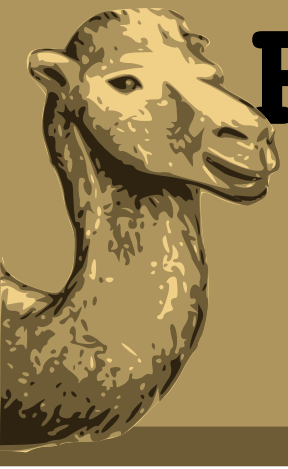
Perl Mongers im Ruhrgebiet



Ruhr.pm

Perl-Plugins fuer Nagios, Icinga & Co

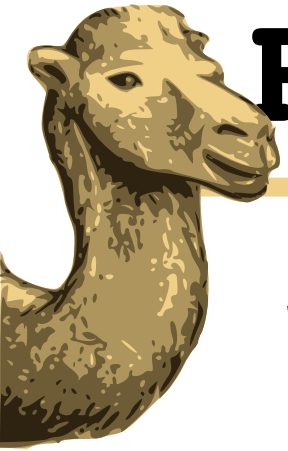
Autor: Veit Wahlich

EMail: veit AT ruhr.pm.org

Datum: 8. Februar 2010

http://ruhr.pm.org/

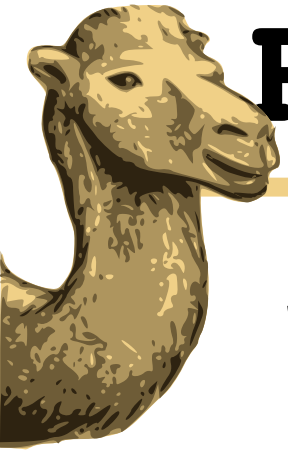
Template Version 0.2



Ruhr.pm

Was ist Nagios?

- sowohl abruf- als auch ereignisbasiertes Ueberwachungssystem
- aktive Ueberwachung von Netzwerkdiensten
- Ueberwachung lokaler Dienste, Prozesse, Hardware, Systemstati etc.
  - aktiv ueber Agenten (nrpe, snmpd, nsclient++, ...)
  - aktiv per Login (ssh, rsh, ...)
  - optional passiv ueber Agenten und/oder Scheduler (z.B. crond und nsca, SNMP-Traps)



Ruhr.pm

Was ist Nagios?

- grosse Auswahl fertiger Ueberwachungs-Plugins (Checks) aus offiziellen und inoffiziellen Quellen
- einfaches Interface zum Erstellen eigener Ueberwachungs-Plugins
- ausserdem zahlreiche Erweiterungen, z.B. fuer
  - das Empfangen und Auswerten passiv bereitgestellter Daten
  - das Sammeln von Performance-Daten
  - das Erzeugen von Graphen/Trends

Ruhr.pm

Screenshot: Nagios 3.0

Nagios

General
- Home
- Documentation

Monitoring
- Tactical Overview
- Service Detail
- Host Detail
- Hostgroup Overview
- Hostgroup Summary
- Hostgroup Grid
- Servicegroup Overview
- Servicegroup Summary
- Servicegroup Grid
- Status Map
- 3-D Status Map
- Service Problems
  - Unhandled
- Host Problems
  - Unhandled
- Network Outages

Show Host:

- Comments
- Downtime
- Process Info
- Performance Info
- Scheduling Queue

Reporting
- Trends
- Availability
- Alert Histogram
- Alert History
- Alert Summary
- Notifications
- Event Log

Configuration
- View Config

Current Network Status
Last Updated: Mon Feb 8 11:23:01 CET 2010
Updated every 90 seconds
Nagios® 3.0.6 - www.nagios.org
Logged in as *nagiosadmin*

View History For This Host
View Notifications For This Host
View Service Status Detail For All Hosts

Host Status Totals

Up	Down	Unreachable	Pending
1	0	0	0

All Problems	All Types
0	1

Service Status Totals

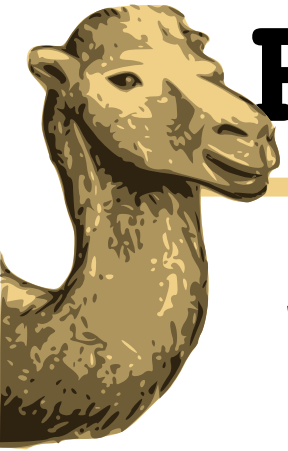
OK	Warning	Unknown	Critical	Pending
14	0	0	0	0

All Problems	All Types
0	14

Service Status Details For Host 'db01'

Host	Service	Status	Last Check	Duration	Attempt	Status Information
db01	CPU	OK	02-08-2010 11:22:42	14d 0h 53m 20s	1/4	CPU used 19.0% (<90) : OK
	Disk /	OK	02-08-2010 11:22:28	20d 21h 15m 33s	1/4	/: 10% used (3062MB/29696MB) (< 80%) : OK
	Disk /boot	OK	02-08-2010 11:22:49	20d 7h 43m 13s	1/4	/boot: 31% used (31MB/99MB) (< 80%) : OK
	Disk /var/lib/pgsql	OK	02-08-2010 11:21:55	20d 21h 15m 6s	1/4	/var/lib/pgsql: 39% used (317722MB/813254MB) (< 50%) : OK
	Disk /var/lib/pgsql/data/pg_xlog	OK	02-08-2010 11:22:06	20d 21h 14m 56s	1/4	/var/lib/pgsql/data/pg_xlog: 1% used (1141MB/93880MB) (< 20%) : OK
	Load	OK	02-08-2010 11:22:42	9d 22h 41m 19s	1/4	Load : 1.72 1.86 2.07 : OK
	PostgreSQL	OK	02-08-2010 11:22:11	1d 16h 45m 50s	1/4	19 processes named postmaster (> 3) (<= 30) : OK
	PostgreSQL Connections	OK	02-08-2010 11:22:33	31d 22h 31m 28s	1/4	17 of 110 connections (15% - 1 DELETE(1 slow), 2 SELECT, 14 idle) : OK
	Processes	OK	02-08-2010 11:21:56	2d 17h 54m 5s	1/4	161 processes (> 130) (<= 175) : OK
	RAM and Swap	OK	02-08-2010 11:22:02	20d 21h 15m 0s	1/4	RAM: 1%, Swap: 2% : OK

10 Matching Service Entries Displayed



Ruhr.pm

Was ist Icinga?

- Nagios-Fork
  - wurde im Mai 2009 von einer Gruppe (mit Nagios unzufriedener) Ex-Nagios-Entwickler ins Leben gerufen
  - soll schnelleren Entwicklungszyklus als Nagios bieten
  - derzeit primaer optische Veraenderungen und Verbesserungen “unter der Haube”
- vollstaendig kompatibel zu Nagios, insbesondere zu Nagios-Ueberwachungs-Plugins

Ruhr.pm

Screenshot: Icinga 1.0

ICINGA

General
- Home
- Documentation
- Search

Monitoring
- Tactical Overview
- Host Detail
- Service Detail
- Hostgroup Overview
- Servicegroup Overview
- Status Map
- Service Problems
- Host Problems
- Network Outages
- Comments
- Downtime
- Process Info
- Performance Info
- Scheduling Queue

Reporting
- Trends
- Availability
- Alert Histogram
- Alert History
- Alert Summary
- Notifications

Current Network Status
Last Updated: Mon Feb 8 11:33:56 CET 2010
Updated every 90 seconds
Icinga 1.0 - www.icinga.org
(Credits to: Nagios® - www.nagios.org)
Logged in as *guest*

Host Status Totals

Up	Down	Unreachable	Pending
1	0	0	0

All Problems	All Types
0	1

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
5	0	0	0	0

All Problems	All Types
0	5

View History For This Host
View Notifications For This Host
View Service Status Detail For All Hosts

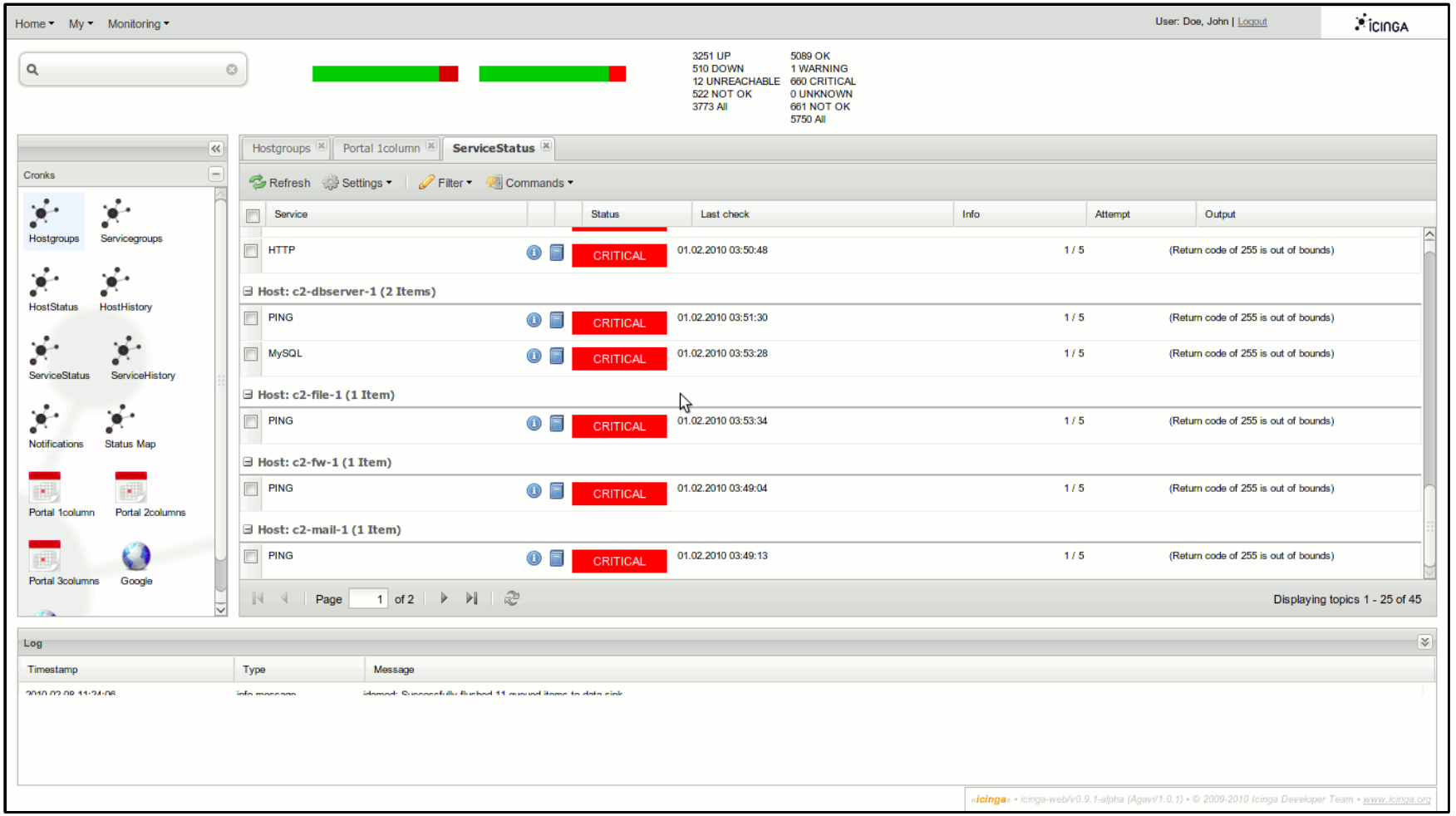
Service Status Details For Host 'monitor'

Host	Service	Status	Last Check	Duration	Attempt	Status Information
monitor	Current Load	OK	02-08-2010 11:29:56	0d 11h 14m 0s	1/4	OK - load average: 0.01, 0.04, 0.06
	Current Users	OK	02-08-2010 11:32:35	6d 20h 24m 35s	1/4	USERS OK - 0 users currently logged in
	PING	OK	02-08-2010 11:31:34	6d 20h 22m 2s	1/4	PING OK - Packet loss = 0%, RTA = 0.05 ms
	Root Partition	OK	02-08-2010 11:32:35	6d 20h 24m 28s	1/4	DISK OK - free space: / 6055 MB (63% inode=94%):
	Total Processes	OK	02-08-2010 11:31:33	6d 20h 21m 55s	1/4	PROCS OK: 25 processes with STATE = RSZDT

5 Matching Service Entries Displayed

Ruhr.pm

Screenshot: Icinga Alpha



Ruhr.pm

Ueberwachungs-Plugins erstellen

- einfache Programme, beliebige Sprache
- Parameteruebernahme per Befehlszeile
  - Hostname bzw. IP-Adresse, Schwellwerte etc.
- Statusmeldung per Exit-Code
  - 0: OK, 1: WARNING, 2: CRITICAL, 3: UNKNOWN
- Mitteilungen ueber genau eine Ausgabezeile
  - immer etwas auf STDOUT ausgeben und mit \n abschliessen
  - uebliche Beispielausgabe: “OK : Works just fine”

Ruhr.pm

Ueberwachungs-Plugins erstellen

- Standard-Befehlszeilenparameter verwenden:
  - **-h** Hilfeseite des Plugins
  - **-H** Hostname/IP-Adresse/Unix-Socket
  - **-I** explizit IP-Adresse
  - **-w** Schwelle fuer Warning
  - **-c** Schwelle fuer Critical
  - **-u** Username
  - **-p** Passwort oder Port
  - **-V** Plugin-Version

Ruhr.pm

ePN: embedded Perl Nagios

- eingebetteter Perl-Interpreter
  - vergleichbar mit mod_perl fuer Apache
- fuehrt Perl-Programme direkt im Prozess aus
  - Ueberwachungs-Plugins (Checks)
  - Ereignis-Handler
  - Benachrichtigungsprogramme
- reduziert Overhead von Perl-Programmen enorm
  - keine Forks zur externen Programmausfuehrung
  - kein Aufstarten des Perl-Interpreters

Ruhr.pm

ePN: embedded Perl Nagios

- Cache fuer Perl-Bytecode
  - beschleunigt Start durch seltene Neukompilierung
  - groessere Speicherbelegung gegenueber “normaler” Plugin-Ausfuehrung
  - Neustart von Nagios notwendig, um Perl-Module neu zu laden

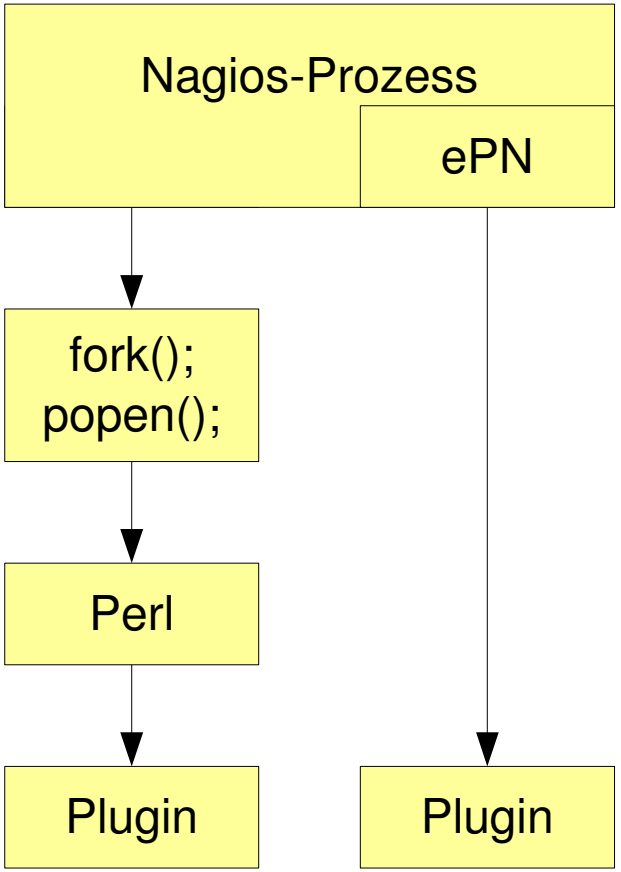
Ruhr.pm

ePN: embedded Perl Nagios

- eventuell Anpassungen fuer ePN notwendig
  - **`use strict; use warnings;`**
  - **`Getopt::Std`** oder **`Getopt::Long`** benutzen
  - keine Variablen in globalem Scope beschreiben
  - Thread Safety beachten
  - **`DATA`**-Handle (**`__DATA__`**) funktioniert nicht
  - **`BEGIN{}`** und **`END{}`** funktionieren nicht
  - Handles und Sockets immer schliessen
  - Timeouts definieren, um Blockaden zu verhindern

Ruhr.pm

ePN: embedded Perl Nagios



Normalerweise forkt Nagios einen eigenen Prozess aus und oeffnet per **`popen()`** ein Programm. Im Fall von Perl-Plugins wird hier zuerst der Perl-Interpreter gestartet, welcher wiederum das eigentliche Plugin-Programm laedt, kompiliert und schliesslich ausfuehrt.

Mit ePN fuehrt der Nagios-Prozess ueber einen eingebetteten Perl-Interpreter selbst das Perl-Plugin-Programm aus. Zusaetzlich wird der Bytecode des Programms und verwendeter Module gecachet.

Ruhr.pm

ePN: embedded Perl Nagios

- ePN muss von Nagios unterstuetzt werden
  - Unterstuetzung mit uebersetzen...
    - **`./configure --enable-embedded-perl --with-perlcache`** ...
  - ... und in der Konfiguration aktivieren
    - "**`enable_embedded_perl`**" zur Aktivierung
    - "**`use_embedded_perl_implicitly`**" zur Regelung impliziter Verwendung von ePN
  - Unterstuetzung bei den meisten Distributionen enthalten

Ruhr.pm

ePN: embedded Perl Nagios

- Steuerung expliziter (Nicht-)Verwendung von ePN ueber optionale Kommentare
  - Steuerkommentar muss in den ersten 10 Zeilen des Plugins festgelegt sein
  - Programm mit ePN starten: **`# nagios: +epn`**
    - fuehrt das Programm mit ePN aus, sofern implizit “normale” Ausfuehrung gewuenscht
  - Programm ohne ePN starten: **`# nagios: -epn`**
    - erzwingt “normale” Ausfuehrung, sofern implizit Ausfuehrung mit ePN gewuenscht

Ruhr.pm

Ein einfaches Beispiel

```
# Dieses Plugin kann mit ePN ausgefuehrt
# werden.
# nagios: +epn

use strict;
use warnings;
use Getopt::Long;

sub main(){
  # Befehlszeile parsen, Warn-Schwelle in
  # $warn und kritische Schwelle in $crit
  # speichern.
  new Getopt::Long::Parser(config =>
      ['no_ignore_case', 'bundling'])
    ->getoptions(
        'warning|w=i'  => \my $warn,
        'critical|c=i' => \my $crit
      )
      || do{
          print("UNKNOWN : Unable to parse"
            . " arguments!\n");
          exit(3);
        }
```

Der Kommentar **`# nagios: +epn`** deklariert das Plugin explizit als ePN-kompatibel.

Wir verwenden **`Getopt::Long`**, dies wird fuer Plugins empfohlen. Huebscher als die uebliche API ist die hier verwendete OO-API.

Definiert werden die Befehlszeilen-Parameter **`-w`** bzw. **`--warning`** sowie **`-c`** bzw. **`--critical`**, die einen ganzzahligen Wert erwarten. Die Benutzung von **`-w`** und **`-c`** fuer Schwellwerte wird empfohlen.

Ruhr.pm

Ein einfaches Beispiel

```
# ACPI-Thermalzone 0 oeffnen...
open(my $fd, '<', '/proc/acpi'
  . '/thermal_zone/TZ00/temperature')
  || do{
      print("UNKNOWN : Unable to open"
        . " ACPI thermal zone 0!\n");
      exit(3);
    };
# ... und Wert sowie Einheit in $temp
#  und $unit speichern.
my($temp, $unit) = map{
    /^temperature:\s+(\d+)\s*(.*)$/
    ? ($1, $2)
    : ()
   } <$fd>;
close($fd);
```

Ueber das procfs wird die Temperatur der ersten “Thermal Zone” ausgelesen. Kann die Datei nicht geoeffnet werden, wird eine UNKNOWN-Meldung erzeugt.

Der Temperaturwert wird in **`$temp`** gespeichert, die Einheit (z.B. “C” fuer Grad Celsius) wird in **`$unit`** gespeichert.
Falls der Regex nicht greift (da der gesuchte Inhalt nicht vorkomme), werden beide Einheiten **`undef`** sein.

Ruhr.pm

Ein einfaches Beispiel

```
    # Auf gueltigen Wert pruefen und mit kri-
    # tischer und Warn-Schwelle vergleichen.
    if(!defined($temp)){
      print("UNKNOWN : Unable to collect"
        . " temperature!\n");
      exit(3);
    }
    elsif(defined($crit) && $temp > $crit){
      printf("CRITICAL : Temperature %d %s"
        . " (> %d)\n", $temp, $unit, $crit);
      exit(2);
    }
    elsif(defined($warn) && $temp > $warn){
      printf("WARNING : Temperature %d %s"
        . " (> %d)\n", $temp, $unit, $warn);
      exit(1);
    }
    else{
      printf("OK : Temperature %d %s\n",
        $temp, $unit);
      exit(0);
    }
}

main();
```

Pruefe den Inhalt von **$temp** zunaechst auf Gueltigkeit, vergleiche anschliessend den Wert mit den Schwellwerten in **$warn** und **$crit**, sofern definiert.

Erzeuge anschliessend Status-meldungen unter Angabe der Temperatur sowie ggf. des ueber-schrittenen Schwellwertes.

Der Exit-Code teilt dem Monitoring-System den Status des Checks mit, 0 fuer OK, 1 fuer WARNING, 2 fuer CRITICAL und 3 fuer UNKNOWN.

Ruhr.pm

Schwellen und Bereiche

- Entwickler sind angehalten, ein einheitliches Format fuer Schwellen und Bereiche zu implementieren: “[@]*Anfang*:*Ende*”
  - ohne @: Alarm falls Wert ausserhalb,
  - mit @: Alarm falls Wert innerhalb *Anfang*:*Ende*
  - *Anfang* muss kleiner-gleich *Ende* sein
  - ohne obere Schwelle: “[@]*Anfang*:”
  - ohne untere Schwelle: “[@]~:*Ende*”
  - wird nur eine Zahl angegeben, ist es *Ende*; es wird *Anfang* == 0 angenommen

Ruhr.pm

Schwellen und Bereiche

- Beispiele:
  - check_foo -w '10' -c '20'
    - CRITICAL, falls Wert groesser als 20 oder kleiner als 0 ist
    - WARNING, falls Wert groesser als 10 oder kleiner als 0 ist
  - check_foo -w '10:20' -c '5:25'
    - CRITICAL, falls Wert groesser als 25 oder kleiner als 5 ist
    - WARNING, falls Wert groesser als 20 oder kleiner als 10 ist

Ruhr.pm

Schwellen und Bereiche

- Beispiele:
  - check_foo -w '5:10' -c '~:20'
    - CRITICAL, falls Wert groesser als 20 ist
    - WARNING, falls Wert groesser als 10 oder kleiner als 5 ist
  - check_foo -w '@10:15' -c '5:'
    - CRITICAL, falls Wert kleiner als 5 ist
    - WARNING, falls Wert groesser-gleich 10 und kleiner-gleich 15 ist

Ruhr.pm

Schwellwertangaben zerlegen

```
# Zerlege eine Threshold-Range in Typ
# (innerhalb/ausserhalb des Bereichs) sowie
# untere und obere Schwelle, sofern
# definiert.
sub parseThreshold($){

  my($range) = @_;

  # Zerlege Schwellwerte/Bereiche per Regex.
  my($type, $min, $max) = ($range =~ /^
        (@|)                          # Type
        (?:(-?\d+(?:\.\d+)?|~):)?     # Min
        (-?\d+(?:\.\d+)?)?            # Max
        $/x);

  # Gebe undef zurueck, falls Parsen nicht
  # erfolgreich (durch ungueltiges Format).
  return(undef) unless(defined($type));

  # $type ist 0 fuer Alarm falls ausserhalb
  # des Bereichs und 1 fuer Alarm falls
  # innerhalb.
  $type = $type ne '' ? 1 : 0;
```

Die Funktion **parseThreshold** dient der Zerlegung einer Threshold wie zuvor beschrieben in fuer uns verwertbare Elemente.

Die primaere Zerlegung findet ueber einen regulaeren Ausdruck statt.

Falls der Typ **undef** ist, hat der Ausdruck nicht gepasst, d.h. das Format war ungueltig.
Sonst wird er **0** fuer Alarm falls ausserhalb und **1** falls innerhalb des Bereichs.

Ruhr.pm

Schwellwertangaben zerlegen

```
    # $min/$max soll undef sein falls keine
    # untere/obere Schwelle definiert.
    $min = defined($min)
      ? ($min eq '~' ? undef : $min)
      : 0;

    # Pruefe, ob $min > $max ist...
    if(defined($min) && defined($max)
         && $min > $max){
      # ... und vertausche ggf. $min und $max.
      ($min, $max) = ($max, $min);
    }

    return($type, $min, $max);

}
```

Die zurueckgegebenen Werte in **$min** und **$max** sind **undef**, falls keine untere bzw. obere Schwelle in der Schwellenangabe enthalten war.
$max ist durch den Regex bereits korrekt gesetzt, fuer **$min** muessen wir hier noch nachhelfen.

Falls **$min** > **$max** ist, ist die Schwellenangabe eigentlich ungueltig, aber wir tauschen hier einfach die Werte der Variablen mit einander.

Ruhr.pm

Performance-Daten

- Ausgabe der Mess- und Grenzwerte in einem einheitlichen Format
  - Weiterverarbeitung durch Plugins und Zusatzprogramme
- ermoeglichen Kapazitaets-/Ressourcenplanung
  - Wie haben sich Werte bei bestimmten Ereignissen in der Vergangenheit entwickelt?
  - Wie ist werden sich die Werte voraussichtlich entwickeln und wann ist mit einem Ueberschreiten der Grenzwerte zu rechnen?

Ruhr.pm

Performance-Daten

- haeufigste Anwendungen: Round-Robin-Datenbanken, Graphing und Trending
  - meist manuelle Ressourcenplanung ohne aufwendige Statistiken
  - zahlreiche Tools und Plugins, z.B.
    - nagiosgraph
    - BrainyPDM
    - PNP4Nagios
    - APAN
  - oft Teil alternativer Nagios-GUIs, z.B. Centreon

Ruhr.pm

Performance-Daten

- Formatdefinition fuer Performance-Daten:
  Name=*Wert*[*Einheit*];[*Warn*];[*Krit*];[*Min*];[*Max*]
  - Performance-Daten werden an die regulaere Ausgabe des Plugins angehaengt, getrennt durch ein Pipe (“|”)
  - das Plugin kann Performance-Daten zu mehreren Werten im o.g. Format auf einmal ausgeben, getrennt durch Leerzeichen
  - Beispielausgabe: **`Temp OK: 49 C |temp=49C`**

Ruhr.pm

Performance-Daten

- Formatdefinition fuer Performance-Daten:
  Name=*Wert*[*Einheit*];[*Warn*];[*Krit*];[*Min*];[*Max*]
  - *Name* darf beliebige Zeichen enthalten, sollten jedoch Leerzeichen, ein Gleich-Zeichen (“=”) oder ein Single-Quote (“ ' ”) enthalten sein, muss er in Single-Quotes gesetzt werden
  - Single-Quotes in *Name* muessen durch zwei Single-Quotes dargestellt werden
  - alleinstehende, am Ende anhaengende Semikolons (“;”) duerfen weggelassen werden

Ruhr.pm

Performance-Daten

- Formatdefinition fuer Performance-Daten:
  Name=*Wert*[*Einheit*];[*Warn*];[*Krit*];[*Min*];[*Max*]
  - *Einheit* darf offiziell nur eine der folgenden sein (die meisten Programme koenne jedoch mit beliebigen Einheiten arbeiten):
    - keine Einheit: einfache Nummer/Anzahl
    - B: Bytes (auch KB, MB, GB, TB, ...)
    - s: Sekunden (auch us, ms, ...)
    - %: Prozent (dann Default fuer *Min*/*Max*: 0/100)
    - c: Zaehler (monoton steigender Wert, z.B. Uptime in Ticks oder Sekunden)

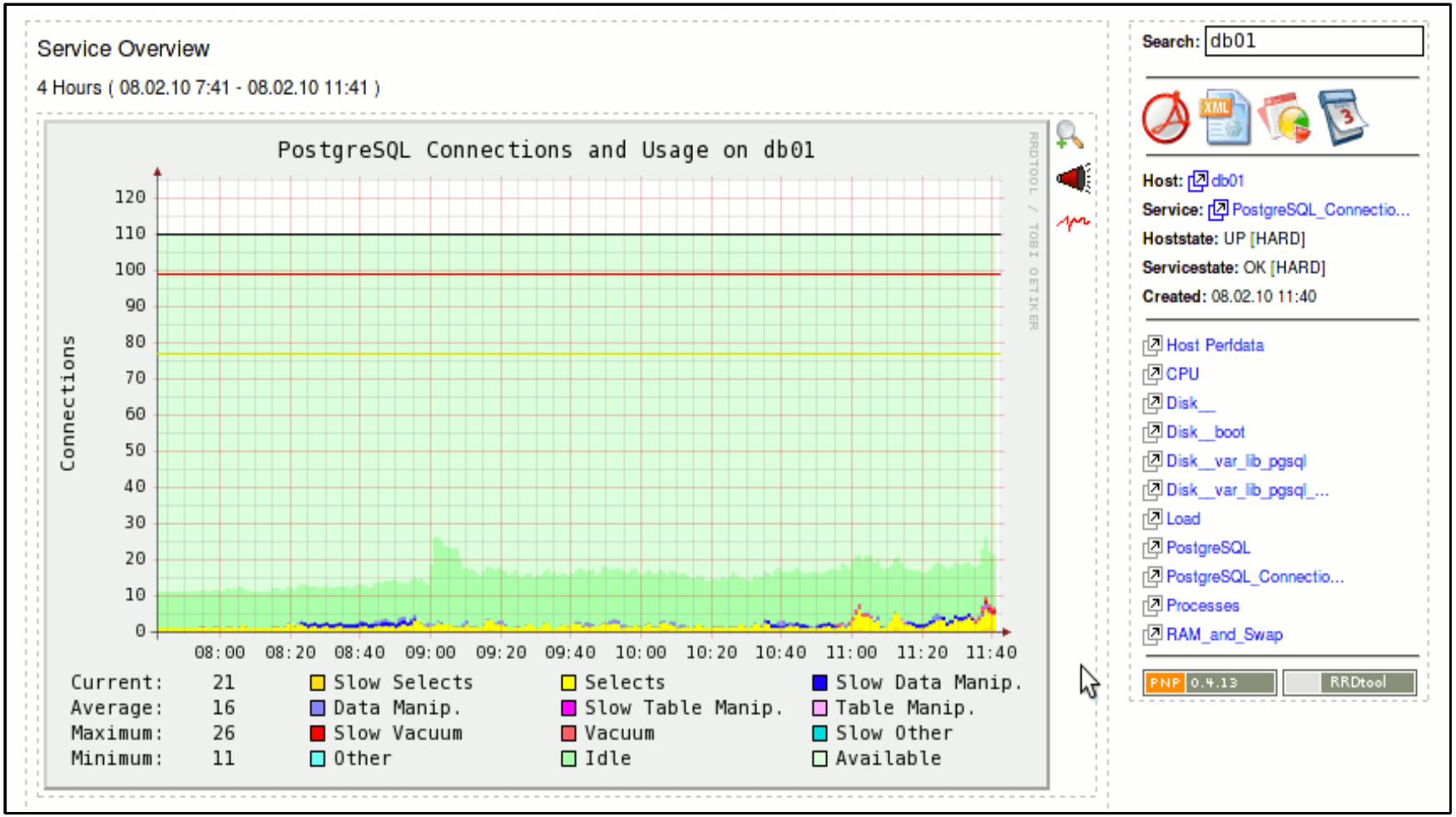
Ruhr.pm

Performance-Daten

- Formatdefinition fuer Performance-Daten:
  Name=*Wert*[*Einheit*];[*Warn*];[*Krit*];[*Min*];[*Max*]
  - *Warn* und *Krit* sind optionale Schwellwerte fuer den Warning- bzw. Critical-Alarm, angegeben im zuvor besprochenen Format fuer Schwellwerte und Bereiche
  - *Min* und *Max* sind die optionalen niedrigsten bzw. hoechsten Werte, die *Wert* annehmen kann
  - *Warn*, *Krit*, *Min* und *Max* werden ohne Einheit angeben

Ruhr.pm

Screenshot: PNP4Nagios



Ruhr.pm

Komplexeres Beispiel

```
# Dieses Plugin kann mit ePN ausgefuehrt
# werden.
# nagios: +epn

use strict;
use warnings;

use Filesys::Df;
use Getopt::Long;

sub main(){
  # Befehlszeile parsen, Warn-Schwelle in
  # $warn und kritische Schwelle in
  # $crit speichern.
  new Getopt::Long::Parser(config =>
      ['no_ignore_case', 'bundling'])
    ->getoptions(
        'warning|w=s'  => \my $warn,
        'critical|c=s' => \my $crit,
        'dir|d=s'      => \my $dir
      )
      || do{
          print("UNKNOWN : Unable to parse"
            . " arguments!\n");
          exit(3);
        };
```

Dieses Plugin soll Verzeichnisse (Mountpoints) hinsichtlich ihres verbleibenden Speicherplatzes ueberwachen. Es ist explizit ePN-kompatibel.

Das CPAN-Modul **`Filesys::Df`** wird verwendet, um die Belegung von Speicherplatz und Inodes zu beziehen.

Wir verwenden auch hier wieder **`Getopt::Long`**, um Befehlszeilenargumente zu parsen.

Ruhr.pm

Komplexeres Beispiel

```
# $dir muss definiert...
unless(defined($dir) && $dir ne ''){
  print("UNKNOWN: No directory given to"
    . " monitor.\n");
  exit(3);
}
# ... und ein existierendes Verzeichnis
# sein.
unless(-d $dir){
  printf("UNKNOWN: Directory not found:"
    . " %s\n", $dir);
  exit(3);
}

# Default-Exitcode.
my $status = 0;

# Wir interessieren uns fuer die Angabe
# in MiB.
my $df = df($dir, 1024*1024);
```

`$dir` enthaelt das zu ueberwachende Verzeichnis. Es muss definiert, d.h. ueber die Befehlszeile uebergeben worden sein und explizit existieren.

`$status` enthaelt den Exit-Code, mit dem das Plugin sich beendet. Default ist **`0`** fuer “OK”.

`df()` erzeugt einen anonymen Hash, der die Daten es uebergebenen Verzeichnisses enthaelt, Speicherplatzangaben moechten wir in MiB auswerten.

Ruhr.pm

Komplexeres Beispiel

```
# Speichere bool'sche Werte, ob eine der
# Schwellen verletzt ist (oder undef,
# falls Parsing fehlschlaegt).
my $isCrit
  = isOutOfRange($df->{per}, $crit);
my $isWarn
  = isOutOfRange($df->{per}, $warn);

# Werte Schwellwertverletzungen aus.
if(!defined($isCrit)
    || !defined($isWarn)){
  print("UNKNOWN: Bad threshold format");
  $status = 3;
}
elsif($isCrit){
  printf("CRITICAL: Disk %s usage %d%%"
    . " (out of threshold %s)",
    $dir, $df->{per}, $crit);
  $status = 2;
}
elsif($isWarn){
  printf("WARNING: Disk %s usage %d%%"
    . " (out of threshold %s)",
    $dir, $df->{per}, $warn);
  $status = 1;
}
```

Die Funktion **`isOutOfRange()`** prueft, ob ein Wert, hier die prozentuale Speicherbelegung, innerhalb einer uebergebenen Schwellwertdefinition liegt. Sie liefert einen wahren oder falschen Wert bzw. **`undef`**, falls die Schwellwerte nicht geparset werden konnten.

In einem **`if-elsif-else`**-Baum werden die zurueckgegebenen Werte entsprechend ihrer Dringlichkeit abgearbeitet, Ausgaben erzeugt und der Exit-Code festgelegt.

Ruhr.pm

Komplexeres Beispiel

```
    else{
      printf("OK: Disk %s usage %d%%",
        $dir, $df->{per});
    }

    # Gebe Performance-Daten zum belegten
    # Speicherplatz sowie, sofern verfuegbar,
    # den belegten Inodes aus.
    printf(" | 'Space Usage'=%dMB;;;0;%d",
      $df->{used}, $df->{blocks});
    if(exists($df->{files})){
      printf(" 'Inode Usage'=%d;;;0;%d",
        $df->{fused}, $df->{files});
    }

    # Schliesse die Zeile mit Newline ab und
    # beende mit dem oben festgelegten
    # Exitcode.
    print("\n");
    exit($status);

}
```

Nun werden noch Performance-Daten erzeugt und ausgegeben, hier zur Platzbelegung in MiB sowie zu den belegten Inodes, sofern das Dateisystem diese definiert.

Hier wird neben dem Messwert nur der Mindestwert von 0 sowie der Maximalwert (MiB bzw. Inodes insgesamt) ausgegeben. Warning- und Critical-Schwellen sind ja fuer diese Werte nicht definiert.

Schliesslich ein Newline ausgeben und mit Exit-Code beenden.

Ruhr.pm

Komplexeres Beispiel

```
# Pruefe, ob $value ausserhalb von $range
# liegt.
sub isOutOfRange($$){
  my($value, $range) = @_;

  # Wenn keine Schwelle definiert ist,
  # liegt der Wert nicht ausserhalb.
  return(0) unless(defined($range));

  # Die Schwelle parsen und undef zurueck-
  # geben, falls das Parsen fehlschlaegt.
  my($type, $min, $max)
    = parseThreshold($range);
  return(undef) unless(defined($type));

  # $oor erhaelt einen wahren bool'schen
  # Wert, falls der Mindest- oder Hoechst-
  # wert der Schwelle, sofern definiert,
  # unter-/ueberschritten ist.
  my $oor
    = ((defined($min) && $value < $min)
    || (defined($max) && $value > $max));
```

Die Funktion **`isOutOfRange()`**.

Wenn keine Schwelle definiert ist, kann der Wert nicht ausserhalb liegen, gebe einen Falsch-Wert zurueck.

Parse die Schwelle und gebe bei einem Fehler **`undef`** zurueck.

`$oor` speichert das Resultat der Vergleiche, ob der Wert ausserhalb der unteren/oberen Schwelle liegt, sofern definiert.

Ruhr.pm

Komplexeres Beispiel

```
  # $oor zurueckgeben, falls die Schwelle
  # vom Typ "Alarm falls innerhalb" ist,
  # negiert.
  return($type ? !$oor : $oor);
}

sub parseThreshold($){
  ...
}

main();
1;
```

Falls die Schwelle invertiert ist (vorangestelltes “@”), muss das Resultat invertiert zurueckgegeben werden.

Die Funktion **`parseThreshold()`** entspricht der vorab besprochenen Funktion mit gleichem Namen.

Ruhr.pm

Vielen Dank
fuer Eure Aufmerksamkeit

Ruhr.pm

Links

- Nagios
  - http://www.nagios.org/
- Icinga
  - http://www.icinga.org/
- Centreon
  - http://www.centreon.com/
- PNP4Nagios
  - http://www.pnp4nagios.org/

Ruhr.pm

Links

- BrainyPDM
  - http://www.brainypdm.org/
- APAN
  - http://apan.sf.net/
- nagiosgraph
  - http://nagiosgraph.sf.net/
- Net-SNMP
  - http://www.net-snmp.org/

Ruhr.pm

Links

- nsclient++ / nscp
  - http://nsclient.org/
- nsca und nrpe
  - http://www.nagios.org/