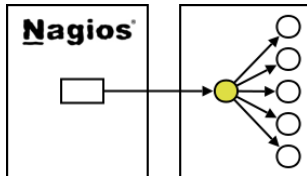


check_multi à la carte

Ein Plugin Menu in 7 Gängen

Matthias Flacke



9. / 10. Juni 2008

4. Nagios Workshop in Oldenburg

Menufolge

- 1 Erst mal einen Aperitif - Motivation
- 2 Die leckere Suppe - check_multi en detail
- 3 Als Vorspeise: die Installation
- 4 Ein bunter Salat: die Konfiguration
- 5 Der Hauptgang: check_multi in voller Aktion
- 6 Alles Käse: Troubleshooting
- 7 Zum Dessert: check_multi advanced
- 8 Zum guten Schluss: der Kaffee

Monitoring in einem großen Unternehmen

- Seit einigen Jahren bin ich im europaweiten Unix-Team eines Telekommunikationskonzerns u.a. fürs Monitoring zuständig.
- Wie fast jeder andere große Konzern *profitieren* auch wir ständig von Umorganisationen.
- Das System-Monitoring hat sich in diesem Zusammenhang immer mehr vom technischen Monitoring hin zu einem reinen Reporting-Mechanismus fürs Management entwickelt.
Hauptsache: **alles grün**
- Die System-Teams und Fachabteilungen brauchen nach wie vor verlässliche Informationen über ihre Infrastruktur und Applikationen.
- Wer kein technisches Monitoring betreibt, riskiert den Rückfall in die Kleinstaaterei: Jeder Administrator pflegt dann sein ganz persönliches Monitoring.

Monitoring in einem großen Unternehmen

- Seit einigen Jahren bin ich im europaweiten Unix-Team eines Telekommunikationskonzerns u.a. fürs Monitoring zuständig.
- Wie fast jeder andere große Konzern *profitieren* auch wir ständig von Umorganisationen.
- Das System-Monitoring hat sich in diesem Zusammenhang immer mehr vom technischen Monitoring hin zu einem reinen Reporting-Mechanismus fürs Management entwickelt.
Hauptsache: **alles grün**
- Die System-Teams und Fachabteilungen brauchen nach wie vor verlässliche Informationen über ihre Infrastruktur und Applikationen.
- Wer kein technisches Monitoring betreibt, riskiert den Rückfall in die Kleinstaaterei: Jeder Administrator pflegt dann sein ganz persönliches Monitoring.

check_multi - Treibstoff fürs U-Boot

Eigentlich unmögliche Anforderungen

- Knappe Ressourcen
- Größenordnung: 500 Server, pro Server 50 Servicechecks
=> 25.000 Services
- Einheitliche Anwendergruppe, 1 Notification pro multi-Service

check_multi - die erste Idee

- Zusammenfassung von Nagios-Checks
-> weniger Ressourcenverbrauch
- Hohe Konfigurierbarkeit und Flexibilität
- Schnelle und einfache Implementierung von Monitoring-Anforderungen
- Delegation der Konfigurationen

check_multi - Treibstoff fürs U-Boot

Eigentlich unmögliche Anforderungen

- Knappe Ressourcen
- Größenordnung: 500 Server, pro Server 50 Servicechecks
=> 25.000 Services
- Einheitliche Anwendergruppe, 1 Notification pro multi-Service

check_multi - die erste Idee

- Zusammenfassung von Nagios-Checks
-> weniger Ressourcenverbrauch
- Hohe Konfigurierbarkeit und Flexibilität
- Schnelle und einfache Implementierung von
Monitoring-Anforderungen
- Delegierung der Konfigurationen

Menufolge

- 1 Erst mal einen Aperitif - Motivation
- 2 Die leckere Suppe - check_multi en detail**
- 3 Als Vorspeise: die Installation
- 4 Ein bunter Salat: die Konfiguration
- 5 Der Hauptgang: check_multi in voller Aktion
- 6 Alles Käse: Troubleshooting
- 7 Zum Dessert: check_multi advanced
- 8 Zum guten Schluss: der Kaffee

Was ist check_multi?

Zuerst mal ein ganz normales Nagios-Plugin

- Returncodes OK, WARNING, CRITICAL, oder UNKNOWN
- Mindestens eine Zeile Output, meistens mehr
- Saubere Performancedaten

Ein Wrapper-Plugin

- engl. to wrap: verpacken, umhüllen, aber auch: umbrechen
- Das check_multi *Parent-Plugin* ruft mehrere *Child-Plugins* auf
- Alle Infos der Child-Plugins werden dank Multiline 1:1 weitergereicht
- *Mehrere* Child-Ergebnisse werden zu *einem* Parent-Status konsolidiert

Was ist check_multi?

Zuerst mal ein ganz normales Nagios-Plugin

- Returncodes OK, WARNING, CRITICAL, oder UNKNOWN
- Mindestens eine Zeile Output, meistens mehr
- Saubere Performancedaten

Ein Wrapper-Plugin

- engl. to wrap: verpacken, umhüllen, aber auch: umbrechen
- Das check_multi *Parent-Plugin* ruft mehrere *Child-Plugins* auf
- Alle Infos der Child-Plugins werden dank Multiline 1:1 weitergereicht
- *Mehrere* Child-Ergebnisse werden zu *einem* Parent-Status konsolidiert

Was ist check_multi nicht?

Kein Mini-Nagios

check_multi versucht nicht, Dinge zu tun, die Nagios besser kann:

- Notifications
- Escalations
- Reporting

Kein Remote Agent

- Um den Transport der Plugin-Informationen muß man sich selber kümmern, z.B. per NRPE oder check_by_ssh
- Auch die check_multi-Konfiguration muß auf den Client gebracht werden, z.B. per rsync, scp, svn, cvs

Was ist check_multi nicht?

Kein Mini-Nagios

check_multi versucht nicht, Dinge zu tun, die Nagios besser kann:

- Notifications
- Escalations
- Reporting

Kein Remote Agent

- Um den Transport der Plugin-Informationen muß man sich selber kümmern, z.B. per NRPE oder check_by_ssh
- Auch die check_multi-Konfiguration muß auf den Client gebracht werden, z.B. per rsync, scp, svn, cvs

Wie wird check_multi konfiguriert?

Konfigurations-Datei im NRPE style

```
# config.cmd
command [ procs ] = check_procs
command [ load ] = check_load -w 5,4,3 -c 10,8,6
```

So sieht das auf der Kommandozeile aus:

```
$ ./check_multi -f config.cmd -r 1
OK - 2 plugins checked, 2 ok
[ 1] procs PROCS OK: 137 processes
[ 2] load OK - load average: 1.29, 1.09, 0.94
```

Übrigens

check_multi kann komplett über die Kommandozeile konfiguriert werden.

Wie wird check_multi konfiguriert?

Konfigurations-Datei im NRPE style

```
# config.cmd
command [ procs ] = check_procs
command [ load ] = check_load -w 5,4,3 -c 10,8,6
```

So sieht das auf der Kommandozeile aus:

```
$ ./check_multi -f config.cmd -r 1
OK - 2 plugins checked, 2 ok
[ 1] procs PROCS OK: 137 processes
[ 2] load OK - load average: 1.29, 1.09, 0.94
```

Übrigens

check_multi kann komplett über die Kommandozeile konfiguriert werden.

Wie wird check_multi konfiguriert?

Konfigurations-Datei im NRPE style

```
# config.cmd
command [ procs ] = check_procs
command [ load ] = check_load -w 5,4,3 -c 10,8,6
```

So sieht das auf der Kommandozeile aus:

```
$ ./check_multi -f config.cmd -r 1
OK - 2 plugins checked, 2 ok
[ 1] procs PROCS OK: 137 processes
[ 2] load OK - load average: 1.29, 1.09, 0.94
```

Übrigens

check_multi kann komplett über die Kommandozeile konfiguriert werden.

Die erste Zeile wie gewohnt in der Status-View

Mit einem Blick sehen, wo es klemmt.

Service Status Details For Host
'demo'

| Host ↑ | Service ↑ | Status ↑ | Last Check ↑ | Duration ↑ | Attempt ↑ | Status Information |
|--------|-----------------|----------|---------------------|--------------|-----------|---|
| demo | Web Application | WARNING | 06-04-2008 17:25:40 | 0d 0h 5m 58s | 3/4 | webapp WARNING - 2 plugins checked, 1 critical (myhost2), 0 warning, 0 unknown, 1 ok |
| | cpu_vmstat | OK | 06-04-2008 17:23:40 | 0d 0h 5m 58s | 1/4 | cpu_vmstat OK - 8 plugins checked, 8 ok |
| | multi_long | CRITICAL | 06-04-2008 17:25:40 | 0d 0h 5m 58s | 3/4 | check_multi_long CRITICAL - 34 plugins checked, 2 critical (system_mysql, dummy_critical), 2 warning (network_if_eth, dummy_warning), 2 unknown (network_rsync, dummy_unknown), 28 ok |
| | multi_nr | CRITICAL | 06-04-2008 17:25:40 | 0d 0h 5m 58s | 3/4 | check_multi_r1 CRITICAL - 3 plugins checked, 2 critical (multi_l2, multi_l3), 1 warning (multi_l1), 0 unknown, 0 ok |

Tipp

Eine Namenskonvention fuer die Child-Checks erleichtert später die Identifikation (z.B. `sys_disk_root`).

Nagios 3 macht's möglich: die Multiline Extended View

Details auf einen Blick

Service State Information

Current Status: **OK** (for 0d 0h 0m 24s)

Status Information: multi_small OK - 3 plugins checked, 3 ok

- 1 rootdisk DISK OK - free space: / 1048 MB (9% inode=74%);
- 2 load OK - load average: 1.80, 0.80, 0.45
- 3 swap SWAP OK - 100% free (2047 MB out of 2048 MB)

Performance Data: multi_small::check_multi::plugins=3 time=1.714757
 rootdisk::check_disk::/=10432MB;11489;11852;0;12094
 load::check_load::load1=1.800;5.000;10.000;0;
 load5=0.800;4.000;8.000;0; load15=0.450;3.000;6.000;0;
 swap::check_swap::swap=2047MB;0;0;0;2048

Current Attempt: 1/4 (HARD state)

Last Check Time: 06-06-2008 18:24:44

Check Type: ACTIVE

Check Latency / Duration: 38.104 / 2.140 seconds

Next Scheduled Check: 06-06-2008 19:24:44

Last State Change: 06-06-2008 18:24:44

Last Notification: N/A (notification 0)

Is This Service Flapping? N/A

In Scheduled Downtime? **NO**

Last Update: 06-06-2008 18:25:04 (0d 0h 0m 4s ago)

check_multi ruft check_multi

Rekursive Aufrufe sind möglich














Service State Information

Current Status:

WARNING (for 0d 0h 0m 10s)

Status Information:

webapp WARNING - 2 plugins checked, 1 critical (myhost2), 0 warning, 0 unknown, 1 ok

- | | | |
|---|---|--|
| 1 |  | myhost1 OK - 6 plugins checked, 0 critical, 0 warning, 0 unknown, 6 ok |
| 1 |  | sys_ping OK - localhost: rta 0.029ms, lost 0% |
| 2 |  | sys_load OK - load average: 1.69, 1.64, 1.26 |
| 3 |  | sys_disk DISK OK - free space: / 2489 MB (21% inode=81%); |
| 4 |  | web_apache PROCS OK: 12 processes with command name 'httpd2-prefork' |
| 5 |  | db_mysql Uptime: 4366 Threads: 1 Questions: 147 Slow queries: 0 Opens: 12 Flush tables: 1 Open tables: 6 Queries per second avg: 0.034 |
| 6 |  | app_myapp HTTP OK - HTTP/1.1 301 Moved Permanently - 0.016 second response time |
| 2 |  | myhost2 CRITICAL - 6 plugins checked, 1 critical (app_myapp), 0 warning, 0 unknown, 5 ok |
| 1 |  | sys_ping OK - localhost: rta 0.029ms, lost 0% |
| 2 |  | sys_load OK - load average: 1.69, 1.64, 1.26 |
| 3 |  | sys_disk DISK OK - free space: / 2489 MB (21% inode=81%); |
| 4 |  | web_apache PROCS OK: 12 processes with command name 'httpd2-prefork' |
| 5 |  | db_mysql Uptime: 4367 Threads: 1 Questions: 148 Slow queries: 0 Opens: 12 Flush tables: 1 Open tables: 6 Queries per second avg: 0.034 |
| 6 |  | app_myapp CRITICAL: HTTP 2 - Unable to open TCP socket |

Auf einen Blick

Warum check_multi einsetzen?

Performance für den Nagios-Server

Viele Child-Checks, nur *ein* Nagios check

Verteiltes Monitoring

Keep Nagios simple: Die Konfiguration wird einfacher

Die Konfiguration ist von Nagios unabhängig

Delegierung ist möglich

Flexible Statusbewertung

durch Boolesche Operatoren AND, OR, NOT

Business Views

mit check_multi einfach zu implementieren

Auf einen Blick

Warum check_multi einsetzen?

Performance für den Nagios-Server

Viele Child-Checks, nur *ein* Nagios check

Verteiltes Monitoring

Keep Nagios simple: Die Konfiguration wird einfacher

Die Konfiguration ist von Nagios unabhängig

Delegierung ist möglich

Flexible Statusbewertung

durch Boolesche Operatoren AND, OR, NOT

Business Views

mit check_multi einfach zu implementieren

Auf einen Blick

Warum check_multi einsetzen?

Performance für den Nagios-Server

Viele Child-Checks, nur *ein* Nagios check

Verteiltes Monitoring

Keep Nagios simple: Die Konfiguration wird einfacher

Die Konfiguration ist von Nagios unabhängig

Delegierung ist möglich

Flexible Statusbewertung

durch Boolesche Operatoren AND, OR, NOT

Business Views

mit check_multi einfach zu implementieren

Auf einen Blick

Warum check_multi einsetzen?

Performance für den Nagios-Server

Viele Child-Checks, nur *ein* Nagios check

Verteiltes Monitoring

Keep Nagios simple: Die Konfiguration wird einfacher

Die Konfiguration ist von Nagios unabhängig

Delegierung ist möglich

Flexible Statusbewertung

durch Boolesche Operatoren AND, OR, NOT

Business Views

mit check_multi einfach zu implementieren

Auf einen Blick

Warum check_multi einsetzen?

Performance für den Nagios-Server

Viele Child-Checks, nur *ein* Nagios check

Verteiltes Monitoring

Keep Nagios simple: Die Konfiguration wird einfacher

Die Konfiguration ist von Nagios unabhängig

Delegierung ist möglich

Flexible Statusbewertung

durch Boolesche Operatoren AND, OR, NOT

Business Views

mit check_multi einfach zu implementieren

Auf einen Blick

Warum check_multi einsetzen?

Performance für den Nagios-Server

Viele Child-Checks, nur *ein* Nagios check

Verteiltes Monitoring

Keep Nagios simple: Die Konfiguration wird einfacher

Die Konfiguration ist von Nagios unabhängig

Delegierung ist möglich

Flexible Statusbewertung

durch Boolesche Operatoren AND, OR, NOT

Business Views

mit check_multi einfach zu implementieren

Sprechen wir von den Nachteilen!

Einschränkungen beim Betrieb von check_multi

Es kann nur einen geben!

Da Nagios nur den check_multi Service sieht, gibt es nur ein(e)

- Notification
- Escalation
- Reporting

HTML Output

Die HTML-Ausgabe ist nur mit einem gewissen Aufwand kompatibel zu den üblichen Notification-Mechanismen.

Man kann sich hier natürlich behelfen, wirklich gelöst wird das HTML-Thema erst mit Nagios 4.

Sprechen wir von den Nachteilen!

Einschränkungen beim Betrieb von check_multi

Es kann nur einen geben!

Da Nagios nur den check_multi Service sieht, gibt es nur ein(e)

- Notification
- Escalation
- Reporting

HTML Output

Die HTML-Ausgabe ist nur mit einem gewissen Aufwand kompatibel zu den üblichen Notification-Mechanismen.

Man kann sich hier natürlich behelfen, wirklich gelöst wird das HTML-Thema erst mit Nagios 4.

Sprechen wir von den Nachteilen!

Einschränkungen beim Betrieb von check_multi

Es kann nur einen geben!

Da Nagios nur den check_multi Service sieht, gibt es nur ein(e)

- Notification
- Escalation
- Reporting

HTML Output

Die HTML-Ausgabe ist nur mit einem gewissen Aufwand kompatibel zu den üblichen Notification-Mechanismen.

Man kann sich hier natürlich behelfen, wirklich gelöst wird das HTML-Thema erst mit Nagios 4.

Menufolge

- 1 Erst mal einen Aperitif - Motivation
- 2 Die leckere Suppe - check_multi en detail
- 3 Als Vorspeise: die Installation**
- 4 Ein bunter Salat: die Konfiguration
- 5 Der Hauptgang: check_multi in voller Aktion
- 6 Alles Käse: Troubleshooting
- 7 Zum Dessert: check_multi advanced
- 8 Zum guten Schluss: der Kaffee

check_multi installieren - man nehme:

Direkt vom Hersteller: ein frisches Nagios

- Ab Nagios 3 gibt es die mehrzeilige Ausgabe für Plugins
- Nagios 2 noch nicht migriert? Macht nix: check_multi kann trotzdem mit der Nagios 2 Report-Option verwendet werden
- Beim Kompilieren von Nagios die Standard-Puffer erhöhen:

[nagios.h](#) MAX_PLUGIN_OUTPUT_LENGTH auf 8192

[common.h](#) MAX_INPUT_BUFFER auf 8192

Aktuelle Transporter

- NRPE und check_by_ssh müssen den Multiline Output beherrschen
- Daher ggf. neue Versionen (>Herbst 2007) kompilieren
- Auch bei NRPE müssen die Puffer-Größen angepasst werden

check_multi installieren - man nehme:

Direkt vom Hersteller: ein frisches Nagios

- Ab Nagios 3 gibt es die mehrzeilige Ausgabe für Plugins
- Nagios 2 noch nicht migriert? Macht nix: check_multi kann trotzdem mit der Nagios 2 Report-Option verwendet werden
- Beim Kompilieren von Nagios die Standard-Puffer erhöhen:

[nagios.h](#) MAX_PLUGIN_OUTPUT_LENGTH auf 8192

[common.h](#) MAX_INPUT_BUFFER auf 8192

Aktuelle Transporter

- NRPE und check_by_ssh müssen den Multiline Output beherrschen
- Daher ggf. neue Versionen (>Herbst 2007) kompilieren
- Auch bei NRPE müssen die Puffer-Größen angepasst werden

Einfach kopieren. . .

Für die ganz Ungeduldigen

- check_multi in das libexec-Verzeichnis kopieren
- Ausführungsrechte setzen
- fertig!

Bei Risiken und Nebenwirkungen: Defaults anpassen

- Wo ist das libexec-Directory?
Default: /usr/local/nagios/libexec
- Wo ist Perl?
Default: #!/usr/bin/perl

Ausblick

- In naher Zukunft wird es auch ein configure-Script geben, mit denen Default-Einstellungen bei der Installation gesetzt werden können.

Einfach kopieren. . .

Für die ganz Ungeduldigen

- check_multi in das libexec-Verzeichnis kopieren
- Ausführungsrechte setzen
- fertig!

Bei Risiken und Nebenwirkungen: Defaults anpassen

- Wo ist das libexec-Directory?
Default: /usr/local/nagios/libexec
- Wo ist Perl?
Default: #!/usr/bin/perl

Ausblick

- In naher Zukunft wird es auch ein configure-Script geben, mit denen Default-Einstellungen bei der Installation gesetzt werden können.

Einfach kopieren. . .

Für die ganz Ungeduldigen

- check_multi in das libexec-Verzeichnis kopieren
- Ausführungsrechte setzen
- fertig!

Bei Risiken und Nebenwirkungen: Defaults anpassen

- Wo ist das libexec-Directory?
Default: /usr/local/nagios/libexec
- Wo ist Perl?
Default: #!/usr/bin/perl

Ausblick

- In naher Zukunft wird es auch ein configure-Script geben, mit denen Default-Einstellungen bei der Installation gesetzt werden können.

Menufolge

- 1 Erst mal einen Aperitif - Motivation
- 2 Die leckere Suppe - check_multi en detail
- 3 Als Vorspeise: die Installation
- 4 Ein bunter Salat: die Konfiguration**
- 5 Der Hauptgang: check_multi in voller Aktion
- 6 Alles Käse: Troubleshooting
- 7 Zum Dessert: check_multi advanced
- 8 Zum guten Schluss: der Kaffee

Allgemeines Format

```
# example.cmd
command [ Tag ] = [Path]Plugin
command [ Tag ] = [Path]Plugin
state   [ RC ] = Expression
```

Bitte nicht vordrängeln!

- Anweisungen werden nacheinander ausgeführt.
- Und zwar von oben nach unten in der Datei-Reihenfolge.

Für die Übersicht

Kommentare mit # beginnende Zeilen werden ignoriert

Leerzeilen können beliebig eingefügt werden

Fortsetzungszeilen werden durch einen Backslash \
am Ende eingeleitet

Allgemeines Format

```
# example.cmd
command [ Tag ] = [Path]Plugin
command [ Tag ] = [Path]Plugin
state   [ RC ] = Expression
```

Bitte nicht vordrängeln!

- Anweisungen werden nacheinander ausgeführt.
- Und zwar von oben nach unten in der Datei-Reihenfolge.

Für die Übersicht

Kommentare mit # beginnende Zeilen werden ignoriert

Leerzeilen können beliebig eingefügt werden

Fortsetzungszeilen werden durch einen Backslash \
am Ende eingeleitet

command - Anweisung

```
command [ Tag ] = [/Pfad/zum/]Plugin
```

command - Format

Tag beliebiger Name des Checks.

Erlaubt sind A-Z, a-z, 0-9, _

Plugin beliebiges Kommando,

es muß nicht einmal ein Plugin sein

/Pfad/zum/ Mögliche Alternativen:

- ① Pfad des Plugins voll angeben
- ② falls nicht: Suche im libexec-Verzeichnis
(default oder Parameter -l)
- ③ falls nicht: Suche im PATH des nagios Users

command - Anweisung

```
command [ Tag ] = [/Pfad/zum/]Plugin
```

command - Format

Tag beliebiger Name des Checks.

Erlaubt sind A-Z, a-z, 0-9, _

Plugin beliebiges Kommando,
es muß nicht einmal ein Plugin sein

/Pfad/zum/ Mögliche Alternativen:

- ① Pfad des Plugins voll angeben
- ② falls nicht: Suche im libexec-Verzeichnis
(default oder Parameter -l)
- ③ falls nicht: Suche im PATH des nagios Users

command - Anweisung

```
command [ Tag ] = [/Pfad/zum/]Plugin
```

command - Format

Tag beliebiger Name des Checks.

Erlaubt sind A-Z, a-z, 0-9, _

Plugin beliebiges Kommando,
es muß nicht einmal ein Plugin sein

/Pfad/zum/ Mögliche Alternativen:

- ① Pfad des Plugins voll angeben
- ② falls nicht: Suche im libexec-Verzeichnis
(default oder Parameter -l)
- ③ falls nicht: Suche im PATH des nagios Users

command - Anweisung

```
command [ Tag ] = [/Pfad/zum/]Plugin
```

command - Format

Tag beliebiger Name des Checks.

Erlaubt sind A-Z, a-z, 0-9, _

Plugin beliebiges Kommando,
es muß nicht einmal ein Plugin sein

/Pfad/zum/ Mögliche Alternativen:

- ① Pfad des Plugins voll angeben
- ② falls nicht: Suche im libexec-Verzeichnis
(default oder Parameter -l)
- ③ falls nicht: Suche im PATH des nagios Users

state - Anweisung

```
state [ OK|UNKNOWN|WARNING|CRITICAL ] = (expression)
```

Wenn man nichts angibt, gilt implizit

```
state [ OK          ] = 1
state [ UNKNOWN    ] = COUNT(UNKNOWN) > 0
state [ WARNING    ] = COUNT(WARNING) > 0
state [ CRITICAL   ] = COUNT(CRITICAL) > 0
```

Es gewinnt immer der schlimmste Bösewicht

expression ist ein beliebiger gültiger Perl-Ausdruck

Status Ersetzung von
OK-0, UNKNOWN-3, WARNING-1, CRITICAL-2

COUNT COUNT(STATUS) ist die Anzahl der jeweiligen Returncodes

Klammern Perl zickt manchmal, im Zweifelsfalle großzügig klammern

state - Anweisung

```
state [ OK|UNKNOWN|WARNING|CRITICAL ] = (expression)
```

Wennn man nichts angibt, gilt implizit

```
state [ OK          ] = 1
state [ UNKNOWN    ] = COUNT(UNKNOWN) > 0
state [ WARNING    ] = COUNT(WARNING) > 0
state [ CRITICAL   ] = COUNT(CRITICAL) > 0
```

Es gewinnt immer der schlimmste Bösewicht

expression ist ein beliebiger gültiger Perl-Ausdruck

Status Ersetzung von

OK-0, UNKNOWN-3, WARNING-1, CRITICAL-2

COUNT COUNT(STATUS) ist die Anzahl der jeweiligen Returncodes

Klammern Perl zickt manchmal, im Zweifelsfalle großzügig klammern

state - Anweisung

```
state [ OK|UNKNOWN|WARNING|CRITICAL ] = (expression)
```

Wennn man nichts angibt, gilt implizit

```
state [ OK          ] = 1
state [ UNKNOWN    ] = COUNT(UNKNOWN) > 0
state [ WARNING    ] = COUNT(WARNING) > 0
state [ CRITICAL   ] = COUNT(CRITICAL) > 0
```

Es gewinnt immer der schlimmste Bösewicht

expression ist ein beliebiger gültiger Perl-Ausdruck

Status Ersetzung von
OK-0, UNKNOWN-3, WARNING-1, CRITICAL-2

COUNT COUNT(STATUS) ist die Anzahl der jeweiligen Returncodes

Klammern Perl zickt manchmal, im Zweifelsfalle großzügig klammern

Wie kommt der Service ins Nagios? Nach Schema F

Service-Definition

```
define service {
    name                multi-service
    check_command       check_multi!-f etc/check_multi.cmd -r 15
    host_name           <hostname>
    use                 generic-service
}
```

Command-Definition

```
define command {
    command_name       check_multi
    command_line       $USER1$/check_multi $ARG1$
}
```

Makros - Übergabe von Werten und Returncodes

Makros innerhalb von check_multi

`$<Tag>$` Ausgabe eines Child-Checks

`$STATE_<Tag>$` Returncode eines Child-Checks

Nagios-Makros an check_multi übergeben

Übergabe durch Parameter `-s NAME=VALUE`

Beispiel `check_multi -s HOSTNAME=$HOSTNAME$`

Weiterverarbeitung in check_multi durch `$HOSTNAME$`

Makros - Übergabe von Werten und Returncodes

Makros innerhalb von check_multi

`$<Tag>$` Ausgabe eines Child-Checks

`$STATE_<Tag>$` Returncode eines Child-Checks

Nagios-Makros an check_multi übergeben

Übergabe durch Parameter `-s NAME=VALUE`

Beispiel `check_multi -s HOSTNAME=$HOSTNAME$`

Weiterverarbeitung in check_multi durch `$HOSTNAME$`

Beispiel 1 - Commands und States

test.cmd

```
command [ disk      ] = check_disk -w 5% -c 2%
command [ swap      ] = check_swap -w 30% -c 20%
command [ load       ] = check_load -w 30,20,10 -c 50,40,30

state [ UNKNOWN     ] = disk == UNKNOWN
state [ WARNING      ] = swap != load
state [ CRITICAL     ] = load == CRITICAL || \
                        (disk != OK && load != OK)
```

Beispiel 2 - Statusbewertung

SNMP

```
# Flexible interpretation of snmp results
#
command [ sensor ] = /usr/bin/snmpget -v 1 -c $COMMUNITY$ \
                    -Oqv $HOSTNAME$ XYZ-MIB::Sensor.0

state [ UNKNOWN  ] = $sensor$ !~ /[4627859]/
state [ OK        ] = $sensor$ == 4 || $sensor$ == 6
state [ WARNING   ] = $sensor$ == 2 || $sensor$ == 7 || \
                    $sensor$ == 8
state [ CRITICAL ] = $sensor$ == 5 || $sensor$ == 9
```

Beispiel 3 - Network Link Settings prüfen

```
# determine default gateway from /proc/net/route
command [ def_gw ] = \
    awk '$2 == "00000000" {print $1}' /proc/net/route

# store ethtool output in temporary file
command [ ethtool ] = \
    sudo /usr/sbin/ethtool $def_gw$ > /tmp/$def_gw$.txt \
    && echo OK

# get interface settings from ethtool output
command [ link      ] = awk '/Link detected:/      {print $3}' /tmp/$def_gw$.txt
command [ speed     ] = awk '/Speed:/              {print $2}' /tmp/$def_gw$.txt
command [ duplex    ] = awk '/Duplex:/             {print $2}' /tmp/$def_gw$.txt
command [ autoneg   ] = awk '/Auto-negotiation:/   {print $2}' /tmp/$def_gw$.txt

# evaluate results
state [ UNKNOWN ] = "$def_gw$" eq ""
state [ WARNING ] = (" $speed$" && "$speed$" !~ /100/) || \
    (" $duplex$" && "$duplex$" !~ /FULL/i)
state [ CRITICAL] = "$link" && "$link" eq "no"
```

Beispiel 3 - Network Link Settings prüfen

Netzwerk-Interface-Monitoring in der GUI

Service State Information

Current Status: **OK** (for 2d 3h 17m 33s)

Status Information: check_if OK - 6 plugins checked, 6 ok

- 1 def_gw eth3
- 2 ethtool OK
- 3 link yes
- 4 speed 100Mb/s
- 5 duplex Full
- 6 autoneg on

Performance Data: check_if::check_multi::plugins=6 time=0.082499

Current Attempt: 1/4 (HARD state)

Last Check Time: 06-06-2008 12:01:23

Check Type: ACTIVE

Check Latency / Duration: 0.059 / 0.241 seconds

Next Scheduled Check: 06-06-2008 13:01:23

Last State Change: 06-04-2008 09:10:18

Last Notification: N/A (notification 0)

Is This Service Flapping? N/A

In Scheduled Downtime? **NO**

Last Update: 06-06-2008 12:27:46 (0d 0h 0m 5s ago)

Menufolge

- 1 Erst mal einen Aperitif - Motivation
- 2 Die leckere Suppe - check_multi en detail
- 3 Als Vorspeise: die Installation
- 4 Ein bunter Salat: die Konfiguration
- 5 Der Hauptgang: check_multi in voller Aktion**
- 6 Alles Käse: Troubleshooting
- 7 Zum Dessert: check_multi advanced
- 8 Zum guten Schluss: der Kaffee

Ausgabe à la carte: `-report`

Format der Report-Option:

- Die Einzel-Werte aller Report-Optionen werden addiert
- Default: 13
1 (Service name) + 4 (Fehlerausgabe) + 8 (Perfdaten)

Die wichtigsten Report-Optionen

- 1 Service name:
4 plugins checked, 1 critical (http), 1 warning (disk), 2 ok
- 2 HTML-Output
- 4 Fehlerausgabe - STDERR wird in eckigen Klammern ausgegeben.
- 8 Performance-Daten (Multi-Label)
- 128 Performance-Daten action link (das Sternchen)
- 512 Nagios2 Kompatibilität (Ausgabe in einer Zeile)

Nicht immer ganz einfach: Performance-Daten

Problem

check_multi vereint Performance-Daten der verschiedensten Plugins.
Wie können diese Daten von einem Performance-Tool
auseinandergelassen werden?

Lösung: Das Multi-Label

Format <Service-Description>::::<Label>

Beispiel system_swap::check_swap::swap=1786MB;0;0;0;204

PNP Das Multi-Label wird von **PNP** unterstützt (danke, Jörg!)

Tipp

Wenn ein Plugin mehrere Typen von Performance-Daten ausgibt (z.B. check_nt),
hilft eine spezielle Form des command-Tags weiter:

```
command [ nt_disk::check_disk ] = check_nt...
```

Nicht immer ganz einfach: Performance-Daten

Problem

check_multi vereint Performance-Daten der verschiedensten Plugins.
Wie können diese Daten von einem Performance-Tool
auseinandergelassen werden?

Lösung: Das Multi-Label

Format <Service-Description>::::<Label>

Beispiel system_swap::check_swap::swap=1786MB;0;0;0;204

PNP Das Multi-Label wird von **PNP** unterstützt (danke, Jörg!)

Tipp

Wenn ein Plugin mehrere Typen von Performance-Daten ausgibt (z.B. check_nt),
hilft eine spezielle Form des command-Tags weiter:

```
command [ nt_disk::check_disk ] = check_nt...
```

Nicht immer ganz einfach: Performance-Daten

Problem

check_multi vereint Performance-Daten der verschiedensten Plugins.
Wie können diese Daten von einem Performance-Tool
auseinandergelassen werden?

Lösung: Das Multi-Label

Format <Service-Description>::<<Plugin>::<<Label>

Beispiel system_swap::check_swap::swap=1786MB;0;0;0;204

PNP Das Multi-Label wird von **PNP** unterstützt (danke, Jörg!)

Tipp

Wenn ein Plugin mehrere Typen von Performance-Daten ausgibt (z.B. check_nt),
hilft eine spezielle Form des command-Tags weiter:

```
command [ nt_disk::check_disk ] = check_nt...
```

Timeout - check_multi hat deren **zwei**: BIG T and small t

Das Timeout für alles: `-TIMEOUT / -T`

- Die Summe der Laufzeiten aller Child-Plugins darf *BIG T* nicht überschreiten
- Default: 30 Sekunden

Das Timeout für den Einzelcheck: `-timeout / -t`

- Die Laufzeit eines Child-Plugins darf *small t* nicht überschreiten
- Default: 10 Sekunden
- *small t* sollte natürlich kleiner sein als *BIG T* :-)

Timeout - check_multi hat deren **zwei**: BIG T and small t

Das Timeout für alles: `-TIMEOUT / -T`

- Die Summe der Laufzeiten aller Child-Plugins darf *BIG T* nicht überschreiten
- Default: 30 Sekunden

Das Timeout für den Einzelcheck: `-timeout / -t`

- Die Laufzeit eines Child-Plugins darf *small t* nicht überschreiten
- Default: 10 Sekunden
- *small t* sollte natürlich kleiner sein als *BIG T* :-)

Timeouts im Beispiel

```
$ check_multi -f demo_timeout.cmd -T 30 -t 10
UNKNOWN - 4 plugins checked, 3 unknown (sleep_10, sleep_11,
sleep_12), 1 ok
[ 1] sleep_09
[ 2] sleep_10 UNKNOWN - sleep cancelled after timeout (10s)
[ 3] sleep_11 UNKNOWN - sleep cancelled after timeout (10s)
[ 4] sleep_12 UNKNOWN - execution cancelled after global
timeout (30s)
```

Was läuft hier genau ab?

- 1 läuft korrekt durch von Sekunde 0-9
- 2 läuft von Sekunde 9-19 und wird vom t-timeout gestoppt
- 3 läuft von Sekunde 19-29 und wird vom t-timeout gestoppt
- 4 wird gar nicht erst gestartet, weil die 10 Sekunden ab Sekunde 29 die 30 Sekunden überschreiten würden.

Timeouts im Beispiel

```
$ check_multi -f demo_timeout.cmd -T 30 -t 10
UNKNOWN - 4 plugins checked, 3 unknown (sleep_10, sleep_11,
sleep_12), 1 ok
[ 1] sleep_09
[ 2] sleep_10 UNKNOWN - sleep cancelled after timeout (10s)
[ 3] sleep_11 UNKNOWN - sleep cancelled after timeout (10s)
[ 4] sleep_12 UNKNOWN - execution cancelled after global
timeout (30s)
```

Was läuft hier genau ab?

- 1 läuft korrekt durch von Sekunde 0-9
- 2 läuft von Sekunde 9-19 und wird vom t-timeout gestoppt
- 3 läuft von Sekunde 19-29 und wird vom t-timeout gestoppt
- 4 wird gar nicht erst gestartet, weil die 10 Sekunden ab Sekunde 29 die 30 Sekunden überschreiten würden.

Menufolge

- 1 Erst mal einen Aperitif - Motivation
- 2 Die leckere Suppe - check_multi en detail
- 3 Als Vorspeise: die Installation
- 4 Ein bunter Salat: die Konfiguration
- 5 Der Hauptgang: check_multi in voller Aktion
- 6 Alles Käse: Troubleshooting**
- 7 Zum Dessert: check_multi advanced
- 8 Zum guten Schluss: der Kaffee

Ein Fehler? check_multi ist (fast) immer nur der Bote

Nochmal zurück zum Wrapper-Plugin

- check_multi führt Plugins aus (= Fremdprogramme)
- check_multi erbt die Laufzeitprobleme der Fremdprogramme
- Ergo: check_multi muß mit Fehlerbedingungen aller Art umgehen können

check_multi ist sehr informativ, wenn man es läßt.

- Ausgabe von STDOUT und STDERR des Child-Plugins
- Prüfung der Performance-Daten des Child-Plugins
- Achtung: fehlerhafte Performance-Daten werden unterdrückt, um nachfolgende Daten nicht zu beeinträchtigen
- Probleme von check_multi werden natürlich auch ausgegeben

Ein Fehler? check_multi ist (fast) immer nur der Bote

Nochmal zurück zum Wrapper-Plugin

- check_multi führt Plugins aus (= Fremdprogramme)
- check_multi erbt die Laufzeitprobleme der Fremdprogramme
- Ergo: check_multi muß mit Fehlerbedingungen aller Art umgehen können

check_multi ist sehr informativ, wenn man es läßt.

- Ausgabe von STDOUT und STDERR des Child-Plugins
- Prüfung der Performance-Daten des Child-Plugins
- Achtung: fehlerhafte Performance-Daten werden unterdrückt, um nachfolgende Daten nicht zu beeinträchtigen
- Probleme von check_multi werden natürlich auch ausgegeben

Strategien zur Fehlersuche

Checkliste

- Immer die Fehlerausgabe `-r 4` einschalten
- Jedes Child-Plugin auf der Kommandozeile testen
 - stets als Nagios-User testen (Rechte)
 - stets mit vollen Pfaden testen
- Aufruf von `check_multi` mit `-verbose`-Option
- Verbose-Stufen 1-3 (`-v -vv -vvv`)
 - 1 Auflistung der Kommandos und Status-Bewertung
 - 2 Detailinfo
 - 3 Debug-Info (sehr ausführlich)
- Im Normalfall reicht Stufe 1 völlig aus.

Menufolge

- 1 Erst mal einen Aperitif - Motivation
- 2 Die leckere Suppe - check_multi en detail
- 3 Als Vorspeise: die Installation
- 4 Ein bunter Salat: die Konfiguration
- 5 Der Hauptgang: check_multi in voller Aktion
- 6 Alles Käse: Troubleshooting
- 7 Zum Dessert: check_multi advanced**
- 8 Zum guten Schluss: der Kaffee

Geschäftsprozesse abbilden

Wozu Business Views?

Einzelinformationen liefern immer nur ein Teilbild.

Gesamtsicht Eine Aussage ist erst in der Gesamtsicht möglich.

Cluster Speziell bei redundanten Systemen sind hostübergreifende Informationen wichtig.

End-To-End Monitoring trifft eine Aussage über Sein oder Nichtsein eines Geschäftsprozesses.

Business Views liefern jedoch die konkrete Ursache einer Störung.

Geschäftsprozesse abbilden

Wozu Business Views?

Einzelinformationen liefern immer nur ein Teilbild.

Gesamtsicht Eine Aussage ist erst in der Gesamtsicht möglich.

Cluster Speziell bei redundanten Systemen sind hostübergreifende Informationen wichtig.

End-To-End Monitoring trifft eine Aussage über Sein oder Nichtsein eines Geschäftsprozesses.

Business Views liefern jedoch die konkrete Ursache einer Störung.

Web-Applikation

Zu überwachende Einzelkomponenten

`sys_ping` `check_icmp -H $HOSTNAME$`

`sys_load` `check_load -w 5,4,3 -c 10,8,6`

`sys_disk` `check_disk -w 5% -c 2% -p / -p /var -p /opt`

`web_apache` `check_procs -c 1: -C httpd`

`db_mysql` `check_mysql`

`app_myapp` `check_http -H $HOSTNAME$ -u http://myhost`

Application Business View

Als remote-Check auf dem Zielsever

```
check_nrpe -H myhost \  
-c check_multi -a -f myapp.cmd -s HOSTNAME=myhost
```

Alle Einzelkomponenten OK -> Business View OK







Service State Information

Current Status:

OK (for 0d 0h 12m 46s)

Status Information:

myapp OK - 6 plugins checked, 0 critical, 0 warning, 0 unknown, 6 ok

- 1  sys_ping OK - demo: rta 0.061ms, lost 0%
- 2  sys_load OK - load average: 1.40, 1.15, 0.98
- 3  sys_disk DISK OK - free space: / 1063 MB (9% inode=74%);
- 4  web_apache PROCS OK: 11 processes with command name 'httpd2-prefork'
- 5  db_mysql Uptime: 24089 Threads: 1 Questions: 51 Slow queries: 0 Opens: 12 Flush tables: 1 Open tables: 6 Queries per second avg: 0.002
- 6  app_myapp HTTP OK - HTTP/1.1 301 Moved Permanently - 0.002 second response time

Eine kleine Erweiterung: Cluster-Überwachung

Zwei Webserver im Cluster

```
command [ myhost1 ] = check_nrpe -H myhost1 \  
    -c check_multi -a -f myapp.cmd -s HOSTNAME=myhost1  
command [ myhost2 ] = check_nrpe -H myhost2 \  
    -c check_multi -a -f myapp.cmd -s HOSTNAME=myhost2  
  
state   [ WARNING   ] = COUNT(WARNING)  >  0 || \  
                                COUNT(CRITICAL) == 1  
state   [ CRITICAL ] = COUNT(CRITICAL) == 2
```

Cluster Application Business View

Ein Host CRITICAL -> Business View WARNING









Service State Information

Current Status:

WARNING (for 0d 0h 13m 45s)

Status Information:

webapp WARNING - 2 plugins checked, 1 critical (myhost2), 0 warning, 0 unknown, 1 ok

- 1  myhost1 OK - 6 plugins checked, 0 critical, 0 warning, 0 unknown, 6 ok
- +
- 2  myhost2 CRITICAL - 6 plugins checked, 1 critical (app_myapp), 0 warning, 0 unknown, 5 ok
- 1  sys_ping OK - localhost: rta 0.056ms, lost 0%
- 2  sys_load OK - load average: 0.60, 1.03, 0.96
- 3  sys_disk DISK OK - free space: / 1054 MB (9% inode=75%);
- 4  web_apache PROCS OK: 11 processes with command name 'httpd2-prefork'
- 5  db_mysql Uptime: 1012551 Threads: 1 Questions: 122 Slow queries: 0 Opens: 12 Flush tables: 1 Open tables: 6 Queries per second avg: 0.000
- 6  app_myapp HTTP CRITICAL - Unable to open TCP socket

Event-Handler à la check_multi

Vorsicht: waffenscheinpflichtig ;-)

```
# check_multi -f start_proc.cmd -s PROC=<process> -s ARGS=<args>

# 1. check for process
command[proc_before] = check_procs -c 1: -C "$PROC$" -a "$ARGSS$"

# 2. start process in background
eval[start_proc] = \
    ( $STATE_proc_before$ != 0 ) \
      ? ( system("$PROC$ $ARGSS$ &") != 0 ) \
        ? "- failed: $?" \
        : "- done" \
      : "- not necessary, already running"

# 3. check process again (ok, maybe redundant ;-))
command[proc_after] = check_procs -c 1: -C "$PROC$" -a "$ARGSS$"

# 4. state evaluation
state[OK          ]= proc_before == OK      && proc_after == OK
state[WARNING    ]= proc_before != OK      && proc_after == OK
state[CRITICAL   ]= start_proc =~/failed/  || proc_after == CRITICAL
```


Was macht eigentlich unser U-Boot?

... es ist aufgetaucht

Hardware Zwei HP DL380 G3 mit 2GB Ram und 2 CPUs

Cluster Active-Active-Cluster mit Heartbeat und DRBD

Nagios 2 Nagios-Instanzen monitoren 800 Hosts
mit mehr als 60 check_multi Services pro Host
=> 50000 Services

Latency stets unter einer Sekunde

Admin-Instanz rollt Nagios-Clients aus und
kontrolliert check_multi Konfigurationen.

NDO & NagVis sorgen fuer die Gesamtsicht auf alle Instanzen
und es gibt ein neues Projekt fuer die Ablösung
des offiziellen Monitorings durch Nagios.

Was macht eigentlich unser U-Boot?

... es ist aufgetaucht

Hardware Zwei HP DL380 G3 mit 2GB Ram und 2 CPUs

Cluster Active-Active-Cluster mit Heartbeat und DRBD

Nagios 2 Nagios-Instanzen monitoren 800 Hosts
mit mehr als 60 check_multi Services pro Host
=> 50000 Services

Latency stets unter einer Sekunde

Admin-Instanz rollt Nagios-Clients aus und
kontrolliert check_multi Konfigurationen.

NDO & NagVis sorgen fuer die Gesamtsicht auf alle Instanzen
und es gibt ein neues Projekt fuer die Ablösung
des offiziellen Monitorings durch Nagios.

Was macht eigentlich unser U-Boot?

... es ist aufgetaucht

Hardware Zwei HP DL380 G3 mit 2GB Ram und 2 CPUs

Cluster Active-Active-Cluster mit Heartbeat und DRBD

Nagios 2 Nagios-Instanzen monitoren 800 Hosts
mit mehr als 60 check_multi Services pro Host
=> 50000 Services

Latency stets unter einer Sekunde

Admin-Instanz rollt Nagios-Clients aus und
kontrolliert check_multi Konfigurationen.

NDO & NagVis sorgen fuer die Gesamtsicht auf alle Instanzen
und es gibt ein neues Projekt fuer die Ablösung
des offiziellen Monitorings durch Nagios.

Fragen?

Vielen Dank für Eure
Aufmerksamkeit!

Hilfe gibt es in der deutschen Nagios-Community
www.nagios-portal.de
und natürlich an der Quelle:
www.my-plugin.de/check_multi