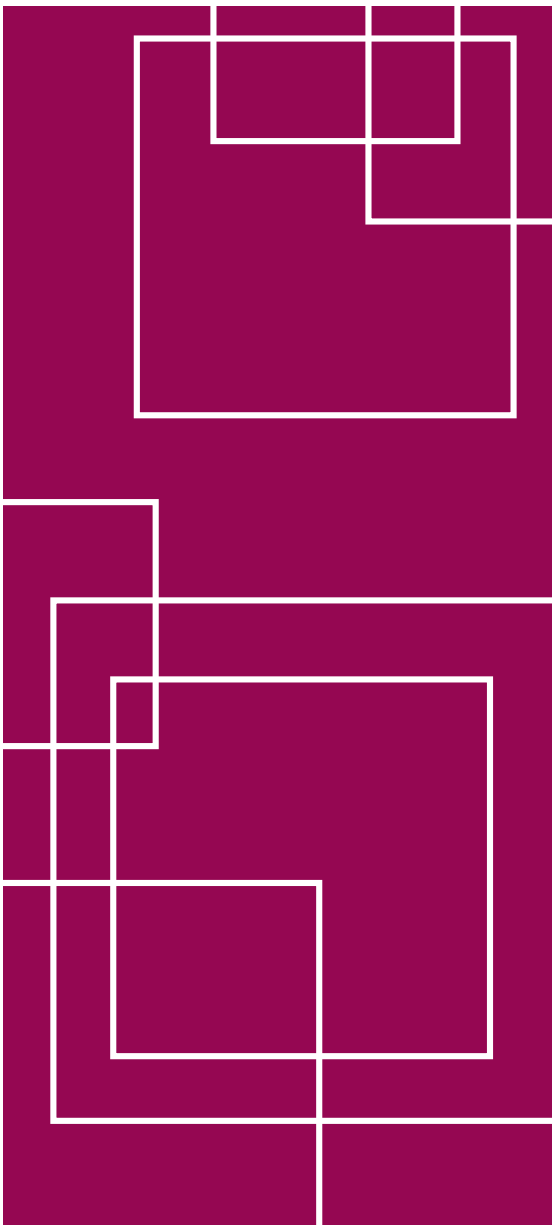
Nagios-Werkstatt:
Eigenbau von Perl-Plugins



Das riesige Angebot fertiger Lösungen ist eine der großen Stärken von Nagios. Wo selbst das nicht reicht, ist es aber auch nicht schwer, den Bedarf mit selbst geschriebenen Skripten zu decken. Wolfgang Barth

Nagios lässt prüfen. Das heißt, es kümmert sich nicht selbst darum, ob ein bestimmter Dienst verfügbar und leistungsfähig ist, sondern delegiert diesen Testjob an kleine Helfer, die so genannten Plugins. Das sind einfache, spezialisierte Prüfprogramme, die, mit ein paar Argumenten auf der Kommandozeile aufgerufen, einen Statuscode zusammen mit einem einzeiligen Text zurückliefern:

```
nagios@swobspace:~/libexec$./check_smtp ➥
-H 192.168.1.9
SMTP OK - 0.021 sec. response time|➥
time=0.020599s;;;0.000000
```

Die Ausgabe erfolgt immer nach dem Schema »*Art_des_Checks Status - Information*«. Der Text nach dem Minuszeichen ist in erster Linie als Information für den Administrator gedacht. Er kann etwa in einer Weboberfläche oder in einer Nachricht Verwendung finden. Hinter dem Pipezeichen »|« folgen Performancedaten – dazu später mehr. Nagios selbst wertet einzig und allein den Rückgabewert aus: Dabei steht der Wert 0 für »OK«, 1 bedeutet »WARNING«, 2 »CRITICAL« und 3 »UNKNOWN«.
Während die ersten drei Zustände – die Web-GUI von Nagios stellt sie in den Farben Grün, Gelb und Rot dar – sich selbst erklären, hat der Zustand »UNKNOWN« eine besondere Bedeutung. Er signalisiert ein Problem beim Ausführen des Plugins. Der Test lief nicht, der tatsächliche Zustand des Services ist unbekannt. In der Weboberfläche wird das durch die Farbe Orange signalisiert (siehe **Abbildung 1**).

Fertige Lösungen für fast jede Überwachungsaufgabe

Mit den offiziellen Nagios-Plugins des Maintainers Ton Voon von der Nagios-Homepage wird bereits ein umfangreicher Werkzeugkasten mitgeliefert. Wer jeweils den aktuellsten Stand einsetzen möchte, kann sich diesen von **(1)** herunterladen und mit dem Dreisatz »configure && make && make install« selbst installieren.

Eine weitere Möglichkeit, an unzählige fertige Plugins zu kommen, ist die offizielle Nagios-Tauschbörse **(2)**, die die Netways GmbH bereitstellt. Trotz des großen Angebots findet man in seltenen Fällen dennoch nichts Passendes. Das ist aber kein Beinbruch. Plugins selbst zu schreiben, ist zum Glück nicht besonders schwer, vorausgesetzt, man beherrscht eine Skriptsprache wie die Bash oder Perl einigermaßen. Dieser Artikel beschreibt die Grundlagen der Plugin-Programmierung in Perl. Die hier vorgestellten Codebeispiele sind unter **(3)** verfügbar.

Plugins nach dem Baukastenprinzip

Jedes Plugin setzt sich aus mehreren Funktionsgruppen zusammen (**Abbildung 2**). Ganz zu Anfang analysiert es seine Kommandozeile und ermittelt die Aufrufparameter. Dabei sollte es auch gleich deren Syntax (Vollständigkeit der Angaben) und wenn möglich Semantik (Zulässigkeit der übergebenen Werte) prüfen. Danach folgt der eigentliche Test, der den Zustand eines Services ermittelt. Das Ergebnis wird wie beschrieben als Text und mit einem Rückgabewert ausgegeben.
Ein ordentliches Plugin enthält außerdem eine Onlinehilfe. Falls es Messwerte ermittelt, sollte es sie auch als Performancedaten in standardisierter Form weiterreichen. Abgesehen von den eigentlichen Funktionstests, lassen sich alle anderen Funktionen weitgehend standardisieren. Für Standardaufgaben hat Perl seine praktischen Module – die Analyse der Kommandozeile erledigt etwa das Modul GetOpt::Long.
Es gibt einige reservierte Optionen (**Tabelle 1**), die nur für einen festgelegten Zweck benutzbar sind. Alle anderen darf man beliebig benennen. Eine Manpage lässt sich als Perl-Online-Dokumentation (POD) gleich in den Quellcode einbauen. Zusätzlich kann das Modul Pod::Usage diese Online-Dokumentation auch als Kurzhilfe in verkürzter Form ausgeben – etwa nach einer Fehlbedienung – ohne den Text doppelt zu be-

Host	Service	Status	Last Check	Duration	Attempt	Status Information
swobspace	PING	OK	2007-01-27 17:19:57	247d 6h 25m 54s	1/3	OK - 192.168.1.9: rta 0.065ms, lost 0%
	PostgreSQL	UNKNOWN	2007-01-27 17:19:27	0d 0h 0m 54s	3/3	check_pgsql: Unknown argument - (null)
	fs_a	WARNING	2007-01-27 17:20:00	29d 6h 32m 7s	3/3	DISK WARNING - free space: /net/swobspace/a 3547 MB (19% inode=94%):
	fs_b	WARNING	2007-01-27 17:19:30	54d 23h 49m 47s	3/3	DISK WARNING - free space: /net/swobspace/b 4322 MB (11% inode=99%):
	fs_c	CRITICAL	2007-01-27 17:20:03	49d 0h 15m 41s	3/3	DISK CRITICAL - free space: /net/swobspace/c 1835 MB (9% inode=82%):

Abbildung 1: Mit einer solchen Übersicht zeigt Nagios dem Admin auf einen Blick, in welchen Zustand sich welcher Service befindet.

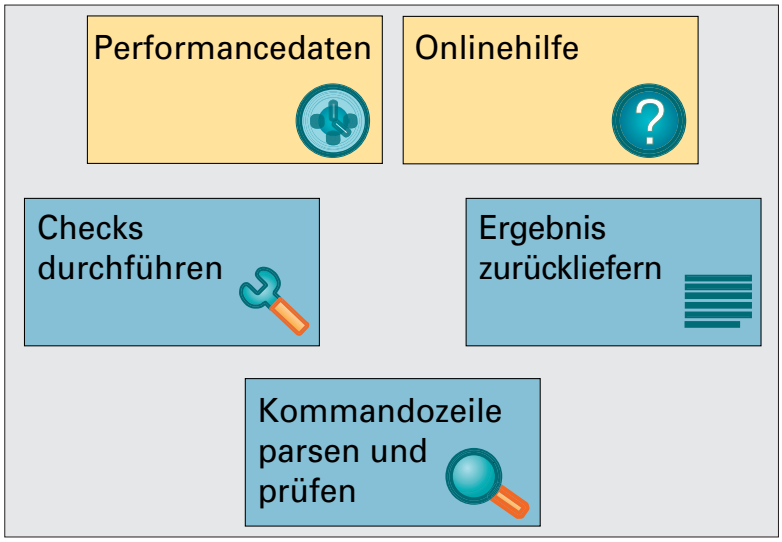


Abbildung 2: In diese Funktionsgruppen gliedert sich intern ein typisches Nagios-Plugin. Für viele immer wiederkehrende Aufgaben bieten sich dabei Bibliotheksfunktionen an, die Nagios::Plugin anbietet.

nötigen. Für die Ergebnisausgabe eignet sich das recht neue Modul Nagios::Plugin, das neben einer Ausgabefunktion auch passende Konstanten für die Zustandswerte oder Funktionen für Performancedaten enthält.

Das Modul Nagios::Plugin installieren

Das Modul Nagios::Plugin ist noch recht neu und daher in den Standard-Linux-Distributionen kaum verbreitet. Dank CPAN (Comprehensive Perl Archive Network, (4)) ist es trotzdem recht einfach zu installieren. Wer bis jetzt noch nie etwas aus dem CPAN installiert hat, sollte allerdings zunächst das Grundpaket Bundle::CPAN laden. Dazu führt man als Benutzer Root lediglich das folgende Kommando aus: »perl -MCPAN -e 'install Bundle::CPAN'«. Dabei fragt der Perl-Interpreter eine Reihe von Einstellungen ab, die man bis auf die Angabe der CPAN-Server alle bedenkenlos mit der Eingabetaste bestätigen kann.

Danach lädt Perl eine Reihe von Modulen nach. Bei der Modulversion 0.15 von Nagios::Plugin sind die Abhängigkeiten noch nicht vollständig implementiert; um Fehler zu vermeiden, gilt es deshalb zuerst noch, das Modul Test::Exception zu installieren: »perl -MCPAN -e 'install Test::Exception'«. Jetzt steht einer Installation von Nagios::Plugin nichts mehr im Wege: »perl -MCPAN -e 'install Nagios::Plugin'«.

Übrigens: Welche Module Perl wann und wohin installiert hat, hält die Datei »/usr/local/perl/*perlversion*/perllocal.pod« fest, die sich mit perldoc formatiert ausgeben lässt: »perldoc perllocal.pod«

Im Blick behalten, ob die DSL-Leitung steht

Als erstes Beispiel (**Listing 1**) dient ein ganz einfaches Plugin, das die Verfügbarkeit einer DSL-Leitung prüft, für die auf dem lokalen Rechner ein PPP-over-Ethernet- (PPPoE)-Prozess startet. Es enthält zwar weder eine Onlinehilfe noch wertet es die Kommandozeile aus, ist dafür aber direkt einsetzbar. Es genügt, das Interface einzutragen, an dem die DSL-Leitung angeschlossen ist. Daraufhin fragt das Plugin mittels »pppoe -I <interface> -A« ab, ob sich die Gegenstelle mel-

Tabelle 1: Reservierte Optionen von Nagios-Plugins

Short	Long	Beschreibung
-V	--version	Ausgabe der Plugin-Version, Plugin beendet sich sofort
-h	--help	Ausgabe einer Onlinehilfe, die bei -h als Kurzhilfe, bei --help als vollständige Hilfe ausfallen kann
-t	--timeout	Timeout, nach dem das Plugin den Test abbricht
-w	--warning	Warnschwellwert
-c	--critical	kritischer Schwellwert
-H	--hostname	Hostname oder IP-Adresse (bei Netzwerkchecks)
-v	--verbose	Debugging-Ausgaben
-C	--community	SNMP-Passwort (nur bei SNMP-Abfragen)
-a	--authentication	Passwort, sonstige Authentifikationsangaben
-l	--logname	Loginname
-p	--port, --passwd, --passwort	normalerweise für die Angabe eines Zielports bei Netzwerkplugins, ist aber auch für die Angabe eines Passwortes verwendbar.
-u	–url –username	URL und/oder Username für die Authentifikation.

det. Dabei wird der Name der Gegenstelle (des so genannten Access Concentrator) und dessen MAC-Adresse ausgegeben (**Abbildung 3**).
Das Laden des Moduls Nagios::Plugin in der vierten Zeile importiert automatisch einige Konstanten: »OK«, »WARNING«, »CRITICAL« und »UNKNOWN«, die sich später direkt für Rückgabewerte verwenden lassen. Das Interface des Moduls ist objektorientiert. Die zehnte Zeile legt eine neue Instanz an. Mit dem Attribut »shortname« verpasst man dem Test gleich einen passenden Namen, den später bei der Textausgabe auch die Methoden »nagios_exit« (Zeile 28 und 32) und »nagios_die« (Zeile 13 und 35) mit ausgegeben (**Abbildung 4**). Dort folgt nach der Testbezeichnung bei beiden Methoden auch der Status »OK«.

DSL-Leitung im Blick behalten

Der Unterschied zwischen den beiden Methoden »nagios_exit« und »nagios_die« ist marginal: Bei »nagios_die« ist der Status »UNKNOWN« voreingestellt, diese Methode verwendet man bei Plugin-Fehlern. Der Status »nagios_exit« erfordert dagegen die explizite Angabe eines Rückgabewertes. Hier kommen die bereits erwähnten Konstanten »OK« und »CRITICAL« zum Zug. Die eigentliche Logik des Plugin ist schnell erklärt: Mit »open« (Zeile 12) wird das Programm »pppoe« für den Verbindungsaufbau

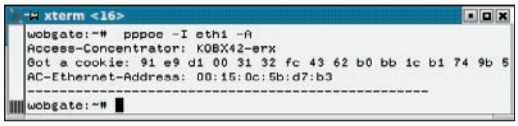


Abbildung 3: DSL-Test mit PPPoE: Am anderen Ende meldet sich der Access Concentrator des Providers. Die Leitung steht.

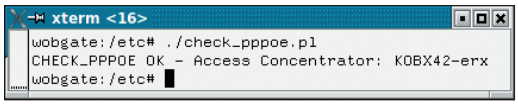


Abbildung 4: Ausgabe von check_pppoe.pl. Das Perl-Skript gibt dieselben Werte in Nagios-Manier zurück.

zum Provider gestartet und seine Ausgabe zu dem Filedeskriptor »OUT« geleitet. Die While-Schleife (Zeile 15-23) arbeitet diesen Output anschließend Zeile für Zeile ab. Bei bestehendem DSL-Link erhält das Skript in Zeile 18 den Namen des Access Concentrator auf der Seite des Providers. Im Fehlerfall gibt »pppoe« jedoch einen Timeout-Text (siehe Zeile 20) an den Aufrufer zurück.
Der DSL-Check liefert im Wesentlichen nur die Information, ob die Verbindung steht oder nicht, aber keine Messwerte und damit auch keine Performancedaten.
Das nächste Plugin, das an dieser Stelle vorgestellt werden soll – »check_du.pl« –, wertet den

Listing 1: check_pppoe.pl

```
01 #!/usr/bin/perl -w
02 use strict;
03 use warnings;
04 use Nagios::Plugin 0.15;
05
06 my $interface = 'eth1';
07 my $accessconcentrator = '';
08 my $failedconn = 0;
09 my $verbose = 0;
10 my $np = Nagios::Plugin->new( shortname =>
   "CHECK_PPPOE" );
11
12 open ( OUT, "/usr/sbin/pppoe -I $interface -A
   2>&1 |" )
13    or $np->nagios_die( "can't start /usr/sbin/
   pppoe" );
14
15 while (<OUT>) {
16    print "$_" if ($verbose);
17    chomp $_;
18    if ( /Access-Concentrator:\s+([^ ]+.*$)/ ) {
19       $accessconcentrator = $1;
20    } elsif ( /Timeout waiting for PADO packets/
   ) {
21       $failedconn++;
22    }
23 }
24 close (OUT);
25
26 SWITCH: {
27    if ( "$accessconcentrator" ne "" ) {
28       $np->nagios_exit( OK, "Access Concentrator:
   $accessconcentrator");
29       last SWITCH;
30    }
31    if ( $failedconn > 0 ) {
32        $np->nagios_exit(CRITICAL, "no DSL link"
   );
33       last SWITCH;
34    }
35    $np->nagios_die( "for unknown reasons" );
36 }
```

Platzbedarf einzelner Files oder aller Dateien in einem bestimmten Verzeichnis aus. Es illustriert als Programmierbeispiel für ein bereits recht vollständiges Plugin den richtigen Umgang mit Schwellwerten und mit Performancedaten, mit Kommandozeilenoptionen und mit einer eigenen Onlinehilfe, die alle Aufrufparameter erklärt.

Datei- und Verzeichnissgrößen per Plugin testen

Das Plugin »check_du.pl« (**Listing 2**) ermittelt und addiert mit Hilfe des Unix-Programms »du« die Größe von Verzeichnissen oder auch von einzelnen Dateien und prüft anschließend, ob die ermittelte Gesamtgröße im Rahmen vorher festgelegter Schwellwerte liegt.

Am Anfang des Skripts befindet sich ein Abschnitt mit der zugehörigen Perl-Online-Dokumentation (POD, Zeile 3-43), die den Text mit einfachen Schlüsselwörtern auszeichnet. Später lässt sich diese Doku mit dem Befehl »perldoc check_du.pl« formatiert als Manpage ausgeben. Der Dokumentationsabschnitt endet mit dem Befehl »=cut« (in Zeile 43).

Die Schwellwerte übergibt man im Nagios-Threshold-Format, das in den Nagios-Developer-Guidlines **(5)** ausführlich beschrieben ist. Im einfachsten Fall ist nur ein Integerwert nötig: »-w 4000« gibt eine Warnung aus, wenn die er-

Listing 2: »check_du.pl«

```
001 #!/usr/bin/perl -w
002
003 =head1 NAME
004
005 Scheck_du.pl - Nagios plugin for checking size
    of directories and files
006
007 =head1 SYNOPSIS
008
009  check_du.pl -P path/pattern [-v] \
010              [-w warning_threshold]  \
                 [-c critical_threshold]
011  check_du.pl [-h|-V]
012
013 =head1 OPTIONS
014
015 =over 4
016
017 =item -P|--path=expression
018
019 Path expression for calculating size. May be a
    shell expression like
020 /var/log/*.log
021
022 =item -w|--warning=threshold
023 threshold can be max (warn if < 0 or > max),
    min:max (warn if < min or >
024 max), min: (warn if < min), or @min:max (warn if
    >= min and <= max). All values must be
025 integer.
026
027 =item -c|--critical=threshold
028
029 see --warning for explanation of threshold
    format
030
031 =item -v|--verbose
032
033 34increases verbosity, specify twice to see the
    original output from sapinfo.
034
035 =item -V|--version
036
037 print version an exit
038
039 =item -h|--help
040
041 print help message and exit
042
043 =cut
044 use strict;
045 use warnings;
046
047 use Nagios::Plugin 0.15;
048 use Getopt::Long qw(:config no_ignore_case
    bundling);
049 use Pod::Usage;
050
051
052 my $np = Nagios::Plugin->new( shortname =>
    "CHECK_DU" );
053 my $line = '';
054 my $warn_threshold = '0:';
055 my $crit_threshold = '0:';
056 my $what = '';
057 my $denied = 0;
058 my $size = 0;
059 my $result = UNKNOWN;
060 my $version = 'V1.0/2007-01-28/wob';
061 my $printversion = 0;
```

mittelte Gesamtgröße größer als 4000 KByte ist. Das Kommando »du« arbeitet dabei per Default mit 1-KByte großen Blöcken. Andere Blockgrößen lassen sich ebenfalls verwenden, wenn man in diesen Fällen alternativ die »du«-Option »--blocksize« benutzt.

Die Funktion »GetOptions()« ab Zeile 66 zerlegt die Kommandozeile und übergibt die vorgefundenen Parameter an die jeweiligen Varia-blen. »w|warning=s« bedeutet, dass die Option wahlweise als Short Option »-w *string*« oder als Long Option »?warning=*string*« zulässig ist.

Ruft man das Plugin mit einer unbekannten Option auf, dann liefert die Funktion »GetOptions()« einen Fehler als Rückgabewert. Der sorgt seinerseits wiederum für den Aufruf der Funktion »pod2usage()« in Zeile 73. Der Verbose-Level 0 führt hier lediglich zu einer sehr kurzen Hilfestellung, der Verbose-Level 2 gibt dagegen eine vollständige Manualseite aus. Die »pod2usage()«-Aufrufe in den Zeilen 77, 81 und 86 kommen allerdings nur dann tatsächlich zum Zug, wenn die zugehörige If-Bedingung erfüllt ist, die sich jeweils an die Funktion anschließt.

Für die Verarbeitung der Schwellwerte im Threshold-Format hält Nagios::Plugin gleich mehrere Methoden bereit. Die Funktion »set_thresholds()« in Zeile 120 übergibt die aus der Kommandozeile oder den Voreinstellungen ermittelten Werte an die Plugin-Instanz »$np«. In der-

Listing 2: »check_du.pl« Fortsetzung

```
062 my $verbose = 0;
063 my $help = 0;
064
065 # -- GetOpt
066 GetOptions(
067    "P|path=s"           => \$what,
068    "w|warning=s"        => \$warn_threshold,
069    "c|critical=s"       => \$crit_threshold,
070    "h|help"             => \$help,
071    "V|version"          => \$printversion,
072    "v|verbose+"         => \$verbose,
073 ) or pod2usage({ -exitval => UNKNOWN,
074                  -verbose => 0,
075                  -msg     => "*** unknown
    argument found ***" });
076
077 pod2usage(-verbose => 2,
078           -exitval => UNKNOWN
079          ) if ( $help );
080
081 pod2usage(-msg     => "\n$0 -- version:
    $version\n",
082           -verbose => 0,
083           -exitval => UNKNOWN
084          ) if ( $printversion );
085
086 pod2usage(-msg     => "*** no path/pattern
    specified ***",
087           -verbose => 0,
088           -exitval => UNKNOWN
089          ) if ( "$what" eq "" );
090
091 # -- thresholds
092 $np->set_thresholds(
093    warning  => $warn_threshold,
094    critical => $crit_threshold);
095
096 # -- main
097 open ( OUT, "LANG=C; /usr/bin/du -cs $what 2>&1
    |" )
098    or $np->nagios_die( "can't start /usr/bin/du"
    );
099
100 while (<OUT>) {
101    print "$_" if ($verbose);
102    chomp $_;
103    my $line = $_;
104    $denied++ if ( /Permission denied/i );
105
106    if ( /^(\d+)\s+total$/i ) { # last line
107       $size = $1;
108       last;
109    }
110 }
111 close (OUT);
112
113 $np->add_perfdata( label => "size", value =>
    $size,
114                    uom => "kB", threshold =>
    $np->threshold() );
115
116 if ( $denied ) {
117    $np->nagios_exit( CRITICAL, "unreadable
    directories or files");
118 }
119
120 $result = $np->check_threshold(  $size );
121
122 $np->nagios_exit( $result, "check size: $size
    kByte");
```

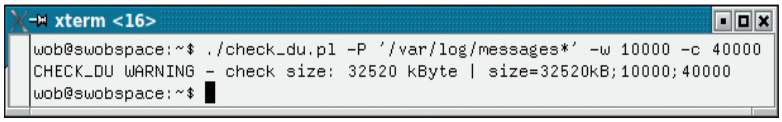


Abbildung 5: Achtung, das Message-File in »/var/log« droht überzulaufen! Das besagt die Ausgabe des hier vorgestellten Plugin »check_du.pl«, das den Platzbedarf von Dateien und Verzeichnissen ermittelt.

selben Zeile 120 ermittelt »check_threshold()s« dann auch die Gesamtgröße aller Files. Der Rückgabewert ist identisch mit den in Nagios generell üblichen Werten für die Statuscodes »OK«, »WARNING« und »CRITICAL« und lässt sich daher auch direkt in »nagios_exit()« verwenden (wie in Zeile 122).

Performancedaten

Die Textausgabe eines Plugin ist, wie schon gesagt, in erster Linie für den Administrator gedacht, und soll ihn per Weboberfläche oder über das Benachrichtigungssystem mit Informationen über das Ergebnis der Überprüfungen versorgen.

Für die Übergabe von Performancedaten an externe Programme, die solche Werte dann grafisch aufbereiten können (ein Beispiel dafür ist der Nagios Grapher, den ein weiterer Beitrag dieser Ausgabe ausführlich vorstellt), gibt es eine besondere Syntax. Bei der Weiterleitung solcher Werte folgt hinter der Textausgabe ein »|«-Zeichen. Alle folgenden Textausgaben unterdrückt Nagios. Das einheitliche Format für Performancedaten lautet: »name=wert[uom]; warn;crit[;min[;max]]«.

Dabei ist »name« die Bezeichnung für eine Variable. Die Abkürzung »uom« hinter dem zugeordneten Wert steht für Unit of Measurement, also eine Maßeinheit. Als Einheiten kommen »%« (Prozentangabe), »s« (eine Zeitangabe in Sekunden, auch Milli- oder Mikrosekunden), »B« (eine Datengröße in Byte, auch KByte, MByte oder TByte) oder »c« für einen monoton wachsenden Zähler (Counter) infrage. Die Angabe von Minimum und Maximum ist optional. Nagios::Plugin stellt die Methode »add_perfdata()« zur Verfügung (Zeile 113), die Information wird beim Aufruf von »nagios_exit()« einfach mit ausgegeben (**Abbildung 5**). Einfacher geht es kaum mehr!

Ausblick

Auch das zweite Beispiel ist noch nicht hundertprozentig perfekt. Noch nachzurüsten wäre etwa ein Timeout-Mechanismus, über den Nagios-Plugins immer verfügen sollten. Er bräche nach einer voreingestellten Zeit ihre Ausführung ab. Das wiederum würde gewährleisten, dass sich Nagios nicht unnötig lange mit aussichtslosen Versuchen aufhält, falls etwas bei der Skriptausführung klemmt. Es hat sich mittlerwile eingebürgert, diesen Timeout auf 10 Sekunden festzusetzen.

Ermittelt man die Gesamtgröße von vielen Verzeichnissen, reicht der Timeout aber nicht. In Perl lässt sich der zeitgesteuerte Abbruch mit der Funktion »alarm()« einrichten, die einen Aufruf nach einer vom Anwender vorgegebenen Zeit abbricht.

Die beiden vorgestellten Plugins arbeiten lokal, das heißt, man installiert sie auf der Maschine, auf der der Test auch auszuführen ist. Um sie alternativ vom Nagios-Server aus über das Netzwerk aufzurufen, muss man auf Mechanismen wie den Dienst NRPE (Nagios Remote Plugin Executor) oder die Secure Shell (SSH) zurückgreifen.

Wer in Perl schon ein paar Erfahrungen gesammelt hat, dem wird der Eigenbau von Nagios-Plugins für spezielle Zwecke nach dem hier vorgestellten Mustersicher nicht mehr schwerfallen.

(jcb)

Infos

(1) Nagios Plugins: (http://nagiosplug.sourceforge.net)

(2) Nagios-Tauschbörse: (http://www.nagiosexchange.org)

(3) Beispiele aus dem Text: (http://linux.swobspace.net/projects/nagios)

(4) CPAN – das Modul-Archiv von Perl: (http://www.cpan.org)

(5) Nagios Developer Guide: (http://nagiosplug.sourceforge.net/developer-guidelines.html#THRESHOLDFORMAT).

Der Autor

Der studierte Physiker Wolfgang Barth ist Systemadministrator mit Leib und Seele. Er betreute oder betreut Systeme unter VMS, Digital Unix (Tru64), SunOS/Solaris, IBM AIX und HP-UX. Linux ist für ihn nur die logische Fortsetzung zum Unix überall. Seine Begeisterung für alles, was mit Systemadministation zu tun hat, gibt er gerne weiter: In Büchern ("Das Firewall-Buch", "Netzwerkanalyse unter Linux" oder "Nagios", in Artikeln, Seminaren und im persönlichen Kontakt.