Supervision avec **Nagios** 1

Nagios est un moniteur de supervision. Successeur de NetSaint, logiciel libre reconnu dans le monde de la supervision, utilisé par les plus grands opérateurs, il permet une supervision active et passive de serveurs, équipements réseaux, et surtout de services divers et variés.
Très puissant, mais utilisant des principes simples, il reste néanmoins complexe à mettre en œuvre dans un environnement de production.

1)Concepts de supervision

La supervision est un ensemble de concepts recouvrant la surveillance du bon fonctionnement d'une production. Les différentes possibilités de supervision sont en gros les suivantes :

- Supervision via SNMP (y compris la métrologie) : Nagios peut utiliser SNMP pour surveiller par exemple les équipements réseau.
- Supervision des journaux : est aussi simple à faire qu'un *tail* sur */var/log/messages*. Plus pratiquement, cela passe maintenant par l'utilisation d'un serveur *syslog* centralisant les messages, couplé à un analyseur temps-réel de ces messages. (Consulter Locust)
- Supervision active des services et infrastructures.

Via des greffons (plugins), Nagios propose de simuler le fonctionnement du client d'une application (comme un client HTTP pour un service Web) pour valider le bon fonctionnement de cette application. Les greffons peuvent aussi effectuer les actions à entreprendre pour vérifier qu'un système de fichier n'est pas saturé.
Mais superviser simplement sans prendre d'action ne sert pas à grand chose.
Nagios intègre un moteur de gestion d'alertes, avec escalades, ces alertes étant diffusées soit par le biais de l'interface Web (avec alerte sonore via un champ EMBED), par courriel (pourvu que le serveur Nagios voie son serveur de messagerie local interconnecté avec la messagerie d'entreprise) ou par tout autre biais imaginable à travers une commande.
On peut même imaginer prendre des actions correctives directement sur la détection d'un problème. Par exemple, redémarrer un processus serveur Web qui montre des signes de faiblesse.
Nagios permet en standard de satisfaire la plupart de ces besoins, hormis la métrologie, mais plusieurs projets peuvent aider sur ce besoin précis.

2)Concepts de supervision avec Nagios

Nagios permet donc une supervision active et passive de serveurs, équipements réseaux, et surtout de services divers et variés.
Il utilise des principes simples, mais reste néanmoins complexe à mettre en oeuvre dans un environnement de production, du fait de sa grande souplesse de configuration et d'utilisation ; et ce d'autant plus que le nombre d'objets (matériels et services) à configurer est important.
Ces derniers peuvent de plus avoir des dépendances plus ou moins fines entre eux, ce qui peut encore augmenter le temps de mise en œuvre.

2.1)Architecture

L'architecture de base de Nagios est simple :

- Un ordonnanceur : Nagios est d'abord un moteur gérant l'ordonnancement des vérifications, ainsi que les actions à prendre sur incidents (alertes, escalades, prise d'action corrective) ;
- Une IHM : la partie graphique visible à travers un simple serveur Web, tel Apache, est basée (pour les versions jusqu'à la 2.0) sur des CGI ;
- Des sondes : les sondes de Nagios (les greffons ou *plugins)* sont de petits scripts ou programmes qui sont la base des vérifications.

Le projet Nagios fournit en standard bon nombre de greffons de base, mais la simplicité de leur mode de fonctionnement permet d'en écrire pour besoins propres, que ce soit pour superviser SAP par exemple ou pour vérifier que des clients peuvent bien se connecter sur l'intranet.

2.2)Principes de base

Nagios est, entre autre, un moteur d'ordonnancement de vérifications diverses et variées.
Ces vérifications sont assurées par des greffons *(plugins* en anglais), dont le développement est séparé du moteur principal (pour les greffons de base).
La relation entre le moteur et les greffons est assurée :

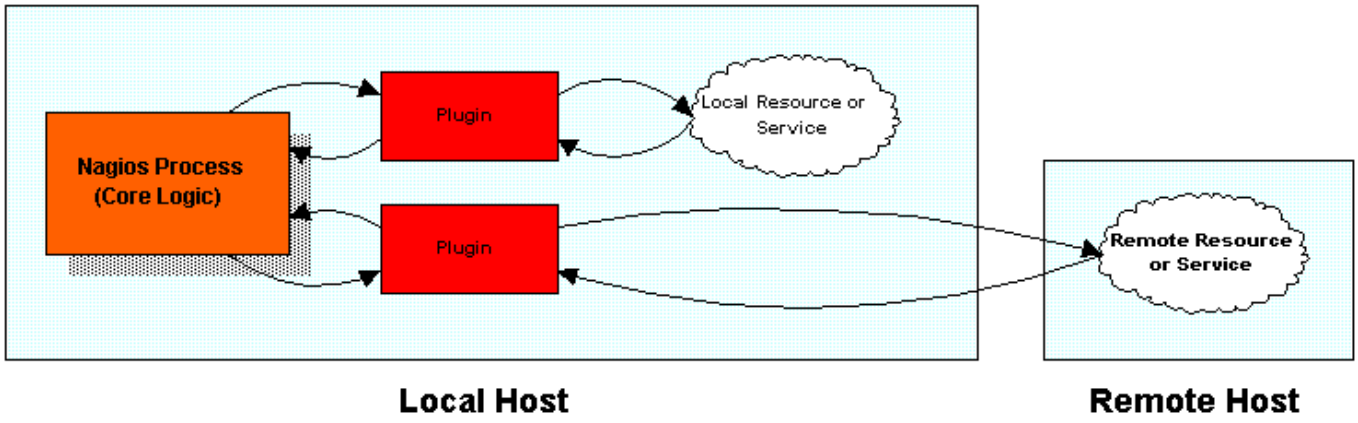
- d'une part dans la configuration de Nagios, pour que Nagios sache quelles vérifications lancer sur ou à destination de quelles machines ou services,
- d'autre part par le code retour ainsi que la sortie standard d'un greffon.

En effet, un greffon n'est qu'un script(programme) qui doit fournir un code retour (**0 : tout va bien ou *OK*, 1 : avertissement ou *WARNING*, 2 : alerte ou *CRITICAL et 3 : UNKNOWN***), et éventuellement un petit message décrivant le déroulement de l'exécution (aide au diagnostic en cas de problème).

Ce sont les états (code retour des greffons) qui seront ensuite remontés au moteur devant prendre les décisions et lancer les actions programmées.

Ces greffons fonctionnent soit en local sur la machine supervisée (par exemple les vérifications sur les disques), soit effectuent des tests à distance (tests sur des protocoles réseaux tels SMTP ou exécution à distance via SSH ou autre).

Les vérifications peuvent être passives (rarement ou dans certains cas où la sécurité impose d'interdire une connexion dans un sens, ou encore dans le cas de supervision hiérarchique), mais le plus souvent actives.



2.3)Mise en réseau de la supervision

Les greffons locaux au serveur de supervision sont exécutés directement par Nagios. La vérification d'un service à distance (par l'exécution d'un greffon situé sur une autre machine, par exemple, ou par SNMP), se fait, elle aussi, par le biais de l'exécution d'un greffon local au serveur Nagios.

Pour l'exécution de greffons à distance, plusieurs possibilités :

- Par le biais d'autres serveurs de supervision Nagios distants : Dans le cas de supervision distribuée qui concentrent les vérifications sur un site distant, et ne remontent que les problèmes.

- Les agents de transport ou d'exécution des tests, tels :
  - NRPE, (le *Nagios Remote Plugin Executor)* permet l'exécution de plugins à distance, à choisir parmi un certain nombre de services disponibles.
  - NSCA (*Nagios Service Check Acceptor)* permet de son côté la remontée d'informations de façon passive (vu du point de vue de Nagios).
  - *check_by_ssh*, qui suppose la création de comptes non privilégiés sur les machines, accessibles par l'utilisateur qui exécute le daemon Nagios, et qui ont les droits suffisants pour lancer les *plugins* nécessaires. Cette dernière solution a des implications en termes de sécurité.
  - NSCIient++, greffon lourd pour des serveurs Windows à surveiller.
  - Winrpe (Nagios NRPE For Wibndows)
  - *check_snmp*, pour la supervision de valeurs SNMP à travers le réseau.

L'ordonnancement des vérifications est assuré de façon locale à chaque machine, et surtout permet d'inverser le sens des connexions entre serveur supervisé et serveur superviseur, ce qui peut avoir un intérêt dans un réseau sécurisé, avec des pares-feu en diode (ne laissant passer les flux que dans un sens).

La différence entre NRPE et NSCA est l'initiateur de la vérification : NRPE reçoit (1) la demande de vérification de la part de Nagios, exécute la vérification (2), puis renvoie le résultat(3).
Alors qu'avec NSCA, la vérification est planifiée en local, exécutée (1), puis le résultat en est envoyé à Nagios (2).
Notons que la plupart de ces moyens supposent un greffon local au serveur Nagios, que ce soit check_nrpe, check_nt pour NSCIient++, ou encore check_by_ssh et check_snmp.

2.4)États hard et soft

Nagios fait la distinction entre les alertes en état soft et celles en état hard. Dès qu'un service remonte une erreur, celle-ci est d'abord définie dans un état soft et Nagios émet ses alertes sonores habituelles. Si après un certain nombre d'essais infructueux, configurable, le service renvoie toujours le même type d'erreur, l'alerte passe alors en état hard et Nagios envoie ses notifications (courriel, SMS, etc.).
L'état hard définit donc une alerte reconnue par Nagios comme posant réellement un problème, il ne s'agit pas d'un greffon qui a accidentellement renvoyé une mauvaise valeur. Ces valeurs seront réutilisées plus bas avec les gestionnaires d'événements.

2.5)Les escalades

Nagios supporte de façon optionnelle l'escalade des notifications envoyées aux contacts pour des services ou hôtes. Ainsi, on peut définir différents groupes de personnes et donc différents moyens de contacter ces personnes en fonction du nombre de notifications déjà envoyées.
Il est possible par exemple, sur certaines alertes, contacter quelques personnes s'occupant de la supervision par courriel, puis si le problème persiste après un temps déterminé, les contacter par SMS ou bien contacter d'autres intervenants.

3)Configuration

3.1)Configuration de Nagios : Apache

Dans le fichier httpd.conf d'Apache, inclure le fichier nagios.conf à écrire.

```
Include conf/nagios.conf
```

Le fichier nagios.conf va contenir les directives suivantes :

- Définition des correspondances entre répertoires et URLs relatives
- ScriptAlias /nagios/cgi-bin/usr/

```
Alias       /nagios     /usr/share/nagios/html
```

Notez que l'URL /nagios/cgi-bin *doit être définies* ***avant*** /nagios *car sinon, elle ne serait pas prise en compte car masquée.*

Cela permettra de pointer un navigateur sur le serveur Web où Nagios est installé, grâce à une URL du style : http://monserver/nagios/.

```
- Création des déclarations des répertoires correspondant aux URLs
-       <Directory /usr/share/nagios/cgi-bin>
-              Allow0verride AuthConfig
-               Options ExecCGl
-               Order deny,allow
-               Allow from all
-               AuthType Basic
-               AuthName "Nagios"
-               AuthUserFile /etc/nagios/htpasswd
-               <Limit GET POST>
-                     Require valid-user
-               </Limit>
-       </Directory>
<Directory /usr/share/nagios/html>
        AllowOverride AuthConfig
        Options Indexes FollowSymLinks
        Order deny,allow
        Allow from all
        AuthType Basic
        AuthName "Nagios"
        AuthUserFile /etc/nagios/htpasswd
        <Limit GET POST>
               Require valid-user
        </Limit>
</Directory>
```

3.2)Authentification

L'accès à Nagios doit être restreint, car il peut montrer des informations importantes voire confidentielles.
De plus, des actions peuvent être entreprises via l'interface Web, actions qui vont de l'acquiescement d'une alarme au redémarrage d'un serveur si *l'event handler* est défini.

La définition des règles d'authentification est déjà écrite ci-dessus, il reste à renseigner le fichier /etc/nagios/htpasswd. Cela se fait par le biais de l'utilitaire htpasswd fourni avec Apache.

```
# htpasswd [-c] /etc/httpd/htpasswd nagios
New password: ******
Re-type new password: ******
Updating password for user nagios
```

Cet utilisateur nagios doit correspondre non à l'utilisateur au sens POSIX créé plus haut, mais à un contact défini dans /etc/nagios/contacts.cfg.

3.3)Configuration Nagios

Les fichiers de configuration de Nagios se trouvent dans le répertoire /etc/nagios.
Ces fichiers de configuration, à l'exception des fichiers nagios.cfg et cgi.cfg, utilisent une structure unique de définition des objets sur le principe suivant :

```
define {
       param1 value
       param2 value
       ...
       paramn value
}
```

Cela implique qu'aucun espace ne pourra être inséré dans les valeurs des paramètres. Il faudra donc y faire attention. Pour tester la cohérence de vos fichiers de paramétrage, utilisez la commande :

```
# /usr/bin/nagios -v ../etc/nagios.cfg
```

ou, si le script de lancement /etc/init.d/nagios a été installé :

```
service nagios reload
```

ou encore, si la distribution n'a pas fournit le script service (qui devrait être obligatoire) :

```
/etc/init.d/nagios reload
```

qui cumule vérification de configuration et lancement de Nagios.

3.4)Objets de Nagios

Au commencement est le serveur *(host)*. Ce serveur n'est pas tout seul, il doit donc pouvoir le définir ainsi que ses congénères, qui peuvent ne pas être des serveurs, mais tout équipement doté d'une interface réseau. Ces hôtes peuvent être regroupés dans un ou plusieurs groupes, par exemple sur critères technologiques (les linux avec les linux, etc.), par clients .... Ce regroupement peut permettre d'agir sur un ensemble de machines, par exemple si l'on a prévu d'arrêter toute la plate-forme d'un client qui a 8 serveurs, nous prévoirons un arrêt sur le groupe du client, plutôt que perdre du temps à prévoir l'arrêt sur chacun des serveurs. Viennent ensuite les services. Ces services correspondent en fait au produit des machines par les greffons et par les paramètres qui peuvent leur être appliqués.

Si une machine a 200 systèmes de fichiers, il sera fait 200 fois appel au greffon `check_disk`, à chaque fois avec en paramètre le nom du système de fichier à vérifier, et avec les seuils de criticité associés. Idem pour le greffon `check_proc`, qui peut compter les processus actifs, les processus zombies, et les processus swapés. Donc trois services différents. Ces services s'appuient sur des greffons, qu'il faut donc définir. C'est le rôle des clauses command, qu'on retrouvera généralement dans un fichier de configuration `checkcommands.cfg`.

L'ordonnancement de la vérification des services se fait selon des calendriers définis par les clauses `timeperiod` (fichier `timeperiods.cfg`).
Le rôle de Nagios est donc de prévenir un contact lorsqu'un problème survient. C'est le rôle des fichiers `contacts.cfg` (clauses `contact`) et `contactgroups.cfg`. L'escalade des avertissements et autres alertes entre les différents groupes d'intervenants se faisant dans le fichier `escalations.cfg`..
Enfin, les commandes externes, non-destinées au lancement de greffons, sont centralisées dans le fichier `misccommands.cfg`, tel que l'envoi de courriels, de SMS (via l'utilisation d'outils tiers), etc.

D'autres fichiers :

- `dependencies.cfg` qui définit les dépendances entre services (sur le même serveur ou sur des serveurs différents, clause `servicedependency`), par exemple tel serveur Web virtuel dépend de la présence d'un processus Apache ;
- `cgi.cfg` pour la configuration des services CGI qui font la partie graphique de Nagios ;
- `hostextinfo.cfg` : les petits plus graphiques (icônes, coordonnées des objets sur la carte graphique) pour les équipements ;
- `serviceextinfo.cfg` : la même chose, mais pour les services.

3.5)Les patrons

L'arrivée en 2002 de Nagios 1.04beta a vu l'introduction des patrons (templates) de configuration. Inutile donc, si l'on a plusieurs serveurs identiques (Linux, Solaris ...) et équipements réseau des mêmes constructeurs, avec des contraintes de disponibilité identiques, de dupliquer manuellement les mêmes paramètres sur chacun. Cela simplifie la lecture (seule l'information directement pertinente est visible à la définition du serveur/service), si l'on doit modifier manuellement les *timeperiods* de 200 services sur plusieurs serveurs différents. Avec les patrons, il suffit de modifier le patron des services de chaque serveur, voire de modifier le patron des patrons si les différents serveurs fournissent une même application.
Exemples :

```
# definition du patron par défaut
define service {
name                            generic-service       ;le nom du patron
register                        0                     ;objet non enregistre car c'est un patron et
                                                             ;non un vrai service
active_checks_enabled           1
passive_checks_enabled          1
parallelize_check               1
check_freshness                 0
notifications_enabled           1
event_handler_enabled           1
flap_detection_enabled          0
retain_status_information       1
retain_nonstatus_information    1
notification_interval           120
notification_period             24x7
notification_options            w,c,u,r
max_check_attempts              3
is_volatile                     0
normal_check_interval           3
retry_check_interval            1
check_period                    24x7
contact_groups                  network-admins
}
```

Un patron voit un grand nombre de paramètres configurés, en fait, exactement les mêmes que pour un service normal.
La seule différence se fait dans le paramètre register, positionné à 0, qui fait que la définition est considérée comme un patron.

Un patron peut étendre les informations d'un autre patron :

```
define service {
       name           switch1-service
       register       0
       use            generic-service       ;On lui dit d'utiliser les info du patron 'generic-service'
       host_name      switch1
  }
```

Enfin, voici comment utiliser le patron pour définir un service Ping :

```
# exemple de service utilisant un patron
define service {
       use                    switch1-service       ;il doit utiliser les info de ce patron
       service_description    Ping
       check_command  check_ping!200,50%!400,80%
  }
```

3.6)Notifications et escalades

Nagios vérifie régulièrement l'ensemble des services sur le parc configuré. Cet intervalle de temps est configurable pour chaque service à l'aide du champ `normal_check_interval`. Si le greffon renvoie un état différent de OK, une alerte en état soft est levée. Nagios vérifiera autant de fois que `max_check_attempts` lui indique le service toutes les `retry_check_interval` minutes d'intervalle. Si celui-ci reste en erreur après tous les essais, une première notification est envoyée.

Si après un intervalle de `notification_interval` minutes, le problème n'est toujours pas réglé, Nagios enverra une autre notification et continuera ainsi jusqu'à ce que le problème soit résolu ou acquitté.

Dans certains cas, il peut être utile de mettre en place des escalades sur les services. Le principe est simple, pour chaque service où l'on souhaite une escalade des notifications, il suffit de préciser à quels groupes de contacts Nagios devra envoyer les notifications. Cela se fait en fonction du nombre de notifications envoyées.
Voici un petit exemple :

```
define serviceescalation {
       host_name                       switch1
       service_description             Ping
       first_notification              3
       last_notification               5
       notification_interval           90
       contact_groups          network-admins,managers
  }
define serviceescalation  {
       host_name               switch1
       service_description     Ping
       first_notification      6
       last_notification       0
       notification_interval   60
       contact_groups          tout-le-monde
  }
```

On continue ici l'exemple des patrons : on possède un équipement réseau désigné par `switch1`. Par défaut, les notifications sont envoyées toutes les deux heures (120 minutes dans le patron par défaut) au groupe de contacts `network-admins`. Grâce aux escalades que l'on vient de configurer, si le problème est toujours présent de la troisième notification (`first_notification`) à la cinquième (`last_notification`) et toutes les 90 minutes (`notification_interval`), celle-ci sera envoyée aux groupes de contacts `network-admins` et `managers`.
Enfin, si au bout de six notifications au total, l'alerte continue, Nagios avertira les gens du groupe de contacts `tout-le-monde` indéfiniment (d'où le 0 dans le champ `last_notification`) à une période de 60 minutes. Dans tous les cas, toutes les personnes qui ont reçu la notification de l'erreur recevront la notification de retour à la normale du service.

3.7)Les gestionnaires d'évènements (*event-handlers*)

Les gestionnaires d'évènements sont des commandes externes optionnelles qui sont exécutées à chaque fois qu'un changement d'état d'un hôte ou d'un service a lieu. Une utilité triviale de ces gestionnaires d'évènements (particulièrement pour les services) réside dans la capacité de Nagios à résoudre les problèmes de manière préventive avant que quelqu'un ne reçoive une notification. Une seconde utilité est celle d'enregistrer les évènements relatifs aux hôtes ou services dans une base de données externe.
Dans la plupart des cas, les commandes de gestionnaires d'évènements seront des scripts écrits en shell ou en Perl. Ils doivent comporter au moins les macros suivantes comme arguments :

```
- Macros de gestionnaire d'évènements de service : $SERVICESTATE$, $STATETYPE$ et $SERVICEATTEMPT$ ;
- Macros de gestionnaire d'évènements d'hôte : $HOSTSTATE$, $STATETYPE$, $HOSTATTEMPT$.
```

Les scripts examinent les valeurs des arguments qui leur sont passés et exécutent les actions nécessaires en fonction de ces valeurs. Voir plus bas le principe de mise en redondance de Nagios comme exemple.

3.8)La cosmétique

A l'aide des informations additionnelles sur les équipements, il est possible d'ajouter des logos, des descriptions, un lien telnet (qui connaît l'URL pour faire du SSH ?) vers les machines, etc. Tout ça vous permettant de reconnaître facilement et accéder directement à d'autres informations sur vos équipements.
Exemple :

```
define hostextinfo {
       host_name             ts1000-1
       icon_image            base/cyclades_ts.gif
       vrml_image            base/cyclades_logo.png
       statusmap_image       base/cyclades_ts.png
       icon_image_alt Serveur Terminaux
  }
```

L'icône donnée dans le champ `icon_image` sera affichée à chaque fois à côté du nom de la machine. Le champ `vrml_image` renseigne sur l'image utilisée dans la vue en 3D. `statusmap_image` est utilisé dans la carte 2D *(status map).* Enfin, le texte donné dans `icon_image_alt` sera affiché en texte alternatif de l'icône.

On pourrait aussi ajouter un lien qui serait affiché dans les informations de l'équipement avec le champ `notes_url` qui pourrait par exemple pointer un logiciel de métrologie.

Ou encore compléter les champs `2d_coords` et `3d_coords` pour fixer l'emplacement de la machine sur les cartes 2D et 3D.

! Attention cependant à la définition du chemin relatif vers les images. Les chemins se donnent de façon relative à un répertoire logos (nom codé en dur, nous semble-t-il) du répertoire image. Ce répertoire image étant lui-même un sous-répertoire de la partie Web de Nagios, ou plus exactement d'un paramètre de configuration des CGI. Ce paramètre est `physical_html_path`, et pourrait pointer sur `/usr/share/nagios`.

4)Les greffons

4.1)Les greffons officiels

Les greffons officiels sont disponibles sur SourceForge, projet *nagios-plug.*
Ils sont divisés en deux catégories : les greffons principaux (« *core plugins* ») et les greffons dits contribués, qui ne sont pas forcément maintenus, mais néanmoins distribués dans l'archive officielle.
D'autres greffons existent, mais sont bien souvent d'un usage limité car destinés à un usage particulier. Il est possible d'interroger un moteur de recherche (« Nagios » et « plugin » mots-clés indispensables).
La compilation des greffons officiels (pour ceux écrits en langage C) ne posent pas de problème particulier, sinon qu'il faut bien évidemment prêter attention aux dépendances, en particulier les bibliothèques permettant d'écrire des clients de service réseau, radius ou LDAP étant des exemples.

4.2)Écrire un greffon

L'écriture d'un greffon doit respecter deux règles : légèreté et simplicité. En effet, ce greffon doit passer inaperçu dans son exécution, au sens où il ne doit pas se faire sentir sur la charge globale de la machine, et doit rester simple dans l'esprit d'un greffon Nagios. À ce titre, il doit accomplir une tâche et une seule, mais le faire bien. Rappelons le but du greffon : renvoyer un code retour, ainsi qu'un petit texte explicatif sur sa sortie standard.
Exemple : certains serveurs n'ont plus de disques SCSI sur une carte RAID. Vérifions que les miroirs logiciels mis en œuvre sous Linux fonctionnent correctement. Le greffon suivant va donc devoir lire le pseudo-fichier /proc/mdstats, analyser son contenu, éventuellement croiser avec /etc/fstab ou /proc/mounts (pour éviter de forker un nouveau processus, ce qui prendra du temps et des ressources) afin de fournir le point de montage du système de fichier en plus du périphérique RAID qui défaille.
Tout d'abord, voici le script :

```
#!/usr/bin/perl -w
use strict;
use warnings;
use English;
use Getopt::Long;
use vars qw($PROGNAME $opt_V $opt_h $opt_t);
use lib "/home/utilisateur/cvs/nagios/nagios-plugins-1.4.0alpha1/plugins-scripts";
use utils qw(%ERRORS &print_revision &support );
use Data::Dumper;
$PROGNAME="check_linuxraid";
 $REVISION='$: check_linuxraid.pl ,v 1.1 2006/03/29 08:33:29 utilisateur Exp $';
 sub print_help ();
 sub print_usage ();
```

```
my (%data, %slices, %types, %status, %alerts, %rcs);
my $msg=''; # empty string
Getopt::Long::Configure('bundling');
GetOptions(
        "V"   => \$opt_V, "version" => \$opt_V,
        "t"   => \$opt_t, "test"    => \$opt_t,
        "h"   => \$opt_h, "help"    => \$opt_h
        );
if ($opt_V) {
        print_revision($PROGNAME, $REVISION);
        exit $ERRORS{'OK'};
        }
if ($opt_h) {
        print_help();
        exit $ERRORS{'OK'};
        }
if (! $opt_t) {
        if (!open DATA, '< /proc/mdstat') {
                print "Unable to open /proc/mdstat\n";
                exit $ERRORS{'UNKNOWN'};
                }
        }
while (<DATA>) {
  my $curdev;
  chomp;
  if (m/^(md\d+)\s*:\s*(.*)$/) {
        $curdev=$1;
        $data{$curdev} = $2;
        }
  $data{$curdev} .= " $1" if m/^\s+(.*)$/;
  }
foreach my $md (keys %data) {
  foreach ( split / /, $data{$md}) {
        push @{$slices{$md}}, $_ if /\w+\[\d+\]/;
        $types{$md} = $_ if /raid\d/;
        $status{$md} = $1 if /\[([U_]+)\]/;
  }
  if ($status{$md} =~/_/){
        my $p=-1;
        while (($p = index($status{$md}, '_', $p+1)) != -1 ) {
        push @{$alerts{$md}}, $slices{$md}[$p];
        }
  }
  # on sort un CRITICAL si des miroirs n'ont qu'une patte
  if ($types{$md} eq 'raid1' && @{$slices{$md}} == 0 ) {
        $rcs{'CRITICAL'}++;
        $msg .=" $md(2few dev)";
        }
}
#print Dumper %slices, \%types, \%alerts, \%status;
foreach my $md(keys %alerts) {
        $msg .= " $md(";
        foreach my $slice ( @{$alerts{$md}} ) {
                $msg .= " $slice";
                if ($slice =~ /\(F\)/) {
                        $rcs{'CRITICAL'}++;
                 }
                elsif ($data{$md} =~  /recovery/) {
                        # ca reconstruit
                        $msg .= " rebuilding";
                        $rcs{'WARNING'}++;
                } else {
                        # il y a des cas ou le device n'est pas failed, et 0 le miroir n'est
                        # pas actif non plus (bug du driver md)
                        $rcs{'CRITICAL'}++;
                }
        }
       $msg .= ")";
}
my $rc = $rcs{'CRITICAL'} ? $ERRORS{'CRITICAL'} : ( $rcs{'WARNING'} ? $ERRORS{'WHARNING'} : $ERRORS{'OK'} ) ;
$msg =~ s/\(\s/\(/g;
my %t = reverse %ERRORS;
```

```
$msg = "ALL RAID devices sane (" . join(', ', keys %data). ")"
 if $rc == $ERRORS{'OK'};
 print "$t{$rc}:$msg\n";
 close DATA;
 exit $rc;
 sub print_usage () {
  print "Usage:\n";
  print " $PROGNAME\n";
  print " $PROGNAME [-h I -help]\n";
  print " $PROGNAME [-V | -version]\n";
  print " $PROGNAME -t\n";
  }
sub print_help () {
  print_revision($PROGNAME, $REVISION);
  print "Copyright @ 2004 utilisaeur\n\n";
  print_usage();
  print "\n";
  print "Use -t to test the script, /proc/mdstat source used is internal to the script\n";
  print "\n"; support();
}

RESULTAT
_DATA_
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 sdb5[1] sda5[0] 4200896 blocks [2/2] [_U]
md1 : active raid1 sdb6[1] sda6[0] 2104384 blocks [2/2] [UU]
md2 : active raid1 sdb7[1] sda7[0] 2104384 blocks [2/2] [UU]
md3 : active raid1 sdc7[1] sdd8[2] sde5[0] 1052160 blocks [2/2] [UU]
md4 : active raid5 hdh1[3] hdg1[2] hdf1[1] hde1[0] 360182016 blocks level 5, 256k chunk, algorithm 2 [4/4] [UUUU]
md5 :  active raid5 sdf1[2] sde1[3] sdg1[1] sdd1[4] sdc1[0]      1638144 blocks level 5, 64k chunk, algorithm 0
      [5/5] [UUUUU]
md6 :  active raid5 sde2[7] sdf2[4] sdg2[3] sdd2[6] sdc2[2] sdb2[1] sda2[0]       104196864 blocks level 5, 64k
      chunk, algorithm 0 [7/6] [UUUUU_U]         [===>.................] recovery = 17.0% (2966244/17366144)
      finish=119.2min speed=2011k/sec
md7 :  active raid1 hde5[0](F) hdf7[1]      39262720 blocks [2/1] [_U]
md8 :  active raid1 sda5[0]      4200896 blocks [1/1] [U]
unused devices: none
```

On utilise dans ce script un module venant des plugins Nagios : `utils.pm`. Il définit le *hash* qui sera utilisé par la suite pour la définition du code retour : %ERRORS, défini comme suit :
`%ERRORS=('OK'=>0,'WARNING'=>1,'CRITICAL'=>2,'UNKNOWN'=>3,'DEPENDENT'=>4);`
Rien de bien sorcier dans l'écriture de ce script, sinon que l'utilisation du *hash* %ERRORS permet d'éviter les impairs et d'avoir un nom sur un code retour (*Attention néanmoins à ne pas faire de faute d'orthographe dans l'écriture de CRITICAL ou de WARNING, voire de UNKNOWN).*

L'option `-t` permet de tester le script en situation avec les données.
Pour utiliser ce script, ne pas oublier de retirer la ligne use lib ou de mettre ce qu'il faut à cet endroit.

5)Configuration d'un agent NRPE

5.1)Principe de fonctionnement

NRPE, *"Nagios Remote Plugin Executor"*, est un des moyens de supervision à distance offert par Nagios.

Il offre en effet la possibilité de profiter de la puissance des greffons, ceux-ci ne s'exécutant pas pour autant sur le serveur Nagios.

Le principe de fonctionnement est le suivant : les greffons sont installés sur la machine à superviser, compilés pour son architecture (car c'est elle qui va les lancer), ainsi que le serveur (daemon) NRPE. Le client de ce serveur, un greffon comme un autre, `check_nrpe`, est lui installé sur le serveur Nagios.

Le serveur `nrpe` voit ensuite sa configuration composée d'une liste de vérifications nommées, auxquelles correspondent un greffon avec ses paramètres, ainsi que quelques autres options de configuration pour le fonctionnement du serveur `nrpe` en lui-même.

Le serveur Nagios voit, lui, sa configuration s'enrichir d'appels au seul greffon `check_nrpe`, auquel on passe en paramètre le nom de la vérification à faire effectuer sur la machine supervisée. Ce `check_nrpe` initiera donc une connexion vers l'agent `nrpe` et lui demandera uniquement l'exécution d'une vérification. L'agent `nrpe` lancera ensuite le greffon considéré et retournera le code retour ainsi que le contenu de la sortie standard du greffon.

Si le serveur Nagios demande une vérification non configurée (i.e. un nom inexistant dans le fichier `nrpe.cfg` de l'agent `NRPE`), il se verra opposer une fin de non-recevoir, et positionnera le statut de la vérification à UNKNOWN.

On peut préciser des listes de machines autorisées à demander des vérifications dans la configuration de `nrpe`, de même qu'un lien SSL peut être mis en place pour éviter les écoutes entre greffon `check_nrpe` et agent `nrpe`.

Enfin, et c'est là le nœud du problème avec `nrpe`, les paramètres de seuils pour qu'une alarme soit remontée sont spécifiés sur la machine supervisée.

Les implications de ce fait sont importantes : cela signifie que la configuration n'est pas centralisée sur le seul serveur Nagios, mais éclatée sur tout le parc.

Il faut donc trouver un moyen d'automatiser la création de la configuration, pour maintenir la cohérence entre les services configurés sur le serveur Nagios et ceux sur l'agent `nrpe`, gérer la quantité de services (par exemple des serveurs avec près de 200 systèmes de fichiers distincts, sans compter les montages d'un même système de fichier à plusieurs endroits), et par-là même centraliser la configuration.

5.2)Déploiement

Il est très simple de mettre en œuvre la supervision via `nrpe`.

Le plus long est, comme nous venons de le voir, de configurer de façon **synchrone** la surveillance des systèmes de fichiers et autres greffons, dont la configuration apparaît à la fois sur le serveur Nagios (dans un fichier `services-machine1.cfg` inclus par `nagios.cfg`, par exemple) et sur la machine supervisée (dans `nrpe.cfg`).

L'idéal est donc d'utiliser un script Perl qui prendra le contenu du fichier `/etc/vfstab` pour en extraire deux fichiers : `new_nrpe.cfg` dont le contenu est à reprendre pour la configuration de l'agent `nrpe`, et `services-hostname.cfg` à insérer dans le fichier de configuration de Nagios sur le serveur de supervision. Ce script est `cnrpe.pl` :

```
#!/usr/local/bin/perl -w
use warnings;
use vars qw/@nrpe @check_nrpe @line/;
my $hostname;
$hostname=qx/hostname/;
$hostname=~ tr/[A-Z]/[a-z]/;
chomp $hostname;
$count=0;
open VFSTAB, "< /etc/vfstab";
while(<VFSTAB>) {
        chomp;
        s/#.*//; # drop comments
        next if /^\s*$/; # empty lines
        @line=split(/\s/);
        if (line[3] =~ m/vxfs|ufs/ ) {
                $nrpe[$count]=$line[0];
                $check_nrpe[$count]=$line[2];
                $count++;
        }
}
close VFSTAB;
-$count;
```

```
open STDOUT, "> new_nrpe.cfg";
print << "EOT";

#
# nrpe.cfg pour $hostname
#
server_port=5666
allowed_hosts=10.81.0.243
nrpe_user=nagios
nrpe_group=nagios
debug=0

# Service checks
command[check_users]=/opt/nagios/libexec/check_users  10 15
command[check_load]=/opt/nagios/libexec/check_load 5 10 15 20 25 30
EOT

foreach $i (0..$count) {
        print "command[check_disk".$i."]=/opt/nagios/libexec/check_disk 80 95";
        print $nrpe[$i]."\n";
    }

open STDOUT, "> services-${hostname}.cfg";
print << "EOT";
# Définition du template pour ${hostname}

define service{
        name                            ${hostname}-service
        use                             generic-service
        host_name                       $hostname
        notification_interval 120
        notification_period   24x7
        notification_options  w,u,c,r
        max_check_attempts    3
        is_volatile           0
        normal_check_interval 3
        retry_check_interval  1
        contact_groups                  sun-admins
        check_period          24x7
        register                        0
}
# Service definition
define service {
        use                             ${hostname}-service
        service_description   SMTP
        check_command         check_smtp
}
define service {
        use                             ${hostname}-service
        service_description   load
        check_command         check_nrpe!check_load
}
define service {
        use                             ${hostname}-service
        service_description   users
        check_command         check_nrpe!check_users
}
EOT

foreach $i (0..$count) {
        print << "EOT";
define service{
        use                             ${hostname}-service
        description           $check_nrpe[$i]
        contact_groups                  sun-admins
        check_command         check_nrpe!check_disk$i
}
EOT

};
```

Ce script est un exemple de ce qu'il est possible de faire pour Solaris. Il est ensuite possible, via des tables de dispatchs (des *hash* Perl), d'affiner les paramétrages des greffons machine par machine.

Ce script, une fois distribué sur toutes les machines du parc, permet de générer à la volée les deux fichiers nécessaires nrpe.conf et services-hostname.cfg. Il suffit ensuite de rapatrier via SSH le second, de le déposer au bon endroit (/etc/nagios par exemple) sur le serveur Nagios, et de relancer Nagios. Toutes les opérations peuvent bien-sûr être encapsulées dans un script shell qui fera toutes les opérations.

6)Interface et utilisation

L'interface utilisateur de Nagios se fait par le biais d'un serveur Web, comme d'ailleurs la plupart des autres outils utilisables. Cela permet une ubiquité de l'outil, ainsi qu'une indépendance de fonctionnement vis-à-vis des matériels et logiciels systèmes d'un poste client.

6.1)Revue de détail des menus

- **Tactical overview**

Vue synthétique où il est possible de trouver l'essentiel des informations importantes sur le bon fonctionnement du système d'information supervisé.
Cette vue très simple rassemble les informations quantitatives essentielles au premier abord : nombre d'équipements supervisés, services fonctionnels, services non fonctionnels, alarmes en cours, acquittées, non acquittées. Elle permet de naviguer facilement et directement vers l'information pertinente. Par exemple, il suffit de cliquer sur un lien sur fond rouge (signifiant qu'un service ou un serveur est tombé) pour aller directement à la liste des problèmes.

- **Service Détail**

La liste exhaustive, par équipement, des services supervisés. Attention, cette vue peut être lourde si vous avez un parc important avec un grand nombre de services supervisés.

- **Host Detail**

La liste des machines, en vert si elles sont vues sur le réseau (soit par le biais d'une vérification par greffon, soit par un ping si aucun service n'est défini).

- **Status overview**

La même liste que précédemment, mais avec les regroupements par *hostgroup.* Cela permet une vue rapide sur un sous-ensemble de du parc (par site, par client, etc.)

- **Status summary**

Une vue encore plus synthétique par *hostgroup,* avec indication du nombre d'équipements et de services dans les différents états possibles.

- **Status grid**

Toujours par groupe d'équipements, une vue des machines, avec en regard les services (par leur nom) et leur état (par la couleur).

- **Status map**

Carte des états des serveurs, en deux dimensions, permettant de voir les relations de dépendances entre objets telles qu'elles ont été configurées. Peu d'intérêt, hormis le côté dépendances.

- **3D Status map**

Vue navigable en 3D de l'infrastructure, plus gadget que réellement utile.

Non seulement ces différentes vues sont pléthoriques, mais en plus, elles peuvent prendre divers paramètres afin de montrer tel ou tel groupe, serveur, service. Leur intérêt est donc limité, sauf dans le cas où l'on veut pouvoir donner l'accès à la vue de son parc à un client externe.

Mais dans la majorité des cas, un PC avec un navigateur sur la « *tactical overview* » suffit amplement.

Voir le but des vues *Service Problems, Host Problems,* et *Network outages* pour les problèmes.

Enfin, un historique des commentaires et des périodes d'arrêt prévues sont disponibles derrière les liens *Comments* et *Downtime.*

Quelques informations sont aussi disponibles sur le fonctionnement de Nagios en lui-même : paramétrage (dans les fichiers et dans ce qu'il est possible de configurer dans l'interface, comme l'arrêt de la notification) via *View config* et *Process Info,* le temps d'un cycle complet de vérifications et quelques statistiques dans *Performance Info,* et enfin l'état actuel de la file des vérifications à venir.

6.2)Que faire quand un problème est remonté ?

Lors de l'indication d'un problème, soit par le biais de l'alarme sonore, par mail ou par SMS (via Kannel par exemple) :

- Se connecter à Nagios puis sélectionner les cases rouges ou orange, et acquittez l'alarme.
- Dans le cas d'une possible correction rapide pour accélérer le retour à la normale, délectionner le lien *Re-schedule the next check of this service* pour un service, ou sur *Schedule An Immédiate Check Of All Services On This Host* afin de remettre en début de file les prochaines vérifications pour le service ou la machine considérée. Cela permet deux choses : faire baisser le temps de détection de résolution du problème, et surtout d'éviter d'attendre les 3 à 5 minutes (suivant configuration) avant la prochaine vérification.
- Dans le cas d'une impossible correction rapide, acquitter le problème afin qu'il ne remonte plus d'alarme, mais sera toujours vérifié régulièrement par Nagios. Puis corriger le problème ultérieurement.

6.3)Reporting

Nagios offre des possibilités de *reporting*, que ce soit du point de vue des machines supervisées que des services.
Les données il est possible d'accéder sont les suivantes, le reste est affaire de présentation : historique des résultats d'exécution des greffons, rapport de disponibilité des machines et services.

- ***Trends***

Les grandes tendances sur le parc supervisé.

- ***Availability***

La disponibilité par machine, groupe de machines, services et bientôt groupe de services. Cette disponibilité est fonction non seulement du temps de bon fonctionnement d'un serveur, mais aussi de **tous** ses services. Attention donc à ne pas plomber le contrat de service *(SLA)* car le calcul ne sera pas sur le bon indicateur.

- ***Alert histogram :***

Génération de graphiques (qui n'ont d'histogrammes que le nom) sur les alertes et les rétablissements.
Les rétablissements sont justes des acquittements sur des syslog.

- ***Alert history***

Une vue synthétique des dernières alertes remontées (que ce soit sur un problème ou un retour à la normale).

- ***Alert summary***

Quasiment la même information que le point précédent, mais sous forme tabulaire, avec en plus les informations remontées par les greffons. Un bon complément à la *Tactical overview*.
Les liens sont le résultat de l'utilisation du greffon `urlize` qui permet d'associer une URL au résultat de l'exécution du vrai greffon. En l'occurrence, un lien vers la gestion des syslog.

- ***Notifications:***

Donne l'historique de tous les échanges entre Nagios et ses utilisateurs (alertes, escalades, acquittements).

- ***Event log:***

Le journal des événements, qui donne un peu la même information que le point précédent, mais sous une forme différente. Le journal contient aussi les arrêts et redémarrages de Nagios. C'est la main courante Nagios, en somme.

D'autres types de rapports sont à imaginer en fonction de l'utilisation et des besoins. Dans les faits, Nagios donne en standard des outils très flexibles en matière de *reporting*.
Il est toujours possible de reprendre toutes les données dans les fichiers de journaux de Nagios, de les analyser, et d'en extraire des 'informations.
Pour ce qui est des informations de performance éventuellement remontées par les greffons, plusieurs projets existent pour les exploiter. Citons-en deux : APAN et perfparse.

7)Nagios redondant

II est possible de configurer Nagios sur deux serveurs différents, de façon à continuer à avoir une supervision lors d'une interruption d'une liaison entre deux sites, pour peu que les moyens de prévenir soient eux-aussi redondants.

7.1)Le principe

On dispose d'une machine maître et d'une esclave. Les deux nagios vérifient chacun de leur côté tous les serveurs et les services.
Seulement le nagios esclave n'envoie aucune notification, ni courriel, ni SMS.
Si le maître tombe, l'esclave le détecte par deux moyens : le maître ne répond pas aux pings ou aucun processus avec le nom nagios ne tourne sur le maître.
Un évènement est alors déclenché sur l'esclave et celui-ci se met alors à envoyer des notifications (courriels et SMS) pour le problème du maître et pour les éventuels autres problèmes.
Si le service nagios ou si le maître recommence à répondre aux requêtes de l'esclave, ce dernier arrête alors les notifications et tout revient dans l'ordre.

7.2)Configuration du maître

On ajoute un service via `nrpe` pour vérifier la présence d'un process nagios (`/etc/nrpe.conf` généré par le script `cnrpe.pl`).

```
command[check_nagios]=/usr/lib/nagios/plugins/check_procs -c 1: -C nagios
```

et c'est tout ! Le maître n'est pas au courant de la présence de l'esclave.

7.3)Configuration de l'esclave

On désactive d'abord la notification (fichier `/etc/nagios/nagios.cfg`)

```
enable_notifications=0
```

On crée ensuite deux gestionnaires d'événements *(events handlers)* qui, suivant l'état du processus ou du serveur, activent ou désactivent la notification (`/usr/lib/nagios/enventhandlers/handler-master-host-event`)

```
#!/bin/sh
# On ne prend des décisions que lorsqu'on est sûr donc en état hard
case "$2" in
HARD)
        case "$1" in
        DOWN)
                # Le maître est tombé !
                # II faut devenir la machine maître et
                # prendre la responsabilité de la supervision
                # du réseau, donc activer les notifications.
                /usr/lib/nagios/eventhandlers/enable_notifications
                ;;
        UP)
                # La machine maître est remontée !
                # Il faut reprendre la place d'esclave et
                # laisser le maître s'occuper de la supervision,
                # on désactive donc les notifications.
                /usr/lib/nagios/eventhandlers/disable_notifications
                ;;
        esac
        ;;
esac
exit 0
```

(et `/usr/lib/nagios/eventhandlers/handle-master-proc-event`)

```
#!/bin/sh
# On ne prend des décisions que lorsqu'on est sûr, donc en état hard
case "$2" in
HARD)
        case "$1" in
        CRITICAL)
                # Le processus principal de Nagios ne tourne pas !
                # II faut devenir la machine maître et
                # prendre la responsabilité de la supervision
                # du réseau, donc activer les notifications.
```

```
                /usr/lib/nagios/eventhandlers/enable_notifications
                ;;
        WARNING)
                ;;
        UNKNOWN)
        # Le processus principal de Nagios tourne peut
                # être... Nous ne faisons rien ici, mais pour
                # être sûr, on pourrait décider que l'esclave
                # devienne maître dans ces situations.
                ;;
        OK)
                # Le processus principal de Nagios refonctionne !
                # I1 faut reprendre la place d'esclave et
                # laisser le maître s'occuper de la supervision,
                # on désactive donc les notifications.
                /usr/lib/nagios/eventhandlers/disable_notifications
                ;;
        *)
                ;;
        esac
        ;;
esac
exit 0
```

On définit ensuite deux nouvelles commandes pour gérer les événements dans `/etc/nagios/misccommands.cfg` :

```
#
# COMMANDE POUR LA MISE EN PLACE DE LA REDONDANCE
#
# Les commandes suivantes sont des "event handlers", en fonction de l'état
# du maître ou du service nagios, il donne la main à l'esclave pour la notification
# ou la retire.
#
define command{
        command_name   handle-master-host-event
        command_line   /usr/lib/nagios/eventhandlers/handle-master-host-event $HOSTSTATE$ $STATETYPE$
}
define command{
        command_name   handle-master-proc-event
        command_line   /usr/lib/nagios/eventhandlers/handle-master-proc-event $SERVICESTATE$ $STATETYPE$ }
```

On ajoute le gestionnaire d'évènements sur le serveur maître (`/etc/nagios/hosts.cfg`):

```
define host {
        use                     unix-host
        host_name               srvmaster
        alias                   srvmaster - Supervision
        address         srvmaster
        event_handler   handle-master-host-event
        parents         cisco1
}
```

Puis le service vérifiant la présence d'un processus nagios chez l'esclave (`/etc/nagios/services-drivers.cfg`) :

```
define service {
        name                            Nagios
        use                             srvmaster-service
        checlk_command          check_nrpe!check_nagios
        event_handler           handle-master-proc-event
        service_description     Redondance Nagios
}
```

7.4)Les limites

La fonctionnalité précédente n'est que la surveillance d'un serveur Nagios par son collègue, avec remplacement automatique. Cela ne dispense nullement de répliquer le reste de la configuration entre les deux serveurs, qui, rappelons-le, ne communiquent pas entre eux.
Et pour que cette mise en œuvre de la redondance soit efficace, il vaut mieux que les configurations soient synchrones.
Cela peut bien évidemment se faire par copie de fichiers de configuration, mais attention aux dépendances réseau (directive parent) : en effet, un routeur ne sera pas forcément vu par la même route, ni même par la même patte, et donc adresse.
Sans parler des parefeu qui peuvent être présents au milieu de tout ça.

8)Conclusion

Nagios est un bon logiciel libre de supervision. Son faible coût d'acquisition négligeable est néanmoins compensé par une intégration moindre par rapport à un logiciel propriétaire souvent plus esthétique .
Mais l'expérience nous a prouvé qu'en quelques dizaines de jours/hommes il était possible de mettre en œuvre Nagios avec tous les prérequis d'une production 24h/24.
Dans une société, il est souvent plus rapide et moins cher de passer par un intégrateur comme une SSLL ou une SSII que de perdre du temps à le mettre en place soi-même.
En effet, le gros du travail avec Nagios est de savoir où commencer, ce qu'il faut superviser, et ce qu'on peut oublier, et surtout de mettre en place un cadre de travail évolutif (télé-distribution). Une fois ce travail fait, il sera relativement simple de faire évoluer les choses, et d'ajouter ou enlever des sondes. Un des gros points non abordés serait la mise en place d'une supervision hiérarchisée, avec un site central recevant les notifications de sites distants, de façon à limiter la bande passante utilisée sur les liaisons louées.

9)Références

- Nagios : http://www.nagios.org/
- Locust : http://home.gna.org/locust/
- Documentation Nagios : http://www.nagios.org/docs/
- NSCIient et NRPE_NT : http://support.tsmgsoftware.com/
- APAN : http://apan.sf.net/
- perfparse : http://perfparse.sf.net/
- NagiosPHP : http://nagios-php.sourceforge.net/
- Kannel : http://www.kannel.org