist dazu gedacht, Ihnen einfache Anweisungen zu liefern, wie Sie Nagios innerhalb von 20 Minuten aus dem Quellcode auf Ubuntu installieren und Ihren lokalen Rechner damit überwachen. Hier werden keine fortgeschrittenen Installationsoptionen vorgestellt - lediglich die Grundlagen, die für 95% aller Benutzer funktionieren, die anfangen wollen.

Diese Anweisungen basieren auf einer **Ubuntu 6.10** (Desktop)-Installation. Sie sollten auch für eine **Ubuntu 7.10**-Installation funktionieren.

Was dabei herauskommt

Wenn Sie diesen Anweisungen folgen, werden Sie am Ende folgendes haben:

- Nagios und die Plugins werden unterhalb von /usr/local/nagios installiert sein
- Nagios wird so konfiguriert sein, dass es einige Dinge auf Ihrem lokalen System überwacht (CPU-Auslastung, Plattenbelegung, usw.)
- Das Nagios-Web-Interface ist erreichbar unter http://localhost/nagios/

Benötigte Pakete

Stellen Sie sicher, dass die folgenden Pakete installiert sind, bevor Sie fortfahren.

- Apache 2
- GCC-Compiler und Development-Libraries
- GD-Development-Libraries

Sie können *apt-get* benutzen, um diese Pakete mit Hilfe der folgenden Befehle zu installieren:

```
sudo apt-get install apache2
sudo apt-get install build-essential
```

Unter Ubuntu 6.10 installieren Sie die gd2-Library mit diesem Befehl:

```
sudo apt-get install libgd2-dev
```

Unter Ubuntu 7.10 hat sich der Name des gd2-Library-Pakets geändert, so dass Sie folgenden Befehl benutzen müssen:

```
sudo apt-get install libgd2-xpm-dev
```

1) Benutzerinformationen erstellen

Werden Sie zum root-Benutzer.

```
sudo -s
```

Erstellen Sie ein neues Benutzerkonto *nagios* und vergeben Sie ein Passwort.

```
/usr/sbin/useradd -m nagios
passwd nagios
```

Auf der Ubuntu-Server-Edition (6.01 und ggf. neueren Versionen) müssen Sie außerdem eine *nagios*-Gruppe anlegen (sie wird nicht automatisch erstellt). Wahrscheinlich können Sie diesen Schritt auf den Ubuntu-Desktop-Editionen überspringen.

```
/usr/sbin/groupadd nagios
/usr/sbin/usermod -G nagios nagios
```

Legen Sie eine neue Gruppe *nagcmd* an, damit über das Web-Interface externe Befehle erteilt werden können. Fügen Sie den Nagios- und den Web-Server-Benutzer zu dieser Gruppe hinzu.

```
/usr/sbin/groupadd nagcmd
/usr/sbin/usermod -a -G nagcmd nagios
/usr/sbin/usermod -a -G nagcmd www-data
```

2) Nagios und Plugins herunterladen

Legen Sie ein Verzeichnis an, um die Dateien zu speichern.

```
mkdir ~/downloads
cd ~/downloads
```

Laden Sie die Quellcode-Archive von Nagios und den Nagios-Plugins herunter (besuchen Sie http://www.nagios.org/download/, um Verweise auf die aktuellsten Versionen zu sehen). Diese Anweisungen wurden mit Nagios 3.0.6 und Nagios-Plugins 1.4.11 getestet.

```
wget http://osdn.dl.sourceforge.net/sourceforge/nagios/nagios-3.0.6.tar.gz
wget http://osdn.dl.sourceforge.net/sourceforge/nagiosplug/nagios-plugins-1.4.11.tar.gz
```

3) Nagios kompilieren und installieren

Entpacken Sie die Nagios-Quellcode-Archivdatei.

```
cd ~/downloads
tar xzf nagios-3.0.6.tar.gz
cd nagios-3.0.6
```

Führen Sie das Nagios-configure-Script aus und übergeben Sie den Namen der Gruppe, die Sie vorhin angelegt haben:

```
./configure --with-command-group=nagcmd
```

Kompilieren Sie den Nagios-Quellcode.

```
make all
```

Installieren Sie die Programme, Init-Scripte, Beispiel-Konfigurationsdaten und setzen Sie die Berechtigungen auf das Verzeichnis für die externen Befehle.

```
make install
make install-init
make install-config
make install-commandmode
```

Starten Sie Nagios noch nicht - es gibt noch ein paar Dinge zu tun...

4) Anpassen der Konfiguration

Die Beispiel-Konfigurationsdateien wurden nun im */usr/local/nagios/etc*-Verzeichnis installiert. Diese Beispieldateien sollten funktionieren, um mit Nagios zu starten. Nun fehlt nur noch eine Änderung, bevor Sie fortfahren können...

Ändern Sie die */usr/local/nagios/etc/objects/contacts.cfg*-Konfigurationsdatei mit Ihrem bevorzugten Editor und passen die e-Mail-Adresse in der *nagiosadmin*-Kontaktdefinition an, so dass sie die Adresse enthält, die im Falle von Alarmen benachrichtigt werden soll.

```
vi /usr/local/nagios/etc/objects/contacts.cfg
```

5) Konfigurieren des Web-Interface

Installieren Sie die Nagios-Web-Konfigurationsdateien im Apache conf.d-Verzeichnis.

```
make install-webconf
```

Legen Sie ein *nagiosadmin*-Konto an, um sich am Web-Interface anmelden zu können. Merken Sie sich das Passwort, das Sie diesem Konto geben - Sie brauchen es später.

```
htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
```

Starten Sie Apache neu, damit die Änderungen wirksam werden.

```
/etc/init.d/apache2 reload
```

 Anmerkung: Prüfen Sie die Implementierung der verbesserten CGI-Sicherheitsmaßnahmen wie hier beschrieben, um sicherzustellen, dass Ihre Web-Authentifizierungsinformationen nicht kompromittiert werden.

6) Kompilieren und installieren der Nagios-Plugins

Entpacken Sie die Nagios-Plugins-Quellcode-Archivdatei.

```
cd ~/downloads
tar xzf nagios-plugins-1.4.11.tar.gz
cd nagios-plugins-1.4.11
```

Kompilieren und installieren Sie die Plugins.

```
./configure --with-nagios-user=nagios --with-nagios-group=nagios
make
make install
```

7) Nagios starten

Fügen Sie Nagios zu der Liste der System-Services hinzu und sorgen Sie für einen automatischen Start, wenn das System hochfährt.

```
ln -s /etc/init.d/nagios /etc/rcS.d/S99nagios
```

Überprüfen Sie die Nagios-Beispielkonfigurationsdateien.

```
/usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

Wenn keine Fehler auftreten, starten Sie Nagios.

```
/etc/init.d/nagios start
```

8) Anmelden am Web-Interface

Sie sollten nun auf das Nagios-Web-Interface zugreifen können. Sie werden nach dem Benutzernamen (*nagiosadmin*) und Passwort gefragt, das Sie vorhin angegeben haben.

```
http://localhost/nagios/
```

Klicken Sie auf den "Service Detail"-Verweis in der Navigationsleiste, um Details darüber zu erhalten, was auf Ihrer lokalen Maschine überwacht wird. Es wird ein paar Minuten dauern, bis Nagios alle mit Ihrer Maschine verbundenen Services geprüft hat, weil die Prüfungen über eine gewisse Zeit verteilt werden.

9) Andere Anpassungen

Wenn Sie e-Mail-Benachrichtigungen über Nagios-Alarme bekommen wollen, installieren Sie das mailx-(Postfix)-Paket.

```
sudo apt-get install mailx
sudo apt-get install postfix
```

Sie müssen die Nagios-e-Mail-Benachrichtigungsbefehle in */usr/local/nagios/etc/objects/commands.cfg* anpassen und alle '/bin/mail'-Verweise in '/usr/bin/mail' ändern. Anschließend müssen Sie Nagios neustarten, damit die Änderungen aktiv werden.

```
sudo /etc/init.d/nagios restart
```

Die Konfiguration von e-Mail-Benachrichtigungen ist nicht Gegenstand dieser Anleitung. Nagios ist konfiguriert, um e-Mail-Benachrichtigungen zu versenden, aber möglicherweise ist auf Ihrem System noch kein Mail-Programm installiert bzw. konfiguriert. Schauen Sie in Ihre Systemdokumentation, suchen Sie im Web oder gucken Sie im NagiosCommunity.org-Wiki nach genauen Anweisungen, wie Ihr System konfiguriert werden muss, damit es e-Mail-Mitteilungen an externe Adressen versendet. Mehr Informationen zu Benachrichtigungen finden Sie hier.

10) Fertig

Glückwunsch! Sie haben erfolgreich Nagios installiert. Ihre Reise in die Überwachung hat gerade begonnen. Sie werden ohne Zweifel mehr als nur Ihre lokale Maschine überwachen wollen, so dass Sie u.a. die folgenden Abschnitte lesen sollten...

- Windows-Rechner überwachen
- Linux/Unix-Rechner überwachen
- Netware-Server überwachen
- Router/Switches überwachen
- Öffentlich zugängliche Dienste (HTTP, FTP, SSH, etc.) überwachen

---

Nagios®

maßgeschneiderte Objektvariablen

Hoch zu: Inhalt

Siehe auch: Objektkonfiguration, Objektvererbung, Makros

Einführung

Benutzer fragen oft nach neuen Variablen in Host-, Service- und Kontaktdefinitionen. Dazu gehören Variablen für die SNMP-Community, MAC-Adressen, AIM-Benutzernamen, Skype-Nummern und Straßennamen. Die Liste ist endlos. Das Problem, was ich darin sehe ist, dass Nagios weniger generisch und mehr infrastrukturspezifisch wird. Nagios war dazu gedacht, flexibel zu sein, was bedeutet, dass die Dinge in einer generischen Art und Weise geplant waren. Host-Definitionen in Nagios zum Beispiel haben eine generische "address"-Variable, die alles von einer IP-Adresse bis zu menschlich-lesbaren Wegbeschreibungen enthalten kann - was immer für die Umgebung des Benutzers angemessen ist.

Trotzdem muss es eine Methode für Administratoren geben, in ihrer Nagios-Konfiguration Informationen zu ihren Infrastrukturkomponenten zu speichern, ohne anderen einen Satz von speziellen Variablen aufzubürden. Nagios versucht dieses Problem zu lösen, indem es Benutzern erlaubt, maßgeschneiderte Variablen in ihren Objektdefinitionen anzugeben. Maßgeschneiderte Variablen erlauben es Benutzern, zusätzliche Eigenschaften in ihren Host-, Service- und Kontaktdefinitionen anzugeben und ihre Werte in Benachrichtigungen, Eventhandlern sowie Host- und Service-Prüfungen zu benutzen.

Grundlagen zu maßgeschneiderten Variablen

Es gibt ein paar wichtige Dinge, die Sie bei maßgeschneiderten Variablen beachten sollten:

- maßgeschneiderte Variablennamen müssen mit einem Unterstrich (_) beginnen, um einen Namenskonflikt mit Standardvariablen zu verhindern
- maßgeschneiderten Variablennamen sind unabhängig von Groß- und Kleinschreibung (case-insensitive)
- maßgeschneiderten Variablen werden von Objektvorlagen wie normale Variablen geerbt
- Scripts können sich mit Makros und Umgebungsvariablen auf die Werte von maßgeschneiderten Variablen beziehen

Beispiele

Hier ein Beispiel, wie maßgeschneiderte Variablen in verschiedenen Arten von Objektdefinitionen definiert werden können:

```
define host{
        host_name       linuxserver
        _mac_address    00:06:5B:A6:AD:AA       ; <-- Custom MAC_ADDRESS variable
        _rack_number    R32                     ; <-- Custom RACK_NUMBER variable
        ...
        }

define service{
        host_name       linuxserver
        description     Memory Usage
        _SNMP_community public                  ; <-- Custom SNMP_COMMUNITY variable
```

```
        _TechContact    Jane Doe                ; <-- Custom TECHCONTACT variable
        ....
        }

define contact{
        contact_name    john
        _AIM_username   john16                  ; <-- Custom AIM_USERNAME variable
        _YahooID        john32                  ; <-- Custom YAHOOID variable
        ...
        }
```

maßgeschneiderte Variablen als Makros

Maßgeschneiderte Variablen können über Makros oder Umgebungsvariablen in Scripts und Programmen eingesetzt werden, die Nagios für Prüfungen, Benachrichtigungen usw. ausführt.

Um Namenskonflikte zwischen maßgeschneiderten Variablen aus verschiedenen Objektarten zu verhindern, stellt Nagios "_HOST", "_SERVICE" oder "_CONTACT" an den Anfang von maßgeschneiderten Host-, Service- oder Kontaktvariablennamen in Makros und Umgebungsvariablen. Die folgende Tabelle zeigt die entsprechenden Namen für maßgeschneiderte Variablen, die im obigen Beispiel definiert wurden.

Objekttyp	Variablenname	Makroname	Umgebungsvariable
Host	MAC_ADDRESS	$_HOSTMAC_ADDRESS$	NAGIOS__HOSTMAC_ADDRESS
Host	RACK_NUMBER	$_HOSTRACK_NUMBER$	NAGIOS__HOSTRACK_NUMBER
Service	SNMP_COMMUNITY	$_SERVICESNMP_COMMUNITY$	NAGIOS__SERVICESNMP_COMMUNITY
Service	TECHCONTACT	$_SERVICETECHCONTACT$	NAGIOS__SERVICETECHCONTACT
Contact	AIM_USERNAME	$_CONTACTAIM_USERNAME$	NAGIOS__CONTACTAIM_USERNAME
Contact	YAHOOID	$_CONTACTYAHOOID$	NAGIOS__CONTACTYAHOOID

maßgeschneiderte Variablen und Vererbung

Maßgeschneiderte Objektvariablen werden genau wie Standard-Host-, Service- oder Kontaktvariablen vererbt.
