Documentation de Nagios

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Nagios Version 2.x Documentation

www.nagios.org

Mise à jour le : 03/05/2006

[Table des matières]

A Propos de Nagios

Qu'est-ce que Nagios?

Nagios est une application permettant la surveillance système et réseau. Elle surveille les hôtes et services que vous spécifiez, vous alertant lorsque les systèmes vont mal et quand ils vont mieux.

Nagios a été prévu à l'origine pour fonctionner sous Linux [Anglais], toutefois il devrait fonctionner également sous les autres systèmes Unix.

Quelques-unes des fonctionnalités incluses dans Nagios:

- Surveillance des services réseaux (SMTP, POP3, HTTP, NNTP, PING, etc.)
- Surveillance des ressources des hôtes (charge processeur, utilisation des disques, etc.)
- Système simple de plugins permettant aux utilisateurs de développer facilement leurs propres v rifications de services.
- Parallélisation de la vérifications des services.
- Possibilité de définir la hiérarchie du réseau en utilisant des hôtes "parents", ce qui permet la détection et la distinction entre les hôtes qui sont à l'arrêt et ceux qui sont injoignables.
- Notifications des contacts quand un hôte ou un service a un problème et quand celui-ci est résolu (via email, pager, sms, ou par tout autre méthode définie par l'utilisateur)
- Possibilité de définir des gestionnaires d'évènements qui s'exécutent pour des évènements sur des hôtes ou des services, pour une résolution pro-active des problèmes
- Rotation automatique des fichiers log
- Support pour l'implémentation de la surveillance redondante des hôtes
- Interface web optionnelle, pour voir l'état actuel du réseau, notification et historique des problèmes, fichiers log, etc.

Pré-requis au niveau du Système

Le seul pré-requis pour le fonctionnement de Nagios est une machine fonctionnant sous Linux (ou une variante Unix) et un compilateur C - si vous désirez le compiler vous-même -. Il faudra également que TCP/IP soit configuré, car la plupart des vérifications de services seront faites par le réseau.

Vous *n'êtes pas obligé* d'utiliser les CGIs inclus dans Nagios. Toutefois, si vous décidez de les utiliser, vous devrez avoir les composants logiciels suivants installés…

1. Un serveur Web (de préférence Apache [Anglais]).
2. La librairie gd [Anglais] de Thomas Boutell's, version 1.6.3 or supérieure (requise par les CGIs statusmap et trends).

Licence

Nagios est distribué sous les termes de la GNU General Public License [Anglais] Version 2 comme publiée par la Free Software Foundation [Anglais]. Cette licence vous donne la permission légale de copier, distribuer et/ou modifier Nagios sous certaines conditions. Référez-vous au fichier 'LICENSE' inclus dans la distribution Nagios ou lisez la version en-ligne de la licence [Anglais] pour plus de détails.

[NdT] : Une traduction française de la GPL version 2 est disponible sur le site français de la Free Software Foundation.

[NdT] : Cette traduction est également distribuée sous les termes de la GNU Free Documentation License [Anglais].

Remerciements

Beaucoup de monde a contribué é Nagios, en signalant des bugs, en suggérant des améliorations, en écrivant des plugins, etc. Une liste de quelques-uns des nombreux contributeurs au développement de Nagios se trouvent sur http://www.nagios.org [Anglais].

Télécharger la derniére version

Vous pouvez trouver les nouvelles versions de Nagios sur http://www.nagios.org [Anglais].

Traduction

Notes de traduction

- Pour certains termes traduits, la version originale anglaise est spécifiée entre crochet : *méthode de délai inter-contrôles de service* [service_inter_check_delay_method] Ceci est utilisé lorsque les termes anglais sont des variables de configurations, des éléments dans l'interface Nagios, ou des anglicismes fréquements utilisés.
- Le "je" a été conservé de la documentation originale et fait bien entendu référence é Ethan Galstad.

La traduction de ce manuel est une mise é jour effectuée é partir de celle de Nagios version 1 crée par Pierre-Antoine Angelini, Johan Moreau et Christian Vanguers, elle m’me tirée de la documentation franéaise de NetSaint réalisée par Xavier Dusart.

Elle est issue d'un effort collectif des personnes suivantes :

Equipe de traduction

- Erwan Ben Souiden
- Xavier Dusart
- Hervé Nicol
- Patrick Proy

Equipe de relecture

- Hervé Bouchind'homme
- Erwan Ben Souiden
- René Coulet
- Philippe Delsol
- Johan Moreau

Quoi de neuf dans la version 2.0 ?

Important : assurez-vous d'avoir lu toute la documentation (et plus particuliérement les F.A.Q.) avant d'envoyer votre question aux listes de diffusion.

Journal des évolutions

Le journal des évolutions de Nagios est consultable en ligne sur http://www.nagios.org/changelog.php [Anglais] ou dans le fichier **Changelog** é la racine de la distribution du code source.

Problémes connus

Il y a un probléme connu qui affecte Nagios 2.0 sur les systémes FreeBSD. Ce probléme sera résolu, espérons-le, dans une version 2.x…

1. **FreeBSD et les threads [NdT : fils d'exécution]**. FreeBSD propose une implémentation native au niveau utilisateur des threads appelée 'pthread', et en option un portage 'linuxthreads' qui utilise les accroches du noyau [NdT : kernel hooks]. Des gars de Yahoo! ont signalé que l'utilisation de la bibliothéque pthread provoque l'arr't de Nagios lorsque la demande d'entrées-sorties est importante, ce qui cause la perte de certains résultats de contrles de services. Passer é la bibliothéque linuxthreads semble améliorer les choses, mais ne résoud pas le probléme. Le blocage se produit dans la fonction __pthread_acquire() de liblthread - le verrou ne peut jamais 'tre acquis. Cela se produit quand le thread principal est embranché [NdT : fork] pour exécuter un contrle actif. Au second embranchement pour créer le processus petit-fils, le petit-fils est créé par fork, mais ne retourne jamais é l'adaptateur [NdT : wrapper] de fork dans liblthread, parce qu'il est coincé dans __pthread_acquire(). Peut-'tre des utilisateurs de FreeBSD peuvent-ils contribuer é la résolution de ce probléme.

Changements et nouveautés

1. **Changements dans les macros** - Les macros ont subi une refonte majeure. Vous devrez mettre é jour la plupart de vos définitions de commandes pour vous adapter aux nouvelles macros. La plupart des macros sont maintenant disponibles en tant que variables d'environnement. De plus, des macros d'hte

et de service "é la demande" ont été ajoutées. Lisez la documentation sur les macros pour de plus amples informations.

2. **Changements dans les groupes d'htes**

   - **Suppression des escalades de groupes d'htes** - Cette fonction peut 'tre remplacée par le paramétre *hostgroup_name* de la définition du groupe d'htes.
   - **Changement du paramétre "members"** - La définition d'un groupe d'htes peut maintenant contenir plusieurs paramétres *members*, ce qui devrait rendre plus facile la saisie des fichiers de configuration lorsque vous avez un grand nombre d'htes membres. Vous pouvez aussi utiliser le paramétre *hostgroups* de la définition dhte pour définir é quel(s) groupe(s) dhtes appartient un hte en particulier.
   - **Changement dans les groupes de contacts** - Le paramétre *contact_groups* a été déplacé de la définition des groupes dhtes é celle des htes. Ceci afin d'tre cohérent avec la définition des contacts de services. Mettez é jour vos fichiers de configuration !
   - **Changement dans les permissions** - Les permissions daccés aux groupes dhtes dans les CGI ont été changées. Vous devez maintenant avoir accés é tous les htes membres du groupe dhtes pour avoir accés au groupe dhtes.

3. **Changements dans les htes**

   - **Contrle de fracheur de lhte** - le contrle de fracheur a été ajouté aux contrles dhtes. Cest le paramétre check_host_freshness qui définit ce comportement, associé au paramétre *check_freshness* de la défintion dhte.
   - **Commande OCHP** - Les contrles dhtes peuvent maintenant 'tre remontés, comme létaient les contrles de services. La commande OCHP est lancée pour tous les htes dont le paramétre *obsess_over_host* est activé dans la définition dhte.

4. **Changements dans le contrle des htes**

   - **Contrles ordonnancés réguliérement** - Vous pouvez maintenant ordonnancer le contrle régulier dhtes en utilisant le paramétre *check_interval* de la définition dhte. **NOTE :** Attention ! Vous devez utiliser parcimonieusement les contrles réguliers dhtes. Ils ne sont pas nécessaires au fonctionnement normal (les contrles é la demande sont déjé réalisés quand cest nécessaire) et peuvent dégrader les performances sils sont utilisés de maniére inappropriée. Vous 'tes prévenus.
   - **Contrles dhtes passifs** - Les contrles dhtes passifs sont maintenant possibles sils sont activés par le paramétre accept_passive_host_checks du fichier de configuration principal et le paramétre *accept_passive_host_checks* de la définition dhte. Les contrles passifs dhtes peuvent simplifier la mise en oeuvre dune supervision redondante ou répartie. **NOTE :** Certains problémes sont liés aux contrles dhtes passifs - pour en savoir plus lisez ceci.

5. **Changements dans la mémorisation**

   - **Mémorisation des informations dordonnancement** - Les information dordonnancement des contrles dhtes et de services (prochains moments de contrle) peuvent maintenant 'tre mémorisées entre les redémarrages du programme grôce au paramétre use_retained_scheduling_info.
   - **Mémorisation plus subtile** - Les valeurs de divers paramétres dhtes et de services qui peuvent 'tre mémorisées entre les redémarrages du programme ne le sont désormais que si elles ont été changées en cours dexécution par une commande externe. Cela rendra les choses plus claires pour ceux qui tentent de modifier les valeurs de paramétres dhtes et de services puis redémarrent Nagios, sattendant é constater des changements.
   - **Plus de choses mémorisées** - Il y a plus dinformations mémorisées entre les redémarrages du programme, y compris lhistorique de la détection doscillation. Youpi !

6. **Changements dans les informations étendues**

    - **Nouvel emplacement** - Les définitions dinformations étendues dhtes et de services sont maintenant stockées dans les fichiers de configuration des objets, avec les définitions dhtes, etc. Il en résulte que les définitions dinformation étendues sont maintenant analysées et validées par le démon Nagios avant le démarrage.
    - **Nouveaux paramétres** - Les définitions dinformations étendues dhtes et de services comportent maintenant deux nouveaux paramétres : *notes* et *action_url*.
7. **Changements dans linterpréteur Perl intégré**

    - **Emplacement de p1.pl** - Vous pouvez maintenant définir lemplacement du fichier daide de linterpréteur Perl intégré (p1.pl) grôce au paramétre p1_file.
8. **Changements dans les notifications**

    - **Notifications doscillation** - Les notifications sont maintenant envoyées au commencement et é la fin de loscillation dhtes et de services. Cette fonction est définie par la valeur *f* dans le paramétre *notification_options* des définitions de contacts, dhtes et de services.
    - **Meilleur algorithme** - Lalgorithme de notification a été légérement amélioré. Cela devrait éviter denvoyer des notifications de rétablissement alors quaucune notification de probléme navait été préalablement envoyée.
    - **Notifications de services** - Avant que les notifications de services ne soient envoyées, les dépendances de notification pour lhte sont désormais vérifiées. Si les notifications dhte ne sont pas jugées valables, les notifications pour le service ne seront pas envoyées non plus.
    - **Options descalade** - Les paramétres de période et détat ont été ajoutés aux escalades dhtes et de services. Cela vous permet de contrler plus finement lutilisation des escalades. Vous trouverez ici plus dinformations sur les escalades.
9. **Ajout des groupes de services** - Les groupes de services ont été ajoutés. Ils vous permettent de grouper des services pour laffichage dans les CGI et peuvent 'tre utilisés dans les définitions de dépendances de services et descalade de services pour faciliter la configuration.
10. **Ajout de larr't planifié déclenché** - La gestion de larr't planifié dit "déclenché" a été ajouté pour les htes et les services. Larr't planifié déclenché vous permet de définir un arr't planifié qui commence en m'me temps quun autre arr't planifié (trés utile pour définir larr't planifié des htes enfants dun hte dont le moment darr't planifié est variable). Vous trouverez ici plus dinformations sur larr't planifié.
11. **Nouvel outil de statistiques** - Un nouvel outil appelé nagiostats fait maintenant partie de la distribution de Nagios. Cest un outil en ligne de commande qui vous donne les statistiques pour un processus Nagios en cours dexécution. Il peut également produire des données compatibles avec MRTG, vous permettant de représenter graphiquement les statistiques. Vous trouverez ici plus dinformation sur lutilisation de cet outil.
12. **Supervision adaptative** - Vous pouvez maintenant changer certains paramétres propres aux contrles dhtes et de services (commandes de contrles, intervalle de contrle, nombre maximal dessais de contrle, etc.) en cours dexécution en envoyant les commandes externes appropriées. Ce type de supervision adaptative ne sera probablement pas dun grand usage pour la majorité des utilisateurs, mais il permet des trucs élégants. Vous trouverez ici plus dinformations sur la supervision adaptative.
13. **Changements dans les données de performance** - La méthode de traitement des données de performance a légérement changé. Vous pouvez maintenant traiter les données de performance en exécutant des commandes externes et/ou en écrivant dans les fichiers sans recompiler Nagios. Lisez la documentation sur les données de performance pour de plus amples informations.
14. **Abandon de la gestion native des bases de données** - La gestion native du stockage de divers types de données (états, mémorisation, commentaires, arr't planifié, etc.) dans MySQL et PostgreSQL a été abandonnée. Arr'tez de vous plaindre. Jespére que quelquun développera bientt une solution alternative grôce au nouveau gestionnaire dévénements. De plus, la gestion des bases de données

nétait pas trés bien implémentée et son abandon va rendre la tôche plus facile pour les nouveaux utilisateurs (une question de moins é se poser).

15. **API de distribution des événements [NdT : event broker]** - Une API [NdT : interface de programmation] a été créée pour permettre aux développeurs indépendants décrire des greffons qui sintégrent avec le coeur du démon Nagios. La documentation de lAPI de distribution dévénements sera créée quand le code de la version 2.x sera mr, et elle sera disponible sur le site web de Nagios.
16. **Changements divers**

    - **Toutes les commandes peuvent avoir des paramétres** - Tous les types de commandes (contrles dhtes, notifications, traitement des données de performance, gestionnaires dévénements, etc.) peuvent recevoir des paramétres (séparés du nom de la commande par le caractére **!**). Les paramétres sont remplacés dans la ligne de commande via les macros **$ARGx**.
    - **Récursivité dans le répertoire de configuration** - Nagios traite maintenant de maniére récursive tous les fichiers de configuration quil trouve dans les sous-répertoires des répertoires définis par le paramétre cfg_dir.
    - **Abandon de lancien style de fichiers de configuration** - La gestion de lancien style (sans templates) de fichiers de configuration des objets et des informations étendues a été abandonnée.
    - **Recherches plus rapides** - Les objets sont maintenant stockés dans une table de hachage chanée pour accélérer les recherches. Cela devrait grandement améliorer la performance des CGI.
    - **Threads de travail** - Quelques threads de travail ont été ajoutés pour artificiellement mettre en mémoire tampon les données pour le fichier de commandes externes et le tube [NdT : pipe] interne utilisé pour traiter les résultats de contrles de services. Cela devrait significativement améliorer les performances pour les grandes installations.
    - **Changements dans la journalisation** - Létat initial des htes et des services est maintenant journalisé légérement différemment. De plus, les états initiaux de tous les htes et services sont immédiatement journalisés aprés toute rotation de journaux. Cela devrait résoudre les problémes de "undetermined time" dans les CGI de disponibilité et de tendance.
    - **Mise en cache du fichier de configuration des objets** - Un fichier de cache des objets est maintenant créé par Nagios au démarrage. Il permet daccélérer un peu les CGI et vous permet de modifier vos fichiers de configuration des objets alors que Nagios sexécute, sans affecter laffichage des CGI.
    - **Limites initiales de contrle** - Vous pouvez définir des limites de temps dans lesquelles le contrle initial de tous les htes et services doit ’tre effectué aprés le démarrage de Nagios. Ces limites de temps sont définie par les paramétres max_host_check_spread et max_service_check_spread.
    - **Acquittements "collants"** - Vous pouvez maintenant déterminer si des acquittements dhtes ou de services sont "collants" [NdT : sticky] ou pas. Les acquittements collants suppriments les notifications jusqué ce quun hte ou un service se rétablisse complétement dans létat UP ou OK. Les acquittements non collants suppriment simplement les notifications jusqué ce quun hte ou un service change détat.
    - **Changements dans le contrle des grappes [NdT : clusters]** - La faéon de superviser des grappes de services et dhtes a changé et est plus fiable quelle nétait. Cela est d é lincorporation des macros é la demande et dun nouveau plugin (check_cluster2). Vous trouverez ici plus dinformations sur le contrle des clusters.
    - **Expressions réguliéres** - Lutilisation dexpressions réguliéres dans de nombreux paramétres des objets peut ’tre activée grôce aux paramétres use_regexp_matching et use_true_regexp_matching. Vous trouverez des informations sur quand et comment utiliser les expressions réguliéres dans la documentation sur les trucs et astuces des modéles.

- **Pseudo-états des services** - La gestion de pseudo-états de services qui étaient redondants a été supprimée du CGI détat. Cela affectera les URL codées en dur qui utilisent le paramétre du CGI servicestatustypes=X. Voyez include/statusdata.h pour la nouvelle liste des états de service que vous pouvez utiliser.
- **Changements dans le contrle de la fracheur** - Lalgorithme de contrle de la fracheur a légérement changé. Le contrle de fracheur ne se fera pas si linstant présent nest pas définit dans le paramétre *check_timeperiod* de lhte ou du service. De plus, les contrles de fracheur nauront plus lieu si les deux paramétres *check_interval* et *freshness_threshold* pour lhte ou le service valent zéro (0).

Informations aux débutants

Félicitation pour avoir choisi d'essayer Nagios! Nagios est vraiment puissant et flexible, mais n'est malheureusement pas trés facile pour les débutants. Pourquoi? Parce que l'installer et le configurer correctement demande beaucoup de travail. Ceci dit, si vous vous y collez et que vous arrivez é le faire fonctionner, vous ne voudrez plus vous en passer. :-) Voici quelques éléments importants é garder en mémoire pour ceux qui, parmi vous, 'tes des utilisateurs de Nagios de la premiére heure:

1. **Relax - Cela va prendre du temps.** N'espérez pas pouvoir compiler Nagios et le démarrer sans rien faire. Ce n'est pas si facile. C'est d'ailleurs assez difficile. Si vous ne voulez pas passer du temps é apprendre comment éa fonctionne et pensez voir les choses tourner facilement, ne prenez pas la peine d'utiliser ce logiciel. Au lieu de cela, payez quelqu'un pour surveiller votre réseau pour vous ou embauchez quelqu'un pour installer Nagios é votre place : -)
2. **Lire la documentation.** Nagios est suffisament difficile é configurer lorsque vous avez une bonne connaissance de son fonctionnement, c'est donc quasiment impossible si vous ne possédez pas ces connaissances . Rendez vous service et lisez avant d'essayer d'installer et faire tourner Nagios é l'aveuglette. Si vous 'tes du genre é ne pas prendre le temps de lire la documentation, vous vous renderez probablement compte que les autres aussi ne trouveront pas de temps pour vous aider quand vous aurez des problémes. RTFM. [**NdT** : **R**ead **T**he **F**ucking **M**anual (lisez ce foutu manuel)]
3. **Utiliser les fichiers d'exemples de configurations.** Des fichiers d'exemple de configurations sont fournit avec Nagios. Regardez les, modifiez les en fonction de vos besoins et testez les ! Les fichiers d'exemple ne sont que cela - des exemples. Il y a de trés fortes chances qu'ils ne fonctionnent pas pour vous sans modifications. Les fichiers d'exemple de configurations peuvent 'tre trouvés dans le sous-répertoire *sample-config/* de la distribution Nagios.
4. **Demandez de l'aide aux autres.** Si vous avez lu la documentation, passé en revu les fichiers d'exemple de configurations, et avez toujours des problémes, essayez d'envoyer un mail décrivant vos problémes é la liste de diffusion *nagios-users*. Avec tout le travail que j'ai [NdT:Ethan Galstad] é faire pour ce projet, je suis dans l'incapacité de répondre é la plupart des questions qui me sont envoyées directement, alors votre meilleure source d'aide sera la liste de diffusion. Si vous avez bien lu auparavant et que vous fournissez une bonne description de votre probléme, il y a de fortes chances que quelqu'un vous donne des pistes pour faire fonctionner les choses proprement. **[NdT** : Un forum francophone existe é l'adresse suivante : http://forums.opsyx.com/].

Installer Nagios

Important: Installer et configurer Nagios nécessite de s'impliquer. Vous ne pouvez pas juste compiler les binaires, exécuter le programme et vous asseoir. Il y a beaucoup de choses é configurer avant de pouvoir superviser quoique ce soit. Relaxez vous, prenez votre temps et lisez toute la documentation - vous allez en avoir besoin. Ok ? Alors commenéons

Devenir Root

Vous aurez besoin d'avoir les droits d'accés root pour installer Nagios comme décrit dans la documentation, vous allez créer des utilisateurs et des groupes, modifier la configuration du serveur web, etc. Utiliser la commande *su* pour vous logger en tant que root é partir d'un autre compte.

Obtenir la Deniére Version

Vous pouvez télécharger la derniére version de Nagios sur http://www.nagios.org/download.

Extraire la Distribution

Pour extraire la distribution Nagios, utiliser la commande suivante :

```
tar xzf nagios-version.tar.gz
```

Lorsque la commande aura été exécutée, vous trouverez un répertoire **nagios-*version*** dans votre répertoire courant. A l'intérieur de celui-ci, vous trouverez tous les fichiers qui constituent le noyau de la distribution Nagios.

Créer l'Utilisateur et le Groupe Nagios

Vous allez probablement vouloir faire tourner Nagios sous un utilisateur standard, donc créer un nouvel utiliser (et un groupe) avec la commande suivante (cette derniére peut varier suivant votre systéme d'exploitation):

```
adduser nagios
```

Créer le Répertoire d'Installation

Créer le répertoire de base o vous souhaitez installer Nagios

```
mkdir /usr/local/nagios
```

Changer le propriétaire du répertoire d'installation par l'utilisateur Nagios et le groupe créés plutt comme suit :

chown nagios.nagios /usr/local/nagios

Identifier l'Utilisateur Web

Vous allez probablement vouloir utiliser les commandes externes (comme les acquittements et les planifications d'arr’ts) é partir de l'interface web. Pour cela, vous avez besoin d'identifier l'utilisateur Web (typiquement *apache*, cependant cela peut ’tre différent pour votre systéme). Cette information se trouve dans les fichiers de configuration du serveur Web. La commande suivante permet de déterminer rapidement quel est l'utilisateur Apache (les chemins peuvent différer suivant votre systéme) :

```
grep "^User" /etc/httpd/conf/httpd.conf
```

Ajouter Un Groupe pour les Commandes

Maintenant vous allez créer un nouveau groupe dont les utilisateurs Web et Nagios feront partis. Appellons ce nouveau groupe '**nagcmd**' (vous pouvez l'appeller différemment si vous le souhaitez). Sur un Linux RedHat vous pouvez utiliser la commande suivante pour créer un groupe (ca peut ’tre différent sur d'autres systémes):

```
/usr/sbin/groupadd nagcmd
```

Ensuite, on ajoute au nouveau groupe créé les utilisateurs Web et Nagios avec les commandes suivantes (je suppose que *apache* et *nagios* sont les utilisateurs respectifs):

```
/usr/sbin/usermod -G nagcmd apache
/usr/sbin/usermod -G nagcmd
  nagios
```

Exécuter le Script Configure

Lancer le script configure comme suit pour initialiser les variables et créer un fichier Makefile(les deux derniéres options : --with-command-*xxx* sont optionnelles mais nécessaires si vous voulez utiliser les commandes externes).

```
./configure --prefix=prefix
  --with-cgiurl=cgiurl --with-htmurl=htmurl
  --with-nagios-user=someuser
  --with-nagios-group=somegroup
  --with-command-group=cmdgroup
```

- Remplacez *prefix* par le répertoire d'installation que vous avez créé plus haut (par défault */usr/local/nagios*).
- Remplacer *cgiurl* par l'URL que vous utiliserez pour accéder aux scripts CGIs (par défault */nagios/cgi-bin*). N'ajoutez PAS de slash (/) é la fin de l'URL.
- Remplacer *htmurl* par l'URL que vous utiliserez pour accéder l'interface principale de Nagios et la documentation (par défaut */nagios/*).
- Remplacer *someuser* par le nom de l'utilisateur qui possédera les fichiers installés(par défault *nagios*).
- Remplacer *somegroup* par le nom du groupe qui possédera les fichiers installés (par défault *nagios*).
- Remplacer *cmdgroup* par le nom du groupe qui fait tourner le serveur Web (par défault *nagios*).

Compiler les Binaires

Compilez Nagios et les CGIs avec la commande suivante:

```
make all
```

Installer les Binaires et les Fichiers HTML

Installez les binaires et les fichiers HTML (documentation et page web principale) avec la commande suivante :

```
make install
```

Installation du Script d'Initialisation

Si vous le souhaitez, vous pouvez aussi installer le script d'initialisation */etc/rc.d/init.d/nagios* avec la commande suivante :

```
make install-init
```

Vous pourrez ’tre obligé d'éditer ce script pour l'adapter é votre systéme d'exploitation et é Nagios en modifiant des chemins,etc.

Strcuture des répertoires et emplacement des fichiers

Placez vous é la racine du répertoire de base de l'installation de Nagios, avec la commande suivante :

```
cd /usr/local/nagios
```

Vous devriez voir 5 sous répertoires. Une bréve description du contenu de chacun de ces répertoires est donnée dans le tableau ci-dessous.

Sous-Répertoire	Contenus
bin/	Ensemble des programmes Nagios
etc/	Les fichiers de configurations principaux, des ressources, des objets, et des CGI doivent ’tre mis ici
sbin/	CGIs
share/	Fichiers HTML (pour l'interface web et la documentation en ligne)
var/	Répertoire vide pour les fichiers de log, les fichiers de status, les les fichiers de retention, etc.
var/archives	Répertoire vide pour les logs archivés
var/rw	Répertoire vide pour le fichier de commandes externes

Installation des Plugins

Pour que Nagios vous soit utile, il va falloir télécharger et installer quelques plugins. Les plugins sont habituellement installés dans le répertoire **libexec/** de votre installation de Nagios (i.e. */usr/local/nagios/libexec*). Les plugins sont des scripts ou des binaires qui réalisent les contrles des services et des htes pour la supervision. Vous pouvez récupérer la derniére version des plugins sur la page de téléchargement de Nagios ou directement sur la la page SourceForge du projet.

Configurer l'Interface Web

Vous allez probablement vouloir utiliser l'interface web, vous devrez donc lire les instructions pour configurer l'interface Web , les accés, etc. Puis

Configuration de Nagios

Maintenant, tout est compilé et installé, mais vous n'avez toujours pas configuré Nagios, ni défini les objets (htes, services, etc..) qui doivent ’tre supervisés. Des informations sur la configuration de Nagios et la définition des objets peuvent ’tre trouvées ici. Il y a beaucoup de choses é configurer mais ne vous laissez pas

décourager - ca vaut le coup.

Configurer l'Interface Web

Notes

Dans ce qui suit, je supposerai que vous utilisez le serveur Web Apache sur votre machine. Si vous utilisez un autre serveur Web, vous devrez faire les changements appropriés. Je supposerai également que vous utilisez */usr/local/nagios* comme préfixe d'installation.

Exemple de configuration

Un fichier de configuration d'Apache est créé lorsque vous exécutez le script de configuration - vous pouvez trouver ce fichier (appelé *httpd.conf*) dans le sous répertoire *sample-config/* de la distribution Nagios. Vous aurez besoin d'ajouter le contenu de ce fichier é votre configuration Apache avant d'accéder é l'interface Web. Les instructions ci-dessous détaillent comment ajouter manuellement la configuration appropriée é Apache.

Configuration des Alias et des Options des Repertoires pour l'Interface Web

Premiérement vous aurez besoin de créer les entrées appropriées dans votre fichier de configuration du serveur Web. Ajouter les lignes suivantes é votre fichier de configuration Web (i.e. **httpd.conf**) en les changeant pour les faires correspondrent aux répertoires de votre systémes.

```
        ScriptAlias /nagios/cgi-bin /usr/local/nagios/sbin

<Directory "/usr/local/nagios/sbin">
    Options ExecCGI
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile /usr/local/nagios/etc/htpasswd.users
    Require valid-user
</Directory>

Alias /nagios /usr/local/nagios/share

<Directory "/usr/local/nagios/share">
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile /usr/local/nagios/etc/htpasswd.users
    Require valid-user
</Directory>
```

Note: L'installation par défaut de Nagios espére trouver les fichiers HTML sur **http://yourmachine/nagios/** et les CGIs sur **http://yourmachine/nagios/cgi-bin/**. Ces emplacements peuvent ’tre modifiés en utilisant les options **--with-htmurl** et **--with-cgiurl** du script de configuration.

Important! Si vous installez Nagios sur un systéme multi-utilisateurs, vous pourriez souhaiter utiliser CGIWrap pour apporter un niveau de sécurité supérieur entre les CGIs et le fichier de commandes externes. Si vous utilisez CGIWrap, l'option ScriptAlias que vous utiliserez sera probablement différente de celle indiquée plus haut. Plus d'informations é ce sujet ici.

Redémarrer le Serveur Web

Une fois que vous aurez fini de modifier le fichier de configuration Apache, vous aurez besoin de redémarrer le serveur web avec une commande comme celle ci

```
/etc/rc.d/init.d/httpd restart
```

Configuration de l'authentification Web

Une fois que vous avez installé proprement l'interface web, vous aurez besoin de spécifier qui peut avoir accés é l'interface Web de Nagios. Suivez ces instructions pour faire cela.

Vérifiez vos modifications

N'oubliez pas de vérifier si les changements que vous faites sur Apache fonctionnent. Vous devriez ’tre capable de pointer votre navigateur sur **http://votremachine/nagios/** et obtenir l'interface Nagios. Les CGIs peuvent ne pas afficher d'informations mais ceci sera résolu une fois que vous aurez tout configuré et lancé Nagios.

Configurer Nagios

Survol de la configuration

Il va falloir créer et éditer plusieurs fichiers de configuration avant de pouvoir surveiller quoique ce soit. Ces fichiers sont décrits ci-dessous.

Fichier de configuration principal

Le fichier de configuration principal (par défaut, */usr/local/nagios/etc/nagios.cfg*) contient un certain nombre de directives qui affectent la maniére dont Nagios fonctionne. Ce fichier est lu par le processus Nagios et par les CGIs. C'est le premier fichier que vous allez créer et éditer.

La documentation du fichier de configuration principal se trouve ici.

Un fichier de configuration principal est généré automatiquement é titre d'exemple quand vous lancez le script **configure** avant de compiler les programmes. Vous le trouverez soit dans la distribution, soit sous le répertoire /etc de votre installation. Quand vous installez les exemples de fichiers de configuration avec la commande **make install-config**, un exemple de fichier de configuration est copié dans votre répertoire de paramétrage (généralement */usr/local/nagios/etc*). Par défaut, son nom est **nagios.cfg**.

Fichier de configuration des ressources

Les fichiers des ressources sont utilisés pour stocker les macros définies par les utilisateurs. Ces fichiers peuvent aussi contenir d'autres informations (telles que la configuration des connexions de la base de données), bien que ceci dépende de la maniére dont vous aurez compilé Nagios. L'avantage de ces fichiers est de pouvoir y mettre des données sensibles de configuration qui ne seront pas accessibles é travers les CGIs.

Vous pouvez définir un ou plusieurs fichiers de ressources avec la directive resource_file dans le fichier de configuration principal.

Fichier de définition des objets

Le fichier de définition des objets définit les htes, services, groupes d'htes, contacts, groupes de contacts, commandes, etc C'est lé que vous définissez les choses que vous souhaitez surveiller et comment vous désirez le faire.

La documentation du fichier de configuration des objets se trouve ici.

Fichier de configuration des CGI

Le fichier de configuration des CGIs (par défaut, */usr/local/nagios/etc/cgi.cfg*) contient un certain nombre de directives qui affectent le mode de fonctionnement des CGIs.

La documentation du fichier de configuration des CGI se trouve ici.

Un fichier de configuration des CGI est généré automatiquement é titre d'exemple quand vous lancez le script **configure** avant de compiler les programmes. Quand vous installez les exemples de fichiers de configuration avec la commande **make install-config**, un exemple de fichier de configuration des CGI est copié dans le m’me répertoire que les fichiers de configuration principal et des htes (généralement */usr/local/nagios/etc*). Par défaut, son nom est **cgi.cfg**.

Fichier de configuration des informations étendues.

Le fichier de configuration des informations étendues est utilisé pour définir des informations supplémentaires pour les htes et les services qui doivent ’tre utilisées par les CGIs. C'est lé que vous définissez, par exemple, les coordonnées de dessin, les "zolies zicones", etc

La documentation pour ce fichier se trouve ici.

Options du fichier de configuration principal

Notes

Quand vous créez ou modifiez des fichiers de configuration, n'oubliez pas que :

1. Les lignes qui commencent avec le caractére '#' sont considérées comme des commentaires et ne sont donc pas traitées
2. Les noms des variables doivent commencer au début de la ligne - ne mettez pas d'espace avant le nom
3. Les noms des variables respectent la casse (majuscule/minuscule)

Configuration d'exemple

Un fichier de configuration d'exemple peut ’tre créé en lanéant la commande '**make config**'. Par défaut, le fichier de configuration principal de Nagios s'appelle **nagios.cfg** - vous le trouverez dans la distribution de Nagios ou dans le sous-répertoire etc/ de votre installation. (*c.a.d /usr/local/nagios/etc/)*

Index

Fichier journal

Format : **log_file=<nom_de_fichier>**

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

Cette variable détermine le chemin d'accés au fichier journal principal de Nagios. Elle doit ’tre la premiére variable définie dans le fichier de configuration, car Nagios essaiera d'enregistrer é cet endroit les erreurs découvertes dans le reste du fichier. Si vous avez activé la rotation des journaux, ce fichier sera automatiquement archivé et remplacé chaque heure, jour, semaine et mois.

Fichier de configuration des objets

Format : **cfg_file=<nom_de_fichier>**

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

Cette variable détermine le fichier de configuration des objets que Nagios doit utiliser pour la supervision. Ce fichier contient les définitions pour les htes, groupes d'htes, contacts groupes de contacts, services, commandes etc Vous pouvez éclater votre fichier de configuration des objets en plusieurs fichiers que vous inclurez, pour leur traitement, en ajoutant une directive **cfg_file=** pour chacun d'eux.

Répertoire des fichiers de configuration des objets

Format : **cfg_dir=<nom_de_répertoire>**

Exemple : **cfg_dir=/usr/local/nagios/etc/commands**
cfg_dir=/usr/local/nagios/etc/services
cfg_dir=/usr/local/nagios/etc/hosts

Cette variable permet de définir un répertoire qui contiendra les fichiers de configuration des objets que Nagios doit utiliser pour la supervision. Tous les fichiers dans ce répertoire avec une extension **.cfg** seront traités comme des fichiers de configuration d'objets. De plus, Nagios va récursivement parcourir tous les fichiers **.cfg** des sous-répertoires de ce répertoire. Vous pouvez distribuer vos fichiers de configurations en plusieurs répertoires que vous inclurez, pour leur traitement, par des directives **cfg_dir=** pour chacun d'eux.

Fichier de cache des objets

Format: **object_cache_file=<nom_de_fichier>**

Exemple : **object_cache_file_file=/usr/local/nagios/var/objects.cache**

Cette variable est utilisée pour indiquer un l'emplacement d'un fichier qui contiendra une copie en cache des définitions des objets. Ce fichier cache est (re)créé é chaque (re)démarrage de Nagios et est utilisé par les CGI. Il permet d'accélérer le cache des fichiers de configuration pour les CGI, et vous permet de modifier les fichiers de configuration des objets, pendant que Nagios tourne, sans que l'affichage des CGI soit modifié.

Fichier de ressources

Format: **resource_file=<nom_de_fichier>**

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

Le fichier de ressources optionnel qui contient des définitions de macros du type $USERn$. Les macros $USERn$ permettent de stocker des noms d'utilisateurs, des mots de passe, et les éléments couramment utilisés dans les commandes (comme les chemins d'accés). Les CGI ne *lisent pas* les fichiers de ressources, si bien que vous pouvez y mettre des droits d'accés restrictifs (600 ou 660) pour protéger les données sensibles. Vous pouvez inclure de nombreux fichiers de ressources en ajoutant des directives **resource_file** au fichier de configuration principal - Nagios les traitera tous. Référez-vous au fichier resource.cfg d'exemple situé é la racine de la distribution de Nagios, pour voir comment utiliser les macros $USERn$.

Fichier temporaire

Format : **temp_file=<nom_de_fichier>**

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

C'est un fichier que Nagios crée périodiquement durant la mise é jour des commentaires de données, des données d'état etc Il est supprimé quand il n'est plus nécessaire.

Fichier d'états (journal des états)

Format : **status_file=<nom_de_fichier>**

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

C'est le fichier utilisé par Nagios, pour stocker l'état courant de tous les services supervisés. L'état de tous les htes associés avec ces services, est également enregistré dans ce fichier. Ce fichier est utilisé par le CGI d'état pour afficher l'état courant de la supervision dans l'interface web. Les CGI doivent avoir le droit d'accéder en lecture é ce fichier, pour fonctionner correctement. Ce fichier est supprimé é l'arr't de Nagios, et recréé au démarrage.

Option d'agrégation des changements d'état

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

Exemple : **aggregate_status_updates=1**

Cette option détermine, si Nagios doit agréger les données de changement d'état des htes, services, et programmes. Par défaut, les données d'état sont immédiatement mises é jour é chaque contrle d'un service ou d'un hte. Ceci peut causer une charge CPU importante, et de nombreuses entrées/sorties disque, si vous contrlez de nombreux services. Si vous voulez que Nagios ne mette é jour les données d'état (dans le journal des états), que toutes les quelques secondes (tel que défini par la variable status_update_interval), activez cette option. Si vous voulez des mises é jour immédiates, désactivez-la. Les valeurs possibles de cette variable sont :

- 0 = Désactiver les mises é jour agrégées (par défaut)
- 1 = Activer les mises é jour agrégées

Intervalle de mise é jour des états agrégés

Format: **status_update_interval=<secondes>**

Exemple : **status_update_interval=15**

Cette variable, détermine la fréquence (en secondes), é laquelle Nagios mettra é jour les données d'état dans le journal des états. L'intervalle minimal est de cinq secondes. Si vous avez désactivé les mises é jour agrégées (grôce é l'option aggregate_status_updates), cette variable est sans effet.

Nagios

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

Exemple : **nagios_user=nagios**

Ceci détermine quel utilisateur doit 'tre le propriétaire du processus de Nagios. Aprés le démarrage du programme, et avant toute supervision, Nagios abandonnera ses priviléges effectifs, et se lancera sous cet

utilisateur. Vous pouvez donner un nom d'utilisateur ou un UID.

Groupe de Nagios

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

Exemple : **nagios_group=nagios**

Ceci détermine quel groupe doit ’tre propriétaire du processus de Nagios. Aprés le démarrage du programme, et avant toute supervision, Nagios abandonnera ses priviléges effectifs et se lancera sous ce groupe. Vous pouvez donner un nom de groupe ou un GID.

Option des notifications

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

Exemple : **enable_notification=1**

Cette option détermine si Nagios envoie ou non une notification quand il (re)démarre. Si cette option est désactivée, Nagios n'enverra aucune notification, quel que soit l'hte ou le service. Note : si vous avez activé la mémorisation d'état, Nagios ignorera ce paramétre au (re)démarrage, et utilisera sa derniére valeur connue (telle qu'elle est stockée dans le fichier de mémorisation d'état), *é moins que* vous ne désactiviez l'option use_retained_program_state. Si vous voulez changer cette variable alors que la mémorisation d'état est activée, (ainsi que l'option use_retained_program_state), vous devrez passer par la commande externe appropriée, ou la changer é travers l'interface web. Les valeurs possibles de cette variable sont :

- 0 = notification désactivée
- 1 = notification activée (par défaut)

Option d'exécution des contrles de service

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

Exemple : **execute_service_checks=1**

Cette option détermine, si Nagios effectuera les contrles des services lorsqu'il (re)démarrera. Si cette option est désactivée, Nagios n'effectuera aucun contrle de service, et restera dans un mode "de sommeil" (il peut quand m’me recevoir les contrles passifs é moins qu'ils ne soient désactivés). Cette option est surtout utile pour définir des serveurs de supervision de secours, comme l'explique la documentation sur la redondance, ou pour mettre en place un environnement de supervision répartie. Note : si vous avez activé la mémorisation d'état, Nagios ignorera ce paramétre au (re)démarrage, et utilisera sa derniére valeur connue (telle qu'elle est stockée dans le fichier de mémorisation d'état), *é moins que* vous ne désactiviez l'option use_retained_program_state. Si vous voulez changer cette variable, alors que la mémorisation d'état est activée (ainsi que l'option use_retained_program_state), vous devrez passer par la commande externe appropriée, ou la changer é travers l'interface web. Les valeurs possibles de cette variable sont :

- 0 = Ne pas exécuter les contrles de service
- 1 = Exécuter les contrles de service (par défaut)

Option d'acceptation des contrles passifs de service

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

Exemple : **accept_passive_service_checks=1**

Cette option détermine si Nagios accepte les contrles passifs de service quand il (re)démarrera. Si cette option est désactivée, Nagios n'acceptera aucun contrle passif de service. Note : si vous avez activé la mémorisation d'état, Nagios ignorera ce paramétre au (re)démarrage et utilisera sa derniére valeur connue (telle qu'elle est stockée dans le fichier de mémorisation d'état), *é moins que* vous ne désactiviez l'option use_retained_program_state. Si vous voulez changer cette variable alors que la mémorisation d'état est activée (ainsi que l'option use_retained_program_state), vous devrez passer par la commande externe appropriée ou la changer é travers l'interface web. Les valeurs possibles de cette variable sont :

- 0 = Ne pas accepter les contrles passifs de service
- 1 = Accepter les contrles passifs de service (par défaut)

Option d'exécution des contrles d'htes

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

Exemple : **execute_host_checks=1**

Cette option détermine si Nagios effectuera les contrles des htes - é la demande ou de maniére réguliére - lorsqu'il (re)démarrera. Si cette option est désactivée, Nagios n'effectuera aucun contrle sur les htes et restera dans un mode "de sommeil" (il peut quand m'me recevoir les contrles passifs d'htes é moins qu'ils ne soient désactivés). Cette option est surtout utile pour définir des serveurs de supervision de secours, comme l'explique la documentation sur la redondance, ou pour mettre en place un environnement de supervision répartie. Note : si vous avez activé la mémorisation d'état, Nagios ignorera ce paramétre au (re)démarrage, et utilisera sa derniére valeur connue (telle qu'elle est stockée dans le fichier de mémorisation d'état), *é moins que* vous ne désactiviez l'option use_retained_program_state. Si vous voulez changer cette variable alors que la mémorisation d'état est activée (ainsi que l'option use_retained_program_state), vous devrez passer par la commande externe appropriée, ou la changer é travers l'interface web. Les valeurs possibles de cette variable sont :

- 0 = Ne pas exécuter les contrles d'htes
- 1 = Exécuter les contrles d'htes (par défaut)

Option d'acceptation des contrles passifs d'hte

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

Exemple : **accept_passive_host_checks=1**

Cette option détermine si Nagios accepte les contrles passifs d'hte quand il (re)démarrera. Si cette option est désactivée, Nagios n'acceptera aucun contrle passif d'hte. Note : si vous avez activé la mémorisation d'état, Nagios ignorera ce paramétre au (re)démarrage, et utilisera sa derniére valeur connue (telle qu'elle est stockée dans le fichier de mémorisation d'état), *é moins que* vous ne désactiviez l'option use_retained_program_state. Si vous voulez changer cette variable alors que la mémorisation d'état est activée (ainsi que l'option use_retained_program_state), vous devrez passer par la commande externe appropriée, ou la changer é travers l'interface web. Les valeurs possibles de cette variable sont :

- 0 = Ne pas accepter les contrles passifs d'hte
- 1 = Accepter les contrles passifs d'hte (par défaut)

Option de gestion d'événement

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

Exemple : **enable_event_handlers=1**

Cette option détermine si Nagios activera les gestionnaires d'événement quand il (re)démarrera. Si cette option est désactivée, Nagios ne lancera aucun gestionnaire d'événement lié aux htes ou aux services. Note : si vous avez activé la mémorisation d'état, Nagios ignorera ce paramétre au (re)démarrage, et utilisera sa derniére valeur connue (telle qu'elle est stockée dans le fichier de mémorisation d'état), *é moins que* vous ne désactiviez l'option use_retained_program_state. Si vous voulez changer cette variable alors que la mémorisation d'état est activée (ainsi que l'option use_retained_program_state), vous devrez passer par la commande externe appropriée ou la changer é travers l'interface web. Les valeurs possibles de cette variable sont :

- 0 = Désactiver les gestionnaires d'événements
- 1 = Activer les gestionnaires d'événements (par défaut)

Méthode de rotation du journal

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

Exemple : **log_rotation_method=d**

C'est la méthode de rotation que vous voulez que Nagios utilise pour le fichier journal. Les valeurs possibles de cette variable sont :

- n = Aucune [None] (ne pas effectuer de rotation sur le journal - par défaut)
- h = Toutes les heures [Hourly] (effectuer une rotation du journal au début de chaque heure)
- d = Tous les jours [Daily] (effectuer une rotation du journal é minuit chaque jour)
- w = Toutes les semaines [Weekly] (effectuer une rotation du journal é minuit le samedi)
- m = Tous les mois [Monthly] (effectuer une rotation du journal é minuit le dernier jour du mois)

Chemin d'accés aux archives du journal

Format : **log_archive_path=<chemin_d_accés>**

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

C'est le répertoire o Nagios doit enregistrer les fichiers journaux ayant fait l'objet d'une rotation. Cette option est ignorée si vous n'avez pas activé la rotation du journal.

Option de contrle des commandes externes

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

Exemple : **check_external_commands=1**

Cette option détermine si Nagios va vérifier le contenu du fichier des commandes é la recherche de commandes é exécuter. Cette option doit ’tre activée si vous avez prévu d'utiliser le CGI de commande pour envoyer des commandes via l'interface web. Des programmes tiers peuvent également envoyer des commandes é Nagios en écrivant dans le fichier de commande, sous réserve que les droits nécessaires é l'accés au fichier ait été donnés, comme le souligne cette FAQ. Vous trouverez plus d'informations sur les commandes externes ici.

- 0 = Ne pas vérifier les commandes externes (par défaut)
- 1 = Vérifier les commandes externes

Intervalle entre les contrles des commandes externes

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

Exemple : **command_check_interval=1**

Si vous spécifiez un nombre suivi d'un "s" (par exemple 30s), c'est le nombre de secondes qui séparera deux contrles de commandes externes. Si vous ne mettez pas de "s", c"est le nombre d'"unités de temps" é laisser entre deux contrles de commandes externes. Si vous n'avez pas modifié la valeur de la durée de l'intervalle qui est par défaut de 60, ce nombre représente des minutes.

Note : en mettant cette valeur é **-1,** Nagios contrlera les commandes externes aussi souvent que possible. Chaque fois que Nagios contrle les commandes externes, il lit et traite toutes les commandes présentes dans le fichier de commandes avant de passer é d'autres tôches. Vous trouverez plus d'informations sur les commandes externes ici.

Fichier de commandes externes

Format : **command_file=<nom_de_fichier>**

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

C'est le fichier que Nagios lit é la recherche de commandes externes. Le CGI de commande écrit les commandes dans ce fichier. Des programmes tiers peuvent écrire dans ce fichier sous réserve qu'ils aient les droits d'accés comme il est décrit ici. Le fichier de commandes externes est implémenté sous forme d'un tube nommé [named pipe] (FIFO), qui est créé quand Nagios démarre et supprimé lorsqu'il s'arr’te. Pour plus d'informations sur les commandes externes, lisez ceci.

Fichier des temps d'indisponibilité programmés des htes et services

Format: **downtime_file=<file_name>**

Exemple : **downtime_file=/usr/local/nagios/var/downtime.dat**

C'est le fichier que Nagios va utiliser pour enregistrer les informations d'indisponibilité programmée des htes et services. Les commentaires peuvent ’tre consultés et ajoutés pour les htes et les services é travers le CGI d'informations complémentaires.

Fichier de commentaire

Format : **comment_file=<nom_de_fichier>**

Exemple : **comment_file=/usr/local/nagios/var/comment.dat**

C'est le fichier utilisé par Nagios pour enregistrer les commentaires sur les services et les htes. Vous pouvez visualiser et ajouter des commentaires sur les services et les htes grôce au CGI d'informations complémentaires.

Fichier verrou

Format : **lock_file=<nom_de_fichier>**

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

C'est l'emplacement du fichier verrou qu'utilise Nagios quand il est lancé en tant que démon (i.e. démarré avec l'argument -d). Ce fichier contient l'identifiant du processus (PID) de Nagios.

Option de mémorisation de l'état

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

Exemple : **retain_state_information=1**

Cette option détermine si Nagios doit mémoriser l'état des htes et des services entre deux démarrages. Si vous activez cette option, vous devez donnez une valeur é la variable fichier de mémorisation de l'état. Une fois l'option activée, Nagios enregistrera toutes les informations concernant l'état des htes et des services avant de s'arr'ter (ou de redémarrer) et il relira les informations d'état préalablement enregistrées quand il redémarrera.

- 0 = Ne pas mémoriser les informations d'état (par défaut)
- 1 = Mémoriser les informations d'état

Fichier de mémorisation de l'état

Format : **state_retention_file=<nom_de_fichier>**

Exemple : **state_retention_file=/usr/local/nagios/var/status.dat**

C'est le fichier utilisé par Nagios pour mémoriser l'état des htes et des services entre les redémarrages. Au démarrage, Nagios positionnera l'état initial des services et des htes é partir des informations contenues dans ce fichier, puis commencera la supervision. Ce fichier est supprimé dés sa lecture effectuée. Pour que Nagios mémorise l'état des htes et des services, vous devez activer l'option de mémorisation de l'état.

Intervalle de mise é jour des états mémorisés

Format : **retention_update_interval=<minutes>**

Exemple : **retention_update_interval=60**

Cette variable détermine la fréquence (en minutes) é laquelle Nagios va automatiquement sauvegarder les données de mémorisation en situation normale. Si vous donnez une valeur de 0, Nagios ne sauvegardera pas les données mémorisées é intervalles réguliers, mais avant de s'arr'ter ou de redémarrer. Si vous avez désactivé la mémorisation des états (grôce é l'option retain_state_information), cette variable est sans effet.

Option d'utilisation des états mémorisés du programme

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

Exemple : **use_retained_program_state=1**

Cette option détermine si Nagios positionnera diverses variables d'état du programme é partir des valeurs enregistrées dans le fichier de mémorisation. Parmi ces variables d'état du programme, normalement sauvegardées par delé les redémarrages du programme si la mémorisation d'état est activée, on trouve les notifications , la détection d'oscillation, l'activation des gestionnaires d'événements, l'exécution des contrles de services, ou l'acceptation des contrles de service passifs. Si vous n'avez pas activé la mémorisation d'état, cette

option est sans effet.

- 0 = Ne pas utiliser la mémorisation des états du programme
- 1 = Utiliser la mémorisation des états du programme (par défaut)

é

Option d'utilisation des états mémorisés de l'ordonnanceur

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

Exemple : **use_retained_scheduling_info=1**

Cette option détermine si Nagios va garder les informations d'ordonnancement (prochain lancement de vérification pour chaque service), lorsqu'il redémarre. Si vous ajoutez un grand nombre (ou pourcentage) de services ou d'htes, je vous recommande de désactiver cette option lorsque vous redémarrez Nagios, car ceci peut biaiser la répartition initiale des services. Dans le cas contraire, vous voudrez probablement laisser cette option active.

- 0 = Ne pas mémoriser les états de l'ordonnanceur
- 1 = Mémoriser les états de l'ordonnanceur (par défaut)

Option de journalisation dans Syslog

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

Exemple : **use_syslog=1**

Cette option détermine si les messages doivent ’tre journalisés via l'utilitaire systéme Syslog. Les valeurs possibles sont :

- 0 = Ne pas utiliser Syslog
- 1 = Utiliser Syslog

Option de journalisation des notifications

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

Exemple : **log_notifications=1**

Cette variable détermine si les messages de notification sont journalisés ou non. Si vous avez de nombreux contacts ou des problémes fréquents sur les services, le fichier journal va rapidement grossir. Utilisez cette option pour éviter de journaliser les notifications faites aux contacts.

- 0 = Ne pas journaliser les notifications
- 1 = Journaliser les notifications

Option de journalisation de tentatives de contrle de service

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

Exemple : **log_service_retries=1**

Cette option détermine si les tentatives répétées de contrle d'un service sont journalisées. Ces tentatives ont lieu lorsqu'un service retourne un état différent de OK, mais que vous avez configuré Nagios pour essayer plus d'une fois avant de considérer cela comme une erreur. Les services dans cette situation sont dits en état "soft". La journalisation des tentatives de contrle permet de déboguer Nagios ou de tester les gestionnaires d'événements.

- 0 = Ne pas journaliser les tentatives de contrle de service
- 1 = Journaliser les tentatives de contrle de service

Option de journalisation de tentatives de contrle d'hte

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

Exemple : **log_host_retries=1**

Cette option détermine si les tentatives répétées de contrle d'un hte sont journalisées. La journalisation des tentatives de contrle permet de déboguer Nagios ou de tester les gestionnaires d'événements.

- 0 = Ne pas journaliser les tentatives de contrle d'hte
- 1 = Journaliser les tentatives de contrle d'hte

Option de journalisation des gestions d'événement

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

Exemple : **log_event_handlers=1**

Cette option détermine si les gestions d'événements liés aux htes ou aux services sont journalisées. Les gestionnaires d'événements sont des commandes optionnelles qu'on peut lancer lors du changement d'état d'un hte ou d'un service. La journalisation des gestions d'événements permet de déboguer Nagios ou de tester les scripts de gestion d'événements.

- 0 = Ne pas journaliser les gestions d'événements
- 1 = Journaliser les gestions d'événements

Option de journalisation des états initiaux

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

Exemple : **log_initial_states=1**

Cette variable détermine si Nagios forcera la journalisation de tous les états initiaux des htes et des services, m'me si leur état est OK. Les états initiaux ne sont normalement journalisés que s'il y a un probléme lors du premier contrle. Cette option peut se révéler utile si vous utilisez une application tierce qui lit le journal pour en tirer des statistiques é long terme pour les htes et services.

- 0 = Ne pas journaliser les états initiaux (par défaut)
- 1 = Journaliser les états initiaux

Option de journalisation des commandes externes

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

Exemple : **log_external_commands=1**

Cette variable détermine si Nagios journalisera les commandes externes reéues via le fichier des commandes externes. Note : cette option ne détermine pas si les contrles passifs de service (qui sont une variante des commandes externes) sont journalisés. Pour définir la journalisation des contrles passifs de service, utilisez l'option de journalisation des contrles passifs de service.

- 0 = Ne pas journaliser les commandes externes
- 1 = Journaliser les commandes externes (par défaut)

Option de journalisation des contrles passifs

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

Exemple : **log_passive_checks=1**

Cette variable détermine si Nagios journalisera les contrles passifs d'hte ou de service reéus via le fichier de commandes externes. Si vous mettez en place un environnement de supervision réparti ou si vous souhaitez utiliser fréquemment un grand nombre de contrles passifs, vous pouvez désactiver cette option pour éviter au journal de trop grossir.

- 0 = Ne pas journaliser les contrles passifs
- 1 = Journaliser les contrles passifs (par défaut)

Gestionnaire global des événements relatifs aux htes

Format : **global_host_event_handler=<commande>**

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

Cette option détermine un gestionnaire d'événement appelé é chaque changement d'état d'un hte. Il s'exécute juste avant le gestionnaire d'événement particulier é l'hte que vous avez précisé de maniére optionnelle dans la définition de l'hte. L'argument *command*e est le nom court d'une définition de commande qui se trouve dans votre fichier de définition des objets. Le temps d'exécution maximal de cette commande est déterminé par la variable event_handler_timeout. Vous trouverez plus d'informations sur les gestionnaires d'événements ici.

Gestionnaire global des événements relatifs aux services

Format : **global_service_event_handler=<commande>**

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

Cette option détermine un gestionnaire d'événement appelé é chaque changement d'état d'un service. Il s'exécute juste avant le gestionnaire d'événement particulier é l'hte que vous avez précisé de maniére optionnelle dans la définition du service. L'argument *commande* est le nom court d'une définition de commande qui se trouve dans votre fichier de définition des objets. Le temps d'exécution maximal de cette commande est déterminé par la variable event_handler_timeout. Vous trouverez plus d'informations sur les gestionnaires d'événements ici.

Temps de sommeil entre les contrles

Format : **sleep_time=<secondes>**

Exemple : **sleep_time=1**

C'est le nombre de secondes pendant lequel Nagios va sommeiller avant de vérifier si le prochain contrle de service ou d'hte en file d'attente doit ’tre exécuté. Notez que Nagios ne s'endormira qu'aprés avoir "liquidé" les contrles de services en retard dans la file.

Méthode de délai inter-contrles de services

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

Exemple : **service_inter_check_delay_method=s**

Cette option détermine comment les contrles de service sont initialement répartis dans la file d'attente. L'option de calcul "débrouillard" [smart] du délai (par défaut), demande é Nagios de calculer un intervalle moyen entre les contrles, et d'ordonnancer les contrles initiaux de tous les services é cet intervalle, ce qui permet d'éviter les pics d'utilisation du processeur. Il *n'est pas* recommandé d'utiliser la méthode "sans délai" [no delay] é moins que vous ne vouliez tester la parallélisation des contrles de service. En effet, cette méthode ordonnance tous les contrles en m’me temps. L'exécution de tous les contrles en paralléle va provoquer d'importants pics d'utilisation du processeur. Vous obtiendrez plus d'informations sur la faéon dont cette variable affecte l'ordonnancement des contrles de service ici. Ses valeurs possibles sont :

- n = Ne pas utiliser de délai [no delay] - ordonnancer le lancement de toutes les contrles maintenant (i.e. en m’me temps !)
- d = Utiliser un délai "irréfléchi" [dumb] d'1 seconde entre les contrles de service
- s = Utiliser un calcul de délai "débrouillard" [smart] pour répartir également les contrles de service (par défaut)
- x.xx = Utiliser le délai fourni de x.xx secondes

Délai maximum de répartition des contrles initiaux de services

Format: **max_service_check_spread=<minutes>**

Exemple : **max_service_check_spread=30**

Cette option détermine le temps maximum (en minutes) entre le démarrage de Nagios et la vérification de tous les services qui sont ordonnancés réguliérement. Cette option va automatiquement ajuster le délai inter-contrles de services (si nécessaire) pour s'assurer que les contrles initiaux des services vont s'effectuer dans le temps imparti. En général, cette option n'a pas d'effet si l'option d'utilisation des états mémorisés de l'ordonnanceur est activée. La valeur par défaut est de **30** (minutes).

Facteur d'entrelacement des services

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

Exemple : **service_interleave_factor=s**

Cette variable détermine comment les contrles de service sont entrelacés. L'entrelacement permet une distribution plus égale des contrles de service, une charge réduite sur les htes *distants*, et une détection globalement plus rapide des problémes liés aux htes. Avec l'introduction de la parallélisation des contrles de service, les htes distants peuvent se retrouver bombardés de contrles si l'entrelacement n'est pas activé. Ceci peut entrainer l'échec de contrles ou des résultats incorrects si l'hte distant est surchargé de requ’tes. Mettre une valeur de 1 est équivalent é ne pas entrelacer les contrles de service (c'est le mode de fonctionnement des

versions de Nagios antérieures é la version 0.0.5). Mettez une valeur de **s** (débrouillard)[smart]) pour que le calcul du facteur d'entrelacement soit automatique, é moins que vous ayez une bonne raison de la changer. Le meilleur moyen de comprendre le fonctionnement du facteur d'entrelacement est d'observer le CGI d'état (vue détaillée) au moment o Nagios vient de démarrer. Vous verrez comment les contrles sont faits au fur et é mesure que les résultats apparaissent. Vous trouverez plus d'informations sur l'entrelacement ici.

- *x* = Un nombre supérieur ou égal é 1 détermine le facteur d'entrelacement é utiliser. Une valeur de 1 revient é ne pas entrelacer les contrles de service.
- s = Utiliser un calcul "débrouillard" du facteur d'entrelacement (par défaut)

Nombre maximal de contrles de service simultanés

Format : **max_concurrent_checks=<maximum_de_contrles>**

Exemple : **max_concurrent_checks=20**

Cette option détermine le nombre maximal de contrles de service pouvant tourner en paralléle é un instant donné. Une valeur de 1 emp'che la parallélisation. Une valeur de 0 (par défaut) n'impose aucune restriction sur le nombre de contrles simultanés. Vous ajusterez cette valeur en fonction des capacités de la machine sur laquelle tourne Nagios, car elle impacte directement la charge du systéme (processeur, mémoire, etc.). Vous trouverez plus d'informations sur le nombre de contrles de services que vous devriez autoriser ici.

Fréquence de consolidation des services

Format : **service_reaper_frequency=<fréquence_en_secondes>**

Exemple : **service_reaper_frequency=10**

Cette option détermine la fréquence *en secondes* des événements de "consolidation" des services. Ces événements traitent les résultats des contrles de service parallélisés dont l'exécution est terminée. Ces événements sont au coeur de la supervision dans Nagios.

Méthode de délai inter-contrles d'htes

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

Exemple : **host_inter_check_delay_method=s**

Cette option détermine comment les contrles d'htes (*pour les htes qui sont réguliérement contrlés)* sont initialement répartis dans la file d'attente. L'option de calcul "débrouillard" [smart] du délai (par défaut) demande é Nagios de calculer un intervalle moyen entre les contrles et d'ordonnancer les contrles initiaux de tous les services é cet intervalle, ce qui permet d'éviter les pics d'utilisation du processeur. Il *n'est pas* recommandé d'utiliser la méthode "sans délai" [no delay]. En effet, cette méthode ordonnance tous les contrles en m'me temps. Vous obtiendrez plus d'informations sur la faéon dont cette variable affecte l'ordonnancement des contrles d'hte ici. Ses valeurs possibles sont :

- n = Ne pas utiliser de délai [no delay] - ordonnancer le lancement de tous les contrles maintenant (i.e. en m'me temps !)
- d = Utiliser un délai "irréfléchi" [dumb] d'1 seconde entre les contrles d'hte
- s = Utiliser un calcul de délai "débrouillard" [smart] pour répartir également les contrles d'hte (par défaut)
- x.xx = Utiliser le délai fourni de x.xx secondes

Délai maximum de répartition des contrles initiaux d'htes

Format: **max_host_check_spread=<minutes>**

Exemple : **max_host_check_spread=30**

Cette option détermine le temps maximum (en minutes) entre le démarrage de Nagios et la vérification de tous les htes qui sont contrlés réguliérement. Cette option va automatiquement ajuster le délai inter-contrles d'htes (si nécessaire) pour s'assurer que les contrles initiaux des services vont s'effectuer dans le temps imparti. En général, cette option n'a pas d'effet si l'option d'utilisation des états mémorisés de l'ordonnanceur est activée. La valeur par défaut est de **30** (minutes).

Valeur de l'intervalle de temps

Format : **interval_length=<secondes>**

Exemple : **interval_length=60**

C'est le nombre de secondes que contient une "unité de temps" utilisée dans la file d'ordonnancement, les re-notifications, etc. Les "unités de temps" sont utilisées dans le fichier de configuration des objets pour déterminer la fréquence d'exécution des contrles de service, la fréquence de re-notification d'un contact, etc.

Important : La valeur par défaut de cette variable est 60, ce qui veut dire qu'une "unité de temps" de 1 dans le fichier de configuration des htes vaut 60 secondes (1 minute). Je n'ai pas vraiment testé d'autres valeurs pour cette variable, donc vous la modifierez é vos risques et périls !

Option de réordonnancement automatique

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

Exemple : **auto_reschedule_checks=1**

Cette option détermine si Nagios va essayer de ré ordonnancer automatiquement les services de vérification actifs des htes et services pour les lisser dans le temps. Ceci peut aider é distribuer la charge sur le serveur de surveillance car il va essayer de conserver un delta de temps cohérent entre deux vérifications. La contrainte étant que l'ordonnancement sera plus strict.

ATTENTION : CETTE OPTION EST EXPERIMENTALE ET PEUT ETRE SUPPRIMEE DANS LES VERSIONS ULTERIEURES. ACTIVER CETTE OPTION PEUT DEGRADER LES PERFORMANCES AU LIEU DE LES AMELIORER, SI ELLE EST MAL UTILISEE.

Intervalle de réordonnancement automatique

Format: **auto_rescheduling_interval=<seconds>**

Exemple : **auto_rescheduling_interval=30**

Cette option détermine la fréquence (en secondes) de réordonnancement automatique. Cette option n'a d'effet que si l'option de réordonnancement automatique est active. La valeur par défaut est de 30 secondes.

ATTENTION : CETTE OPTION EST EXPERIMENTALE ET PEUT ETRE SUPPRIMEE DANS LES VERSIONS ULTERIEURES. ACTIVER CETTE OPTION PEUT DEGRADER LES PERFORMANCES AU LIEU DE LES AMELIORER, SI ELLE EST MAL UTILISEE.

Fen’tre de réordonnancement automatique

Format: **auto_rescheduling_window=<seconds>**

Exemple : **auto_rescheduling_window=180**

Cette option détermine la fen’tre de temps (en seconde) sur laquelle Nagios va porter le réordonnancement automatique. Seules les vérifications d'htes et de services qui doivent avoir lieu durant les prochaines N secondes seront affectées par le réordonnancement. Cette option n'a d'effet que si l'option de réordonnancement automatique est active. La valeur par défaut est de 180 secondes (3 minutes).

ATTENTION : CETTE OPTION EST EXPERIMENTALE ET PEUT ETRE SUPPRIMEE DANS LES VERSIONS ULTERIEURES. ACTIVER CETTE OPTION PEUT DEGRADER LES PERFORMANCES AU LIEU DE LES AMELIORER, SI ELLE EST MAL UTILISEE.

Option de contrle agressif des htes

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

Exemple : **use_aggressive_host_checking=0**

[NdT] : cette option comporte deux "g" é "aggressive" depuis la version 2. Elle n'en comprenait qu'un auparavant.

Nagios essaye d’’tre plus malin dans la faéon et le moment de contrler l'état des htes. En général, désactiver cette option permet é Nagios d’’tre un peu plus malin et de faire les contrles plus vite. Activer cette option revient é ralentir le contrle des htes, mais peut améliorer la sreté de fonctionnement. A moins que Nagios ne parvienne pas é détecter le rétablissement d'un de vos htes, je vous suggére de **ne pas** activer cette option.

- 0 = Ne pas utiliser le contrle agressif des htes (par défaut)
- 1 = Utiliser le contrle agressif des htes

Option de détection de l'oscillation d'un service ou d'un hte[flap]/strong>

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

Exemple : **enable_flap_detection=0**

Cette option détermine si Nagios essaiera de détecter les htes et les services qui oscillent [ou "bagotent, yoyotent de la touffe" en bon argot]. L'oscillation apparat lorsqu'un hte ou un service change d'état trop fréquemment, causant l'émission d'une montagne de notifications. Quand Nagios détecte qu'un hte ou un service oscille, il supprime temporairement les notifications pour cet hte/service jusqu'é ce qu'il arr’te d'osciller. La détection de l'oscillation est encore au stade expérimental, utilisez donc cette fonctionnalité avec prudence ! Pour plus d'informations sur la détection et la gestion des oscillations, lisez ceci. Note : si vous avez activé la mémorisation d'état, Nagios ignorera cette option é son (re)démarrage et utilisera la derniére valeur connue (telle qu'elle est enregistrée dans le fichier de mémorisation des états), *é moins que* vous ne désactiviez l'option use_retained_program_state. Si vous voulez changer cette option alors que la mémorisation d'état est activée (ainsi que l'option use_retained_program_state), vous devrez passer par la commande externe appropriée ou l'interface web.

- 0 = Ne pas activer la détection des oscillations (par défaut)
- 1 = Activer la détection des oscillations

Seuil inférieur d'oscillation d'un service

Format : **low_service_flap_threshold=<pourcentage>**

Exemple : **low_service_flap_threshold=25.0**

Cette option permet de donner le seuil inférieur pour la détection de l'oscillation d'un service. Pour plus d'informations sur la détection et la gestion des oscillations (et comment cette variable les affecte), lisez ceci.

Seuil supérieur d'oscillation d'un service

Format : **high_service_flap_threshold=<pourcentage>**

Exemple : **high_service_flap_threshold=50.0**

Cette option permet de donner le seuil supérieur pour la détection de l'oscillation d'un service. Pour plus d'informations sur la détection et la gestion des oscillations (et comment cette variable les affecte), lisez ceci.

Seuil inférieur d'oscillation d'un hte

Format : **low_host_flap_threshold=<pourcentage>**

Exemple : **low_host_flap_threshold=25.0**

Cette option permet de donner le seuil inférieur pour la détection de l'oscillation d'un hte. Pour plus d'informations sur la détection et la gestion des oscillations (et comment cette variable les affecte), lisez ceci.

Seuil supérieur d'oscillation d'un hte

Format : **high_host_flap_threshold=<pourcentage>**

Exemple : **high_host_flap_threshold=50.0**

Cette option permet de donner le seuil supérieur pour la détection de l'oscillation d'un hte. Pour plus d'informations sur la détection et la gestion des oscillations (et comment cette variable les affecte), lisez ceci.

Option de dépendance "Soft" des services

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

Exemple : **soft_state_dependencies=0**

Cette option détermine si Nagios utilisera les informations d'état "soft" des services lors du contrle des dépendances de service. En temps normal, Nagios n'utilise que le dernier état "hard" du service lors du contrle des dépendances. Si vous voulez utiliser le tout dernier état (que ce soit un état de type hard ou soft), activez cette option.

- 0 = Ne pas utiliser les états soft dans les dépendances de service (par défaut)
- 1 = Utiliser les états soft dans les dépendances de service

Dépassement de délai du contrle de service

Format : **service_check_timeout=<secondes>**

Exemple : **service_check_timeout=60**

C'est le nombre maximal de secondes pendant lequel Nagios laissera tourner un contrle de service. Si le contrle dépasse cette limite, il est tué et un état CRITICAL est retourné. Une erreur de dépassement de délai est également journalisée.

Il existe la confusion la plus totale sur ce que cette option fait vraiment. Elle est lé comme dernier rempart, pour tuer les plugins qui se comportent mal, ou ne se sont pas terminés correctement. Il faut la positionner é une valeur haute (quelque chose comme 60s), de maniére é ce que tout contrle ait le temps de se terminer avant cette limite. Si le contrle du service prend plus de temps pour s'exécuter, Nagios le tuera, pensant que c'est un processus ayant des problémes d'exécution.

Dépassement de délai du contrle d'hte

Format : **host_check_timeout=<secondes>**

Exemple : **host_check_timeout=60**

C'est le nombre maximal de secondes pendant lesquelles Nagios laissera tourner un contrle d'hte. Si le contrle dépasse cette limite, il est tué et un état CRITICAL est retourné, et l'hte sera supposé ’tre dans l'état DOWN. Une erreur de dépassement de délai est également journalisée.

Il existe également la confusion la plus totale sur ce que cette option fait vraiment. Elle est lé comme dernier rempart, pour tuer les plugin qui se comportent mal ou ne se sont pas terminés correctement. Il faut la positionner é une valeur haute (quelque chose comme 60s), de maniére é ce que tout contrle ait le temps de se terminer avant cette limite. Si le contrle de l'hte prend plus de temps pour s'exécuter, Nagios le tuera, pensant que c'est un processus ayant des problémes d'exécution.

Dépassement de délai de contrle du gestionnaire d'événement

Format : **event_handler_timeout=<secondes>**

Exemple : **event_handler_timeout=60**

C'est le nombre maximal de secondes pendant lequel Nagios laissera tourner un gestionnaire d'événement. Si un gestionnaire d'événement dépasse cette limite il sera tué et une alerte sera journalisée.

Il existe lé aussi, comme précédemment, la confusion la plus totale sur ce que cette option fait vraiment. Elle est lé comme dernier rempart, pour tuer les plugins qui se comportent mal ou ne se sont pas terminés correctement. Il faut le positionner é une valeur haute (quelque chose comme 60s), de maniére é ce que tout contrle ait le temps de se terminer avant cette limite. Si le contrle du gestionnaire d'événement prend plus de temps pour s'exécuter, Nagios le tuera, pensant que c'est un processus ayant des problémes d'exécution.

Dépassement de délai de notification

Format : **notification_timeout=<secondes>**

Exemple : **notification_timeout=60**

C'est le nombre maximal de secondes pendant lequel Nagios laissera tourner une commande de notification. Si une commande de notification dépasse cette limite elle sera tuée et une alerte sera journalisée.

Il existe la confusion la plus totale (**NdT**: toute ressemblance avec des phrases précédentes serait purement intentionnelle :-)) sur ce que cette option fait vraiment. Elle est lé comme dernier rempart, pour tuer les plugins qui se comporte mal ou ne se sont pas terminés correctement. Il faut le positionner é une valeur haute (quelque chose comme 60s), de maniére é ce que tout contrle ait le temps de se terminer avant cette limite. Si

la notification prend plus de temps pour s'exécuter, Nagios la tuera, pensant que c'est un processus ayant des problémes d'exécution.

Dépassement de délai de la commande de remontée de contrle de service

Format : **ocsp_timeout=<secondes>**

Exemple : **ocsp_timeout=5**

C'est le nombre maximal de secondes pendant lequel Nagios laissera tourner une commande de remontée de contrle de service. Si une commande dépasse cette limite, elle sera tuée et une alerte sera journalisée.

Dépassement de délai de la commande de remontée de contrle d'hte

Format: **ochp_timeout=<seconds>**

Exemple : **ochp_timeout=5**

C'est le nombre maximal de secondes pendant lequel Nagios laissera tourner une commande de remontée de contrle d'hte. Si une commande dépasse cette limite, elle sera tuée et une alerte sera journalisée.

Dépassement de délai de la commande de traitement des données liées aux performances

Format : **perfdata_timeout=<secondes>**

Exemple : **perfdata_timeout=5**

C'est le nombre maximal de secondes pendant lequel Nagios laissera tourner une commande de traitement des données liées aux performance d'un hte ou de traitement des données liées aux performances d'un service. Si une commande dépasse cette limite, elle sera tuée et une alerte sera journalisée.

Option de remontée de contrle de service

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

Exemple : **obsess_over_services=1**

Cette variable détermine si Nagios remontera les résultats de contrles de service et lancera la commande de remontée de contrle de service que vous avez définie. Je sais que c'est un drle de nom [en anglais "obsessive compulsive service processor"], mais c'est tout ce qui m'est venu é l'esprit. Cette option est utile dans le cadre de la supervision répartie. Si vous ne faites pas de supervision répartie, n'activez pas cette option.

- 0 = Ne pas remonter les contrles de service (par défaut)
- 1 = Remonter les contrles de service

Commande de remontée de contrle de service

Format : **ocsp_command=<commande>**

Exemple : **ocsp_command=obsessive_service_handler**

Cette option définit la commande é lancer aprés *chaque* contrle de service, ce qui peut ’tre utile dans une supervision répartie. Elles est exécutée aprés les éventuelles commandes de gestion d'événement ou de notification. L'argument *command*e est le nom court d'une définition de commande que vous avez définie

dans le fichier de configuration des htes. Cette option sert dans le cadre de la supervision répartie. Le temps d'exécution maximal de cette commande est déterminé par la variable ocsp_timeout. Vous trouverez plus d'informations sur la supervision répartie ici.

Option de remontée de contrle d'hte

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

Exemple : **obsess_over_hosts=1**

Cette variable détermine si Nagios remontera les résultats de contrles d'htes et lancera la commande de remontée de contrle d'hte que vous avez définie. Je sais que c'est un drle de nom [en anglais "obsessive compulsive host processor"], mais c'est tout ce qui m'est venu é l'esprit. Cette option est utile dans le cadre de la supervision répartie. Si vous ne faites pas de supervision répartie, n'activez pas cette option.

- 0 = Ne pas remonter les contrles d'hte (par défaut)
- 1 = Remonter les contrles de service

Commande de remontée de contrle d'hte

Format: **ochp_command=<command>**

Exemple : **ochp_command=obsessive_host_handler**

Cette option définit la commande é lancer aprés *chaque* contrle d'hte, ce qui peut ’tre utile dans une supervision répartie. Elle est exécutée aprés les éventuelles commandes de gestion d'événement ou de notification. L'argument *command*e est le nom court d'une définition de commande que vous avez définie dans le fichier de configuration des htes. Cette option sert dans le cadre de la supervision répartie. Le temps d'exécution maximal de cette commande est déterminé par la variable ochp_timeout. Vous trouverez plus d'informations sur la supervision répartie ici. Cette commande n'est exécutée que si l'option de remontée de contrle d'hte est activée.

Option de traitement des données liées aux performances

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

Exemple : **process_performance_data=1**

Cette valeur détermine si Nagios traitera les données liées aux performances des contrles d'htes et de services.

- 0 = Ne pas traiter les données de performance (par défaut)
- 1 = Traiter les données de performance

Commande de performance liée aux htes

Format: **host_perfdata_command=<command>**

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

Cette option vous permet de spécifier une commande qui sera lancée aprés *chaque* vérification d'hte pour traiter les données de performance qui peuvent ’tre retournées aprés la vérification. L'argument *command*e est le nom court d'une définition de commande que vous avez définie dans le fichier de configuration des objets. Cette commande est exécutée si l'option de traitement des données de performance est activée et si la directive

process_perf_data (**[NdT] : traitement des données de performance**) est activée dans la définition de l'hte.

Commande de performance liée aux services

Format: **service_perfdata_command=<command>**

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

Cette option vous permet de spécifier une commande qui sera lancée aprés *chaque* vérification de service pour traiter les données de performance qui peuvent 'tre retournées aprés la vérification. L'argument *command*e est le nom court d'une définition de commande que vous avez définie dans le fichier de configuration des objets. Cette commande est exécutée si l'option de traitement des données de performance est activée et si la directive *process_perf_data* (**[NdT] : traitement des données de performance**) est activée dans la définition du service.

Fichier de performance des htes

Format: **host_perfdata_file=<file_name>**

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

Cette option vous permet de spécifier un fichier dans lequel les données de performance vont 'tre écrites aprés chaque vérification d'hte. Les données seront écrites dans ce fichier comme spécifié dans le patron du fichier de performance des htes. Les données ne seront écrites que si l'option de traitement des données de performance est activée et si la directive *process_perf_data* (**[NdT] : traitement des données de performance**) est activée dans la définition de l'hte.

Fichier de performance des services

Format: **service_perfdata_file=<file_name>**

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

Cette option vous permet de spécifier un fichier dans lequel les données de performance vont 'tre écrites aprés chaque vérification de service. Les données seront écrites dans ce fichier comme spécifié dans le patron du fichier de performance des service. Les données ne seront écrites que si l'option de traitement des données de performance est activée et si la directive *process_perf_data* (**[NdT] : traitement des données de performance**) est activée dans la définition du service.

Patron du fichier de performance des htes

Format: **host_perfdata_file_template=<template>**

Exemple **host_perfdata_file_template=[HOSTPERFDATA]\t$TIMET$\t$HOSTNAME$\t$HOSTEXECUTION**
:

ette option détermine ce qui va 'tre écrit (et comment) dans le fichier de performance des htes. Le patron peu contenir des macros, des caractéres spéciaux (\t pour une tabulation, \r un retour charriot, \n pour un retour é la ligne). Un retour é la ligne est ajouté aprés chaque écriture des données de performance.

Patron du fichier de performance des services

Format: **service_perfdata_file_template=<template>**

Exemple **service_perfdata_file_template=[SERVICEPERFDATA]\t$TIMET$\t$HOSTNAME$\t$SERVICEDE**
:

Cette option détermine ce qui va 'tre écrit (et comment) dans le fichier de performance des services. Le patron

peu contenir des macros, des caractéres spéciaux (\t pour une tabulation, \r un retour charriot, \n pour un retour é la ligne). Un retour é la ligne est ajouté aprés chaque écriture des données de performance.

Mode d'ouverture du fichier de performance des htes

Format: **host_perfdata_file_mode=<mode>**

Exemple : **host_perfdata_file_mode=a**

Cette option détermine si le fichier de performance des htes est ouvert en mode écrasement ou ajout. A moins que le fichier ne soit un tube nommé [named pipe], vous voudrez certainement utiliser l'option par défaut ajout.

- a = Ouvrir le fichier en mode ajout [append] (par défaut)
- w = Ouvrir le fichier en mode écrasement [write]

Mode d'ouverture du fichier de performance des services

Format: **service_perfdata_file_mode=<mode>**

Exemple : **service_perfdata_file_mode=a**

Cette option détermine si le fichier de performance des services est ouvert en mode écrasement ou ajout. A moins que le fichier ne soit un tube nommé [named pipe], vous voudrez certainement utiliser l'option par défaut ajout.

- a = Ouvrir le fichier en mode ajout [append] (par défaut)
- w = Ouvrir le fichier en mode écrasement [write]

Intervalle de traitement du fichier de performance des htes

Format: **host_perfdata_file_processing_interval=<seconds>**

Exemple : **host_perfdata_file_processing_interval=0**

Cette option vous permet de spécifier l'intervalle (en secondes) entre deux traitements du fichier de performance des htes par la commande de performance liée aux htes. Une valeur de 0 signifie que le traitement n'aura pas lieu de maniére réguliére.

Intervalle de traitement du fichier de performance des services

Format: **service_perfdata_file_processing_interval=<seconds>**

Exemple : **service_perfdata_file_processing_interval=0**

Cette option vous permet de spécifier l'intervalle (en secondes) entre deux traitements du fichier de performance des services par la commande de performance liée aux services. Une valeur de 0 signifie que le traitement n'aura pas lieu de maniére réguliére.

Commande de traitement du fichier de performance des htes

Format: **host_perfdata_file_processing_command=<command>**

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

Cette option vous permet de définir la commande qui sera exécutée pour le tratement du fichier de performance des htes. L'argument *command*e est le nom court d'une définition de commande que vous avez définie dans le fichier de configuration des objets. L'intervalle entre deux traitements est indiqué par

l'intervalle de traitement du fichier de performance des htes

Commande de traitement du fichier de performance des services

Format: **service_perfdata_file_processing_command=<command>**

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

Cette option vous permet de définir la commande qui sera exécutée pour le tratement du fichier de performance des services. L'argument *command*e est le nom court d'une définition de commande que vous avez définie dans le fichier de configuration des objets. L'intervalle entre deux traitements est indiqué par l'intervalle de traitement du fichier de performance des services

Option de vérification des contrles de service orphelins

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

Exemple : **check_for_orphaned_services=0**

Cette option vous permet d'activer ou désactiver la vérification des contrles de service orphelins. Les contrles de service orphelins sont des contrles ayant été exécutés et supprimés de la file des événements, mais dont les résultats n'ont pas été remontés depuis longtemps. Comme aucun résultat n'a été remontéf pour ce service, il n'est pas réordonnancé dans la file d'événements. Cela peut causer l'arr't des contrles du service. Normalement, c'est un phénoméne trés rare - il peut se produire si un utilisateur ou un processus extérieur a tué le processus utilisé pour exécuter le contrle de service. Si cette option est activée et que Nagios s'aperéoit qu'un résultat de contrle de service particulier ne revient pas, il journalisera un message d'erreur, et réordonnancera le contrle de service. Si vous constatez que certains contrles de service semblent n''tre jamais réordonnancés, activez cette option et cherchez dans les journaux, des messages concernant des services orphelins.

- 0 = Ne pas vérifier les contrles de service orphelins (par défaut)
- 1 = Vérifier les contrles de service orphelins

Option de contrle de la validité des données d'un service

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

Exemple : **check_service_freshness=0**

Cette option détermine si Nagios va contrler ou non périodiquement la validité des données d'un service. L'activation de cette option aide é contrler que les contrles de service passifs sont reéus en temps et en heure. On peut trouver plus d'informations sur cette option ici.

- 0 = Ne pas contrler
- 1 = Contrler (par défaut)

Intervalle de contrle de la validité des données d'un service

Format: **freshness_check_interval=<seconds>**

Exemple : **freshness_check_interval=60**

Cette option détermine l'intervalle de temps (en secondes) entre deux contrles de validité des données d'un service. Si vous avez désactivé ce service, avec l'option de contrle de validité des données d'un service, cette option n'a pas d'effet. Vous pouvez trouver plus d'informations sur ce service ici.

Option du contrle de la validité des données d'un hte

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

Exemple : **check_host_freshness=0**

Cette option détermine si Nagios va contrler ou non périodiquement la validité des données d'un hte. L'activation de cette option aide é contrler que les contrles passifs des htes sont reéus en temps et en heure. On peut trouver plus d'informations sur cette option ici.

- 0 = Ne pas contrler
- 1 = Contrler (par défaut)

Intervalle de contrle de la validité des données d'un hte

Format: **host_freshness_check_interval=<seconds>**

Exemple : **host_freshness_check_interval=60**

Cette option détermine l'intervalle de temps (en secondes) entre deux contrles de validité des données d'un hte. Si vous avez désactivé ce service, avec l'option de contrle de validité des données d'un hte, cette option n'a pas d'effet. Vous pouvez trouver plus d'informations sur ce service ici.

Format de date

Format: **date_format=<option>**

Exemple : **date_format=us**

Cette option spécifie le format de date que Nagios utilisera dans l'interface web. Les options possibles sont :

Option	Format de sortie	Exemple
us	MM/DD/YYYY HH:MM:SS	06/30/2002 03:15:00
euro	DD/MM/YYYY HH:MM:SS	30/06/2002 03:15:00
iso8601	YYYY-MM-DD HH:MM:SS	2002-06-30 03:15:00
strict-iso8601	YYYY-MM-DDTHH:MM:SS	2002-06-30T03:15:00

Caractéres illégaux dans les noms d'objets

Format: **illegal_object_name_chars=<chars>**

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

Cette option vous permet de spécifier quels sont les caractéres illégaux dans les noms d'objets, tels qu'htes, services et autres. Nagios vous autorisera la plupart des caractéres dans les définitions d'objets, mais je recommande de ne pas utiliser les caractéres ci-dessus. Le faire vous expose é des problémes dans l'interface web, les notifications de commandes, etc.

Caractéres illégaux dans la sortie des macros

Format: **illegal_macro_output_chars=<chars>**

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

Cette option vous permet de spécifier les caractéres illégaux qui seront filtrés dans les macros, avant qu'elles ne soient utilisées dans les notifications, les gestionnaires d'événements et autres commandes. Ceci n'affecte pas les macros utilisées dans les contrles des services ou des htes. Vous pouvez choisir de ne pas filtrer les caractéres donnés en exemple ci-dessus, mais je vous le le déconseille. Quelques uns d'entre eux sont interprétés par le shell (par exemple, le `) et peuvent poser des problémes de sécurité. Les macros suivantes

sont débarrassées des caractéres spécifiés dans l'option :

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

Option de concordance par expressions rationnelles

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

Exemple : **use_regexp_matching=0**

Cette option détermine si les directives dans vos définitions d'objet seront traitées comme des expressions rationnelles. Vous pouvez trouver plus d'informations sur ce service ici.

- 0 = Ne pas utiliser la concordance par expressions rationnelles (defaut)
- 1 = Utiliser la concordance par expressions rationnelles

Option de concordance systématique par expressions rationnelles

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

Exemple : **use_true_regexp_matching=0**

Si vous avez activé la concordance par expression rationnelle, cette option va déterminer quand les directives vont ’tre traitées comme des expressions rationnelles :

- Si cette option est désactivée (par défaut), les directives seront traitées comme des expression rationnelles uniquement si elles contiennent un ***** ou un **?**
- Si cette option est activée, toutes les directives seront traitées comme des expressions rationnelles. Fates trés attention lorsque vous activez cette option !

Pour plus d'information, lisez ceci

- 0 = Ne pas utiliser la concordance systématique par expressions rationnelles (par défaut)
- 1 = Utiliser la concordance systématique par expressions rationnelles

Adresse email de l'administrateur

Format : **admin_email=<adresse_email>**

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

C'est l'adresse mail de l'administrateur local de la machine (i.e. celle sur laquelle Nagios tourne). Cette valeur peut ’tre utilisée dans les commandes de notification grôce é la macro **$ADMINEMAIL$**.

Pager de l'administrateur

Format : **admin_pager=<numéro_de_pager_ou_passerelle_pager_email>**

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

C'est le numéro du pager (ou la passerelle pager-email) de l'administrateur de la machine locale (i.e. celle sur laquelle Nagios tourne). Le numéro ou l'adresse de pager peut ’tre utilisé dans les commandes de notification grôce é la macro **$ADMINPAGER$**.

Définition des objets

Que sont les données des objets ?

Les données des objet sont simplement un terme générique que j'utilise pour décrire les différentes définitions de données dont vous avez besoin pour superviser quelque chose. Les types de défintions d'objet sont les suivants :

- Services
- Groupes de services
- Htes
- Groupes d'htes
- Contacts
- Groupes de contacts
- Commandes
- Périodes de temps
- Escalade de services
- Dépendances de services
- Escalade d'htes
- Dépendances d'htes
- Informations complémentaires sur les htes
- Informations complémentaires sur les services

O sont définies les données des objets ?

Les données des objets sont définies dans un ou plusieurs fichiers de configuration que vous déclarez en utilisant les paramétres cfg_file et/ou cfg_dir dans le fichier de configuration principal. Vous pouvez inclure de multiples fichiers et/ou répertoires de configuration des objets en utilisant de multiples paramétres *cfg_file* et/ou *cfg_dir*.

Comment définir les données des objets ?

Les définitions des objets se font é travers un systéme de gabarit (template). Cliquez ici pour plus d'information sur cette méthode de définition des données des objets.

Options du fichier de configuration des CGI

Notes

Quand vous créez ou modifiez des fichiers de configuration, n'oubliez pas que :

1. Les lignes qui commencent avec le caractére '#' sont considérées comme des commentaires et ne sont donc pas traitées
2. Les noms des variables doivent commencer au début de la ligne - ne mettez pas d'espace avant le nom
3. Les noms des variables respectent la casse (majuscule/minuscule)

Exemple de configuration

Un exemple de fichier de configuration est créé lorsque vous lancez le script "configure". Vous pouvez trouver le fichier exemple dans le répertoire *sample-config* de la distribution Nagios. En lanéant la commande '**make install-config**', les fichiers seront copiés dans le répertoire Nagios destination.

Localisation du fichier de configuration

Par défaut, Nagios s'attend é trouver le fichier de configuration des CGI sous le nom **cgi.cfg** dans le répertoire de configuration avec le fichier de configuration principal. Si vous changez le nom ou la localisation de du fichier, vous devez configurer Apache pour qu'il passe une variable d'environnement nommée NAGIOS_CGI_CONFIG - contenant la localisation correcte du fichier - aux CGI de Nagios. Ceci peut ’tre réalisé avec la directive *SetEnv*. Voir la configuration d'Apache pour plus de détails.

Index

Emplacement du fichier de configuration principal

Format : **main_config_file=<nom_fichier>**

Exemple : **main_config_file=/usr/local/nagios/etc/nagios.cfg**

Cette option détermine le chemin d'accés é votre fichier de configuration principal. Les CGI doivent savoir o trouver ce fichier pour récupérer les informations de configuration, l'état courant des htes et des services, etc.

Chemin d'accés physique aux fichiers HTML

Format : **physical_html_path=<chemin>**

Exemple : **physical_html_path=/usr/local/nagios/share**

C'est le chemin du répertoire *physique* de votre serveur o sont stockés les fichiers HTML de Nagios. Nagios suppose que la documentation et les images (utilisées par les CGI) sont stockées dans des sous-répertoires nommés respectivement *docs/* et *images/*.

URL d'accés aux pages HTML

Format : **url_html_path=<chemin>**

Exemple : **url_html_path=/nagios**

Si, lors de l'accés é Nagios via un navigateur web, vous pointez sur une URL du type **http://www.monserveur.com/nagios**, cette variable doit avoir pour valeur */nagios*. En fait, il s'agit de la partie contenant le chemin d'accés aux pages HTML de Nagios dans l'URL utilisée pour accéder aux pages HTML de Nagios.

Utilisation de l'authentification

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

Exemple : **use_authentication=1**

Cette option détermine si les CGI utiliseront l'authentification et les autorisations pour déterminer les informations et les commandes auxquelles les utilisateurs auront accés. Je vous recommande vivement d'utiliser l'authentification dans les CGI. Si vous choisissez de ne pas le faire, assurez-vous de supprimer le CGI de commande pour emp'cher les utilisateurs non autorisés d'envoyer des commandes é Nagios. Ce CGI ne devrait pas envoyer de commandes é Nagios si l'authentification est désactivée, mais deux précautions valent mieux qu'une. Vous trouverez plus d'informations sur la faéon de configurer l'authentification et les autorisations dans les CGI ici.

- 0 = Ne pas utiliser l'authentification
- 1 = Utiliser l'authentification et les autorisations (par défaut)

Nom d'utilisateur par défaut

Format : **default_user_name=<nom_utilisateur>**

Exemple : **default_user_name=guest**

Cette variable définit un nom d'utilisateur par défaut pour accéder aux CGI. Ainsi les utilisateurs d'un domaine sécurisé (i.e., derriére un firewall) peuvent accéder aux CGI sans avoir é s'authentifier auprés du serveur web. Vous pouvez choisir cette fonctionnalité pour éviter le recours é l'authentification de base si vous n'utilisez pas un serveur web sécurisé, car l'authentification de base transmet les mots de passe en clair sur Internet.

Important : *Ne définissez pas* un utilisateur par défaut é moins que vous n'utilisiez un serveur web sécurisé et que vous soyez sr que tous ceux qui ont accés aux CGI ont été authentifiés d'une maniére ou d'une autre ! Si vous définissez cette variable, ceux qui ne se sont pas authentifiés auprés du serveur web hériteront de tous les droits que vous donnez é cet utilisateur !

Accés aux informations sur le systéme et le processus

Format : **authorized_for_system_information=<utilisateur1>,<utilisateur2>,<utilisateur3>,<utilisateur*n*>**

Exemple : **authorized_for_system_information=nagiosadmin,theboss**

C'est une liste de noms d'*utilisateurs authentifiés*, séparés par des virgules, qui peuvent voir les informations sur le systéme et le processus dans les CGI d'informations complémentaires. Les utilisateurs de cette liste *ne sont pas* automatiquement autorisés é passer des commandes systéme/processus. Si vous voulez que des utilisateurs puissent aussi passer ces commandes, il faut les ajouter é la variable authorized_for_system_commands. Vous trouverez plus d'informations sur la faéon de configurer l'authentification et les autorisations des CGI ici.

Accés aux commandes du systéme/processus

Format : **authorized_for_system_commands=<utilisateur1>,<utilisateur2>,<utilisateur3>,<utilisateur*n*>**

Exemple : **authorized_for_system_commands=nagiosadmin**

C'est une liste de nom d'*utilisateurs authentifiés*, séparés par des virgules, qui peuvent passer des commandes systéme/processus via le CGI de commande. Les utilisateurs de cette liste *ne sont pas* automatiquement autorisés é visualiser les informations sur le systéme et le processus. Si vous voulez que des utilisateurs puissent visualiser ces informations aussi, il faut les ajouter é la variable authorized_for_system_information. Vous trouverez plus d'informations sur la faéon de configurer l'authentification et les autorisations des CGI ici.

Accés aux informations de configuration

Format : **authorized_for_configuration_information=<utilisateur1>,<utilisateur2>,<utilisateur3>,<utilisateur*n*>**

Exemple : **authorized_for_configuration_information=nagiosadmin**

C'est une liste de noms d'*utilisateurs authentifiés*, séparés par des virgules, qui peuvent voir les informations liées é la configuration via le CGI de configuration. Les utilisateurs de cette liste peuvent voir les informations sur tous les htes configurés, les groupes d'htes, les services, les contacts, les groupes de contacts, les périodes, et les commandes. Vous trouverez plus d'informations sur la faéon de configurer l'authentification et les autorisations dans les CGI ici.

Accés global aux informations sur les htes

Format : **authorized_for_all_hosts=<utilisateur1>,<utilisateur2>,<utilisateur3>,<utilisateur*n*>**

Exemple : **authorized_for_all_hosts=nagiosadmin,theboss**

C'est une liste de noms d'*utilisateurs authentifiés*, séparés par des virgules, qui peuvent voir l'état et la configuration de tous les htes. Les utilisateurs de cette liste sont également automatiquement autorisés é voir les informations de tous les services. Les utilisateurs de cette liste *ne sont pas* automatiquement autorisés é envoyer des commandes aux htes ou aux services. Si vous voulez que des utilisateurs puissent aussi envoyer des commandes, il faut les ajouter é la variable authorized_for_all_host_commands. Vous trouverez plus d'informations sur la faéon de configurer l'authentification et les autorisations des CGI ici.

Accés global aux commandes des htes

Format : **authorized_for_all_host_commands=<utilisateur1>,<utilisateur2>,<utilisateur3>,<utilisateur*n*>**

Exemple **authorized_for_all_host_commands=nagiosadmin**
:

C'est une liste de noms d'*utilisateurs authentifiés*, séparés par des virgules, qui peuvent envoyer des commandes é tous les htes via le CGI de commande. Les utilisateurs de cette liste sont également automatiquement autorisés é envoyer des commandes é tous les services. Les utilisateurs de cette liste *ne sont pas* automatiquement autorisés é voir l'état ou la configuration de tous les htes ou services. Si vous voulez que des utilisateurs puissent voir ces informations aussi, il faut les ajouter é la variable authorized_for_all_hosts. Vous trouverez plus d'informations sur la faéon de configurer l'authentification et les autorisations des CGI ici.

Accés global aux informations sur les services

Format : **authorized_for_all_services=<utilisateur1>,<utilisateur2>,<utilisateur3>,<utilisateur*n*>**

Exemple : **authorized_for_all_services=nagiosadmin,theboss**

C'est une liste de noms d'*utilisateurs authentifiés*, séparés par des virgules, qui peuvent voir l'état et la configuration de tous les services. Les utilisateurs de cette liste *ne sont pas* automatiquement autorisés é voir les informations de tous les htes. Les utilisateurs de cette liste *ne sont pas* automatiquement autorisés é envoyer des commandes é tous les services. Si vous voulez que des utilisateurs puissent aussi envoyer des commandes, il faut les ajouter é la variable authorized_for_all_service_commands. Vous trouverez plus d'informations sur la faéon de configurer l'authentification et les autorisations des CGI ici.

Accés global aux commandes des services

Format : **authorized_for_all_service_commands=<utilisateur1>,<utilisateur2>,<utilisateur3>,<utilisateur*n*>**

Exemple **authorized_for_all_service_commands=nagiosadmin**
:

C'est une liste de noms d'*utilisateurs authentifiés*, séparés par des virgules, qui peuvent envoyer des commandes é tous les services via le CGI de commande. Ils *ne sont pas non plus* automatiquement autorisés é envoyer des commandes aux htes. Les utilisateurs de cette liste *ne sont pas* automatiquement autorisés é voir l'état ou la configuration de tous les services. Si vous voulez que des utilisateurs puissent aussi voir ces informations, il faut les ajouter é la variable authorized_for_all_services. Vous trouverez plus d'informations sur la faéon de configurer l'authentification et les autorisations des CGI ici.

Image de fond du CGI de cartographie des états (Statusmap)

Format: **statusmap_background_image=<image_file>**

Exemple: **statusmap_background_image=smbackground.gd2**

Cette option permet de spécifier une image qui sera utilisée comme fond d'image dans le CGI de cartographie des états. Il suppose que l'image est située dans le chemin des images HTML (c.a.d /usr/local/nagios/share/images). Ce chemin est automatiquement déterminé en ajoutant "/images" au chemin défini dans la directive chemin d'accés physique aux fichiers html. Note: Cette image peut ’tre au format GIF, JPEG, PNG, ou GD2. Cependant, le format GD2 (de préférence en format compressé) est recommandé, en raison de la faible charge CPU requise quand le CGI génére l'image.

Dessin de la cartographie des états: valeur par défaut

Format: **default_statusmap_layout=<layout_number>**

Exemple: **default_statusmap_layout=4**

Cette option définit la méthode de dessin utilisée par défaut par le CGI de cartographie des états. Les valeurs autorisées sont:

<layout_number> Valeur	Méthode de dessin
0	Coordonnées définies par l'utilisateur
1	Couches imbriquées
2	Arbre réduit
3	Arbre équilibré
4	Circulaire
5	Circulaire (avec marque supérieure)
6	Circulaire (sous forme de ballon)

Monde inclus dans le CGI du monde des états (Statuswrl)

Format: **statuswrl_include=<vrml_file>**

Exemple: **statuswrl_include=myworld.wrl**

Cette option permet d'inclure ses propres objets dans le monde VRML généré. Elle suppose que le fichier est situé dans le chemin défini par la directive d'accés aux fichiers html. Note: Ce fichier doit ’tre un monde VRML valide (c.a.d que vous devez pouvoir le visualiser avec un fouineur [browser] VRML)

Dessin du monde des états: valeur par défaut

Format: **default_statuswrl_layout=<layout_number>**

Exemple: **default_statuswrl_layout=4**

Cette option définit la méthode utilisée par défaut pour dessiner le monde VRML avec le CGI concerné (Statusvrml). Les options autorisées sont :

<layout_number> Valeur	Méthode de dessin
0	Coordonnées définies par l'utilisateur
2	Arbre réduit
3	Arbre équilibré
4	Circulaire

Fréquence de rafrachissement des CGI

Format : **refresh_rate=<délai_en_secondes>**

Exemple : **refresh_rate=90**

Cette option vous permet de spécifier le délai en secondes entre deux rafrachissements de page dans les CGI d'état, de cartographie des états, et d'informations complémentaires.

Alertes sonores

Formats :	**host_unreachable_sound=<fichier_son>** **host_down_sound=<fichier_son> service_critical_sound=<fichier_son>** **service_warning_sound=<fichier_son>** **service_unknown_sound=<fichier_son>**
Exemples :	**host_unreachable_sound=hostu.wav host_down_sound=hostd.wav** **service_critical_sound=critical.wav** **service_warning_sound=warning.wav** **service_unknown_sound=unknown.wav**

Cette option vous permet de spécifier un fichier audio é jouer dans votre navigateur lorsqu'il y a des problémes dans le CGI d'état. En cas de problémes multiples, le fichier audio joué est celui du probléme le plus critique. Un probléme est considéré comme le plus critique lorsqu'un ou plusieurs htes sont inaccessibles, alors qu'il est le moins critique lorsqu'un ou plusieurs services sont dans un état inconnu (voyez l'ordre dans l'exemple ci-dessus). Les fichiers audios sont censés se trouver dans le sous-répertoire **media/** de votre répertoire HTML (i.e. */usr/local/nagios/share/media*).

Ping Syntax

Format: **ping_syntax=<command>**

Exemple: **ping_syntax=/bin/ping -n -U -c 5 $HOSTADDRESS$**

Cette option définit quelle syntaxe doit ’tre utilisée quand on veut tester un hte avec ping é travers l'interface WAP en utilisant le CGI statuswml .Vous devez inclure le chemin complet vers le fichier binaire exécutable de ping, ainsi que les paramétres passés é la commande. La macro $HOSTADDRESS$ est remplacée par l'adresse de l'hte avant que la commande ne soit exécutée.

Authentification et autorisations dans les CGI

Notes

Dans ces instructions, je pars du principe que vous utilisez le serveur web Apache [Anglais]. Dans le cas contraire, il vous faudra faire les adaptations appropriées.

Définitions

J'utiliserai les termes qui suivent dans ces instructions, il est donc important que vous en compreniez le sens

- Un *utilisateur authentifié* est quelqu'un qui s'est authentifié auprés du serveur web avec un nom d'utilisateur et un mot de passe, et é qui le serveur web a donné accés é l'interface web de Nagios.
- Un *contact authentifié* est un utilisateur authentifié dont le nom correspond é celui d'une définition de contact du fichier de configuration.

Index

Déclarer des utilisateurs authentifiés

Si vous ne l'avez pas déjé fait, vous devez modifier la configuration de votre serveur web pour activer l'authentification pour les CGI et les pages HTML de l'interface Nagios. Vous pouvez trouvez les instructions pour cette configuration ici.

Maintenant que votre serveur web est configuré pour réclamer un accés authentifié é l'interface web de Nagios, vous devez déclarer les utilisateurs autorisés é y accéder. C'est ce que fait la commande Apache **htpasswd**.

La commande suivante crée un fichier *htpasswd.users* dans le répertoire */usr/local/nagios/etc*. Elle crée également une entrée nom d'utilisateur/mot de passe pour *nagiosadmin*. Un mot de passe pour l'authentification de *nagiosadmin* dans le serveur web vous sera alors demandé.

```
htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
```

Ajoutez ainsi tous les utilisateurs qui doivent avoir accés aux CGI. Utilisez la commande suivante pour cela, en remplaéant <nom_utilisateur> par le nom de l'utilisateur que vous voulez ajouter. Notez que l'option **-c** n'est plus utilisée, dans la mesure o le fichier initial est déjé créé.

```
htpasswd /usr/local/nagios/etc/htpasswd.users <nom_utilisateur>
```

Voilé, la premiére partie du travail est faite. En pointant votre navigateur sur les CGI de Nagios, un nom d'utilisateur et un mot de passe vous seront demandés. Si vous rencontrez des problémes, veuillez vous référer é la documentation de votre serveur web.

Activer l'authentification/l'autorisation dans les CGI

Maintenant il s'agit de s'assurer que les CGI sont configurés pour filtrer les informations et les commandes auxquelles les utilisateurs peuvent accéder. Pour cela la variable use_authentication du fichier de configuration des CGI doit ’tre positionnée é une valeur différente de zéro. Par exemple:

```
use_authentication=1
```

Voilé, la fonctionnalité d'authentification/d'autorisation de base est activée dans les CGI.

Droits d'accés par défaut aux informations des CGI

Quelles sont maintenant les droits d'accés aux CGI par défaut ?

Données des CGI	Contacts authentifiés*	Autres utilisateurs authentifiés*
Information sur l'état des htes	Oui	Non
Information sur la configuration des htes	Oui	Non
Historique des htes	Oui	Non
Notifications des htes	Oui	Non
Commandes des htes	Oui	Non
Information sur l'état des services	Oui	Non
Information sur la configuration des services	Oui	Non
Historique des services	Oui	Non
Notifications des services	Oui	Non
Commandes des services	Oui	Non
Toutes informations de configuration	Non	Non
Information sur le systéme/processus	Non	Non
Commandes systéme/processus	Non	Non

Les *Contacts authentifiés** ont les droits suivants sur chaque **service** dont ils sont un contact (mais pas sur ceux dont ils ne sont pas un contact)

- Droit de voir l'état du service
- Droit de voir la configuration du service
- Droit de voir l'historique et les notifications de ce service
- Droit de passer des commandes é ce service

*Les Contacts authentifiés** ont les droits suivants sur chaque **hte** dont ils sont un contact (mais pas sur ceux dont ils ne sont pas un contact)

- Droit de voir l'état de l'hte
- Droit de voir la configuration de l'hte
- Droit de voir l'historique et les notifications de cet hte

- Droit de passer des commandes é cet hte
- Droit de voir l'état de tous les services de cet hte
- Droit de voir la configuration de tous les services de cet hte
- Droit de voir l'historique et les notifications de tous les services de cet hte
- Droit de passer des commandes é tous les services de cet hte

Il est important de noter que par défaut**, personne** n'a le droit de

- Voir le fichier journal brut via la CGI d'affichage du journal
- Voir les informations sur le processus de Nagios via le CGI d'informations complémentaires
- Passer des commandes é Nagios via la CGI de commande
- Voir les définitions des groupes d'htes, contacts, groupes de contacts, périodes, et commandes via la CGI de configuration

Vous aurez sans doute besoin de ces informations, c'est pourquoi vous devrez vous donner des droits d'accés supplémentaires (et éventuellement é d'autres utilisateurs) comme décrit ci-dessous

Donner des droits d'accés supplémentaires aux informations des CGI

Vous pouvez donner aux *contacts authentifiés* et autres *utilisateurs authentifiés* des droits d'accés é d'autres informations des CGI en les ajoutant é diverses variables d'autorisation dans le fichier de configuration des CGI. Je m'aperéois que ces variables ne permettent pas d''tre trés précis dans les autorisations, mais c'est mieux que rien.

Des droits supplémentaires peuvent ’tre accordés aux utilisateurs en les ajoutant aux variables suivantes

- autorisé pour les informations systéme
- autorisé pour les commandes systéme
- autorisé pour les informations de configuration
- autorisé pour tous les htes
- autorisé pour toutes les commandes aux htes
- autorisé pour tous les services
- autorisé pour toutes les commandes aux services

Autorisations requises par les CGI

Si les droits d'accés aux diverses informations des CGI ne vous paraissent pas clairs, lisez la section *Autorisations requises* de chaque CGI qui se trouve ici.

Authentification sur des serveurs web sécurisés

Si votre serveur web se trouve dans un domaine sécurisé (i.e., derriére un firewall) ou si vous utilisez SSL, vous pouvez définir un utilisateur par défaut pour accéder aux CGI. C'est le rle de la variable default_user_name du fichier de configuration des CGI. En définissant un nom d'utilisateur par défaut, vous pouvez autoriser les utilisateurs é accéder aux CGI sans qu'ils ne s'authentifient auprés du serveur web. Ceci vous permet d'éviter d'utiliser l'authentification web de base, qui transmet les mots de passe en clair sur Internet.

Important : *Ne définissez pas* d'utilisateur par défaut, é moins que vous n'utilisiez un serveur web sécurisé et que vous soyez sr que les utilisateurs qui accédent aux CGI ont été authentifiés d'une maniére ou d'une autre ! Si vous utilisez cette variable, un utilisateur non authentifié héritera des droits de cet utilisateur par défaut !

Configuration des informations étendues

Que signifient les informations étendues?

Les informations étendues correspondent é des définitions d'attributs *optionnels* d'htes et services utilisés dans les CGI pour :

- Fournir des URLs pour des informations additionnelles é propos des htes et services
- Ajouter de jolies icnes aux htes et services affichés dans l'interface web interface
- Positionner les htes dans les CGIs statusmap et statuswrl aux positions 2-D et 3-D définies par l'utilisateur

O sont définis les Informations étendues?

Les définitions des informations étendues sont stockées dans le fichier de configuration des htes, services, contacts, etc Vous pouvez utiliser des patrons pour définir des attributs pour plusieurs htes et services facilement et rapidement.

Vérifier votre configuration de Nagios

Vérification de la Configuration en Ligne de Commande

Une fois que vous avez saisi toutes les informations nécessaires dans le fichier de configuration, il est temps de faire un contrle "sanitaire". Tout le monde commet des erreurs de temps en temps, il vaut donc mieux vérifier ce que vous avez saisi. Nagios exécute automatiquement un "contrle avant le décollage" avant de commencer é superviser, mais vous pouvez aussi lancer ce contrle manuellement avant d'essayer de lancer la supervision. Pour cela, vous devez, sur la ligne de commande, lancer Nagios avec l'argument de ligne de commande **-v** comme suit

/usr/local/Nagios/bin/Nagios -v <fichier_de_configuration_principal>

Notez que vous devez spécifier le chemin/nom de votre fichier de configuration principal (i.e.*/usr/local/Nagios/etc/Nagios.cfg*) comme second argument. Nagios lira votre fichier de configuration principal et tous vos fichiers de configuration des objets et vérifiera qu'ils contiennent des données valides.

Relations Vérifiées au Cours du "Contrle Avant Décollage"

Au cours du "contrle avant décollage", Nagios vérifie que vous avez défini les relations entre les données nécessaires é la supervision. Les objets sont tous liés et doivent ’tre correctement définis pour que les choses fonctionnent. Voici une liste des contrles de base que tente d'effectuer Nagios avant de commencer la supervision

1. Vérifier que tous les contacts sont membres d'au moins un groupe de contacts.
2. Vérifier que tous les contacts spécifiés dans chaque groupe de contacts sont valides.
3. Vérifier que tous les htes sont membres d'au moins un groupe d'htes.
4. Vérifier que tous les htes spécifiés dans chaque groupe d'htes sont valides.
5. Vérifier que tous les htes sont associés é au moins un service.
6. Vérifier que toutes les commandes utilisées dans les contrles de services et d'htes sont valides.
7. Vérifier que toutes les commandes utilisées dans les gestionnaires d'événement de services et d'htes sont valides.
8. Vérifier que toutes les commandes utilisées dans les notifications de contacts, services et htes sont valides.
9. Vérifier que toutes les périodes de notification spécifiées pour les services, htes et contacts sont valides.
10. Vérifier que toutes les périodes de contrle de service spécifiées pour les services sont valides.

Correction des Erreurs de Configuration

Si vous avez oublié de saisir des données critiques ou si vous vous ’tes tout simplement mélangé les pinceaux, Nagios affichera un message d'alerte ou d'erreur vous donnant la localisation de l'erreur. Les messages d'erreur contiennent généralement la ligne du fichier de configuration qui semble ’tre é l'origine du probléme. En cas d'erreur, Nagios sortira souvent du "contrle avant décollage" et retournera é l'invite de commande aprés avoir affiché seulement la premiére erreur qu'il a rencontrée. Ceci afin qu'une erreur n'en entrane pas de multiples autres au fur et é mesure que le reste de la configuration est vérifiée. Si vous recevez des messages d'erreur, vous devrez modifier vos fichiers de configuration pour y remédier. Les messages d'avertissement peuvent généralement ’tre ignorés sans risque, car il s'agit de recommandations et non d'obligations.

Que faire maintenant

Une fois que vous aurez vérifié et corrigé vos fichiers de configuration, vous pouvez ’tre assuré que Nagios commencera la supervision des services que vous avez spécifié. Passons maintenant é Démarrer Nagios!

Démarrer Nagios

IMPORTANT : Avant de démarrer Nagios, vous devez ’tre sr de l'avoir configuré correctement et d'avoir vérifié les données de configurations !

Méthodes pour démarrer Nagios

Il y a quatre moyens différents de lancer Nagios:

1. Manuellement, en tant que processus prioritaire (utile pour les tests initiaux et le débuggage)
2. Manuellement, en tant que processus d'arriére-plan
3. Manuellement, en tant que un démon
4. Automatiquement au démarrage (boot) du systéme

Examinons briévement chacune des méthodes

Lancer Nagios manuellement en tant que processus prioritaire

Si vous avez validé les options de débuggage dans le script de configuration (et recompilé Nagios), cela devrait ’tre votre premier choix pour le test et le débuggage. Lancer Nagios comme processus prioritaire dans un terminal de commande vous permettra de voir plus facilement ce qui se passe dans les processus de supervision et de notification. Pour lancer Nagios en tant que processus prioritaire, tapez ceci dans un terminal :

```
/usr/local/nagios/bin/nagios <main_config_file>
```

Remarquez que vous devez spécifier le chemin d'accés au fichier de configuration principal (c.-é-d. */usr/local/nagios/etc/cfg*) sur la ligne de commande.

Pour arr’ter Nagios, tapez simplement CTRL-C. Si vous avez activé les options de débuggage, vous souhaiterez probablement rediriger la sortie vers un fichier afin de le consulter plus facilement plus tard.

Lancer manuellement Nagios en processus d'arriére-plan

Pour démarrer Nagios en processus d'arriére-plan, lancez-le comme suit

```
/usr/local/nagios/bin/nagios <main_config_file> &
```

Remarquez que vous devez spécifier le chemin d'accés au fichier de configuration principal (c.-é-d. */usr/local/nagios/etc/cfg*) sur la ligne de commande.

Lancer manuellement le démon Nagios

Pour lancer manuellement Nagios sous forme de démon, vous devez préciser l'option **-d** sur la ligne de commande comme suit

```
/usr/local/nagios/bin/nagios -d <main_config_file>
```

Remarquez que vous devez spécifier le chemin d'accés au fichier de configuration principal (c.-é-d. */usr/local/nagios/etc/cfg*) sur la ligne de commande.

Lancer Nagios automatiquement au démarrage de la machine

Une fois que vous aurez testé Nagios et que vous serez é peu prés sr qu'il ne se plantera pas, vous souhaiterez probablement le lancer au boot de la machine. Pour cela (sous Linux), il faudra écrire un script de démarrage dans votre répertoire **/etc/rc.d/init.d/**. Il faudra également créer un lien vers ce script dans les runlevels pour lesquels Nagios doit démarrer. Je suppose que vous savez de quoi je parle et que vous ’tes capable de le faire.

Un exemple de script d'initialisation (appelé **daemon-init**) est créé dans le répertoire racine de la distribution de Nagios lorsque vous exécutez le script de configuration. Vous pouvez installer ce script d'exemple dans votre répertoire */etc/rc.d/init.d* grôce é la commande

```
make install-daemoninit
```

comme il est décrit dans les instructions d'installation.

Les exemples de scripts d'initialisation sont prévus pour tourner sous Linux, ce qui fait que si vous voulez les utiliser sous FreeBSD, Solaris, etc. Vous devrez peut-’tre les modifier un peu

Arr’t et redémarrage de Nagios

Les instructions relatives é l'arr’t et au redémarrage de Nagios se trouvent ici.

Arr’ter et Redémarrer Nagios

Une fois que Nagios fonctionne, vous pouvez avoir besoin d'arr’ter le processus ou de recharger les données de configuration "é la volée". Cette section décrit comment le faire.

IMPORTANT: Avant de redémarrer Nagios, assurez vous d'avoir vérifié les données de configuration é l'aide de la commande -v, *surtout* si vous avez fait des changements dans vos fichiers de configuration.

Arr’ter et Redémarrer avec le Script d'Initalisation

Si vous avez installé le script d'initialisation dans votre répertoire /etc/rc.d/init.d, vous pouvez arr’ter et redémarrer Nagios facilement. Si ce n'est pas le cas, passez cette section et lisez plus bas comment le faire manuellement. Dans les exemples ci-dessous, je supposerai que vous appelez le script d'initialisation **Nagios**.

Action souhaitée	Commande	Description
Arr’ter Nagios	`/etc/rc.d/init.d/nagios stop`	Arr’te Nagios
Redémarrer Nagios	`/etc/rc.d/init.d/nagios restart`	Arr’te puis redémarre un nouveau processus Nagios
Recharger les Données de Configuration	`/etc/rc.d/init.d/nagios reload`	Envoie un signal SIGHUP au processus Nagios, lui imposant de purger ses données actuelles de configuration, relire les fichiers de configuration et redémarrer la supervision

Arr’ter, redémarrer et recharger Nagios est vraiment simple avec un script d'initialisation et je vous recommande vivement de l'utiliser dans la mesure du possible.

Arr’ter et Redémarrer Manuellement Nagios

Si vous n'utilisez pas le script d'initialisation pour démarrer Nagios, vous devrez le faire manuellement. D'abord vous devrez trouver l'identifiant du processus Nagios et utiliser la commande *kill* pour arr’ter l'application ou recharger les données de configuration en envoyant le bon signal. Les instructions relatives é ces opérations se trouvent ci-dessous

Trouver l'identifiant du processus Nagios

Il faut d'abord connatre le numéro du processus de Nagios. Pour cela, taper juste la commande suivante :

```
ps axu | grep nagios
```

Vous devriez obtenir quelque chose qui ressemble a ca :

```
nagios  6808  0.0  0.7   840   352  p3 S    13:44   0:00 grep nagios
nagios 11149  0.2  1.0   868   488  ?  S   Feb 27   6:33 /usr/local/nagios/bin/
```

De ces informations, vous déduisez que Nagios a été lancé par l'utilisateur **nagios** et qu'il tourne sous l'identifiant de processus **11149**.

Stopper Manuellement Nagios

Pour arr'ter Nagios, utilisez la commande *kill* comme suit

```
kill 11149
```

Vous devrez remplacer **11149** par l'identifiant du processus Nagios de votre machine.

Redémarrer Manuellement Nagios

Si vous avez modifié les données de configuration, vous voudrez redémarrer Nagios et lui faire relire la nouvelle configuration. Si vous avez changé le code source et recompilé l'exécutable principal nagios, vous ne *devez pas* utiliser cette méthode. Il vaut mieux arr'ter Nagios avec la commande kill (cf. plus haut) et le redémarrer manuellement. Redémarrer Nagios en utilisant la méthode ci-dessous ne recharge pas réellement Nagios - cela le force é purger son fichier de configuration, relire le nouveau et reprendre la supervision. Pour redémarrer Nagios, vous aurez besoin d'envoyer le signal **SIGHUP** é Nagios. En supposant que Nagios tourne sous l'identifiant de processus 11149 (en prenant l'exemple ci-dessus), il faut utiliser la commande suivante :

```
kill -HUP 11149
```

Souvenez-vous que vous aurez besoin de remplacer **11149** avec l'identifiant du processus Nagios qui tourne sur votre machine.

Les plugins de Nagios

Qu'est-ce qu'un plugin ?

Les plugins sont des programmes compilés ou des scripts (Perl, shell, etc..) qui peuvent ’tre exécutés en ligne de commande pour tester l'état d'un hte ou d'un service. Nagios utilise le résultat des plugins pour déterminer le statut actuel des htes ou services sur le réseau. Vous ne pouvez pas vous passer des plugins - Nagios est inutile sans eux.

Récupération des plugins

Le développement des plugins pour Nagios est fait sur SourceForge. La page du projet de développement de plugins pour Nagios (o vous trouverez toujours les derniéres versions des plugins) se trouve sur http://sourceforge.net/projects/nagiosplug/.

Comment utiliser le plugin *x* ?

La documentation sur la maniére d'utiliser chaque plugin particulier *n'est pas* fournie avec la distribution principale de Nagios. Vous devez vous reporter é la derniére distribution des plugins pour toute information sur leur utilisation. Karl DeBisschop, principal développeur de plugins met l'accent sur les points suivants :

Tous les plugins qui respectent les consignes minimales de développement pour ce projet contiennent une documentation interne. Cette documentation peut ’tre affichée en exécutant le plugin avec l'option '-h' ('--help' si les paramétres longs sont activés). Si l'option '-h' ne fonctionne pas, c'est un bug.

Par exemple, si vous voulez savoir comment fonctionne le plugin check_http ou quelles options sont disponibles, vous devez essayez une des commandes suivantes :

```
./check_http --help
```

ou

```
./check_http -h
```

Exemples de définition de commande pour des services

Il est important de noter que les définitions de commandes contenues dans les fichiers d'exemples de configuration de la distribution principale de Nagios ne sont probablement *pas* en phase avec les paramétres du plugin utilisé. Ce sont de simples exemples de la faéon de définir des commandes.

Créer ses propres plugins

Créer ses propres plugin pour les adapter é des services ou htes particuliers est facile. Vous pouvez trouver des informations sur comment développer ses plugins sur http://sourceforge.net/projects/nagiosplug/. Vous trouverez le guide du développeur sur http://nagiosplug.sourceforge.net/developer-guidelines.html.

Compléments é Nagios

Voici une description de certains compléments disponibles pour Nagios. Ces ajouts ainsi que d'autres peuvent ’tre obtenus :

- Sur la page "downloads" (téléchargements) du site web de Nagios : www.nagios.org [Anglais]
- Sur le site regroupant la plupart des plugins pour Nagios : Nagios Exchange [Anglais].

Voici un aperéu de certains d'entre eux.

Plugins

- NRPE - Démon et plugin pour l'exécution de plugins sur des machines distantes.
- NSCA - Démon et programme client pour envoyer des résultats de contrles passifs via le réseau.

NRPE

NRPE - Démon et plugin pour l'exécution de plugins sur des machines distantes

Auteur: Ethan Galstad

Aperéu: Permet d'exécuter des plugins sur des machines distantes de maniére transparente et relativement aisée.

Fichiers:

check_nrpe
: Plugin utilisé pour envoyer des requ’tes sur l'agent nrpe de la machine distante.

nrpe
: PAgent qui tourne sur la machine distante et exécute les requ’tes du plugin.

nrpe.cfg
: Fichier de configuration pour les agents des machines distantes.

Description: Cet ajout est conéu pour permettre l'exécution de plugins sur une machine distante. Le plugin check_nrpe tourne sur la machine Nagios et est utilisé pour envoyer les requ’tes d'exécution du plugin é l'agent nrpe de la machine distante. L'agent nrpe exécutera le plugin approprié sur la machine distante, et retournera les données de sortie et le code de retour au plugin check_nrpe de la machine Nagios. Le plugin check_nrpe envoie la sortie du plugin distant et le code de retour é Nagios comme si c'était le sien. Cela permet d'exécuter les plugins de maniére transparente sur les machines distantes. L'agent nrpe peut soit fonctionner en mode démon standalone, soit comme un service inetd.

Notes:

- Quand il tourne en mode démon, l'agent nrpe identifie les requ’tes d'exécution des plugins en effectuant une comparaison rudimentaire de l'adresse IP de la machine appelante avec une liste d'adresses IP présentes dans le fichier de configuration.
- Quand il tourne sous inetd, des encapsuleurs TCP [TCP wrappers] peuvent ’tre utilisés pour restreindre l'accés é l'agent nrpe.

NSCA

Démon et programme client, pour envoyer des résultats de contrles passifs via le réseau

Autheur: Ethan Galstad

Aperéu: Permet d'envoyer des résultats de contrles passifs de services é un autre serveur sur le réseau sur lequel tourne Nagios.

Fichiers:

nsca
: Démon qui tourne sur le serveur central de Nagios et qui traite les résultats des contrles faits sur les services passifs soumis par les clients.

nsca.cfg
: Fichier de Configuration pour le démon nsca.

send_nsca
: Programme client exécuté é partir des machines distantes, et qui envoie des informations de contrles de services passifs au démon nsca sur le serveur principal Nagios.

send_nsca.cfg
: Fichier de configuration pour le client send_nsca.

Description: Cet ajout vous permet d'envoyer des résultats de contrles de services passifs é partir de machines distantes é une machine centrale sur laquelle Nagios tourne. Le client peut ’tre utilisé comme un programme séparé, ou peut ’tre intégré dans des serveurs NAGIOS distants qui exécutent la commande ocsp pour créer un environnement de contrles distribués. La communication entre le client et le démon peut ’tre cryptée par différents algorithmes (DES, 3DES, CAST, xTEA, Twofish, LOKI97, RJINDAEL, SERPENT, GOST, SAFER/SAFER+, etc.) si les bibliothéques mcrypt sont installées sur vos systémes.

Détermination de l'état et de l'accessibilité des htes du réseau

Supervision des services sur des htes hors fonction ou inaccessibles

Le but principal de Nagios est de superviser des services qui tournent sur ou sont proposés par des htes physiques ou des équipements de votre réseau. Il est évident que si un hte ou un équipement du réseau s'arr’te, tous les services qu'il offre s'arr’tent avec lui. De la m’me maniére, si un hte devient inaccessible, Nagios ne pourra pas superviser les services associés é cet hte.

Nagios reconnat cette situation et tente de vérifier ce genre de scénario quand un probléme survient sur un service. Chaque fois qu'un contrle de service retourne un niveau d'état non-OK, Nagios essaiera de contrler si l'hte supportant ce service est "vivant". Pratiquement, ceci consiste é envoyer un ping é l'hte et é vérifier si une réponse est retournée. Si la commande de contrle de l'hte retourne un état non-OK, Nagios suppose qu'il y a un probléme lié é l'hte. Dans ce cas, Nagios "taira" toutes les alarmes potentielles pour les services qui tournent sur cet hte et se contentera de notifier les contacts appropriés que l'hte est hors fonction ou inaccessible. Si la commande de contrle de l'hte retourne l'état OK, Nagios verra que l'hte est en fonction et enverra une alerte pour le service qui présente un probléme.

Htes locaux

Les htes "locaux" sont ceux qui se trouvent sur le m’me segment de réseau que l'hte qui héberge Nagios - aucun routeur ou firewall ne se trouve entre eux. La figure 1 représente un exemple de topologie de réseau. L'hte A fait tourner Nagios et supervise tous les autres htes ou routeurs représentés sur le schéma. Les htes B, C, D, E et F sont tous considérés comme "locaux" par rapport é l'hte A.

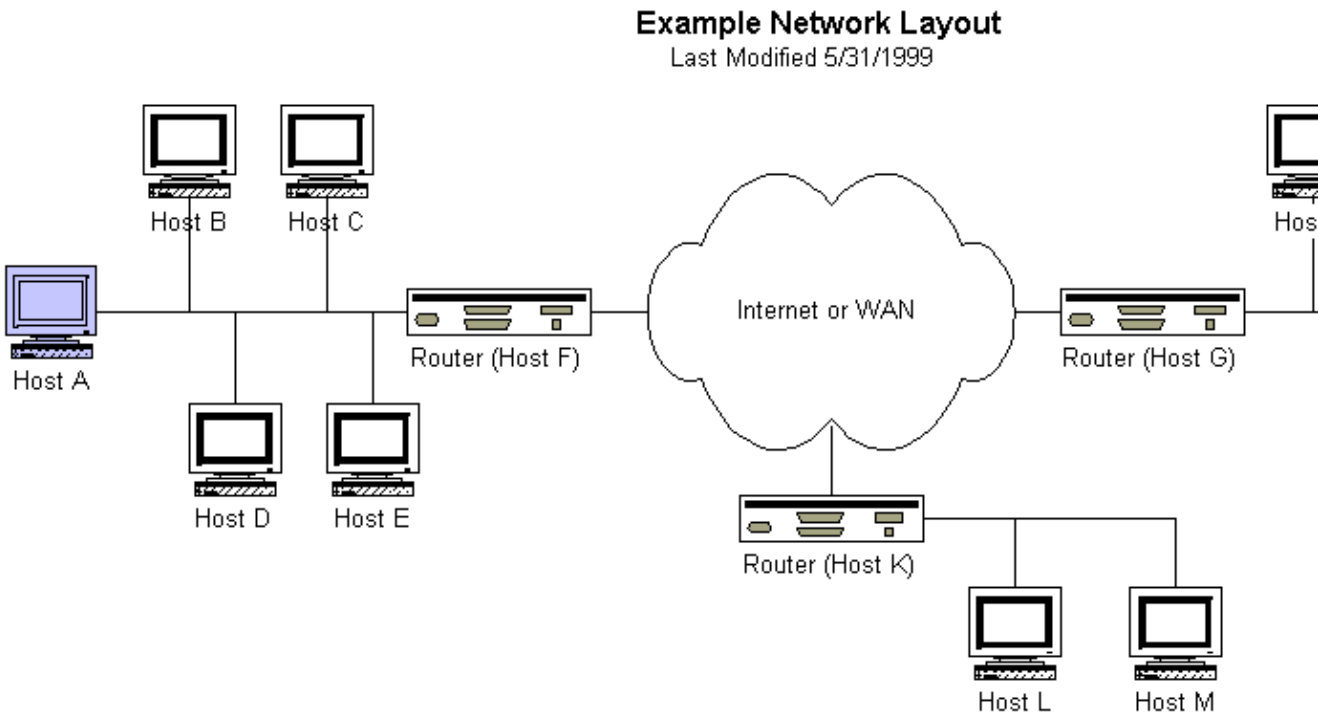
L'option <*parents*> de la définition d'hte pour un hte "local" doit ’tre laissée vide, car les htes locaux n'ont pas de dépendances ou de "parents" - c'est ce qui les rend locaux.

Supervision d'htes locaux

Contrler des htes qui sont sur votre réseau local est assez simple. Sorti de quelqu'un qui débranche accidentellement (ou intentionnellement) le côble réseau d'un de vos htes, peu de choses peuvent causer un probléme lors du contrle de la connexion au réseau. Il n'y a pas de routeurs ou de réseaux externes entre l'hte chargé de la supervision et les autres htes du réseau local.

Si Nagios a besoin de contrler qu'un hte est "vivant", il lancera simplement la commande de contrle de cet hte. Si la commande retourne un état OK, Nagios suppose que l'hte est en fonction. Si la commande retourne n'importe quel autre état, Nagios suppose que l'hte est hors fonction.

Figure 1.



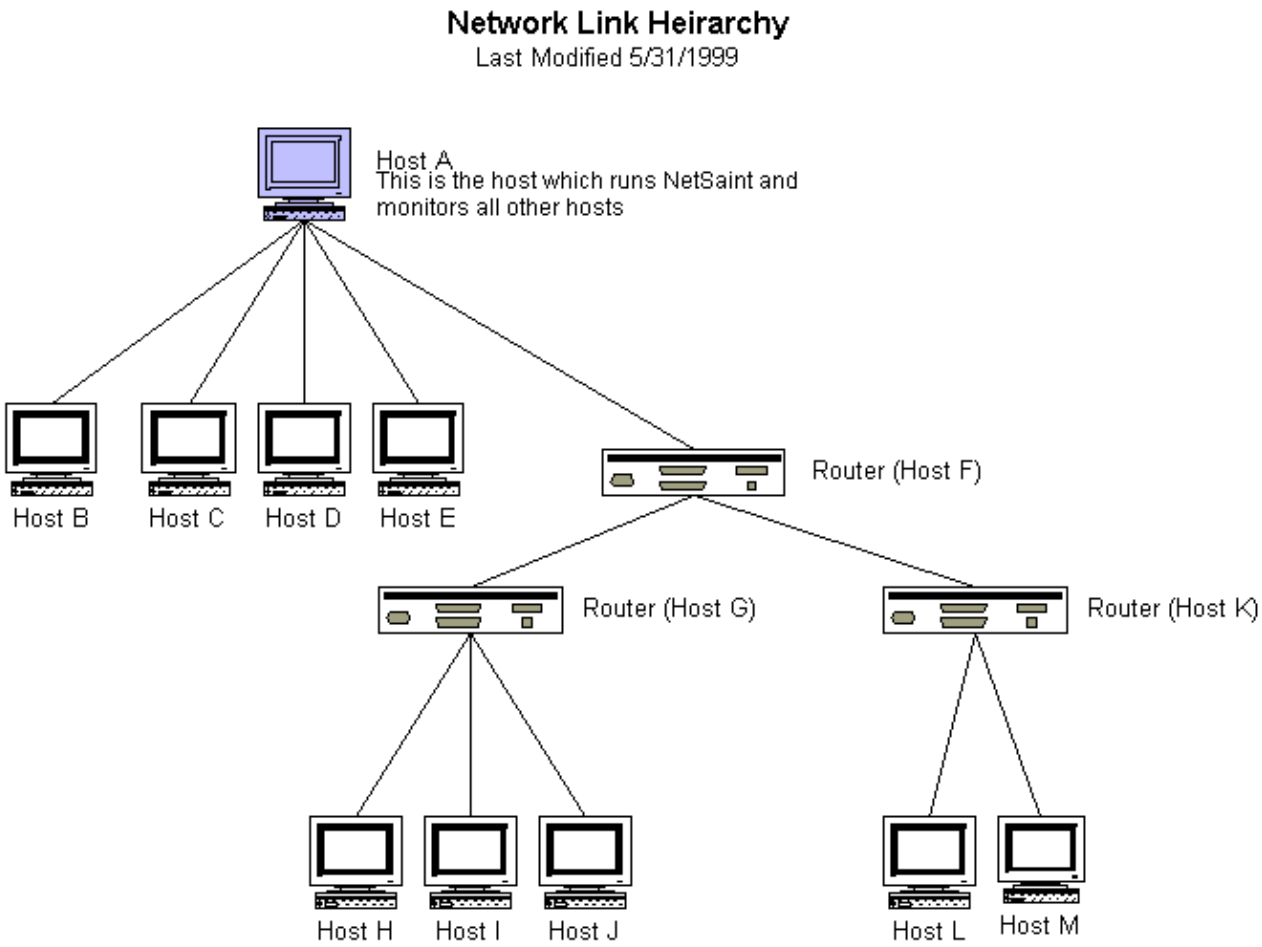
Htes distants

Les htes "distants" sont ceux qui se trouvent sur un segment de réseau différent de celui de l'hte qui héberge Nagios. Dans le schéma ci-dessus, les htes G, H, I, J, K, L et M sont tous considérés comme "distants" par rapport é l'hte A.

Notez que certains htes sont "plus loin" que d'autres. Les htes H, I et J se trouvent un saut [NdT : hop] plus loin de l'hte A que l'hte G (le routeur). De cette constatation, nous pouvons déduire un arbre de dépendance des htes comme indiqué figure 2. Cette arbre va nous aider é configurer chaque hte dans Nagios.

L'option **<parents>** de la définition d'hte d'un hte "distant" doit ’tre le nom court de l'hte directement au-dessus dans l'arbre de dépendance (comme indiqué ci-dessous). Par exemple, l'hte parent de l'hte H est l'hte G. Celui de l'hte G est F. F n'a pas d'hte parent, car il est sur le m’me segment de réseau que l'hte A - c'est un hte "local".

Figure 2.



Supervision d'htes distants

Contrler l'état d'htes distants est légérement plus complexe que pour des htes locaux. Si Nagios ne peut pas superviser des services distants, il doit déterminer si l'hte distant est hors fonction ou s'il est inaccessible. Heureusement, l'option **<parents>** permet é Nagios de le faire.

Si la commande de contrle d'un hte distant retourne un état non-OK, Nagios va parcourir l'arbre de dépendance (comme indiqué dans le schéma ci-dessus) jusqu'au sommet (ou jusqu'é ce que le contrle d'un hte parent retourne l'état OK). Ce faisant, Nagios peut déterminer si un probléme sur un service résulte de l'arr't d'un hte, de la rupture d'un lien réseau, ou est simplement une erreur du service.

Type de notification DOWN opposé é UNREACHABLE

Je reéois de nombreux courriels de personnes demandant pourquoi Nagios envoie des notifications au sujet d'htes inaccessibles. La réponse est que vous l'avez configuré pour qu'il le fasse. Si vous voulez désactiver les notifications UNREACHABLE pour les htes, modifiez le paramétre *notify_options* de chaque définition d'hte en supprimant le paramétre *u* ("unreachable"). Vous trouverez plus d'information dans cette FAQ [NdT : la FAQ n'existe plus dans cette version de la documentation].

Rupture de la continuité du réseau

Introduction

Le CGI de rupture a été fait pour aider au diagnostic des ruptures de la continuité du réseau. Pour des réseaux de petite taille, ce CGI peut ne pas ’tre trés utile, mais pour les plus grands, il le sera. Le diagnostic des ruptures de la continuité du réseau aidera les administrateurs é trouver et résoudre plus rapidement les problémes qui causent le plus de dégôts au réseau.

Notez que le CGI de rupture n'essaiera pas de trouver la cause exacte du probléme, mais localisera plutt les htes de votre réseau qui semblent causer le plus de problémes. Creuser plus profondément le probléme est laissé aux bons soins de l'utilisateur, car le nombre de causes possibles pour un probléme est illimité.

Diagrammes

Les diagrammes ci-dessous montrent comment le CGI de rupture fonctionne pour déterminer leur cause. Vous pouvez cliquer sur une image pour l'obtenir en plus grand format

Diagramme 1

Ce diagramme sera la base de notre exemple. Tous les htes apparaissant en rouge sont soit hors fonction, soit inaccessibles (du point de vue de Nagios). Tous les autres sont en fonctionnement.

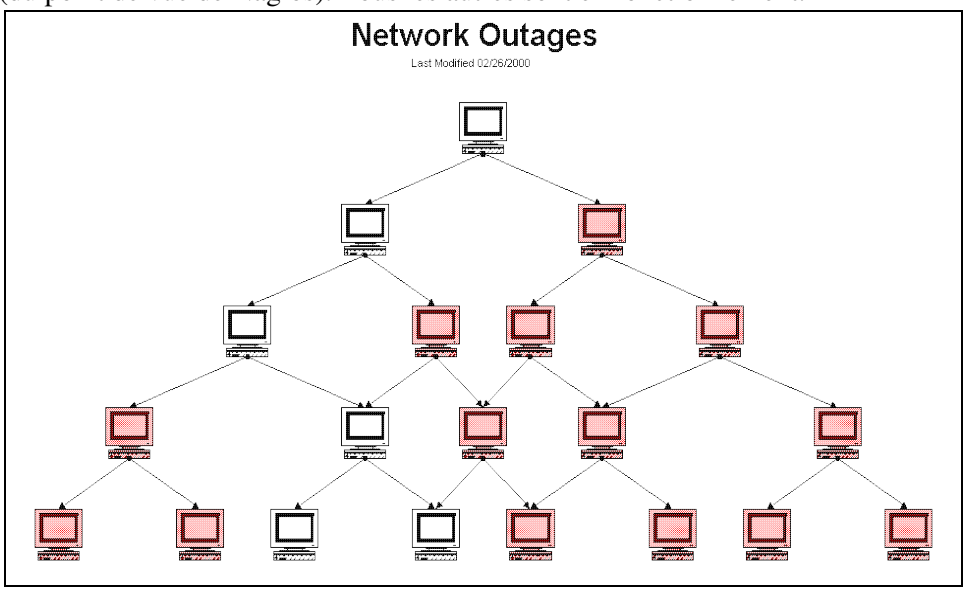
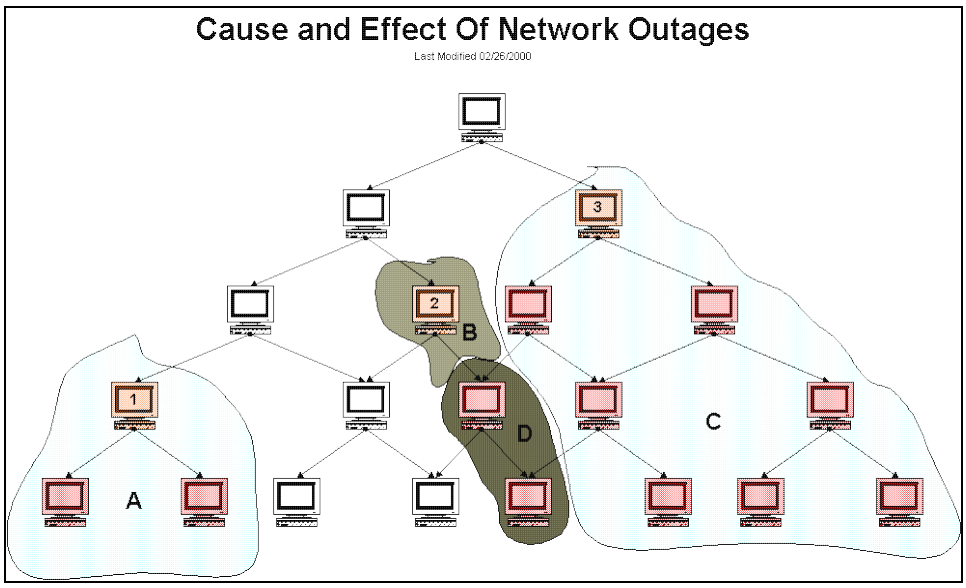


Diagramme 2

Ce diagramme met en exergue les causes de rupture du réseau (du point de vue de Nagios), et montre les divers groupes d'htes affectés par le probléme.



Détermination de la cause d'une rupture du réseau

Maintenant, comment le CGI de rupture détermine-t'il les htes é la source du probléme ? *Les htes "é probléme" doivent ’tre dans l'état DOWN ou UNREACHABLE* ***et*** *au moins un de leur parents immédiats doit ’tre dans l'état UP*. Les htes remplissant ce critére sont marqués comme étant une cause potentielle de la rupture.

Pour déterminer si ces htes marqués sont la cause de la rupture du réseau, nous devons procéder é d'autres tests

Si *tous* les enfants immédiats d'un de ces htes marqués sont DOWN ou UNREACHABLE *et* qu'ils n'ont aucun parent immédiat qui soit UP, l'hte marqué est la cause de la rupture du réseau. Si ne serait-ce qu'un des enfants immédiats d'un hte marqué *ne remplit pas* ces conditions, alors l'hte marqué *n'est pas* la cause de la rupture du réseau.

Détermination des effets d'une rupture du réseau

En plus de vous dire quel hte est é l'origine d'une rupture du réseau, le CGI de rupture vous dira aussi combien d'htes et de services sont affectés par un hte posant probléme. Comment cela est-il déterminé ? Regardez le diagramme 2 ci-dessus

Le diagramme montre clairement que l'hte 1 bloque 2 enfants (dans le domaine A). L'hte 2 est seulement responsable de son propre blocage (domaine B) et l'hte 3 est responsable du blocage de 7 htes (domaine C). Les effets de la rupture sur les deux htes du domaine D sont "partagés" entre les htes 2 et 3, car il n'est pas possible de déterminer la cause réelle de la rupture. Si un des htes 2 ou 3 était UP, alors les htes du domaine D pourraient ne pas ’tre bloqués.

Le nombre d'htes affectés par chaque probléme se décompose comme suit (l'hte causant probléme étant inclus dans ces chiffres) :

- Hte 1 : 3 htes affectés
- Hte 2 : 3 htes affectés
- Hte 3 : 10 htes affectés

Classement des problémes par niveau de gravité

Le CGI de rupture affiche tous les htes é probléme, qu'ils causent des ruptures du réseau ou non. Quoiqu'il en soit, le CGI vous dira combien parmi les htes é problémes (s'il y en a) causent des ruptures du réseau.

Pour afficher les htes é problémes de maniére utile, le tri est effectué selon la gravité de la répercussion de leurs effets sur le réseau. Le niveau de gravité est déterminé par deux éléments : le nombre d'htes affectés par le probléme et le nombre de services affectés. Les htes pésent plus lourd que les services dans le niveau de gravité. La version actuelle répartit le poids é 4 contre 1 (c.-é-d. que les htes sont 4 fois plus importants que les services).

En supposant que tous les htes du diagramme 2 ont le m’me nombre de services associés, l'hte 3 serait classé comme posant le probléme le plus grave, alors que les htes 1 et 2 auraient le m’me niveau de gravité.

Notifications

Introduction

J'ai reéu de nombreuses questions sur le fonctionnement des notifications. Ce document essaiera d'expliquer exactement quand et comment les notifications pour les htes et les services sont émises, et qui les reéoit.

Quand y a-t'il notification ?

La décision d'émettre des notifications est prise dans le cadre du contrle de service et du contrle d'hte. Les notifications d'hte et de service ont lieu dans les cas suivants

- Lors d'un changement d'état HARD. Pour plus d'informations sur les états et les changement d'état HARD, lisez ceci.
- Lorsqu'un hte ou un service demeure dans un état hard et non-OK, et que le délai défini dans le paramétre <*notification_interval*> de la définition de l'hte ou du service est écoulé depuis que la derniére notification a été émise (pour ce m'me hte ou service). Si vous ne voulez pas de notifications réitérées, mettez é 0 le paramétre <*notification_interval*> - ceci emp'che les notifications d''tre envoyées plus d'une fois pour un probléme donné.

Qui est notifié ?

Chaque définition de service comprend un paramétre <*contact_groups*> qui définit quels groupes de contacts recevront les notifications de ce service. Chaque groupe de contacts peut contenir un ou plusieurs contacts individuels. Quand Nagios émet une notification de service, il notifie chaque contact membre d'un des groupes de contacts spécifiés dans le paramétre <*contactgroups*> de la définition du service. Nagios est conscient qu'un contact peut 'tre membre de plus d'un groupe, donc il commence par supprimer les doublons.

Chaque définition d'hte comprend un paramétre <*contact_groups*> qui définit quels groupes de contacts recevront les notifications de cet hte. Quand Nagios émet une notification d'hte, il notifie chaque contact membre d'un des groupes de contacts é notifier pour cet hte. Nagios supprime les doublons de la liste des contacts avant toute chose.

Quels sont les filtres é traverser avant qu'une notifications ne soit émise ?

Le simple fait qu'une notification d'hte ou de service doive 'tre émise ne signifie pas que des contacts vont la recevoir. Il y a plusieurs filtres qu'une notification doit traverser avant d''tre jugée valable pour l'émission. M'me alors, des contacts peuvent ne pas la recevoir si leurs filtres de notification ne le permettent pas. Voyons en détail les filtres é traverser

Filtre global au programme :

Le premier filtre que les notifications doivent traverser est un test pour savoir si les notifications sont activées au niveau global du programme ou non. Ceci est spécifié par le paramétre enable_notifications du fichier de configuration principal, mais peut 'tre modifié en cours d'exécution via l'interface web. Si les notifications

sont désactivées de maniére globale, aucune notification ne sera envoyée - point final. Si elles sont activées, il y a encore d'autres tests é réussir

Filtres d'hte et de service :

Le premier filtre des notifications d'hte et de service consiste é vérifier que l'hte ou le service n'est pas dans une période d'arr't planifié. Si c'est le cas, **personne n'est notifié**. S'il n'est pas dans une période d'arr't planifié, la notification est passée au filtre suivant. Notez également que les notifications de services sont supprimées si l'hte auquel est associé le service est dans une période d'arr't planifié.

Le deuxiéme filtre des notifications d'hte et de service consiste é vérifier si l'hte ou le service oscille (é condition que vous ayez activé la détection d'oscillation). Si le service ou l'hte oscille, **personne n'est notifié**. Sinon, la notification est passée au filtre suivant.

Le troisiéme filtre é traverser pour les notifications d'hte et de service est formé par les paramétres de notification. Chaque définition de service contient des paramétres qui déterminent si les notifications doivent 'tre envoyées pour les états WARNING, CRITICAL, et RECOVERY. De la m'me maniére, chaque définition d'hte contient des paramétres qui déterminent si les notifications doivent 'tre envoyées quand l'hte s'arr'te [NdT: état DOWN], devient inaccessible [UNREACHABLE], ou se rétablit [RECOVERY]. Si la notification d'hte ou de service est bloquée par ces paramétres, **personne n'est notifié**. Dans le cas contraire, la notification est passée au filtre suivant Note : les notifications concernant les rétablissement d'htes et de services ne sont émises que si une notification a été envoyée é l'apparition du probléme. Cela n'a pas de sens de recevoir une notification de rétablissement pour un probléme dont vous n'aviez pas connaissance

Le quatriéme filtre des notifications d'hte et de service é traverser concerne la période. Chaque définition d'hte et de service comporte un paramétre <*notification_period*> qui spécifie quelle période contient les heures de notification valides pour cet hte ou service. Si le moment o la notification apparat n'est pas dans une plage valide de la période spécifiée, **personne n'est contacté**. Dans le cas contraire, la notification est passée au filtre suivant Note : si le filtre de période n'est pas traversé, Nagios réordonnancera la prochaine notification pour l'hte ou le service (s'il est dans un état non-OK) dans la prochaine plage de temps valide pour la période. Cela permet de s'assurer que les contacts sont notifiés des problémes dés que possible quand arrive le prochain moment valide de la période.

Le dernier jeu de filtres d'hte et de service est conditionnée par deux éléments : (1) une notification a déjé été émise par le passé concernant un probléme avec l'hte ou le service, et (2) l'hte ou le service est resté dans le m'me état non-OK depuis la derniére notification. Si ces deux conditions sont réunies, Nagios vérifie que le temps écoulé depuis l'émission de la derniére notification est supérieur ou égal é la valeur spécifiée par le paramétre <*notification_interval*> de la définition de l'hte ou du service. Si le temps écoulé depuis la derniére notification est insuffisant, **personne n'est contacté**. Si un temps suffisant s'est écoulé, ou si les deux conditions de ce filtre n'ont pas été réunies, la notification sera émise ! Le fait qu'elle parvienne ou non aux contacts individuels reléve d'un autre jeu de filtres

Filtres de contact :

A ce point la notification a traversé le filtre du mode de programme et tous les filtres d'hte et de service, et Nagios commence é envoyer des notifications é tous ceux qui doivent en recevoir. Cela signifie-t-il que tous les contacts vont recevoir la notification ? Non ! Chaque contact posséde son propre jeu de filtres que la notification doit traverser avant qu'ils ne la reéoivent. Note : les filtres de contact sont propres é chaque contact et n'affectent pas la faéon dont les autres contacts reéoivent les notifications.

Le premier filtre é passer pour chaque contact concerne les paramétres de notification. Chaque définition de contact comprend des paramétres qui déterminent si les notifications de service peuvent ’tre émises pour les états WARNING, CRITICAL, et RECOVERY. Chaque définition de contact contient également des options qui déterminent si les notifications d'hte peuvent ’tre émises lorsqu'un hte passe dans les états DOWN, UNREACHABLE, ou RECOVERY. Si la notification d'hte ou de service ne remplit pas ces conditions, **les contacts ne recevront pas de notification**. Dans le cas contraire, la notification est passée au prochain filtre Note : les notifications concernant les rétablissement d'htes et de services ne sont émises que si une notification a été envoyée é l'apparition du probléme. Cela n'a pas de sens de recevoir une notification de rétablissement pour un probléme dont vous n'aviez pas connaissance

Le dernier filtre é passer pour chaque contact concerne la période. Chaque définition de contact comprend un paramétre <*notification_period*> qui spécifie la période durant laquelle on peut envoyer des notification é ce contact. Si l'heure é laquelle la notification est faite n'est pas comprise dans la plage de temps de la période spécifiée, **le contact ne recevra pas la notification**. Dans le cas contraire, le contact la reéoit !

Pourquoi aucune méthode de notification n'est-elle intégrée directement é Nagios ?

On m'a plusieurs fois demandé pourquoi les méthodes de notification (pager, etc.) ne sont pas directement intégrées au code de Nagios. La réponse est simple - cela n'aurait pas beaucoup de sens. Nagios n'est pas prévu pour ’tre une application "tout-en-un". Si les contrles de service étaient intégrés dans le cur de Nagios, il serait trés difficile pour les utilisateurs d'ajouter de nouvelles méthodes de contrle, de modifier celles qui existent, etc. Les notifications fonctionnent de la m’me maniére. Il y a mille et une faéons d'envoyer des notifications et de nombreux modules ont déjé été développés pour faire le sale boulot, alors pourquoi réinventer la roue, et vous contenter d'un pneu de vélo ? Il est bien plus facile de déléguer é une application externe (c.-é-d. un simple script ou un systéme de messagerie complet) ce travail complexe. Des modules de messagerie qui peuvent traiter les notifications pour les pagers ou les téléphones portables sont listés ci-dessous, dans la section des ressources.

Macro de type de notification

Lorsque vous bricolez vos commandes de notification, vous devez prendre en compte le type de notification qui se présente. La macro $NOTIFICATIONTYPE$ contient une chane de caractéres qui identifie précisément cela. Le tableau ci-dessous liste les valeurs possibles de cette macro et leur description :

Valeur	Description
PROBLEM	Un service ou un hte vient de passer (ou est encore) dans un état dénotant un probléme. S'il s'agit d'une notification de service, cela signifie que le service est dans un état WARNING, UNKNOWN ou CRITICAL. Si c'est une notification d'hte, cela signifie que l'hte est dans l'état DOWN ou UNREACHABLE.
RECOVERY	Un service ou un hte s'est rétabli. S'il s'agit d'une notification de service, cela signifie que le service vient de retourner dans l'état OK. Si c'est une notification d'hte, cela signifie que l'hte vient de reprendre l'état UP.
ACKNOWLEDGEMENT	Cette notification est liée é l'acquittement d'un probléme d'hte ou de service. Les notifications d'acquittement sont générées via l'interface web par les contacts de l'hte ou du service concerné.
FLAPPINGSTART	L'hte ou le service vient de commencer é osciller.
FLAPPINGSTOP	L'hte ou le service vient de s'arr’ter d'osciller.

Ressources utiles

Il y a bien des moyens de configurer Nagios pour envoyer des notifications. C'et é vous de décider de la (des) méthode(s) que vous voulez utiliser. Une fois ce choix fait, vous devrez installer les logiciels nécessaires, et définir les commandes de notification dans vos fichiers de configuration avant de pouvoir les utiliser. Voici quelques méthodes de notification possibles :

- Email
- Pager
- Téléphone (SMS)
- Message WinPopup
- Messageries instantanées Yahoo, ICQ, ou MSN
- Alertes sonores
- etc

En gros, tout ce que vous pouvez faire en ligne de commande peut ’tre adapté sous forme de commande de notification.

Si vous voulez envoyer une notification alphanumérique é votre pager ou é votre téléphone portable via email, vous trouverez peut-’tre les informations suivantes utiles [NdT : et si vous ’tes aux USA]. Voici quelques hyperliens vers divers fournisseurs de service de messagerie, qui expliquent comment envoyer des messages alphanumériques aux pagers et aux téléphones portables

- Cigular
- PageNet
- SprintPCS (téléphones SMS)

Si vous cherchez é envoyer des messages é votre pager ou é votre téléphone portable sans email, jetez un coup d'il aux applications suivantes. Elles peuvent ’tre utilisées en conjonction avec Nagios pour envoyer une notification via un modem quand un probléme arrive. De cette faéon, vous n’’tes pas dépendant de l'email pour émettre des notifications (gardez é l'esprit que l'email peut *ne pas* fonctionner s'il y a des problémes réseau). Je n'ai pas personnellement essayé ces applications, mais certains m'ont dit l'avoir fait avec succés

- Gnokii [Anglais] (logiciel SMS pour contacter des téléphones Nokia via le réseau GSM)
- QuickPage [Anglais] (logiciel de pager alphanumérique)
- Sendpage [Anglais] (logiciel de pager)
- SMS Client [Anglais] (outil en ligne de commande pour envoyer des messages aux pagers et aux téléphones portables)

Si vous désirez une méthode atypique, vous pouvez essayer de vous amuser avec les alertes sonores. Si vous voulez des alertes sonores sur le serveur de supervision (avec voix synthéthique), essayez Festival [Anglais]. Si vous voulez les recevoir sur une autre machine, testez le Network Audio system (NAS) [Anglais] et rplay [Anglais].

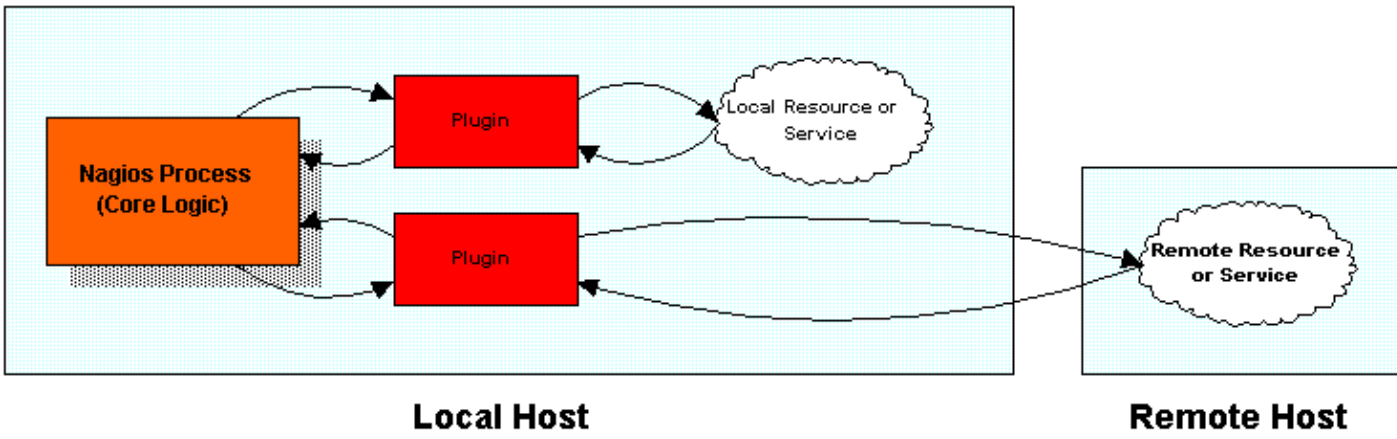
Enfin, il y a un espace dans la section des contributions de la page d'accueil de Nagios [Anglais] pour les scripts de notification réalisés par des utilisateurs. Vous pouvez y trouver votre bonheur, dans la mesure o ils gérent le sale boulot nécessaire é l'émission de notifications alphanumériques

Principe des plugins

Introduction

A la différence de beaucoup d'autres outils de supervision, Nagios ne dispose pas de mécanisme interne pour vérifier l'état d'un service, d'un hte, etc. A la place, il utilise des programmes externes (appelés plugins) pour faire le "sale boulot". Nagios exécutera un plugin dés qu'il a besoin de vérifier un service ou un hte supervisé. Le plugin fait *quelque chose* (remarquez ce terme trés général) pour exécuter le contrle et simplement renvoyer le résultat é Nagios. Nagios analysera le résultat provenant du plugin et prendra les mesures nécessaires (lancer les gestionnaires d'événements, envoyer des notifications, etc).

L'image ci-dessous montre comment les plugins sont séparés du coeur logique de Nagios. Nagios exécute les plugins qui vérifient ensuite des services ou des ressources locales ou distantes. Lorsque les plugins ont fini de vérifier la ressource ou le service, ils transmettent simplement le résultat de la vérification é Nagios. Un schéma plus complexe sur la faéon de travailler des plugins se trouve dans la documentation é la page des contrles de services passifs.



Avantage

Grôce é cette architecture, vous pouvez contrler n'importe quoi, du moment que vous y pensez. Si vous pouvez automatiser le processus de contrle de quelque chose, vous pouvez le superviser avec Nagios. Il existe déjé beaucoup de plugins crées pour superviser les ressources basiques telles que : la charge du processeur, l'espace d'un disque dur, les réponses aux pings, etc. Si vous voulez superviser quelque chose d'autre, regardez la documentation sur la page comment écrire ses plugins et lancez vous !

Inconvénient

Le seul réel inconvénient é cette architecture de plugin est le fait que Nagios n'a absolument aucune idée de ce que vous supervisez. Vous pouvez trés bien superviser des statistiques de trafic réseau, des taux d'erreur, la température d'une piéce, la tension du CPU, la vitesse des ventilateurs, la charge du processeur, l'espace disque, ou la capacité de votre fantastique toaster a bien griller votre pain pour le matin Nagios ne peut pas créer des graphes des ressources que vous supervisez. Il peut seulement suivre les changements d'*état* de ces ressources. Seul les plugins savent exactement ce qu'ils supervisent et comment ils réalisent les vérifications. Cependant les plugins peuvent retourner optionnellement des informations de performances liés aux données

en m’me temps que l'état. Ces information peuvent ’tre ensuite passées é une application externe qui peut créer des graphes des services (i.e. utilisation des espaces disques, charge du processeur, etc.). Vous trouverez plus d'informations é ce sujet ici.

Utilisation des Plugins pour Contrler des Services

La corrélation entre les plugins et les contrles de services devrait ’tre évidente. Quand Nagios a besoin de vérifier létat d'un service que vous avez défini, il exécutera le plugin spécifié dans l'argument <*check_command*> de la définition du service. Le plugin vérifiera létat du service ou de la ressource et retournera le résultat é Nagios.

Utilisation des Plugins pour contrler des htes

L'utilisation des plugins pour vérifier les états d'un hte peut ’tre un peu plus difficile é comprendre. Dans chaque définition d'hte vous utilisez l'argument <*host_check_command*> pour spécifier un plugin qui devra ’tre utilisé pour vérifier l'état de l'hte. Les contrles des htes ne sont pas effectués réguliérement : ils sont exécutés uniquement au besoin, typiquement lorsqu'il il y a des problémes avec un ou plusieurs services associes é l'hte.

Les contrles des htes peuvent s'effectuer avec les m’mes plugins que les contrles des services. La seule réelle différence entre les deux types de contrles est l'interprétation des résultats du plugin. Si le résultat d'un plugin utilisé pour le contrle d'un hte est un état différent de OK, alors Nagios croira que l'hte n'est pas actif.

Dans la plupart des cas, vous voudrez utiliser un plugin permettant de contrler si l'hte est répond é une requ’te ICMP (i.e. il répond au "ping") car c'est la méthode la plus courante pour savoir si un hte est actif ou non. Cependant, si vous supervisez un super grille pain, vous pourriez vouloir utiliser un plugin [NdT : check_paing :-) ?] afin de contrler si les résistances fonctionnent lorsqu'on abaisse la manette. Cela donnerait une bonne indication de l'état du grille-pain, 'vivant' ou non.

Ordonnancement du contrle des services

Index

Introduction

Jai reéu de nombreuses questions concernant la faéon dont les contrles de service sont ordonnancés dans certaines situations, ainsi que la différence dordonnancement entre lexécution réelle des contrles et le traitement de leur résultat. Cette page détaille ces mécanismes particuliers.

Options de configuration

Avant de commencer, il y a plusieurs options de configuration qui affectent la maniére dont les contrles de service sont ordonnancés, exécutés, et traités. Pour les débutants, je rappelle que chaque définition de service contient trois options qui déterminent quand et comment chaque contrle de service est ordonnancé et exécuté. Ces trois options sont :

- *intervalle de contrle [normal_check_interval]*
- *intervalle de réessai [retry_check_interval]*
- *période de contrle [check_period]*

Il y a également quatre options de configuration dans le fichier de configuration principal qui affectent les contrles de services. Ce sont :

- méthode de délai inter-contrles [service_inter_check_delay_method]
- facteur dentrelacement des services [service_interleave_factor]
- nombre maximal de contrles simultanés [max_concurrent_checks]
- fréquence de consolidation des services [service_reaper_frequency]

Nous verrons plus en détails comment ces options affectent les contrles de service au fur et é mesure de notre explication. Tout dabord, voyons comment les services sont ordonnancés lorsque Nagios démarre ou redémarre

Ordonnancement initial

Quand Nagios (re)démarre, il essaie dordonnancer le contrle initial de tous les services de maniére é minimiser la charge imposée é lhte local et aux htes distants. Ceci est fait en espaéant les contrles initiaux, et en les entrelaéant. Lespacement des contrles (aussi appelé le délai inter-contrles) est utilisé pour minimiser/égaliser la charge sur lhte supportant Nagios et lentrelacement permet de minimiser/égaliser la charge imposée aux htes distants. Le délai inter-contrles et lentrelacement sont tous deux décrits plus loin.

Bien que les contrles des services soient initialement ordonnancés pour répartir la charge sur les htes, les choses vont évoluer é terme vers le chaos et ’tre un peu faites au hasard. Cela est d au fait que les services ne sont pas tous contrlés é la m’me fréquence, que certains services mettent plus longtemps que dautres é ’tre contrlés, que des problémes sur les htes et/ou les services peuvent modifier lordonnancement des contrles, etc. Au moins, nous essayons de faire partir les choses du bon pied. Avec un peu de chance, lordonnancement initial permettra de garder la charge assez équilibrée au fil du temps

Note : si vous voulez connatre lordonnancement initial, démarrez Nagios avec loption **-s**. Cela affichera des informations sommaires sur lordonnancement (délai inter-contrles, facteur dentrelacement, heure du premier et du dernier contrle de service, etc) et créera un nouveau journal des états qui montre é quelle heure exactement sont ordonnancés les contrles initiaux de chaque service. Comme cette option écrase le journal des états, vous ne devez pas lutiliser lorsquune autre instance de Nagios tourne. Nagios *ne démarre aucune supervision* lorsque cette option est utilisée.

Délai inter-contrles

Comme indiqué précédemment, Nagios tente déquilibrer la charge imposée é la machine qui le supporte en espaéant réguliérement les contrles de services. Le temps dattente entre deux contrles consécutifs est appelé délai inter-contrles. En donnant une valeur é la variable méthode de délai inter-contrles du fichier de configuration principal, vous pouvez modifier la faéon dont ce délai est calculé. Jexpliquerai comment le calcul "débrouillard" [smart] fonctionne, car cest le paramétre que vous utiliserez normalement.

En utilisant la valeur "débrouillard" [smart] pour la variable *service_inter_check_delay_method*, Nagios calculera le délai inter-contrles de la maniére suivante :

délai inter-contrles = (total des intervalles de contrle normaux de tous les services) / (nombre total de services)

Prenons un exemple. Supposons que vous ayez 1.000 services ayant chacun un intervalle normal de contrle de 5 minutes (normalement les services ne sont pas tous contrlés é la m’me fréquence, mais prenons le cas le plus simple). Le total des intervalles de contrle de tous les services est de 5.000 (1.000 * 5). Ce qui signifie que le délai moyen entre deux contrles de chaque service est de 5 minutes (5.000 / 1.000). Partant de cette information, nous réalisons que (en moyenne) il nous faut contrler 1.000 services toutes les 5 minutes. Cela signifie quil nous faut utiliser un délai inter-contrles de 0.005 minutes (0.3 secondes) lors de la répartition initiale des contrles de services. En espaéant chaque contrle de 0.3 secondes, nous pouvons plus ou moins garantir que Nagios ordonnance et/ou exécute 3 nouveaux contrles de services chaque seconde. En espaéant réguliérement les contrles dans le temps de cette maniére, nous pouvons espérer que la charge sur lhte qui supporte Nagios reste équilibrée.

Entrelacement des services

Comme décrit ci-dessus, le délai inter-contrles aide é répartir la charge que Nagios impose é lhte local. Quen est-il des htes distants ? Est-il nécessaire dy répartir également la charge ? Pourquoi ? Oui, cest important et oui, Nagios peut y contribuer. Répartir la charge sur les htes distants est particuliérement important avec la mise en place de la parallélisation des contrles de services. Si vous supervisez un grand nombre de services sur un hte distant et que les contrles ne sont pas répartis, lhte distant peut se croire victime dune attaque SYN sil y a de nombreuses connexions ouvertes sur le m'me port. De plus, tenter de répartir la charge sur les htes distants est une bonne chose

En donnant une valeur é la variable facteur dentrelacement des services du fichier de configuration principal, vous pouvez modifier le mode de calcul de lentrelacement. Jexpliquerai comment le calcul "débrouillard" fonctionne, comme cest probablement le mode que vous utiliserez en général. Vous pouvez toutefois utiliser un facteur dentrelacement prédéfini plutt que de laisser Nagios le calculer pour vous. Notez par ailleurs que si vous utilisez un facteur dentrelacement de 1, le contrle entrelacé des services est désactivé.

Quand vous utilisez la valeur "débrouillard" pour la variable *service_interleave_factor*, Nagios calculera un facteur dentrelacement selon la méthode suivante :

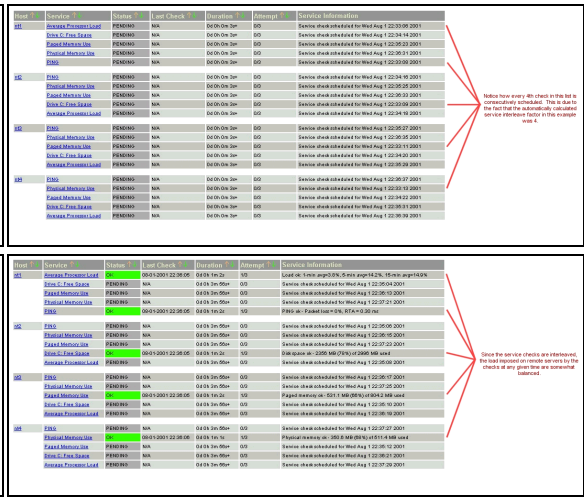
facteur dentrelacement = plafond(nombre total de services / nombre total dhtes)

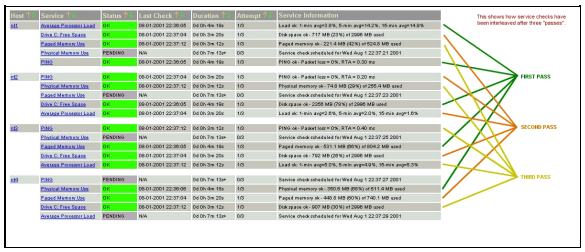
Prenons un exemple. Mettons que vous ayez un total de 1.000 services et 150 htes é superviser. Nagios calculera un facteur dentrelacement de 7. Ceci signifie que quand Nagios ordonnance les contrles initiaux des services, il ordonnancera le premier quil trouvera, sautera les 6 suivants, ordonnancera le suivant, et ainsi de suite Ce processus se répétera jusqué ce que tous les contrles de service soient ordonnancés. Comme les services sont triés (et donc ordonnancés) selon le nom de lhte auquel ils sont associés, cela aidera é minimiser/répartir la charge sur les htes distants.

Les deux images suivantes montrent comment les contrles de services sont ordonnancés quand ils sont ne sont pas entrelacés (*service_interleave_factor=1*) et quand ils sont entrelacés avec la variable *service_interleave_factor* égal é 4.

Non-Interleaved Checks:

Interleaved Checks:





Nombre maximal de contrles de services simultanés

Pour éviter que Nagios ne consomme toute votre CPU, vous pouvez restreindre le nombre maximal de contrles de services qui peuvent sexécuter é un instant donné. Ceci est défini grôce é loption nombre maximal de contrles de service simultanés *(max_concurrent_checks)* du fichier de configuration principal.

Le bon cté de cette option est que vous pouvez contrler lusage que fait Nagios de votre CPU. Le mauvais cté est que des contrles de services peuvent ne pas pouvoir sexécuter si cette valeur est trop basse. Quand arrive le moment de faire un contrle de service, Nagios sassurera que pas plus de *x* contrles de services ne sont en cours dexécution ou en attente de résultat (*x* étant la valeur que vous avez donné é loption *max_concurrent_checks*). Si cette limite est atteinte, Nagios retardera lexécution des contrles é faire jusqué ce que les précédents contrles aient terminé. Dans ce cas, comment déterminer une valeur raisonnable pour loption *max_concurrent_checks* ?

Tout dabord, rassemblez les informations suivantes

- Le délai inter-contrles quutilise Nagios pour ordonnancer initialement les contrles de services (utilisez largument **-s** de la ligne de commande pour le connatre)
- La fréquence (en secondes) des événements de consolidation des services, spécifiée par la variable fréquence de consolidation des services du fichier de configuration principal.
- Une idée du temps moyen dexécution des contrles de services (la plupart des plugins ont un timeout de 10 secondes, ce qui fait que la moyenne est probablement plus basse)

Ensuite, faites le calcul suivant pour déterminer une valeur raisonnable pour le nombre maximal de contrles simultanés

max. contrles simultanés = plafond(max(fréquence de consolidation des services, temps moyen dexécution dun contrle) / délai inter-contrles)

Le résultat devrait donner une valeur de départ raisonnable pour la variable *max_concurrent_checks*. Vous pouvez avoir é augmenter un peu cette valeur si les contrles de services continuent é dépasser lheure ordonnancée ou é la réduire si Nagios consomme trop de CPU.

Mettons que vous supervisez 875 services, chacun avec un intervalle de contrle de 2 minutes. Cela signifie que votre délai inter-contrles vaut 0,137 secondes. Si vous mettez la fréquence de consolidation des services é 10 secondes, vous pouvez calculer une valeur approximative pour le nombre maximal de contrles simultanés comme suit (je suppose que le temps dexécution moyen dun contrle est de moins de 10 secondes)

max. contrles simultanés = plafond(10 / 0,137)

Dans ce cas, le résultat est 73. Ceci est logique, car (en moyenne) Nagios exécutera un peu plus de 7 nouveaux contrles de service par seconde et il traite les résultats des contrles toutes les 10 secondes. Ce qui veut dire qué nimporte quel moment, il y aura un peu plus de 70 contrles de service en cours dexécution ou en

attente de traitement des résultats. Dans ce cas, je recommanderais de pousser la valeur du nombre maximal de contrles simultanés jusqué 80, étant donné quil y aura de lattente quand Nagios traite les résultats des contrles et fait le reste de son travail. Visiblement, vous devrez tester et régler les choses petit é petit pour que tout tourne correctement sur votre systéme, mais vous avez lé des régles de configuration générales

Restrictions temporelles

Loption *check_period* détermine la période durant laquelle Nagios peut exécuter des contrles de service. Sans parler de létat dans lequel se trouve un service, si lheure de son exécution réelle nest pas une heure valide telle que spécifiée dans la période, le contrle *ne sera pas* exécuté. Nagios réordonnancera alors le contrle é la prochaine heure valide de la période. Si le contrle peut ’tre exécuté (e.g. lheure est validée dans la période), le contrle est exécuté.

Note : m’me si un contrle de service ne peut pas ’tre exécuté é une certaine heure, Nagios peut quand m’me l*ordonnancer* é cette heure. Ce cas de figure se produira trés probablement durant lordonnancement initial des contrles de services, mais il peut aussi se produire dans dautre cas. Cela *ne signifie pas* que Nagios exécutera le contrle ! Au moment d*exécuter* un contrle de service, Nagios vérifiera que le contrle peut se dérouler é cette heure. Si ce nest pas le cas, Nagios nexécutera pas le contrle, mais le réordonnancera plus tard. Que cela ne vous induise pas en erreur ! Lordonnancement et lexécution des contrles sont deux choses clairement distinctes (bien quen relation).

Ordonnancement normal

Dans un monde parfait, vous naurez pas de problémes de réseau. Mais si cela arrivait, vous nauriez pas besoin dun outil de supervision réseau. Quoi quil en soit, quand tout tourne et que les services sont dans létat OK, nous appellerons cela "normal". Les contrles de service sont normalement ordonnancés é la fréquence spécifiée par loption *check_interval*. Cest tout. Simple, non ?

Ordonnancement en cas de probléme

Quarrive-t-il en cas de probléme sur un service ? Eh bien, entre autres lordonnancement des contrles est modifié. Si vous avez configuré loption *max_attempts* de la définition du service é plus de 1, Nagios contrlera le service é nouveau avant de remonter le probléme. Pendant que le service est é nouveau contrlé (jusqué *max_attempts* fois), il est considéré comme étant dans un état "soft" (comme décrit ici) et des contrles du service sont réordonnancés é la fréquence déterminée par loption *retry_interval*.

Si Nagios contrle le service *max_attempts* fois et quil le trouve toujours dans un état non-OK, Nagios mettra le service en état "hard", enverra des notifications aux contacts (le cas échéant), et commencera le ré ordonnancement des futurs contrles du service é la fréquence déterminée par loption *check_interval*.

Comme toujours, il y a des exceptions é la régle. Quand un contrle de service retourne un état non-OK, Nagios contrlera lhte auquel est associé le service pour déterminer sil est en fonction ou pas (voyez la note ci-dessous pour savoir de quelle faéon). Si lhte nest pas en fonction (i.e. il est soit hors fonction soit inaccessible), Nagios mettra immédiatement le service dans un état hard non-OK et remettra le compteur dessais é 1. Comme le service est dans un état hard non-OK, le contrle de service sera réordonnancé é la fréquence normale spécifiée par loption *check_interval*, plutt que celle de loption *retry_interval*.

Contrles dhtes

A la différence des contrles de services, les contrles dhtes *ne sont pas* ordonnancés de maniére réguliére. Ils sont lancés é la demande, quand Nagios en ressent le besoin. Cest une question courante des utilisateurs, qui a donc besoin d'tre clarifiée.

Un des cas o Nagios contrle létat dun hte est quand un contrle de service retourne un état non-OK. Nagios contrle lhte pour déterminer sil est en fonction, hors fonction, ou inaccessible. Si le premier contrle dhte retourne un état non-OK, Nagios continuera é demander des contrles de lhte jusqué ce que (a) le nombre maximal de contrles dhte (spécifié par loption *max_attempts* de la définition dhte) soit atteint, ou (b) un contrle de lhte retourne létat OK.

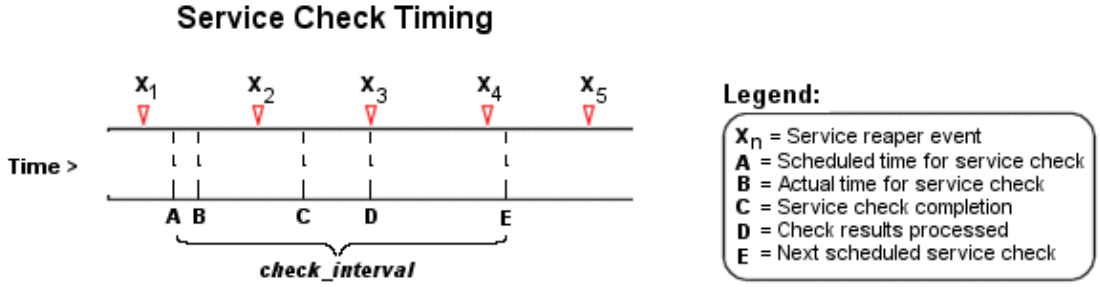
Notez également - quand Nagios contrle létat dun hte, il suspend tout autre action (exécution de nouveaux contrles de services, traitement des résultats des contrles de services, etc). Ceci peut ralentir un peu les choses et retarder les contrles de service pendant quelques temps, mais il est indispensable de déterminer létat de lhte avant que Nagios puisse agir sur le(s) service(s) posant probléme.

Délais dordonnancement

Il est é noter que lordonnancement et lexécution des contrles de service sont faits "au mieux". Les contrles de service individuels sont considérés comme étant des événements de basse priorité dans Nagios, ils peuvent donc 'tre retardés si des événements de haute priorité doivent 'tre exécutés. Sont par exemple des événements de haute priorité les rotations de fichier journal, les commandes de contrle externes, et les événements de consolidation de service. Qui plus est, les contrles dhtes ralentissent lexécution et le traitement des contrles de services.

Exemple dordonnancement

Lordonnancement de contrles de services, leur exécution, et le traitement de leurs résultats peuvent 'tre un peu ardus é comprendre, alors prenons un petit exemple. Regardez le diagramme ci-dessous - je my référerai en expliquant comment fonctionnent ces choses.



Tout dabord, les événements $\mathbf{X}_n$ sont des événements de consolidation de services qui sont ordonnancés é une fréquence spécifiée par loption fréquence de consolidation des services du fichier de configuration principal. Les événements de consolidation des services rassemblent et traitent les résultats des contrles de service. Ils sont au cur de Nagios, lanéant les contrles dhte, les gestionnaires dévénements et les notifications si nécessaire.

Dans notre exemple, un service a été ordonnancé au moment **A**. Toutefois, Nagios a pris du retard dans sa file dévénements, et donc le contrle na été exécuté quau moment **B**. Le contrle de service a terminé son exécution

au moment **C**, donc la différence entre les points **C** et **B** est le temps pendant lequel le contrle a effectivement tourné.

Les résultats du contrle de service ne sont pas traités immédiatement é la fin de lexécution. Ils sont sauvegardés pour un traitement ultérieur par lévénement de consolidation des services. Le prochain événement de ce type se produit au moment **D**, ce qui fait que cest approximativement le moment auquel le résultat est traité (le moment réel peut se trouver un peu plus tard que **D** dans la mesure o dautres résultats de contrle de service peuvent ’tre traités avant celui-ci).

Au moment o lévénement de consolidation des services traite le résultat du contrle de service, il réordonnance le prochain contrle de service et le place dans la file dévénements de Nagios. Supposons que le contrle de service retourne un état OK, le prochain moment de contrle **E** se trouve é lheure du contrle original plus le temps spécifié par loption *check_interval*. Notez que le service *nest pas* réordonnancé selon le moment o il a été effectivement exécuté ! Il y a é ceci une exception (ny en a til pas toujours ?) - si le moment ou le contrle de service est réellement exécuté (point **B**) se trouve aprés le prochain moment dexécution du contrle (point **E**), Nagios compensera en ajustant le prochain moment de contrle. Ceci permet de sassurer que Nagios ne semballera pas é essayer de maintenir les contrles de service en cas de charge importante. Qui plus est, quel intér’t dordonnancer quelque chose dans le passé?

Options de définition de service affectant lordonnancement

Chaque définition de service contient les options *check_interval* et *retry_interval*. Jespére ici clarifier le rle de ces options, leur relation avec loption *max_attempts* de la définition de service, et comment elles affectent lordonnancement du service.

Tout dabord, loption *check_interval* est lintervalle de temps entre deux contrles de service en temps "normal". "Normal" signifie lorsque le service est dans létat OK ou quand il est dans un état hard non-OK.

Quand un service passe pour la premiére fois de létat OK é létat non-OK, Nagios vous offre la possibilité daugmenter ou de réduire temporairement lintervalle de temps dont seront espacés les prochains contrles de ce service. Quand un service change détat pour la premiére fois, Nagios fera jusqué *max_attempts*-1 tentatives de contrle du service avant de décider que le probléme est avéré. Pendant que le service est contrlé é nouveau , il est ordonnancé selon loption *retry_interval*, qui peut ’tre plus courte ou plus longue que loption *check_interval*. Pendant que le service est contrlé (jusqué *max_attempts*-1 fois), il est dans un état soft. Si le service est contrlé *max_attempts*-1 fois et quil est encore en état non-OK, le service passe en état hard et est ensuite ré ordonnancé é la fréquence normale spécifiée dans loption *check_interval*.

A ce sujet, si vous spécifiez une valeur de 1 pour loption *max_attempts*, le service ne sera jamais contrlé é lintervalle spécifié par loption *retry_interval*. Il passera immédiatement dans un état hard et sera ensuite ré ordonnancé é la fréquence spécifiée par loption *check_interval*.

Types d'états

Introduction

L'état courant des services et des htes est déterminé par deux composants : l'état du service ou de l'hte (c.a.d OK, WARNING, UP, DOWN, etc) et le type d'état dans lequel il se trouve. Il y a deux types d'état dans Nagios - les états "soft" et les états "hard". Les types d'état sont une partie cruciale de la logique de supervision de Nagios. Ils sont utilisés pour déterminer quand les gestionnaires d'événement sont exécutés et quand les notifications sont envoyées.

Réessais de Contrle de Service et d'Hte

Pour éviter les alarmes indésirables, Nagios vous permet de définir combien de fois un contrle de service ou d'hte sera retenté avant que le service ou l'hte soit considéré comme ayant réellement un probléme. Le nombre maximum de tentatives avant qu'un service ou un hte soit considéré comme ayant un réel probléme est configuré par l'option <*max_check)attempts*> dans les définitions de service et d'hte. Le numéro de la tentative de contrle de service ou d'hte en cours détermine le type d'état dans lequel ce dernier se trouve. Il y a quelques exceptions é cela dans la logique de supervision, mais nous les ignorerons pour l'instant. Jetons un coup d'oeil aux différents types d'état

Etat Soft

Un service ou un hte est dans un état soft dans les situations suivantes

- Quand un contrle de service ou d'hte retourne un état non-OK et qu'il n'a pas encore été (re)contrlé autant de fois que le spécifie l'option <*max_check_attempts*> de la définition du service ou de l'hte. Appelons éa un état d'erreur soft
- Quand un service ou un hte se rétablit suite é un état d'erreur soft. Ceci est considéré comme un rétablissement soft.

Evénements d'Etat Soft

Que se passe-t'il lorsqu'un service ou un hte est dans un état d'erreur soft ou en rétablissement soft ?

- L'erreur ou le rétablissement soft est enregistré dans un log si vous avez activé les options log_service_retries ou log_host_retries dans le fichier de configuration principal.
- Les gestionnaires d'événement sont exécutés (si vous en avez défini) pour traiter l'erreur ou le rétablissement soft du service ou de l'hte. (Avant toute exécution de gestionnaire d'événement, les macros **$HOSTSTATETYPE$** ou **$SERVICESTATETYPE$** sont mises "**SOFT**").
- Nagios *n'envoie pas* de notifications aux contacts, car il n'y a pas (ou il n'y a pas eu) de "réel" probléme avec le service ou l'hte.

Comme vous pouvez le voir, la seule chose importante qui se passe lors d'un état soft est l'exécution des gestionnaires d'événement. L'utilisation des gestionnaires d'événement peut se révéler particuliérement pratique si vous voulez réparer préventivement un probléme avant qu'il ne passe en état hard. Vous trouverez plus d'informations sur les gestionnaires d'événement ici.

Etat Hard

Les états hards surviennent pour les services dans les situations suivantes (les états hard des htes sont décrits plus loin)

- Quand un contrle de service retourne un état non-OK et qu'il a été (re)contrlé autant de fois que spécifié par l'option <*max_check_attempts*> de la définition du service. C'est un état d'erreur hard.
- Quand un service se rétablit d'un état d'erreur hard. Ceci est considéré comme un rétablissement hard.
- Quand un contrle de service retourne un état non-OK et que l'hte correspondant est soit DOWN soit UNREACHABLE. C'est une exception é la logique générale de la supervision, mais c'est tout é fait cohérent. Si l'hte n'est pas UP, pourquoi essayer de recontrler le service?

Les états hard surviennent pour les *htes* dans les situations suivantes

- Quand un contrle d'hte retourne un état non-OK et qu'il a été (re)contrlé autant de fois que spécifié par l'option <*max_check_attempts*> de la définition de l'hte. C'est un état d'erreur hard.
- Quand un hte se rétablit suite é un état d'erreur hard. Ceci est considéré comme un rétablissement hard.

Changements d'Etat Hard

Avant d'expliquer ce qui se passe quand un hte ou un service est en état hard, vous devez connatre les changements d'état hard. Les changements d'état hard surviennent quand un service ou un hte

- passe de l'état hard OK é l'état hard non-OK
- passe de l'état hard non-OK é l'état hard OK
- passe d'un état hard non-OK quelconque é un état hard non-OK d'un autre type (i.e. d'un état hard WARNING é un état hard UNKNOWN)

Evénements d'Etat Hard

Que se passe-t'il lorsqu'un service ou un hte est dans un état d'erreur hard ou en rétablissement hard ? Eh bien, éa dépend s'il y a eu ou non un changement d'état hard (tel que décrit ci-dessus).

Si un changement d'état hard s'est produit *et* que le service ou l'hte est dans un état non-OK, les actions suivantes sont enclenchées

- Le probléme hard du service ou de l'hte est enregistré dans le log.
- Les gestionnaires d'événement sont exécutés (si vous en avez défini) pour traiter le probléme hard du service ou de l'hte. (Avant toute exécution de gestionnaire d'événement, les macros **$HOSTSTATETYPE$** ou **$SERVICESTATETYPE$** sont mises "**HARD**").
- Les contacts seront notifiés du probléme du service ou de l'hte (si la politique de notification le permet).

Si un changement d'état hard s'est produit *et* que le service ou l'hte est dans un état OK, les actions suivantes sont enclenchées.

- Le rétablissement hard du service ou de l'hte est enregistré dans le log.

- >Les gestionnaires d'événement sont exécutés (si vous en avez défini) pour traiter le rétablissement hard du service ou de l'hte. (Avant toute exécution de gestionnaire d'événement, les macros **$HOSTSTATETYPE$** ou **$SERVICESTATETYPE$** sont mises "**HARD**").
- Les contacts seront notifiés du rétablissement du service ou de l'hte (si la politique de notification le permet).

Si un changement d'état hard NE S'EST PAS produit et que le service ou l'hte est dans un état non-OK, les actions suivantes sont enclenchées

- Les contacts seront re-notifiés du probléme du service ou de l'hte (si la politique de notification le permet).

Si un changement d'état hard NE S'EST PAS produit *et* que le service ou l'hte est dans un état OK, il ne se passe rien. En effet, le service ou l'hte est dans un état OK et l'était aussi lors du dernier contrle.

Les Périodes

ou
"Est-ce le bon moment ?"

Introduction

Les périodes vous permettent une meilleure matrise sur le moment o les contrles de service peuvent ’tre lancés, celui o les notifications d'htes et de services peuvent ’tre envoyées, et celui o les contacts peuvent recevoir les notifications. Cette nouvelle fonctionnalité vient avec quelques problémes potentiels, que je l'expliquerai plus loin. J'ai été trés réticent au début é introduire les périodes é cause de ces "*snafus*" [NdT : acronyme de Situation Normal All Fucked Up, qui peut se traduire par : La situation est complétement normale, puisque tout le systéme est mort]. Je vous laisse décider de ce qui est bon dans votre propre cas

Comment les périodes fonctionnent pour les contrles de service

Sans l'implémentation des périodes, Nagios superviserait tous les services que vous auriez défini 24 heures sur 24, 7 jours sur 7. Bien que cela convienne é la plupart des services é superviser, cela ne fonctionne pas si bien pour certains. Par exemple, avez-vous réellement besoin de superviser les imprimantes en permanence alors qu'elles ne sont utilisées uniquement pendant les heures de bureau ? Peut-’tre avez-vous des serveurs de développement dont vous préféreriez qu'ils soient en fonctionnement, mais qui ne sont pas "critiques" et qui n'ont donc pas besoin d’’tre contrlés pendant le weekend. Les définitions de périodes vous permettent maintenant de mieux matriser les horaires de supervision de ces machines

L'argument <*check_period*> de chaque définition de service vous permet de spécifier une période qui renseigne Nagios sur le moment o le service doit ’tre contrlé. Quand Nagios essaie de réordonnancer un contrle de service, il s'assurera que la prochain vérification tombe dans une plage de temps valide é l'intérieur de la période définie. Dans le cas contraire, Nagios ajustera le moment du prochain contrle de service pour concider avec le prochain moment "valide" dans la période spécifiée. Cela signifie que le service peut ne pas ’tre recontrlé avant une heure, un jour, ou une semaine, etc.

Problémes potentiels liés aux contrles de services

Si vous utilisez des périodes qui ne couvrent pas une plage de 24h/24, 7j/7, vous aurez des problémes, surtout si un service (ou l'hte correspondant) est hors fonction alors que le contrle a été décalé au prochain moment valide de la période. Voici quelques uns de ces problémes

1. Les contacts ne seront pas notifié é nouveau de problémes sur un service jusqu'é ce que le prochain contrle puisse avoir lieu.
2. Si un service se rétablit é un moment qui a été exclu de la période de contrle, les contacts ne recevront pas de notification du rétablissement.
3. L'état du service ne changera pas (dans le journal des états et le CGI) jusqu'é ce qu'il puisse ’tre recontrlé.
4. Si tous les services associés avec un hte particulier ont la m’me période de contrle, les problémes de l'hte ou son rétablissement ne seront pas détectés jusqu'é ce qu'un des services puisse ’tre contrlé (et donc les notifications peuvent ’tre retardées ou ne pas ’tre envoyées du tout).

Limiter la période de contrle de service é autre chose que 24 heures sur 24, 7 jours sur 7 peut causer de nombreux problémes. En fait, pas tant des problémes que des désagréments et des inexactitudes A moins que vous n'ayez une bonne raison de le faire, je vous recommande fortement de définir dans l'argument <*check_period*> de chaque service une période de type "24x7".

Comment les périodes fonctionnent pour les notifications aux contacts

Le meilleur usage que vous puissiez probablement faire des périodes est de gérer le moment auquel les notifications peuvent ’tre envoyées aux contacts. En utilisant les arguments <*service_notification_period*> et <*host_notification_period*> dans les définitions de contact, vous pouvez définir une période de disponibilité pour chaque contact. Notez que vous pouvez définir des périodes différentes pour les notifications d'hte et de service. Ceci peut ’tre utile si vous voulez que les notifications d'htes soient envoyées au contact n'importe quel jour de la semaine, mais que les notifications de service ne soient envoyées que les jours ouvrables. Il faut savoir que ces deux périodes de notification doivent couvrir tous les moments o le contact peut recevoir la notification. Vous pouvez définir les périodes de notification pour des services ou des htes de maniére spécifique de la maniére suivante

En définissant l'argument <*notification_period*> de la définition d'hte, vous gérez les moments o Nagios est autorisé é envoyer des notifications concernant les problémes ou les rétablissements de cet hte. Quand une notification d'hte est pr’te é partir, Nagios s'assurera que le moment présent fait partie d'une plage valide de la période <*notification_period*>. Si c'est un moment valide, alors Nagios tentera de notifier chaque contact du probléme de l'hte. Certains contacts peuvent ne pas recevoir la notification d'hte si leur argument <*host_notification_period*> n'autorise pas les notifications d'htes é ce moment. Si le moment *n'est pas* valide au sein de l'argument <*notification_period*> défini pour l'hte, Nagios n'enverra la notification é *aucun* contact.

Vous pouvez spécifier les moments de notification pour les services de la m’me maniére que pour les htes. En définissant l'argument <*notification_period*> vous pouvez gérer les moments o Nagios est autorisé é envoyer des notifications concernant les problémes ou les rétablissement de ce service. Quand une notification de service est pr’te é partir, Nagios s'assurera que le moment présent fait partie d'une plage valide de la période <*notification_period*>. Si c'est un moment valide, alors Nagios tentera de notifier chaque contact du probléme du service. Certains contacts peuvent ne pas recevoir la notification de service si leur argument <*svc_notification_period*> n'autorise pas les notifications de service é ce moment. Si le moment *n'est pas* valide au sein de l'argument <*notification_period*> défini pour le service, Nagios n'enverra la notification é *aucun* contact.

Problémes potentiels liés aux notifications de contact

Il n'y a pas réellement de problémes majeurs liés é l'utilisation de périodes pour la notification des contacts. Vous devez toutefois ’tre conscients que des contacts peuvent ne pas toujours recevoir notification de problémes ou de rétablissements d'htes ou de services. Si le moment n'est pas correct é la fois pour la période de notification de l'hte ou du service, et la période de notification du contact, la notification ne partira pas. Une fois que vous avez bien pesé les problémes potentiels qu'améne la restriction des moments de notification par rapport é vos besoins, vous devriez pouvoir mettre en place une configuration adaptée é votre situation.

Conclusion

Les périodes vous donnent une meilleure matrise de la faéon dont Nagios réalise la supervision et les notifications, mais peut amener des problémes. Si vous ne savez pas trop quelles périodes implémenter, ou si votre implémentation pose des problémes, je vous suggére d'utiliser des périodes "24x7" (o toutes les heures sont valides tous les jours de la semaine). N'hésitez pas é me contacter si vous avez des questions ou si vous rencontrez des problémes.

Gestionnaires d'événements

Introduction

Les gestionnaires d'événements sont des commandes optionnelles qui sont exécutées é chaque fois qu'un changement d'état d'un hte ou d'un service a lieu. Une premiére utilité de ces gestionnaires d'événements (particuliérement pour les services) réside dans la capacité de Nagios é résoudre les problémes de maniére préventive avant que quelqu'un ne reéoive une notification. Une seconde utilité est celle d'enregistrer les événements relatifs aux htes ou services dans une base de données externe.

Types de gestionnaires d'événements

Deux types principaux de gestionnaires d'événements peuvent ’tre définis : les gestionnaires d'événements de services et les gestionnaires d'événements d'htes. Les commandes de gestionnaires d'événements sont définies (de maniére optionnelle) dans chaque définition d'hte et de service. Comme ces gestionnaires d'événements ne sont associés qu'avec des htes ou des services particuliers, je les appellerai "locaux". Si un gestionnaire d'événement local a été défini pour un service ou un hte, il sera exécuté lorsque cet hte ou ce service changera d'état.

Vous pouvez aussi spécifier des gestionnaires d'événements globaux qui doivent fonctionner é *chaque* changement d'état d'hte ou de service en utilisant les options global_host_event_handler et global_service_event_handler de votre fichier de configuration principal. Les gestionnaires d'événements globaux sont exécutés immédiatement, *avant m’me* d'exécuter un gestionnaire local d'événement d'hte ou de service.

Quand les commandes de gestionnaires d'événements sont-elles exécutées ?

Les commandes de gestionnaires d'événements de service et d'hte sont exécutées lorsqu'un service ou un hte :

- est dans un état d'erreur "soft"
- entre dans un état d'erreur "hard"
- se rétablit aprés un état d'erreur "soft" ou "hard"

A quoi correspondent les états d'erreur "soft" et "hard" dont vous parlez ? Ils sont décrits ici .

Ordre d'exécution des gestionnaires d'événements

Les gestionnaires globaux d'événements sont exécutés avant les gestionnaires locaux que vous avez configurés pour des htes ou services spécifiques.

Comment écrire des commandes de gestionnaires d'événements

Dans la plupart des cas, les commandes de gestionnaires d'événements seront des scripts écrits en shell ou en perl. Ils doivent accepter au moins les macros suivantes comme arguments:

Macros de gestionnaire d'événement de service : **$SERVICESTATE$**, **$SERVICESTATETYPE$**, **$SERVICEATTEMPT$**
Macros de gestionnaire d'événement d'hte : **$HOSTSTATE$**, **$HOSTSTATETYPE$**, **$HOSTATTEMPT$**

Les scripts doivent examiner les valeurs des arguments qui leur sont passés et exécuter les actions nécessaires en fonction de ces valeurs. Le meilleur moyen de comprendre comment les gestionnaires d'événements doivent fonctionner est de voir un exemple. Heureusement pour vous, il y en a un ci-dessous. Il y a aussi des exemples de scripts de gestionnaires d'événements dans le sous-répertoire **eventhandlers/** de la distribution de Nagios. Certains de ces scripts montrent l'usage des commandes externes pour implémenter la supervision redondante des htes.

Autorisations d'exécution des commandes de gestionnaires d'événements

Les commandes de gestionnaires d'événements que vous configurerez s'exécuteront avec les permissions de l'utilisateur grôce auquel Nagios tourne sur votre machine. Cela présente un probléme pour les scripts qui essaient de redémarrer les services du systéme, car, pour ce genre de tôches, les priviléges de root sont généralement nécessaires.

Ideally you should evaluate the types of event handlers you will be implementing and grant just enough permissions to the Nagios user for executing the necessary system commands. You might want to try using sudo to accomplish this. Implementation of this is your job, so read the docs and decide if its what you need. Vous devrez donc évaluer le type de gestionnaires d'événements que vous implémentez et donner juste les permissions requises é l'utilisateur nagios pour exécuter les commandes du systéme nécessaires. Vous pourrez essayer d'utiliser sudo pour cela. L'implémentation, c'est votre travail. Donc lisez les docs et décidez si éa correspond é vos besoins.

Débogage des commandes de gestionnaires d'événements

Lorsque vous déboguez des commandes de gestionnaires d'événements, je vous conseille fortement d'autoriser la journalisation des réessais de service, réessais d'htes, et commandes de gestionnaires d'événements. Toutes ces options sont configurées dans le fichier de configuration principal. Permettre la journalisation de ces options vous autorisera é voir exactement quand et pourquoi les commandes de gestionnaires d'événements sont exécutées.

Quand vous aurez achevé le débogage des commandes de gestionnaires d'événement, vous voudrez probablement désactiver la journalisation des réessais d'htes et de services. Ils peuvent rapidement remplir votre fichier journal, mais si vous avez autorisé la rotation des journaux, vous pouvez négliger cela.

Exemple de gestionnaire d'événement de service

L'exemple ci-dessous suppose que vous supervisez le serveur HTTP de la machine locale et que vous avez spécifié **restart-httpd** comme commande de gestionnaire d'événement pour la définition du service HTTP. Je supposerai également que vous avez donné é l'option <max_check_attempts> une valeur supérieure ou égale é 4 (i.e le service est contrlé 4 fois avant qu'on ne considére qu'il a un réel probléme). Un exemple de définition (avec uniquement les champs concernés) ressemblerait é ceci

```
define service{ host_name somehost service_description HTTP max_check_attempts
```

Une fois que le service a été défini avec un gestionnaire d'événement, nous devons définir le gestionnaire d'événement comme une commande. Remarquez les macros de la ligne de commande que je passe au gestionnaire d'événements, elles sont importantes !

```
define command{
        command_name    restart-httpd
        command_line    /usr/local/nagios/libexec/eventhandlers/restart-httpd
        }
```

Maintenant, nous allons écrire le script de gestionnaire d'événement (c'est le fichier **/usr/local/nagios/restart-httpd**).

```
#!/bin/sh
#
# Event handler script for restarting the web server on the local machine
#
# Note: This script will only restart the web server if the service is
#       retried 3 times (in a "soft" state) or if the web service somehow
#       manages to fall into a "hard" error state.
#


# What state is the HTTP service in?
case "$1" in
OK)
        # The service just came back up, so don't do anything
        ;;
WARNING)
        # We don't really care about warning states, since the service is proba
        ;;
UNKNOWN)
        # We don't know what might be causing an unknown error, so don't do any
        ;;
CRITICAL)
        # Aha!  The HTTP service appears to have a problem - perhaps we should

        # Is this a "soft" or a "hard" state?
        case "$2" in

        # We're in a "soft" state, meaning that Nagios is in the middle of retr
        # check before it turns into a "hard" state and contacts get notified
        SOFT)

                # What check attempt are we on?  We don't want to restart the w
                # check, because it may just be a fluke!
                case "$3" in

                # Wait until the check has been tried 3 times before restarting
                # If the check fails on the 4th time (after we restart the web
                # type will turn to "hard" and contacts will be notified of the
                # Hopefully this will restart the web server successfully, so t
```

```
                # result in a "soft" recovery.  If that happens no one gets not
                # fixed the problem!
                3)
                        echo -n "Restarting HTTP service (3rd soft critical sta
                        # Call the init script to restart the HTTPD server
                        /etc/rc.d/init.d/httpd restart
                        ;;
                        esac
                ;;

        # The HTTP service somehow managed to turn into a hard error without ge
        # It should have been restarted by the code above, but for some reason
        # Let's give it one last try, shall we?
        # Note: Contacts have already been notified of a problem with the servi
        # point (unless you disabled notifications for this service)
        HARD)
                echo -n "Restarting HTTP service"
                # Call the init script to restart the HTTPD server
                /etc/rc.d/init.d/httpd restart
                ;;
        esac
        ;;
esac
exit 0
```

Le script donné é titre d'exemple ci-dessus essaiera de redémarrer le serveur web sur la machine locale é deux occasions différentes : aprés que le le service HTTP soit essayé pour la troisiéme fois (dans un état d'erreur "soft") et aprés que le service soit tombé dans un état "hard". L'état "hard" ne devrait pas arriver, car le script devrait redémarrer le service quand il se trouve encore dans un état "soft" (i.e la troisiéme tentative de contrle), mais est laissé au cas o.

Il faut noter que le gestionnaire d'événement de service sera seulement exécuté la premiére fois que le service tombe dans un état "hard". Cela emp’che Nagios de poursuivre l'exécution du script pour redémarrer le serveur web quand il est dans un état "hard".

Commandes externes

Introduction

Nagios peut traiter des commandes d'applications externes (y compris les CGIs, voir CGI de commande é titre d'exemple) et modifier de nombreux aspects de ses fonctions de supervision suivant les commandes qu'il reéoit.

Autoriser les commandes externes

Par défaut, Nagios ne contrle, ni ne traite les commandes externes. Si vous voulez autoriser le traitement des commandes externes, il faut effectuer les actions suivantes

- Activer le contrle des commandes externes é l'aide de l'option check_external_commands
- Régler la fréquence des contrles de commande é l'aide de l'option command_check_interval
- Préciser l'emplacement du fichier de commandes é l'aide de l'option command_file . Il vaut mieux mettre le fichier de commandes externes dans son propre répertoire (i.e. */usr/local/nagios/var/rw*).
- configurer les bonnes autorisations pour le répertoire contenant les fichiers de commandes. Des détails sur la maniére de faire cela se trouvent ici.

Quand Nagios contrle-t-il les commandes externes ?

- A intervalles réguliers précisés par l'option command_check_interval du fichier de configuration principal.
- Immédiatement aprés que les gestionnaires d'événements soient exécutés. Cela s'effectue en plus du cycle régulier des contrles de commandes externes et sert é fournir une action immédiate si un gestionnaire d'événement soumet des commandes é Nagios.

L'utilisation des commandes externes

Les commandes externes peuvent ’tre utilisées pour mener é bien un certain nombre de choses pendant que Nagios fonctionne. A titre d'exemple, ce qui peut ’tre effectué comprend : la désactivation temporaire des notifications pour les services et les htes, la désactivation temporaire des tests de service, l'obligation de contrler immédiatement un service, l'ajout de commentaires aux htes et services, etc.

Format des commandes

Les commandes externes écrites pour le fichier de commande suivent le format suivant :

[*time*] *command_id*;*command_arguments*

o *time* est l'heure (au format *time_t*) é laquelle l'application externe ou le CGI a envoyé la commande externe au fichier de commande. Certaines des commandes disponibles sont décrites dans le tableau ci-dessous, ainsi que leur *command_id* et une description de leurs *command_arguments*.

Commandes implémentées

Un listing complet des commandes externes qui peuvent ’tre utilisées (ainsi que des exemples d'utilisation) sont en ligne é l'adresse suivante:

http://www.nagios.org/developerinfo/externalcommands/

Contrles Indirects des Htes et des Services

Introduction

Par chance, la plupart des services que vous allez superviser sur votre réseau peuvent ’tre contrlés en utilisant directement par un plugin depuis la machine Nagios. Les services qui peuvent ’tre directement contrlés sont par exemple la disponibilité des serveurs web, email et FTP. Ces services peuvent ’tre contrlés par un plugin depuis l'hte Nagios parce que ce sont des ressources accessibles directement. Cependant, il y a un certain nombre de choses dont vous aimeriez superviser mais qui ne sont pas publiquement accessibles comme les autres services. Ces services/ressources "privés" inclus le taux d'utilisation d'un disque, la charge d'un processeur,etc d'une machine distante. Ce type de ressources ne peuvent pas ’tre contrlées sans utiliser un agent intermédiaire. Les contrles de service qui nécessitent un agent intermédiaire (pour réaliser le contrle) sont appelés contrles *indirects*.

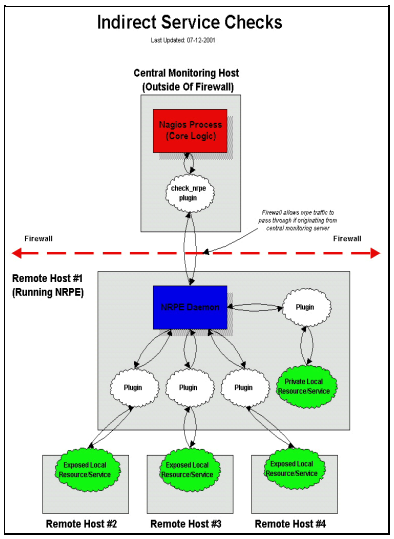
Les contrles indirects sont utiles pour :

- Superviser les ressources "locales" (telles que le taux d'utilisation d'un disque, la charge d'un processeur, etc.) sur des machines distantes
- Superviser des services et des htes derriére un pare-feu
- Obtenir des résultats plus réalistes des contrles de services sensibles aux délais entre des machines distantes (par exemple le temps de réponse d'un ping entre deux htes distants)

Il existe plusieurs méthodes pour réaliser des contrles actifs indirects (on ne discutera pas des contrles passifs), mais j'expliquerai uniquement comment ils peuvent ’tre effectués en utilisant l'addon nrpe.

Contrles Indirects des Services

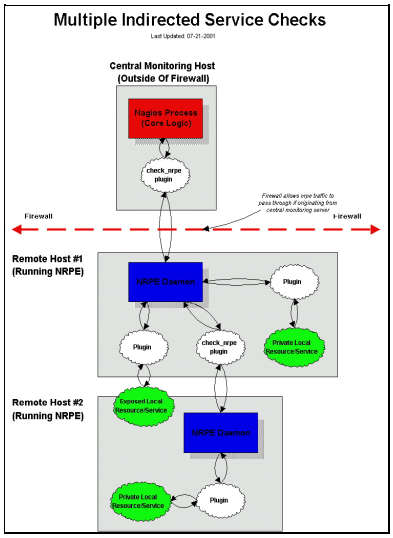
Le diagramme ci-dessous montre comment les contrles indirects des services fonctionnent. Cliquez sur l'image pour l'agrandir



Contrles Multiples et Indirects des Services

Si vous supervisez des serveurs se trouvant derriére un pare-feu (et que la machine sur laquelle Nagios est installée se trouve en dehors du pare-feu) le contrle des services sur ces machines peut s'avérer un compliqué. Il y a des chances pour que vous bloquiez la plupart du trafic entrant qui serait normalement requis pour la supervision. Une solution pour réaliser les contrles actifs (ceci est aussi valable pour les contrles passifs) sur les machines derriére le pare-feu serait d'enfoncer un petit trou dans les filtres du pare-feu pour autoriser l'hte Nagios é appeler le démon *nrpe* d'un hte protégé par le pare-feu. La machine protégée par le pare-feu pourrait aussi ’tre utilisée comme intermédiaire dans la réalisation de contrles sur d'autres serveurs protégés par le pare-feu.

Le diagramme ci-dessous montre comment les contrles de services multiples et indirects fonctionnent. Remarquez que le démon *nrpe* tourne sur les machines #1 et #2. La copie fonctionnant sur la machine #2 est utilisée pour permettre é l'agent *nrpe* sur l'hte #1 de contrler un service "privé" sur la machine #2. Les services "privés" sont des services tel que la charge CPU, l'utilisation de l'espace disque, etc. qui ne sont pas directement exposés comme SMTP, FTP, et les services web. Cliquez sur l'image pour l'agrandir

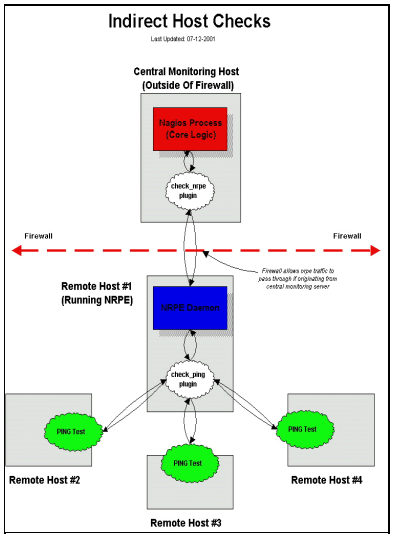


Controles Indirects d'Hotes

Les contrles indirects d'htes fonctionnent de la m’me faéon que les contrles indirects services. Typiquement, le plugin utilisé dans la commande de contrle d'hte demande é un agent intermédiaire (c.a.d. un démon fonctionnant sur une machine distante) de réaliser le contrle d'hte pour lui. Les contrles indirects d'hte sont utiles lorsque l'hte distant é superviser se trouvent derriére un pare-feu et que vous souhaitez restreindre le trafic de supervision é une machine donnée. Cette machine (l'hte #1 sur le diagramme ci-dessous) réalisera les contrles d'htes et retournera les résultats au plugin *check_nrpe* de plus haut niveau (sur le serveur central). Notez qu'avec cette configuration des problémes peuvent survenir. Si l'hte distant #1 s'effondre, le plugin *check_nrpe* ne sera pas capable de contacter le démon *nrpe* et Nagios croira que les htes distants #2, #3, et #4 sont "down", m’me si ce n'est pas le cas. Si l'hte #1 est votre firewall, alors le probléme n'en sera pas vraiment un parce que Nagios le détectera "down" et considéra les htes #2, #3 et #4 comme inaccessibles.

Le diagramme ci-dessous montre comment un contrle indirect d'hte peut ’tre effectué en utilisant le

démon nrpe et le plugin *check_nrpe*. Cliquez sur l'image pour l'agrandir.



Contrles passifs d'htes et de services

Introduction

Une des fonctionnalités de Nagios est de traiter le résultat de contrles de service soumis par des applications tierces. Les contrles d'htes et de service réalisés par des applications tierces et traités par Nagios sont appelés contrles *passifs*. Ces contrles sont dits passifs par opposition aux contrles *actifs*, qui sont des contrles d'hotes ou de services réalisés é l'initiative de Nagios.

Des contrles passifs pour quoi faire?

Les contrles passifs sont utiles pour superviser des services qui sont :

- situés derriére un firewall, et ne peuvent donc pas ’tre contrlés depuis l'hte supportant Nagios
- asynchrones par nature et ne peuvent donc pas ’tre contrlés activement de maniére fiable (p.ex. les traps SNMP, les alertes de sécurité, etc.)

Les contrles passifs d'htes et de services sont également utiles dans le cadre d'une supervision répartie.

Contrles passifs de service contre contrles passifs d'hte

Les contrles passifs d'hte et de service fonctionnent de maniére similaire, mais avec d'importantes limitations pour les contrles passifs d'hte. Voyez ci-dessous pour plus d'informations sur ces limitations.

Comment les contrles passifs de service fonctionnent-ils ?

La seule réelle différence entre les contrles actifs et passifs est que les contrles actifs sont initiés par Nagios, alors que les contrles passifs sont réalisés par des applications tierces. Une fois qu'une application tierce a réalisé un contrle de service (que ce soit activement ou en ayant reéu un événement asynchrone comme un trap SNMP ou une alerte de sécurité), elle envoie le résultat du "contrle" de service é Nagios é travers le fichier de commande externe.

Lorsque Nagios traite le contenu du fichier de commande externe, il place les résultats de tous les contrles passifs de service dans une file d'attente pour un traitement ultérieur. C'est la m’me file qui est utilisée pour stocker les résultats des contrles actifs et passifs.

Nagios exécute réguliérement un événement de consolidation des services et lit le contenu de la file de résultat des contrles. Chaque résultat de contrle de service, qu'il soit actif ou passif, est traité de la m’me faéon. L'algorithme de contrle de service est exactement le m’me pour les deux types de contrles. Ceci permet d'appliquer une seule méthode pour la gestion des résultats de contrles actifs et passifs.

Comment les applications tierces soumettent-elles le résultat des contrles de service ?

Les applications tierces peuvent soumettre les résultats de contrles de service é Nagios en écrivant une commande externe PROCESS_SERVICE_CHECK_RESULT dans le fichier de commandes externes.

Le format de la commande est le suivant :

```
[<timestamp>]
  PROCESS_SERVICE_CHECK_RESULT;<host_name>;<description>;<return_code>;<plugin_
```

o

- *timestamp* est le moment au format time_t (secondes écoulée depuis l'époque UNIX) auquel le contrle du service a été réalisé (ou envoyé). Veuillez noter l'espace nécessaire aprés le crochet fermant.
- *host_name* est le nom court de l'hte associé au service dans la définition du service
- *description* est la description du service telle que spécifiée dans la définition du service
- *return_code* est le code renvoyé par le contrle (0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN)
- *plugin_output* est le texte affiché par le contrle (c.-é-d. l'affichage de sortie du plugin)

Notez que pour soumettre des contrles de service é Nagios, un service doit avoir été défini préalablement dans le fichier de configuration des objets ! Nagios ignorera tous les résultats de contrles de service si ceux-ci n'ont pas été configurés avant son dernier (re)démarrage.

Si vous voulez que seuls des résultats passifs soient fournis pour un service particulier (c.-é-d. que les contrles actifs ne doivent pas avoir lieu), mettez simplement le paramétre *active_check_enabled* de la définition du service é 0. Ceci emp'chera définitivement Nagios de réaliser un contrle du service. Assurez vous également que le paramétre *passive_check_enabled* de la définition du service est é 1, sinon Nagios ne traitera jamais les contrles passifs pour ce service !

Vous pouvez trouver un exemple de script Shell sur la faéon de soumettre des résultats de contrles passifs de services é Nagios dans la documentation sur les services volatils.

Soumission de résultats de contrles passifs de service depuis des htes distants

Si l'application qui soumet les résultats de contrles passifs se trouve sur le m'me hte que Nagios, elle peut directement écrire ces résultats dans le fichier de commandes externes comme décrit ci-dessus. Mais les applications se trouvant sur des htes distants ne peuvent pas le faire aussi simplement. Pour que des htes distants puissent envoyer des résultats de contrles passifs é l'hte sur lequel tourne Nagios, j'ai développé l'addon nsca. Cet addon consiste en un démon qui tourne sur l'hte de Nagios et un client exécuté sur les htes distants. Le démon attend les connexions des clients distants, valide sommairement les résultats soumis, et les écrit directement dans le fichier de commandes externe (de la maniére décrite ci-dessus). Vous trouverez plus d'information sur l'addon nsca ici

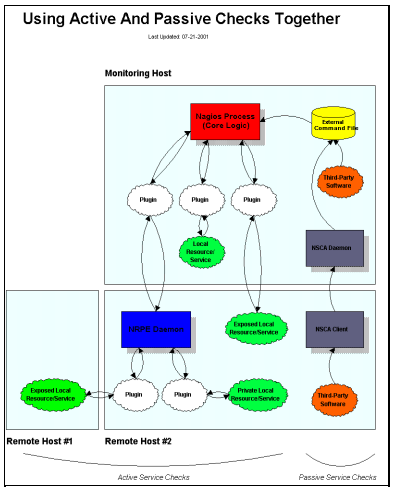
Utilisation commune de contrles actifs et passifs de service

A moins que vous n'implémentiez un environnement de supervision répartie avec un serveur central n'acceptant que les contrles passifs (et ne réalisant aucun contrle actif), vous utiliserez probablement les deux types de contrles. Comme je l'ai déjé dit, les contrles actifs sont plus adaptés aux services qui se pr'tent au contrle régulier (disponibilité d'un serveur FTP ou web, etc.), alors que les contrles passifs conviennent mieux pour gérer des événement asynchrones survenant é des fréquences variables (alertes de sécurité, etc.).

L'image ci-dessous donne une représentation visuelle de la faéon dont les contrles actifs et passifs de service peuvent tous deux ’tre employés pour superviser les ressources du réseau (cliquez sur l'image pour la voir en grand format).

Les patatodes oranges é droite de l'image sont des applications tierces qui soumettent des résultats de contrles passifs dans le fichier de commandes externes de Nagios. Une des applications se trouve sur le m’me hte que Nagios, ce qui fait qu'elle peut écrire directement dans ce fichier. L'autre application se trouve sur un hte distant et se sert du client et du démon nsca pour transférer les résultats de contrles passifs é Nagios.

Les éléments é gauche de l'image représentent des contrles actifs de services que Nagios réalise. J'ai montré comment les contrles peuvent ’tre réalisés pour des ressources locales (utilisation du disque, etc.), pour des ressources "publiques" sur des htes distants (serveur web, serveur FTP, etc.), et pour des ressources "privées" sur des htes distants (utilisation du disque de l'hte distant, charge du processeur, etc.). Dans cet exemple, les ressources privées des htes distants sont en fait contrlées grôce é l'addon nrpe, qui facilite l'exécution de plugins sur les htes distants.



Comment les contrles passifs d'htes fonctionnent-ils ?

Les contrles passifs d'hte fonctionnent de la m’me maniére que les contrles passifs de service. Quand une application tierce a réalisé un contrle d'hte, elle envoie les résultats de ce "contrle" d'hte é Nagios via le fichier de commandes externes. Quand Nagios lira le contenu du fichier de commandes externes, il traitera le résultat de contrle d'hte qui a été envoyé.

ATTENTION ! Les contrles d'htes passifs ont des limitations. A la différence des contrles d'htes actifs, Nagios ne tente pas de déterminer si l'hte est dans l'état DOWN ou UNREACHABLE lors d'un contrle passif. Nagios prend plutt le résultat du contrle passif comme étant l'état réel de l'hte, sans essayer de déterminer cet état réel. Au contraire, dans le cas de contrle d'hte actif (initié par Nagios), Nagios tente de déterminer l'état correct (DOWN ou UNREACHABLE) des htes qui ne sont pas dans l'état UP. Cela peut poser des problémes si vous envoyez des contrles passifs depuis un hte distant ou si vous avez une supervision répartie o les relations entre les htes parent/enfant sont différentes. Voyez la documentation sur l'accessibilité des htes pour de plus amples informations sur la faéon dont sont déterminés les états DOWN et UNREACHABLE lors de contrles actifs d'hte.

Comment les applications tierces soumettent-elles le résultat des contrles d'hte ?

Les applications tierces peuvent envoyer des résultats de contrles d'hte é Nagios en écrivant une commande externe PROCESS_HOST_CHECK_RESULT dans le fichier des commandes externes.

Le format de la commande est le suivant :

```
[<timestamp>]éPROCESS_HOST_CHECK_RESULT;<host_name>;<host_status>;<plugin_outpu
```

o

- *timestamp* est le moment au format time_t (secondes écoulée depuis l'époque UNIX) auquel le contrle de l'hte a été réalisé (ou envoyé). Veuillez noter l'espace nécessaire aprés le crochet fermant.
- *host_name* est le nom court de l'hte (tel qu'il est spécifié dans la définition de l'hte)
- *host_status* est l'état de l'hte (0=UP, 1=DOWN, 2=UNREACHABLE)
- *plugin_output* est le texte affiché par le contrle d'hte

Notez que pour envoyer un contrle d'hte é Nagios, l'hte correspondant doit ’tre défini dans le fichier de configuration des objets ! Nagios ignorera tous les résultats de contrles d'hte si ceux-ci n'ont pas été configurés avant son dernier (re)démarrage.

Soumission de résultats de contrles passifs d'hte depuis des htes distants

Si l'application qui soumet les résultats de contrles passifs se trouve sur le m’me hte que Nagios, elle peut directement écrire ces résultats dans le fichier de commandes externes comme décrit ci-dessus. Mais les applications se trouvant sur des htes distants ne peuvent pas le faire aussi simplement. Pour que des htes distants puissent envoyer des résultats de contrles passifs é l'hte sur lequel tourne Nagios, vous pouvez utiliser l'addon nsca. Cet addon consiste en un démon qui tourne sur l'hte de Nagios et un client exécuté sur les htes distants. Le démon attend les connexions des clients distants, valide sommairement les résultats soumis, et les écrit directement dans le fichier de commandes externe (de la maniére décrite ci-dessus). Vous trouverez plus d'information sur le addon nsca ici

Services volatiles

Introduction

Nagios a la capacité de faire la distinction entre les services "normaux" et les services "volatiles". L'option *is_volatil* de chaque définition de service vous permet de spécifier si ou non un service spécifique est volatile. Pour la plupart des gens, la majorité des services supervisés sera de type non-volatile (i.e. "normal"). Toutefois, des services volatiles peuvent se révéler trés utiles lorsqu'ils sont bien utilisés

A quoi servent-ils ?

Les services volatiles sont utiles pour superviser

- des choses qui se remettent automatiquement en état "OK" chaque fois qu'ils sont contrlés
- des événements comme les alertes de sécurité qui réclament de l'attention é chaque probléme (et pas seulement la premiére fois)

Qu'est-ce que les services volatiles ont de si particulier ?

Les services volatiles différent des services "normaux" de trois faéons importantes. *Chaque fois* qu'ils sont contrlés quand ils sont dans un état hard non-OK, et que le contrle retourne un état non-OK (i.e. aucun changement d'état n'a eu lieu)

- l'état non-OK du service est journalisé
- les contacts re?oivent notification du probléme (si c'est ce qui doit ’tre fait)
- le gestionnaire d'événement du service est lancé (s'il a été défini)

Ces événements ne se produisent normalement que lorsque des services sont dans un état non-OK et qu'un changement d'état hard vient de se produire. En d'autre termes, ils ne se produisent que la premiére fois que le service passe dans un état non-OK. Si des contrles ultérieurs du service conduisent au m’me état non-OK, il n'y a aucun changement d'état hard et aucun des événements mentionnés ne se reproduit.

La puissance de deux

Si vous combinez les fonctionnalités des services volatiles avec les contrles passifs de service, vous pouvez faire des choses trés utiles. Par exemple, gérer des traps SNMP, des alertes de sécurité, etc.

Que diriez-vous d'un exemple ? Disons que vous exécutiez le produit PortSentry pour scanner les ports de votre machine et les intrus potentiels. Si vous voulez que Nagios soit averti des scans de ports, vous pouvez mettre en place ce qui suit

Dans Nagios:

- Configurez un service appelé *Port Scans* et associez-le avec l'hte sur lequel tourne PortSentry.
- Mettez l'option *max_check_attempts* de la définition du service é 1. Ceci dira é Nagios de passer immédiatement le service en état hard quand un état non-OK est retourné.

- Mettez l'option *active_check_enabled* é 0 ou mettez l'option *check_time* de la définition du service é une période qui ne contient *aucune* plage de temps valide. Cela évitera que Nagios ne contrle activement le service. M'me si le contrle du service est ordonnancé, il ne sera jamais réellement contrlé.

Dans PortSentry:

- Modifiez votre fichier de configuration de PortSentry (portsentry.conf), et définissez une commande pour la directive **KILL_RUN_CMD** comme suit :

```
KILL_RUN_CMD="/usr/local/Nagios/libexec/eventhandlers/submit_check_result
    <host_name> 'Port Scans' 2 'Port scan from host
    $TARGET$ on port $PORT$. Host has been
    firewalled.'"
```

  Assurez-vous de remplacer <*host_name*> avec le nom court de l'hte avec lequel le service est associé.

Créez un script Shell dans le répertoire */usr/local/Nagios/libexec/eventhandlers* que vous appelez *submit_check_result*. Le contenu de ce script Shell doit ressembler é ceci

```
#!/bin/sh

# Write a command to the Nagios command file to cause
# it to process a service check result

echocmd="/bin/echo"

CommandFile="/usr/local/Nagios/var/rw/Nagios.cmd"

# get the current date/time in seconds since UNIX epoch
datetime=`date +%s`

# create the command line to add to the command file
cmdline="[$datetime] PROCESS_SERVICE_CHECK_RESULT;$1;$2;$3;$4"

# append the command to the end of the command file
`$echocmd $cmdline >> $CommandFile`
```

Notez que si vous exécutez PortSentry en tant que root, vous devrez ajouter au script la modification du propriétaire du fichier et les permissions de faéon é ce que Nagios et les CGI puissent lire/modifier le fichier de commande. Vous trouverez des détails sur les permissions/l'appartenance du fichier de commande ici.

Et donc qu'arrive-t'il lorsque PortSentry détecte un scan de port sur la machine?

- Il bloque l'hte (c'est une fonction du logiciel PortSentry)
- Il exécute le script *submit_check_result* pour envoyer l'information d'alerte de sécurité é Nagios
- Nagios lit le fichier de commande, reconnat l'entrée du scan de port comme un contrle passif de service
- Nagios traite les résultats du service en journalisant l'état CRITICAL, en envoyant des notifications aux contacts (s'il a été configuré pour le faire), et exécute le gestionnaire d'événement pour le service

Port Scans (s'il a été défini).

Contrle de la fracheur des résultats d'htes et de services

Introduction

Nagios propose une fonctionnalité de vérification de la "fracheur" des résultats de contrles d'htes et de services. Cette fonctionnalité est utile pour vous assurer que des contrles passifs sont reéus é la fréquence que vous souhaitez. Bien que le contrle de fracheur puisse ’tre utilisé dans de nombreuses situations, son emploi premier est dans la configuration d'un environnement de supervision réparti.

Le but du contrle de "fracheur" est de s'assurer que les contrles d'hte et de service sont soumis réguliérement de maniére passive par des applications externes. Si les résultats d'un contrle d'hte ou de service (pour lequel vous avez activé le contrle de fracheur) sont considérés comme "périmés", Nagios forcera un contrle actif de l'hte ou du service.

Contrle de fracheur d'hte ou de service

La documentation ci-dessous décrit le contrle de fracheur de service. Le contrle de fracheur d'hte (qui n'est pas documenté individuellement) fonctionne de maniére similaire au contrle de fracheur de service - é part évidemment qu'il se rapporte é la fracheur des htes et pas é celle des services. Si vous souhaitez configurer le contrle de fracheur d'hte, adaptez les instructions ci-dessous.

Configuration du contrle de fracheur de service

Avant de paramétrer le seuil de fracheur de chaque service, vous devez activer le contrle de fracheur par les paramétres check_service_freshness et freshness_check_interval du fichier de configuration principal. Si vous deviez configurer le contrle de fracheur d'hte, vous utiliseriez les paramétres check_host_freshness et host_freshness_check_interval.

Maintenant, comment activer le contrle de fracheur d'un service en particulier ? Vous devez configurer la définition de service comme suit.

- Le paramétre **check_freshness** de la définition du service doit valoir 1. Cela active le contrle de "fracheur" du service.
- Le paramétre **freshness_threshold** de la définition du service doit avoir une valeur (en secondes) qui refléte quel "fracheur" les résultats de ce service doivent présenter.
- Le paramétre **check_command** de la définition du service doit ’tre une commande valide, permettant de contrler activement le service quand il est vu comme "périmé".
- Le paramétre **normal_check_interval** de la définition du service doit ’tre supérieur é zéro (0) si le paramétre **freshness_threshold** vaut zéro (0).
- Le paramétre **check_period** de la définition du service doit ’tre une période valide. L'intervalle de temps que spécifie la période détermine quand les contrles de fracheur peuvent ’tre exécutés pour le service.

Fonctionnement du seuil de fracheur

Nagios contrle réguliérement la "fracheur" des résultats de tous les services dont le contrle de fracheur est activé. Le paramétre *freshness_threshold* de la définition de chaque service permet de déterminer quelle "fracheur" ces résultats doivent présenter. Par exemple, si vous donnez la valeur 60 au paramétre *freshness_threshold* de l'un de vos services, Nagios considérera que le service est "périmé" si ses resultats sont ôgés de plus de 60 secondes (1 minute). Si vous ne spécifiez pas de valeur pour le paramétre *freshness_threshold* (ou si vous le mettez é zéro), Nagios calculera automatiquement un seuil de "fracheur" é utiliser en se basant soit sur le paramétre *normal_check_interval* soit sur *retry_check_interval* (selon le type d'état dans lequel se trouve le service actuellement).

Ce qui se passe lorsqu'un résultat de contrle de service devient "périmé"

Si le résultat d'un contrle de service devient "périmé" (selon la définition ci-dessus), Nagios forcera un contrle actif du service en exécutant la commande spécifiée par le paramétre *check_command* de la définition du service. Il est important de noter qu'un contrle de service actif déclenché parce que le service est vu comme "périmé" sera exécuté *m'me si les contrles actifs de service sont désactivés de maniére globale ou pour le service en particulier*.

Travailler avec des contrles purement passifs

Comme je l'ai déjé dit, le contrle de fracheur est surtout utile pour les services dont les résultats sont issus d'un contrle passif. Il se peut bien souvent (comme dans le cas d'une supervision répartie) que ces services ne reéoivent pas *tous* les résultats de leur contrle passif - aucun résultat n'étant obtenu par contrle actif.

Un exemple de service purement passif pourrait 'tre un rapport d'état de vos travaux de sauvegarde de nuit. Vous avez peut-'tre un script externe qui soumet les résultats du travail de sauvegarde é Nagios une fois que la sauvegarde est terminée. Dans ce cas, tous les résultats des contrles pour ce service sont fournis par une application externe, en utilisant des contrles passifs. Pour vous assurer que l'état des travaux de sauvegarde est bien remonté chaque jour, vous activerez le contrle de fracheur pour ce service. Si le script externe ne soumet pas les resultats du travail de sauvegarde, vous pouvez faire en sorte que Nagios simule un résultat "critical" de la maniére suivante

Voici ce é quoi la définition du service pourrait ressembler (certains paramétres obligatoires ont été omis)

```
define service{
        host_name               backup-server
        service_description     ArcServe Backup Job
        active_checks_enabled   0                       ; active checks are NOT
        passive_checks_enabled  1                       ; passive checks are en
        check_freshness         1
        freshness_threshold     93600                   ; 26 hour threshold, si
        check_command           no-backup-report        ; this command is run o
        other options
        }
```

Notez que les contrles actifs sont désactivés pour ce service. En effet les résultats du service sont uniquement fournis par une application externe, en utilisant des contrles passifs. Le contrle de fracheur est activé et le seuil de fracheur a été positionné é 26 heures. C'est un peu plus de 24 heures parce que la durée des travaux de sauvegarde varie selon les jours (en fonction du volume de données é sauvegarder, de l'encombrement du réseau, etc.). La commande *no-backup-report* est exécutée seulement si les résultats du service sont considérés comme "périmés". La définition de la commande *no-backup-report* pourrait ressembler é ceci

```
define command{
        command_name    no-backup-report
        command_line    /usr/local/nagios/libexec/nobackupreport.sh
        }
```

Le script **nobackupreport.sh** dans votre répertoire */usr/local/nagios/libexec* pourait ressembler é ceci :

```
#!/bin/sh

/bin/echo "CRITICAL: Results of backup job were not reported!"

exit 2
```

Si Nagios détecte que les résultats du service sont périmés, il lancera la commande **no-backup-report** comme un contrle actif de service (m’me si les contrles actifs sont désactivés pour ce service - rappelez-vous que nous sommes dans un cas particulier). Cela lance le script */usr/local/nagios/libexec/nobackupreport.sh*, qui retourne un état "critical". Le service passe en état "critical" (s'il n'y était pas déjé) et quelqu'un sera probablement notifié du probléme.

Supervision répartie

Introduction

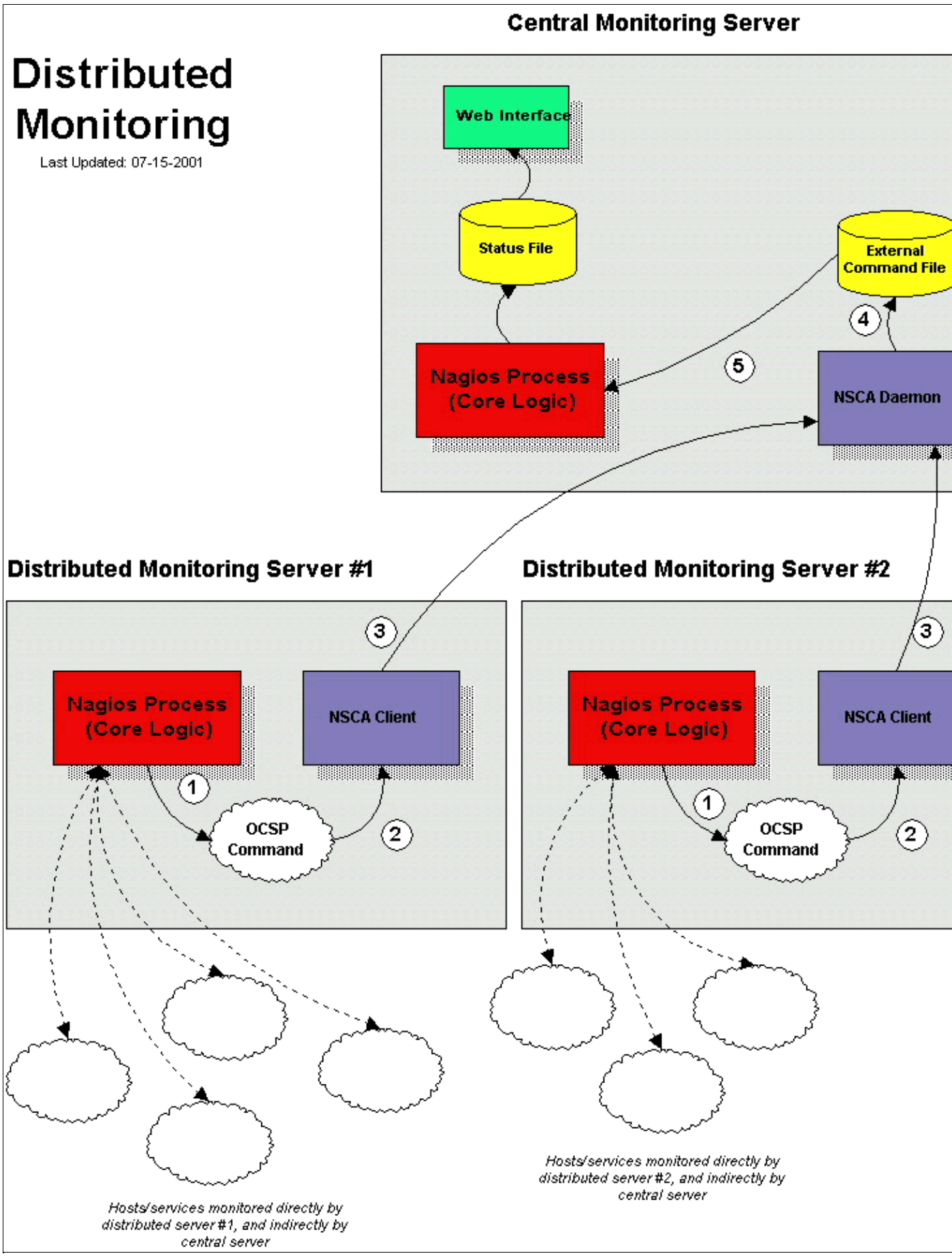
Nagios peut ’tre configuré pour supporter la supervision répartie des services et ressources du réseau. Je vais essayer d'expliquer briévement comment cela peut s'effectuer

Buts

Le but de l'environnement de supervision réparti que je vais décrire est de décharger l'excés de charge induit par les contrle de services (sur la CPU, etc.) du serveur central sur un ou plusieurs serveurs "répartis". La plupart des petites et moyennes entreprises n'auront pas réellement besoin de mettre en uvre cet environnement. Cependant, quand vous voulez superviser des centaines, voire des milliers d'*htes* (et plusieurs fois cette valeur en termes de services) é l'aide de Nagios, cela commence é devenir important.

Diagramme de référence

Le diagramme ci-dessous devrait vous aider é vous faire une idée du fonctionnement de la supervision répartie avec Nagios. Je ferai référence aux éléments du diagramme quand j'expliquerai les choses


Distributed Monitoring
Last Updated: 07-15-2001
Central Monitoring Server
Web Interface
Status File
External Command File
Nagios Process (Core Logic)
NSCA Daemon
1
2
3
4
5
Distributed Monitoring Server #1
Nagios Process (Core Logic)
NSCA Client
OCSP Command
Hosts/services monitored directly by distributed server #1, and indirectly by central server
Distributed Monitoring Server #2
Nagios Process (Core Logic)
NSCA Client
OCSP Command
Hosts/services monitored directly by distributed server #2, and indirectly by central server

Serveur central ou serveurs répartis

Quand on installe l'environnement de supervision réparti avec Nagios, il y a des différences entre la configuration du serveur central et celle des serveurs répartis. Je vous montrerai comment configurer ces deux types de serveurs et j'expliquerai les effets des changements sur la supervision en général. A l'intention des débutants, décrivons d'abord l'utilité des différents serveurs

Le rle d'un *serveur réparti* est de contrler tous les services définis pour une grappe [cluster] d'htes. J'utilise ce terme "grappe" de maniére inappropriée : il désigne simplement un groupe d'htes de votre réseau. Selon la topographie de votre réseau, vous pouvez avoir plusieurs grappes en un seul lieu, ou chaque grappe peut ’tre séparée par un WAN, un pare-feu, etc. Il faut surtout se souvenir d'une chose, c'est que pour chaque groupe d'htes (de quelque maniére que vous le définissiez), il y a un serveur réparti sur lequel Nagios tourne et qui supervise les services des htes du cluster. Un serveur réparti est généralement une installation simplifiée de Nagios. Il n'est pas besoin d'installer l'interface web, d'envoyer des notifications, de faire tourner les scripts de gestionnaires d'événements ou de faire autre chose que l'exécution des contrles de service si vous ne le souhaitez pas. De plus amples explications relatives é la configuration du serveur réparti suivront

Le but du *serveur central* est d'écouter simplement les résultats des contrles de service d'un ou de plusieurs serveurs répartis. M’me si les services sont occasionnellement contrlés activement par le serveur central, les contrles actifs sont seulement exécutés dans des circonstances particuliéres ; disons donc pour l'instant que le serveur central n'accepte que les contrles passifs. Comme le serveur central obtient des résultats des contrles de services passifs d'un ou plusieurs serveurs répartis, il est utilisé comme point central pour la logique de supervision (i.e il envoie des notifications, exécute les scripts de gestionnaires d'événements, détermine l'état des htes, son interface web est installée, etc.).

Obtention des informations de contrle de service é partir de moniteurs répartis

Avant de sauter é pieds joints dans les détails de la configuration, il faut savoir comment envoyer les résultats des contrles de service des serveurs répartis au serveur central. J'ai déjé expliqué comment soumettre des résultats de contrles passifs é Nagios é partir de la machine m’me sur laquelle Nagios tourne (cf. documentation sur les contrles passifs), mais je n'ai pas fourni d'information sur la maniére d'envoyer des résultats de contrles de service é partir d'autres htes.

Afin de faciliter l'envoi de résultats de contrles passifs é un hte distant, j'ai écrit le module complémentaire nsca. Il contient deux parties. La premiére est un programme client (send_nsca) qui tourne sur un hte distant et envoi les résultats de contrles de service é un autre serveur. La seconde est le démon nsca (nsca) qui fonctionne, soit comme un démon autonome, soit sous inetd, et écoute les connections des programmes du client. Aprés avoir reéu l'information de contrle de service de la part d'un client, le démon enverra l'information de contrle é Nagios (sur le serveur central) en insérant une commande *PROCESS_SVC_CHECK_RESULT* dans le fichier de commande externe, avec les résultats du contrle. La prochaine fois que Nagios contrlera les commandes externes, il trouvera l'information de contrle de service passif qui avait été envoyée par le serveur réparti et la traitera. Elémentaire, non ?

Configuration des serveurs répartis

Bon, comment Nagios est-il configuré sur un serveur réparti ? A la base, c'est juste une simple installation. Il n'est pas nécessaire d'installer l'interface web ou de faire envoyer des notifications par le serveur, comme c'est le cas pour le serveur central.

Changements principaux dans la configuration :

- Seuls les services et les htes qui sont supervisés directement par le serveur réparti sont définis dans le fichier de configuration d'hte.
- Le serveur réparti a sa directive d'activation de notification initiale fixée é 0. Cela évitera d'envoyer des notifications é partir du serveur.
- Le serveur réparti est configuré avec l'option de remontée de contrle de service activée.
- Le serveur réparti a une commande ocsp définie (cf. ci-dessous).

Afin que tout fonctionne ensemble de maniére correcte, nous voulons que le serveur réparti renvoie les résultats de *tous* les contrles de service é Nagios. Nous pourrions utiliser les gestionnaires d'événements pour envoyer les *changements* de l'état d'un service, mais cela ne fait pas l'affaire. Afin d'obliger le serveur réparti é envoyer tous les résultats des contrles de service, il faut autoriser l'option de remontée de contrle de service dans le fichier de configuration principal et permettre qu'une commande ocsp soit lancée aprés chaque contrle de service. Nous utiliserons cette commande ocsp pour envoyer les résultats de tous les contrles de service au serveur central, en utilisant le client send_nsca et le démon nsca (comme décrit ci-dessus) pour gérer la transmission.

Pour mener tout cela é bien, il faut définir la commande ocsp de cette faéon :

```
ocsp_command=submit_check_result
```

La définition de la commande *submit_check_result* ressemble é ceci :

```
define command{
        command_name    submit_check_result
        command_line    /usr/local/nagios/libexec/eventhandlers/submit_check_re
        }
```

Le script shell *submit_check_result* ressemble é cela (remplacez *central_server* par l'adresse IP du serveur central) :

```
   #!/bin/sh
        # Arguments:
        #  $1 = host_name (Short name of host that the service is
        #       associated with)
        #  $2 = svc_description (Description of the service)
        #  $3 = state_string (A string representing the status of
        #       the given service - "OK", "WARNING", "CRITICAL"
        #       or "UNKNOWN")
        #  $4 = plugin_output (A text string that should be used
        #       as the plugin output for the service checks)
        #
        # Convert the state string to the corresponding return code
```

```
return_code=-1
case "$3" in
    OK)
                return_code=0
            ;;
        WARNING)
            return_code=1
                ;;
        CRITICAL)
            return_code=2
                ;;
        UNKNOWN)
            return_code=-1
                ;;
esac
# pipe the service check info into the send_nsca program, which
# in turn transmits the data to the nsca daemon on the central
# monitoring server
/bin/echo -e "$1\t$2\t$return_code\t$4\n" | /usr/local/nagios/bin/send_
```

Le script ci-dessus suppose que vous avez le programme send_nsca et son fichier de configuration (send_nsca.cfg) placés respectivement dans les répertoires */usr/local/nagios/bin/* et */usr/local/nagios/var/*.

C'est tout ! Nous venons de configurer avec succés un hte distant sur lequel tourne Nagios pour agir comme un serveur de supervision réparti. Voyons maintenant ce qui se passe exactement avec le serveur réparti et comment il envoie des résultats de contrle de service é Nagios (les étapes soulignées ci-dessous correspondent aux numéros du schéma de référence ci-dessus) :

1. Aprés que le serveur réparti a terminé l'exécution d'un contrle de service, il exécute la commande définie par la variable ocsp_command. Dans notre exemple, c'est le script */usr/local/nagios/libexec/eventhandlers/submit_check_result*. Remarquez que la définition de la commande *submit_check_result* a envoyé quatre éléments d'information au script : le nom de l'hte auquel le service est associé, la description du service, le code de retour du contrle de service, et la sortie du plugin de contrle du service.
2. Le script *submit_check_result* envoie dans un tube [pipe] l'information du contrle de service (nom de l'hte, description, code de retour et sortie) au programme client *send_nsca*.
3. Le programme *send_nsca* transmet l'information de contrle de service au démon *nsca* qui se trouve sur le serveur de supervision central.
4. Le démon *nsca* du serveur central prend l'information de contrle de service et l'écrit dans le fichier de commande externe pour qu'elle soit reprise ultérieurement par Nagios.
5. Le processus Nagios du serveur central lit le fichier de commande externe et analyse l'information de contrle de service provenant du serveur de supervision réparti.

Configuration du serveur central

Nous avons vu comment les serveurs de supervision répartis doivent ’tre configurés, occupons nous maintenant du serveur central. Pour accomplir toutes ses missions, il est configuré de la m’me maniére que vous configureriez un serveur seul. Il est installé de la maniére suivante :

- L'interface web du serveur central est installée (optionnel, mais recommandé)

- Sur le serveur central, la directive d'activation de notifications est fixée é 1. Ceci activera les notifications (optionnel, mais recommandé)
- Les contrles de service actifs sont désactivés sur le serveur central (optionnel, mais recommandé voir notes ci-dessous)
- Les contrles de commandes externes sont activés sur le serveur central (obligatoire)
- Les contrles de service passifs sont activés sur le serveur central (obligatoire)

Il y a trois autres éléments importants que vous devez conserver é l'esprit en configurant le serveur central :

- *Tous les services* qui sont supervisés par les serveurs répartis doivent comporter des définitions de service sur le serveur central.. Nagios ignorera les résultats des contrles de service passifs s'ils ne correspondent pas é un service qui a été défini.
- Si vous n'utilisez le serveur central que pour traiter des services dont les résultats sont fournis par des htes répartis, vous pouvez simplement désactiver tous les contrles de service pour l'ensemble du logiciel en mettant la variable execute_service_checks é 0.
- Si vous utilisez le serveur central pour superviser activement quelques services par lui-m'me (sans l'intervention des serveurs répartis), la variable *enable_active_check* de chaque définition de service pour les services dont les résultats sont fournis par les htes répartis doit 'tre positionnée é 0. Ceci emp'chera Nagios de vérifier activement ces services.

Il est important que vous désactiviez soit tous les contrles de service pour l'ensemble du logiciel, soit l'option *enable_active_checks* dans la définition de tout service surveillé par un serveur réparti. Cela assure que les contrles de service actifs ne sont jamais exécutés en temps normal. Les services continueront é 'tre ordonnancés é leur intervalle de contrle normal (3 minutes, 5 minutes, etc), mais ils ne seront jamais exécutés. Cette boucle de ré ordonnancement continuera aussi longtemps que Nagios tourne. Je vais expliquer bientt pourquoi ce type de fonctionnement

Et voilé ! Facile, non ?

Problémes rencontrés lors des contrles passifs

Pour toutes les utilisations intensives, nous pouvons dire que le serveur central s'appuie uniquement sur les contrles passifs pour la supervision. Faire totalement confiance aux contrles passifs pour superviser pose un probléme majeur : Nagios doit se fier é quelque chose d'autre pour fournir les données supervisées. Que se passe-t-il si l'hte distant qui envoie les résultats s'effondre ou devient injoignable ? Si Nagios ne contrle pas activement les services de cet hte, comment saura-t-il qu'il y a un probléme ?

Nous pouvons prévenir ce type de problémes en utilisant un autre module complémentaire pour superviser les résultats des contrles passifs qui arrivent

Le contrle de validité des données

Nagios offre une fonctionnalité qui teste la validité des résultats d'un test. On peut trouver plus d'informations é ce sujet ici. Cette fonctionnalité apporte une solution aux situations ou les htes distants peuvent arr'ter d'envoyer le résultat des tests passifs au serveur central. Elle permet d'assurer que le test est soit fourni passivement par les serveurs répartis, soit exécuté activement par le serveur central si nécessaire. Si les résultats fournis par le test du service deviennent "figés", Nagios peut 'tre configuré pour forcer un contrle actif depuis le serveur central.

Comment fait-on cela ? Sur le serveur central, il faut configurer ainsi les services qui sont surveillés par les serveurs répartis:

- L'option *check_freshness* dans la définition des services doit ’tre é 1. Ceci active le test de validité
- L'option *freshness_threshold* dans la définition des services doit ’tre positionné é une valeur qui indique le niveau de validité (**[NdT]** : *la traduction littérale est la "fracheur", mais il ne me semble pas adapté)* des données (telles que fournies par les serveurs répartis)
- L'option *check_command* dans la définition des services doit indiquer les commandes valides qui seront employées pour tester activement les services depuis le serveur central.

Nagios teste réguliérement la validité des résultats pour lesquels cette option est validée. L'option *freshness_threshold* dans chaque service détermine le niveau de validité pour celui-ci.

Par exemple, si sa valeur est 300 pour un de vos services, Nagios va considérer que les résultats du service sont "figés" s'ils ont plus de 5 minutes (300 secondes). Si vous ne spécifiez pas la valeur de *freshness_threshold,* Nagios calculera un seuil é partir de la valeur des options *normal_check_interval* ou de *retry_check_interval* (en fonction de l'état du service). Si les résultats sont "figés", Nagos exécutera la commande spécifiée dans *check_command* dans la définition du service, vérifiant ainsi activement ce service.

N'oubliez pas que vous devez définir l'option *check_command* dans la définition des services, pour pouvoir tester activement l'état d'un service depuis le serveur central. Dans des conditions normales, cette commande *check_command* ne sera pas exécutée (parce que les test actifs auront été désactivés globalement au niveau du programme, ou pour des services spécifiques). A partir du moment ou le contrle de validité des données est activé, Nagios exécutera cette commande, *m’me si les tests actifs ont été désactivés globalement au niveau du programme ou pour des services spécifiques.*

Si nous n'arrivez pas é définir des commandes pour tester activement un service depuis le serveur central (ou bien cela est un casse-t’te douloureux), vous pouvez simplement définir toutes les options *check_command* d'aprés un simple script qui retourne un état "critique". Voici un exemple : supposons que vous ayez défini une commande "service_fige" et utilisez cette commande dans l'option *check_command* de vos services. Elle pourrait ressembler é cela ..

```
define command{
        command_name    service-fige
                command_line /usr/local/nagios/libexec/service-fige.sh
        }
```

Le programme "service_figé.sh " dans */usr/local/nagios/libexec* pourrait ressembler é cela:

```
#!/bin/sh
/bin/echo "CRITICAL: Les resultats du service sont figes!"
exit 2
```

Ensuite, quand Nagios détecte que les résultats sont figés et lance la commande **service_fige.sh,** le script */usr/local/nagios/libexec/service-fige.sh* est exécuté et le service passe dans un état critique. Ceci déclenchera l'envoi de notifications, donc vous saurez finalement qu'il y a un probléme.

Test des htes

Maintenant, vous savez comment obtenir des résultats de contrles passifs depuis des serveurs répartis. Ceci signifie que le serveur central ne teste activement que ses propres services. Mais qu'en est-il des htes ? Vous en avez toujours besoin, non ?

Comme les tests des htes n'ont qu'un impact faible sur la surveillance (ils ne sont faits que s'ils sont vraiment nécessaires), je vous recommanderai bien de faire ces tests, de maniére active, depuis le serveur central. Ceci signifie que vous définirez le test des htes sur le serveur central de la m’me maniére que vous l'avez fait sur les serveurs répartis (également de la m’me maniére que sur un serveur normal, non répartis).

Les résultats de contrle passifs des htes sont disponible (ici), donc vous pourriez les utiliser dans votre architecture répartie mais cette méthode comporte certains problémes. Le principal étant que Nagios ne traduit pas les états problémes (arr’tés [down] ou injoignables [unreachable]) retournés par les vérifications passives des htes quand ils sont traités. Par conséquent, si vos serveurs de supervision ont une structure différentes en terme de parents ou enfants (et c'est ce qui se passe lorsque vos serveurs de supervisions ne sont pas exactement au m’me endroit), le serveur central va avoir une vision incorrecte des états des htes.

Si vous voulez vraiment envoyer des résultats passifs de contrle d'hte é un serveur de supervision central, vérifiez que :

- Le serveur central a activé la réception passive de vérification d'hte (requis)
- Le serveur réparti est configuré avec l'option de remontée de contrle d'hte activée.
- Le serveur réparti a une commande ochp définie

La commande ochp utilisé pour le traitement des vérifications d'hte, fonctionne de maniére similaire é la commande ocsp, utilisée pour le traitement des vérification des services (cf documentation ci dessus). Pour ’tre sur que les vérifications passives d'hte sont valides et é jour, il est nécessaire d'activer la validité des vérification pour les htes de la m’me maniére que pour les services.

Gestion de panne et redondance pour la supervision réseau

Introduction

Cette section décrit quelques scénarii dimplémentation de systémes de supervision redondante ainsi que plusieurs topologies réseau. Avec la redondance des systémes, vous pouvez maintenir la possibilité de surveiller votre réseau alors que le premier systéme sur lequel tourne Nagios pose probléme ou lorsque des parties du réseau deviennent injoignables.

Note: Si vous apprenez é utiliser Nagios, je suggére de ne pas essayer dimplémenter la redondance tant que vous n'tes pas habitué aux pré-requis déjé présentés. La redondance est un sujet relativement complexe, et il est encore plus difficile de limplémenter correctement.

Index

Pré-requis

Avant de penser pouvoir implémenter la redondance avec Nagios, vous devez 'tre familier avec ce qui suit

- Implémenter les Gestionnaires dévénements pour les htes et services
- Présenter une commande externe é Nagios via des scripts shell
- Exécuter des plugins sur des htes distants en utilisant soit NRPE, soit dautres méthodes
- Vérifier létat du processus Nagios avec le plugin check_nagios.

Exemples de scripts

Tous les exemples que jutilise dans cette documentation se trouvent dans le répertoire *eventhandlers/* de la distribution Nagios. Vous devrez probablement les modifier pour les faire fonctionner sur votre systéme

Scénario 1 - Supervision Redondante

Introduction

Ceci est une méthode facile (et nave) pour implémenter le monitoring redondant dhtes sur votre réseau, qui protégera seulement contre un nombre limité de problémes. Des réglages plus complexes sont nécessaires pour fournir une redondance plus pratique, une meilleure redondance é travers des segments réseau, etc.

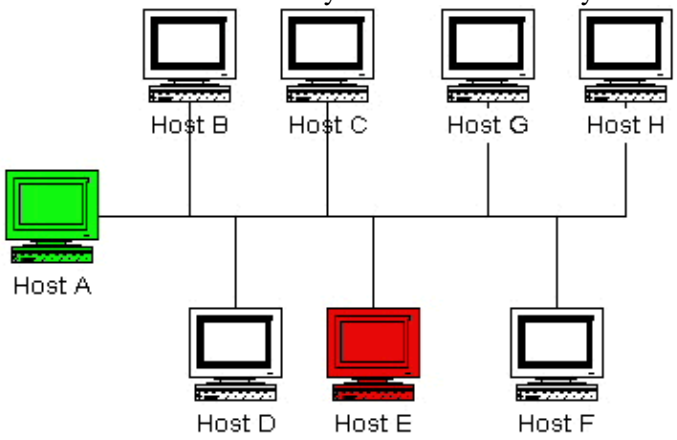
Buts

Le but de ce type dimplémentation de redondance est simple. Les htes "matre" et "esclave" surveillent les m'mes systémes et services sur le réseau. Dans des circonstances normales, le systéme "matre" prendra en charge lenvoi des notifications aux contacts concernant les problémes détectés. Nous voulons que systéme "esclave" fasse fonctionner Nagios et prenne en charge la notification des problémes si:

1. Soit le systéme "Matre" faisant fonctionner Nagios ne répond plus
2. Soit le processus Nagios sur le systéme "Matre" arr'te de fonctionner.

Diagramme de topologie réseau

Le diagramme ci-dessous montre une configuration réseau trés simple. Pour ce scénario, je vais supposer que les systémes A et E font tourner tous deux Nagios et surveillent tous les systémes que lon y voit. Le systéme A sera considéré comme étant le systéme "matre" et le systéme E sera considéré comme étant lesclave.



Réglages initiaux du programme

Le systéme esclave (systéme E) a sa directive initiale de notification enable_notifications désactivée, afin de lui éviter denvoyer des notifications autant pour les htes que pour les services. Il faut aussi sassurer que le systéme esclave a sa directive check_external_commands activée. Cest plutt simple

Configuration Initiale

Il faudra ensuite considérer les changements entre le(s) fichier(s) de configuration des objets sur les systémes matre et esclave

Je vais supposer que le systéme matre (systéme A) est configuré pour surveiller des services sur tous les systémes montrés dans le diagramme ci-dessus. Le systéme esclave (systéme E) doit 'tre configuré pour surveiller les m'mes systémes et services, avec les ajouts suivants aux fichiers de configuration

- La définition du systéme A (dans le fichier de configuration du systéme E) devrait avoir un gestionnaire dévénement défini. Supposons que le nom du gestionnaire dévénement pour le systéme sappelle handle-master-host-event.
- Le fichier de configuration sur le systéme E devrait avoir un service défini pour surveiller létat du processus Nagios sur le systéme A. Supposons que vous ayez défini cette vérification de service pour

lancer le plugin *check_nagios* sur le systéme A.
- La définition du service pour la surveillance du processus Nagios sur le systéme A devrait 'tre un gestionnaire dévénement. Supposons que ce service gestionnaire dévénement soit handle-master-proc-event.

Il est important de noter que le systéme A (le systéme matre) ne sait rien du systéme E (le systéme esclave). Dans ce scénario, il nen a simplement pas besoin. Bien évidemment, vous pouvez surveiller les services du systéme E depuis le systéme A, mais cela na rien é voir avec limplémentation de la redondance

Définition de commandes de Gestion dEvénements

Faisons une petite pause, et décrivons les définitions de commandes de gestion dévénement sur lesclave. Voici un exemple

```
define command{
        command_name    handle-master-host-event
        command_line    /usr/local/nagios/libexec/eventhandlers/handle-master-h
        }

define command{
        command_name    handle-master-proc-event
        command_line    /usr/local/nagios/libexec/eventhandlers/handle-master-p
        }
```

Cela implique que vous ayez placé les scripts de gestion dévénements dans le répertoire */usr/local/nagios/libexec/eventhandlers*. Vous pouvez les placer ailleurs, mais vous devrez modifier les exemples que jai donnés ici.

Scripts de Gestion dEvénements

Ok, regardons é quoi ressemble ce script

Gestionnaire dEvénement de systéme (handle-master-host-event):

```
#!/bin/sh
# Only take action on hard host states
case "$2" in
HARD)
        case "$1" in
        DOWN)
                # The master host has gone down!
                # We should now become the master host and take
                # over the responsibilities of monitoring the
                # network, so enable notifications
                /usr/local/nagios/libexec/eventhandlers/enable_notifications
                ;;
        UP)
                # The master host has recovered!
                # We should go back to being the slave host and
                # let the master host do the monitoring, so
```

```
                # disable notifications
                /usr/local/nagios/libexec/eventhandlers/disable_notifications
                ;;
        esac
        ;;
esac
exit 0
```

Gestionnaire dEvénements de Service (handle-master-proc-event):

```
#!/bin/sh
# Only take action on hard service states
case "$2" in
HARD)
        case "$1" in
        CRITICAL)
                # The master Nagios process is not running!
                # We should now become the master host and
                # take over the responsibility of monitoring
                # the network, so enable notifications
                /usr/local/nagios/libexec/eventhandlers/enable_notifications
                ;;
        WARNING)
        UNKNOWN)
                # The master Nagios process may or may not
                # be running.. We wont do anything here, but
                # to be on the safe side you may decide you
                # want the slave host to become the master in
                # these situations
                ;;
        OK)
                # The master Nagios process running again!
                # We should go back to being the slave host,
                # so disable notifications
                /usr/local/nagios/libexec/eventhandlers/disable_notifications
                ;;
        esac
        ;;
esac
exit 0
```

Que fait ce script pour nous ?

La notification sur le systéme esclave (systéme E) est désactivée, donc il nenverra pas de notifications pour les systémes autant que pour les services tant que le processus Nagios fonctionne sur le systéme matre (systéme A).

Le processus Nagios sur le systéme esclave (systéme E) devient matre quand

- Le systéme matre (systéme A) tombe et le gestionnaire dévénement de systéme *handle-master-host-event* est exécuté.
- Le processus Nagios sur le systéme matre (systéme A) sarr'te de fonctionner et le gestionnaire dévénement du service *handle-master-proc-event* est exécuté.

Dés que le processus Nagios sur le systéme esclave (systéme E) a la notification activée, il sera capable denvoyer des notifications quant aux services ou problémes systéme ou encore les retours é la normale. A ce moment, le systéme E a effectivement pris la responsabilité de notifier les contacts des problémes de systémes et services!

Le processus Nagios sur le systéme E retourne é son état desclave quand

- Le systéme A revient é la normale et le gestionnaire dévénement de systéme *handle-master-host-event* est exécuté.
- Le processus Nagios sur le systéme A revient é la normale et que le gestionnaire dévénement de service *handle-master-proc-event* est exécuté.

Dés que le processus Nagios sur le systéme E est désactivé, il nenverra plus de notification concernant les problémes liés aux services et aux systémes ou encore les retours é la normale. Dés ce moment, le systéme E a pris la responsabilité de notifier les contacts des problémes du processus Nagios sur le systéme A. Tout revient maintenant dans le m'me état que lorsque lon a démarré!

Délais

La redondance dans Nagios nest en rien parfaite. Un des nombreux problémes est le délai entre le moment o le matre tombe et que lesclave prend le relai. En voici les raisons

- Lintervalle entre la rupture du systéme matre et la premiére fois que le systéme esclave détecte le probléme.
- Le temps quil faut pour vérifier que le systéme matre a réellement un probléme (en utilisant une nouvelle fois la commande check dun service ou dun systéme sur le systéme esclave)
- Le temps entre lexécution du gestionnaire dévénement et la fois suivante o Nagios va vérifier la présence dune commande externe

Vous pouvez minimiser ce délai en

- Sassurant que le processus Nagios sur le systéme E (re)vérifie un ou plusieurs services avec une fréquence élevée. Ceci peut 'tre fait en utilisant les arguments *check_interval* et *retry_interval* dans chaque définition de service.
- Sassurer que le nombre de re-vérifications de la présence du systéme A (sur le systéme E) permette une détection des problémes liés au systéme plus rapidement. Ceci peut 'tre fait en utilisant largument *max_check_attempts* dans la définition du systéme.
- Augmenter la fréquence de vérification des commandes externes sur le systéme E. Ceci peut 'tre fait en modifiant loption command_check_interval dans le fichier de configuration principal.

Quand Nagios revient é la normale sur le systéme A, il y a aussi un délai avant que le systéme E ne redevienne esclave. Cest du aux faits suivants

- Le temps entre le retour é la normale du systéme A la fois suivante o le processus Nagios sur le systéme E détecte le retour é la normale.

- Lintervalle entre lexécution du gestionnaire dévénement sur le systéme E et la fois suivante o le processus Nagios sur le systéme E vérifie la présence de commandes externes

Les intervalles exacts entre le transfert des responsabilités de supervision dépendent du nombre de services définis, lintervalle auquel les services sont vérifiés, et un peu de chance. A tous niveaux, cest mieux que rien.

Cas spéciaux

Il y a une chose é laquelle il faut ’tre attentif Si le systéme A tombe, le systéme de notifications sur le systéme E sera activé et prendra la responsabilité de notifier les contacts de problémes. Lorsque le systéme A revient é la normale, le systéme E aura sa notification désactivée. Si, quand le systéme A revient é la normale, le processus Nagios ne redémarre pas correctement, il y aura une période de temps o aucun systéme ne notifiera les contacts de problémes! Heureusement on peut compter sur la logique de vérification de services de Nagios. La fois suivante o le processus Nagios sur le systéme E vérifie létat du processus Nagios sur le systéme A, il verra quil ne fonctionne pas. Le systéme E aura donc sa notification activée et prendra la responsabilité de notifier les contacts des problémes.

Le temps o aucun systéme ne surveille est assez difficile é déterminer. Toutefois, cette période peut ’tre minimisée en augmentant la fréquence de vérification (sur le systéme E) du processus Nagios sur le systéme A. Le reste est une question de chance, mais le temps de "blackout" total ne devrait pas ’tre trop mauvais.

Scénario 2 - Supervision en mode haute disponibilité

Introduction

La supervision avec gestion de panne est pratiquement identique. Il existe quand m’me des différences avec le systéme précédent (scénario 1).

Buts

Le but principal de la gestion de panne est davoir le processus Nagios sur le systéme esclave en hibernation tant que le processus Nagios sur le systéme maitre fonctionne. Si le processus sur le systéme matre arr’te de fonctionner (ou si le systéme tombe), le processus Nagios sur le systéme esclave commence é tout surveiller.

Bien que la méthode décrite dans la partie scénario 1 permette de continuer é recevoir la notification si le systéme maitre tombe, il y a quelques piéges. Le plus gros probléme est que le systéme esclave surveille les m’mes systémes que le maitre *au m’me moment*! Ceci peut causer des problémes de trafic excessif et charger les machines surveillées si vous avez beaucoup de services définis. Voici une maniére de contourner ce probléme

Réglages initiaux du programme

Désactiver la vérification active des services et la notification sur le systéme esclave en utilisant les directives execute_service_checks et enable_notifications. Ceci évitera au systéme esclave de surveiller les services et les systémes et denvoyer des notifications tant que le processus Nagios sur le systéme matre fonctionne. Assurez-vous davoir la directive check_external_commands activée sur le systéme esclave.

Vérification du processus principal

Créer un tôche programmée [cron job] sur le systéme esclave qui lance périodiquement un script (disons toutes les minutes) qui vérifie létat du processus Nagios sur le systéme matre (en utilisant le plugin *check_nrpe* sur le systéme esclave et le démon nrpe sur le systéme matre). Le script va vérifier le code de retour du *plugin nrpe*. Sil retourne un état non-OK, le script va envoyer les commandes appropriées au fichier de commandes externes pour activer la notification et la surveillance des services. Si le plugin retourne un état OK, le script enverra les commandes pour désactiver la surveillance active des services et la notification.

En procédant comme suit, vous nutilisez quun processus de surveillance de systéme et de service é la fois, ce qui est plus efficace que de surveiller en double.

Notez aussi que vous *ne* devez *pas* définir de gestionnaires dévénements comme défini dans le scénario 1 , car les contraintes sont surmontées de maniére différente.

Cas Supplémentaires

Vous avez maintenant implémenté une gestion de panne de maniére plutt basique. Il y a toutefois dautres maniéres de procéder pour que cela fonctionne de maniére plus douce.

Le gros probléme avec cette technique est surtout le fait que lesclave ne connait pas létat courant des services ou des systémes au moment m'me o il prend é sa charge le travail de surveillance. Une maniére de solutionner ce probléme est dactiver la commande ocsp sur le systéme matre et de lui demander de rapporter les résultats des vérifications é lesclave en utilisant lajout nsca. De cette maniére, le systéme esclave posséde un statut mis é jour des informations de tous les services et des systémes sil venait é prendre en charge la surveillance. Tant que les vérifications actives ne sont pas activées sur le systéme esclave, il neffectuera aucune vérification active. Malgré tout, il exécutera les vérifications si nécessaire. Cela signifie quautant le matre que lesclave exécuteront les vérifications de systéme comme il le faut, ce qui nest pas vraiment une bonne affaire puisque la majorité des surveillances fonctionnent par rapport aux services.

Voilé é peu prés tout ce quil y a é configurer.

Détection et gestion de l'oscillation d'état

Introduction

Nagios supporte la détection optionnelle des htes et des services qui "oscillent" [NdT: ou bagotent]. L'oscillation intervient quand un service ou un hte change d'état trop fréquemment, provoquant une temp’te de notifications de problémes et de rétablissement. L'oscillation peut ’tre l'indice de problémes de configuration (i.e. des seuils positionnés trop bas) ou de vrais problémes sur le réseau.

Avant d'aller plus loin, permettez-moi de signaler qu'implémenter la détection de l'oscillation a été assez difficile. Comment déterminer ce que "trop fréquemment" veut dire concernant les changements d'état de tel hte ou service ? La premiére fois que je me suis penché sur la détection de l'oscillation, j'ai cherché des informations sur la faéon dont on peut ou doit procéder. Voyant que je n'en trouvais pas, j'ai décidé de définir ce qui pourrait ’tre une solution raisonnable. Les méthodes utilisées par Nagios pour détecter l'oscillation de l'état des htes et des services sont décrites ci-dessous

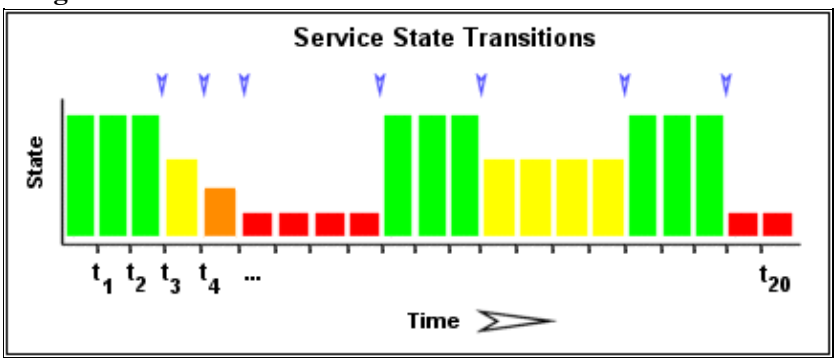
Détection de l'oscillation de service

Chaque fois d'un contrle de service résulte dans un état hard ou un état de rétablissement soft, Nagios contrle si le service a commencé ou arr’té d'osciller. Il le fait en stockant les 21 derniers résultats de contrle de service dans un tableau. Les résultats les plus récents écrasent les anciens dans le tableau.

Le contenu du tableau d'historique des états est parcouru (depuis le plus ancien résultat jusqu'au plus récent) pour déterminer le pourcentage total de changements d'état survenus durant les 21 derniers contrles du service. Un changement d'état survient quand un état archivé est différent de l'état archivé qui le précéde immédiatement dans le tableau. Comme nous conservons les résultats des 21 derniers contrles de service dans le tableau, il y a 20 changements d'état possibles.

L'image 1 ci-dessous montre un tableau chronologique d'états de service. Les états OK sont en vert, les WARNING en jaune, les CRITICAL en rouge, et les UNKNOWN en orange. Des fléches bleues marquent les moments o des changements d'états sont survenus.

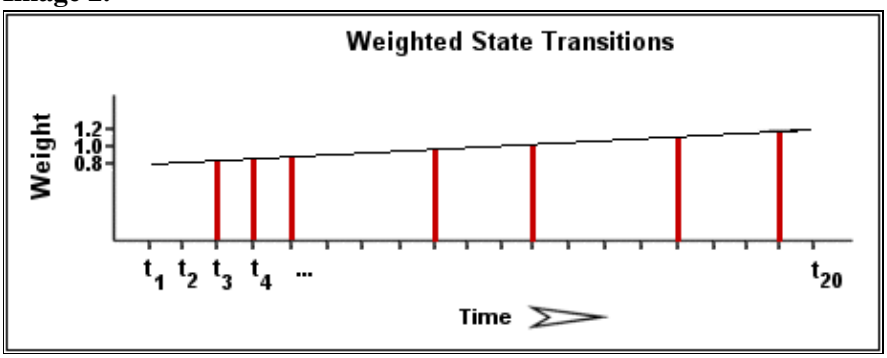
Image 1.



Les services qui changent rarement d'état auront un pourcentage moins élevé de changement d'état que ceux qui changent beaucoup d'état. Comme l'oscillation est associée avec des changements d'état fréquents, nous pouvons utiliser la valeur de changement d'état calculée pour une période donnée (dans notre cas, les 21 derniers contrles de service) pour déterminer si un service oscille ou non. Mais ce n'est pas encore assez précis /p>

Il parat évident que les changements d'état les plus récent ont plus de poids que les anciens, si bien qu'il nous faut recalculer le pourcentage total de changements d'état du service selon une espéce de courbe Pour simplifier, j'ai décidé d'utiliser un rapport linéaire entre le temps et la pondération pour le calcul de ce pourcentage. Les fonctions de détection de l'oscillation sont conéues actuellement pour que le changement d'état le plus récent possible pése 50% plus lourd que le plus ancien. L'image 2 montre combien le poids des changements d'état récents est supérieur é celui des anciens lors du calcul du pourcentage total de changements d'état d'un service particulier. Si vous voulez savoir exactement comment ce calcul pondéré est réalisé, regardez le code dans *base/flapping.c*

Image 2.



Prenons un rapide exemple de détection de l'oscillation. L'image 1 montre le tableau d'historique des résultats de contrle d'un service particulier. Les résultats les plus anciens sont é gauche et les plus récents é droite. Nous voyons dans cet exemple qu'il y a eu au total 7 changements d'état (en t_3, t_4, t_5, t_9, t_{12}, t_{16}, et t_{19}). Sans pondération des changements d'état en fonction du temps, cela nous donnerait un total de 35% de changements d'état (7 changements d'état sur un maximum possible de 20). Quand on applique la pondération en fonction du moment d'apparition, le pourcentage est de moins de 35%. C'est logique dans la mesure o la plupart des changements d'états sont plutt anciens. Disons que le pourcentage pondéré est finalement de 31%

Ainsi donc, que signifient 31% de changements d'états ? Hé bien, si le service n'oscillait pas auparavant et que 31% est supérieur ou égal é la valeur spécifiée par le paramétre high_service_flap_threshold de la définition du service, Nagios considére que le service vient de commencer é osciller. Si le service *oscillait* auparavant et que 31% est inférieur ou égal é la valeur spécifiée par le paramétre low_service_flap_threshold de la définition du service, Nagios considére que le service vient de s'arr'ter d'osciller. Si aucune de ces deux conditions n'est remplie, Nagios ne fait rien de plus concernant le service, car soit il n'oscille pas, soit il oscille toujours

Détection de l'oscillation d'hte

La détection de l'oscillation d'hte fonctionne de maniére similaire é la détection d'oscillation de service, avec une différence importante : Nagios essaiera de déterminer si un hte oscille é chaque contrle de l'état de l'hte *et* é chaque contrle d'un service associé é cet hte. Pourquoi cela ? Hé bien, parce qu'avec les services, nous savons que l'intervalle minimal de temps entre deux détections d'oscillation consécutives sera égal é l'intervalle de contrle du service. Avec les htes, nous n'avons pas d'intervalle de contrle, du fait que les htes ne sont pas supervisés de maniére réguliére ô ils ne sont contrlés que lorsque c'est nécessaire. L'oscillation d'un hte sera contrlée si son état a changé depuis la derniére détection d'oscillation de cet hte *ou* si son état n'a pas changé, mais qu'au moins *x* temps s'est écoulé depuis la derniére détection d'oscillation. Ce temps *x*est égal é la moyenne des intervalles de contrles de tous les services associés avec l'hte. C'est la meilleure méthode que j'ai pu imaginer pour déterminer la fréquence de la détection d'oscillation d'un hte

Comme pour les services, Nagios stocke les résultats des 21 derniers contrles d'oscillation d'hte dans un tableau destiné é l'algorithme de détection d'oscillation. Les changements d'état sont pondérés en fonction du moment, et le pourcentage total de changements d'état est calculé de la m'me maniére que dans l'algorithme de détection d'oscillation des services.

Si un hte *n'oscillait pas* précédemment et que le calcul de son pourcentage total de changement d'état est *supérieur ou égal* é la valeur spécifiée dans le paramétre high_host_flap_threshold, Nagios considére que l'hte commence juste é osciller. Si l'hte *oscillait* précédemment et que le calcul de son pourcentage total de changement d'état est *inférieur ou égal* é la valeur spécifiée par le paramétre low_host_flap_threshold, Nagios considére que l'hte vient d'arréter d'osciller. Si aucune de ces deux conditions n'est remplie, Nagios ne fait rien de plus concernant l'hte, car soit il n'oscille pas, soit il oscille toujours

Seuils de détection d'oscillation pour les htes et les services

Si vous utilisez le fichier de définition des htes é base de modéles, vous pouvez spécifier un seuil de détection d'oscillation pour les htes et les services en ajoutant les directives **low_flap_threshold** et **high_flap_threshold** dans les définitions individuelles d'htes et services. Si ces directives *ne sont pas présentes* dans la définition d'un hte ou d'un service, les valeurs globales de seuil seront utilisées.

De la m'me maniére, vous pouvez activer/désactiver la détection d'oscillation pour des htes ou services particuliers avec la directive **enable_flap_detection** appliquée é la définition d'un objet. Notez que la détection d'oscillation doit 'tre activée pour l'ensemble du programme Nagios (avec la directive enable_flap_detection du fichier de configuration principal) si vous souhaitez que celle-ci fonctionne.

Gestion de l'oscillation

Quand un service ou un hte commence é osciller, Nagios fait trois choses :

1. il journalise un message indiquant que le service ou l'hte oscille
2. il ajoute un commentaire non persistant é l'hte ou au service indiquant qu'il oscille
3. il supprime les notifications pour le service ou l'hte concerné (c'est l'un des filtre de l'algorithme de notification)

Quand un service ou un hte s'arr'te d'osciller, Nagios fait les chose suivantes :

1. il journalise un message indiquant que le service ou l'hte n'oscille plus
2. il supprime le commentaire qui avait été ajouté au service ou é l'hte lorsqu'il avait commencé é osciller
3. il léve le blocage des notifications sur le service ou l'hte concerné (les notifications restant assujetties é l'algorithme de notification)

Parallélisation des contrles de service

Introduction

Une des fonctionnalités de Nagios est la possibilité d'effectuer des contrles de service en paralléle. Cette documentation essaiera d'expliquer en détail ce que cela signifie et quel est son impact sur les services que vous avez définis.

Comment fonctionne la parallélisation

Avant que je n'explique comment fonctionne la parallélisation du contrle des services, vous devez d'abord comprendre comment Nagios effectue l'ordonnancement des événements. Tous les événements internes de Nagios (c.-é-d. rotation des fichiers journaux, contrle des commandes externes, contrle des services, etc.) sont placés dans une file d'attente. Chaque membre de la file d'attente est ordonnancé pour une exécution é un instant donné. Nagios fait de son mieux pour que tous les événements soient exécutés au bon moment, bien que certains puissent ’tre retardés si Nagios est occupé é autre chose.

Les contrles de services sont un des types d'événement qui sont placés dans la file d'attente de Nagios. Lorsqu'un contrle de service doit ’tre effectué, Nagios démarre un nouveau processus (avec un appel é la fonction fork()) et effectue le contrle de service (c.-é-d. un quelconque plugin). Cependant, Nagios *n'attend pas* que le contrle de service s'achéve ! Il retourne plutt s'occuper des autres événements qui se trouvent dans la file d'attente

Que se passe-t-il donc lorsque le contrle de service s'achéve ? Hé bien, le processus lancé par Nagios pour effectuer ce contrle envoie un message é Nagios avec les résultats du contrle. C'est alors au tour de Nagios de vérifier la présence et d'analyser les résultats de ce contrle de service quand il aura le temps.

Afin que Nagios supervise effectivement, il doit analyser les résultats des contrles de services qui sont terminés. Ceci est effectué via un processus de "récolte". Les "récoltes" de service sont un autre type d'événement placé dans la file d'attente de Nagios. La fréquence de ces événements de "récolte" est déterminée par le paramétre service_reaper_frequency du fichier de configuration principal. Lorsqu'un événement de "récolte" est exécuté, il cherche tous les messages qui contiennent le résultat d'un contrle de service terminé. Ces résultats sont alors traités par l'algorithme de contrle des services. A partir de cela, Nagios détermine si des htes doivent ou non ’tre contrlés, si des notifications doivent ’tre envoyées, etc. Lorsque les résultats des contrles de service ont été analysés, Nagios reprogramme le prochain traitement des contrles de service et le place dans la file d'attente pour une exécution ultérieure. Ce qui termine le cycle contrle de service/traitement !

Pour ceux d'entre vous qui veulent vraiment savoir, mais qui n'ont pas regardé le code source, Nagios utilise des files de messages pour traiter les communications entre Nagios et le processus qui effectue le contrle de service

Attrape-nigauds possibles

Vous devez savoir qu'il y a quelques ennuis potentiels liés é la parallélisation du contrle des services. Comme plusieurs contrles de service peuvent s'exécuter simultanément, ils peuvent interférer les uns avec les autres. Vous devrez évaluer le type de contrles de services qui tournent et prendre les mesures appropriées pour vous prémunir des conséquences indésirables. Ceci est particuliérement important si vous avez plus d'un contrle de

service qui accéde au m’me matériel (comme un modem). De m’me, si au plusieurs contrles de services se connectent au m’me démon sur un hte distant pour vérifier une information, assurez-vous que le démon peut traiter les connexions simultanées.

Heureusement, vous pouvez effectuer certaines manipulations pour éviter les "collisions" de contrles de services

1. Le moyen le plus simple pour éviter les collisions est d'utiliser le paramétre service_interleave_factor. Entrelacer les services aidera é réduire la charge imposée par les contrles de services sur les htes distants. Réglez la variable pour utiliser le calcul "débrouillard" [NdT : smart] du facteur d'entrelacement, puis réglez-la manuellement si vous estimez que c'est nécessaire.
2. Ensuite, vous pouvez régler le paramétre *max_check_attempts* de chaque définition de service é une valeur supérieure é un. Si le contrle de service entre en collision avec un autre contrle en cours, Nagios réessaiera le contrle de service *max_check_attempts - 1* fois avant d'envoyer une notification du probléme é quelqu'un.
3. Vous pouvez aussi essayer d'implémenter un algorithme du type "retrait et réessai" [NdT : back-off and retry] dans le code de contrle de service, bien que vous puissiez trouver cela trop difficile ou trop consommateur de temps.
4. Si rien de tout cela ne fonctionne, vous pouvez emp’cher la parallélisation des contrles de service en réglant l'option max_concurrent_checks é 1. Cela ne permettra le contrle que d'un seul service é la fois, ce n'est donc pas une solution spectaculaire. S'il y a suffisamment de demandes, j'ajouterai une option aux définitions de services qui vous permettra de préciser si le contrle de chaque service doit ’tre ou non parallélisé. S'il n'y a pas assez de demandes, je ne le ferai pas

Un autre élément é noter est l'effet de la parallélisation des contrles de service sur les ressources de la machine hébergeant Nagios. Effectuer un grand nombre de contrles de services en paralléle peut grever les ressources mémoire et processeur. Le paramétre inter_check_delay_method essaiera de minimiser la charge imposée sur votre machine en répartissant les contrles de maniére homogéne dans le temps (si vous utilisez la méthode "débrouillard" [NdT : smart]), mais ce n'est pas une solution totalement sre. Afin d'avoir un regard sur le nombre de contrles de services qui peuvent s'exécuter en m’me temps, utilisez le paramétre max_concurrent_checks. Vous devrez affiner sa valeur, basée sur le nombre total de services que vous contrlez, les ressources systéme disponibles (cadence du processeur, mémoire, etc.) et les autres processus qui tournent sur votre machine. Pour avoir davantage d'informations sur la faéon de régler le paramétre *max_concurrent_checks* pour qu'il corresponde é vos exigences, lisez la documentation relative é l'ordonnancement des contrles.

Ce qui n'est pas parallélisé

Il faut se souvenir que seule *l'éxecution* des contrles de services a été parallélisée. Il y a une bonne raison é cela - les autres choses ne peuvent pas l’’tre d'une maniére trés sre ou trés saine. En particulier les gestionnaires d'événements, les notifications de contact, le traitement des contrles de services et des contrles d'htes *ne sont pas* parallélisés. Voici pourquoi

Les *gestionnaires d'événements* ne sont pas parallélisés du fait de leur conception m’me. Une grande part de leur puisssance provient de leur capacité é résoudre les problémes de maniére préventive. Par exemple, redémarrer le serveur web quand le service HTTP de la machine locale est détecté DOWN. Pour éviter que plusieurs gestionnaires d'événements n'essayent de "résoudre" des problémes en paralléle (sans savoir ce que font les autres), j'ai décidé de ne pas les paralléliser.

Les *notifications de contact* ne sont pas parallélisées é cause des méthodes de notification que vous pouvez utiliser. Si, par exemple, une notification de contact utilise un modem pour composer le numéro de votre pager et lui envoyer un message, elle a besoin de l'accés exclusif au modem pendant que la notification s'effectue. Si plusieurs notifications étaient exécutées en paralléle, une seule serait effectuée car les autres n'auraient pas accés au modem. Il y a des moyens de contourner cela, comme employer une méthode du style "retrait et réessai" dans le script de notification, mais j'ai décidé de ne pas me fier aux utilisateurs qui pourraient implémenter ce type de fonction dans leurs scripts. Une note rapide : si vous avez des contrles de services qui utilisent un modem, assurez-vous que les scripts de notification qui effectuent la numérotation peuvent réessayer d'accéder au modem aprés un échec. C'est nécessaire, car un contrle de service peut s'exécuter en m’me temps qu'une notification !

Le *traitement des résultats des contrles de services* n'a pas été parallélisé. Et cela pour éviter les situations o de multiples notifications relatives é des problémes ou au rétablissement d'htes ou de service peuvent ’tre envoyées lorsqu'un hte passe dans l'état DOWN, UNREACHABLE, ou RECOVERY.

L'escalade des notifications

Introduction

Nagios supporte l'escalade *optionelle* des notifications envoyées aux contacts pour des services ou htes. Je vais en expliquer rapidement le fonctionnement, bien que cela se comprenne facilement

Les escalades des notifications de service

L'escalade des notifications de service est effectuée en définissant les escalades de service dans votre fichier de configuration d'objet. Ces définitions sont utilisées pour que les notifications relatives é un service particulier escaladent.

Les escalades des notifications d'htes

L'escalade des notifications d'htes est effectuée en définissant les escalades d'htes dans votre fichier de configuration d'objets. Les exemples que je donne ci-dessous utilisent tous les définitions d'escalade de service, mais les escalades d'htes fonctionnent de la m'me maniére (é l'exception du fait qu'elles sont utilisées pour les notifications d'htes et non de services).

Quand y a-t-il escalade des notifications?

Cette escalade a lieu *si et seulement si* au moins une définition d'escalade correspond é la notification qui est envoyée. Si é une notification d'hte ou de service *ne s'applique aucune* définition d'escalade valide, le groupe de contacts précisé soit dans le groupe d'hte ou de service sera utilisé pour la notification. Regardons l'exemple ci-dessous :

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   90
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      6
        last_notification       10
        notification_interval   60
        contact_groups          nt-admins,managers,everyone
        }
```

Remarquez qu'il y a des "trous" dans les définitions d'escalade de notification. En particulier, les notifications 1 et 2 ne sont pas prises en compte par les escalades, ni celles au-delé de 10. Pour la premiére et la seconde

notification, de m’me que pour celles au-delé de la dixiéme, le groupe de contacts *par défaut* précisé dans la définition de service est utilisé. Dans tous les exemples que j'utiliserai, je considérerai que le groupe de contacts par défaut des définitions de service s'appelle *nt-admins*.

Groupes de contact

Lorsqu'on définit les escalades de notification, il est important de garder é l'esprit que tous les groupes de contact qui appartenaient aux escalades "plus basses" (i.e celles avec les plus bas numéros de notification) doivent aussi ’tre inclus dans les définitions d'escalade "plus hautes". Cela doit ’tre effectué pour s'assurer que ceux qui se voient notifier un probléme *continuent* de recevoir les notifications lorsque le probléme est escaladé. Exemple:

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   90
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      6
        last_notification       0
        notification_interval   60
        contact_groups          nt-admins,managers,everyone
        }
```

Le premier (ou "plus bas") niveau d'escalade comprend é la fois les groupes de contact *nt-admins* et *managers*. Le dernier (ou "plus haut") niveau d'escalade comprend les groupes de contact *nt-admins*, *managers*, et *everyone*. Remarquez que le groupe de contact *nt-admins* fait partie des deux définitions d'escalade. C'est pour qu'il continue é ’tre prévenu s'il reste des problémes aprés que les deux premiéres notifications de service aient été envoyées. Le groupe de contact *managers* apparat d'abord dans la définition d'escalade la "plus basse" - il reéoit sa premiére notification lorsque la troisiéme notification de probléme est envoyée. Nous voulons que le groupe *managers* continue de recevoir des notifications si le probléme persiste aprés cinq notifications, il fait donc partie de la "plus haute" définition d'escalade.

Recoupement des portées des escalades

Les définitions d'escalade de notification peuvent avoir des portées qui se recoupent. Prenons l'exemple suivant :

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
```

```
        notification_interval   20
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      4
        last_notification       0
        notification_interval   30
        contact_groups          on-call-support
        }
```

Dans l'exemple ci-dessus :

- Les groupes de contact *nt-admins* et *managers* reéoivent la troisiéme notification
- Les trois groupes de contact reéoivent les quatriéme et cinquiéme notifications
- Seul le groupe de contact *on-call-support* reéoit les notifications é partir de la sixiéme notification

Notifications de reprise d'activité

Les notifications de reprise d'activité sont légérement différentes des notifications de probléme lorsqu'il s'agit d'escalade. Prenons l'exemple suivant :

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   20
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      4
        last_notification       0
        notification_interval   30
        contact_groups          on-call-support
        }
```

Si, aprés trois notifications de probléme, une notification de reprise d'activité est envoyée au service, qui reéoit la notification ? La reprise d'activité est la quatriéme notification envoyée. Cependant, le code d'escalade est suffisamment bien fait pour que seules les personnes qui ont reéu la troisiéme notification reéoivent celle de reprise d'activité. Dans ce cas, les groupes de contact *nt-admins* et *managers* recevront la notification de reprise d'activité.

Intervalles de notification

Vous pouvez modifier la fréquence é laquelle les notifications escaladées sont émises pour un hte ou un service particulier en utilisant le paramétre *notification_interval* de la définition d'escalade de groupe d'htes ou de service. Par exemple :

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   45
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      6
        last_notification       0
        notification_interval   60
        contact_groups          nt-admins,managers,everyone
        }
```

Dans cet exemple nous voyons que l'intervalle de notification par défaut pour les services est de 240 minutes (c'est la valeur donnée dans la définition du service). Quand la notification de ce service est escaladée lors des 3éme, 4éme, et 5éme notifications, un intervalle de 45 minutes sera utilisé entre les notifications. Lors de la 6éme notification et des suivantes, l'intervalle de notification sera de 60 minutes, comme il est spécifié dans la seconde définition d'escalade.

Comme il est possible d'avoir des définitions d'escalade qui se chevauchent pour un groupe d'htes ou un service donné, et comme un hte peut ’tre membre de plusieurs groupes d'htes, Nagios doit décider quel intervalle de notification utiliser quand des définitions d'escalade se chevauchent. Dans tous les cas de chevauchement, Nagios choisira l'intervalle de notification le plus court. Prenez l'exemple suvant :

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   45
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      4
        last_notification       0
        notification_interval   60
```

```
        contact_groups          nt-admins,managers,everyone
        }
```

Nous voyons que les deux définitions d'escalade se chevauchent sur les 4éme et 5éme notifications. Pour ces notifications, Nagios utilisera un intervalle de notification de 45 minutes, car c'est le plus petit intervalle présent dans les définitions d'escalade valides de ces notifications.

Une derniére remarque é propos des intervalles de notification concerne les intervalles de 0. Un intervalle de 0 signifie que Nagios ne doit émettre une notification que pour la premiére notification valide durant cette définition d'escalade. Toutes les notifications suivantes pour le groupe d'hte ou le service seront supprimées. Prenez cet exemple :

```
define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      3
        last_notification       5
        notification_interval   45
        contact_groups          nt-admins,managers
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      4
        last_notification       6
        notification_interval   0
        contact_groups          nt-admins,managers,everyone
        }

define serviceescalation{
        host_name               webserver
        service_description     HTTP
        first_notification      7
        last_notification       0
        notification_interval   30
        contact_groups          nt-admins,managers
        }
```

Dans l'exemple ci-dessus, il y aurait au maximum 4 notifications de probléme envoyées é propos du service. Ceci est d au fait que l'intervalle de notification de 0 dans la seconde définition d'escalade indique qu'une seule notification doit ’tre émise (é partir de et en incluant la 4éme notification) et que toutes les notifications suivantes doivent ’tre supprimées. A cause de cela, la troisiéme définition d'escalade du service est sans effet, car il n'y aura jamais plus de quatre notifications.

Restrictions de période de temps

Dans des circonstances normales, les escalades peuvent servir aux heures o une notification pourrait normalement ’tre envoyée pour le service. Cette "fen’tre d'heures de notification" est déterminée par la directive *notification_period* de la definition de service.

Vous pouvez optionnellement restreindre l'escalade pour qu'elle ne soit prise ne compte pendant une période de temps spécifique en utilisant la directive *escalation_period* dans la définition de l'escalade de service. Si vous utilisez la directive *escalation_period* pour spécifier une timeperiod pendant laquelle utiliser l'escalade, l'escalade sera prise en compte uniquement pendant ces heures. Si vous ne spécifiez aucune directive *escalation_period*, l'escalade peut ’tre prise en compte é n'importe quelle heure pendant la "fen’tre d'heures de notification" pour le service.

Notez que la notification est toujours soumise aux restriction d'heure normales imposées par la directive *notification_period* de l'escalade de service, et donc que la période de temps que vous spécifiez dans l'escalade doit ’tre comprise dans une plus grande "fen’tre d'heures de notification".

Restrictions d'état

Si vous voulez restreindre la définition d'escalade pour qu'elle soit prise en compte uniquement quand le service est dans un état donné, vous pouves utiliser la directive *escalation_options* dans la définition de l'escalade de service. Si vous n'utilisez pas la directive *escalation_options*, l'escalade peut ’tre prise en compte quel que soit l'état du service.

Supervision des grappes de services et d'htes

Introduction

Plusieurs personnes m'ont demandé comment superviser les grappes d'htes ou de services, c'est pourquoi j'ai décidé d'écrire une petite documentation é ce propos. C'est plutt simple, donc vous devriez trouver ceci facile é comprendre.

Tout d'abord, nous devons définir ce que nous entendons par "grappe". La faéon la plus simple de comprendre est de prendre un exemple. Supposons que votre société dispose de 5 htes qui lui offrent des services DNS redondants. Si l'un d'eux a un défaut, ce n'est pas une catastrophe majeure, car les serveurs restants continueront é fournir des services de résolution de noms. Si vous ’tes concerné par la supervision de la disponibilité du service DNS de votre société, vous voudrez superviser cinq serveurs DNS. C'est ce que je considére comme une grappe de *services*. Cette grappe de services consiste en 5 services DNS distincts que vous supervisez. Bien que vous vouliez superviser chaque service individuellement, votre préoccupation premiére est l'état général de la grappe de service DNS, plutt que la disponibilité d'un service en particulier.

Si votre société a un groupe d'htes qui procure une solution é haute-disponibilité (type grappe), je les considérerai comme étant une grappe *d'htes*. Si un hte particulier ne répond plus, un autre prendra é son compte toutes les tôches du serveur en panne. A ce sujet, vous pouvez consulter le site High-Availability Linux Project [Anglais] pour des informations concernant la redondance d'htes sous Linux.

Plan d'attaque

Il y a plusieurs faéons de superviser des grappes de services ou d'htes. Je décrirai la méthode qui me semble ’tre la plus simple. Superviser des grappes de services ou d'htes implique deux choses :

- Superviser individuellement les éléments de la grappe
- Superviser la grappe en tant qu'entité collective

Superviser individuellement des éléments d'une grappe d'htes ou de services est plus simple que vous ne l'imaginez. En fait, vous le faites sans doute déjé. Pour les grappes de services, assurez-vous de superviser chacun de ses éléments. Si vous avez une grappe de cinq serveurs DNS, vérifiez que vous avez bien cinq définitions de service (probablement en utilisant le plugin *check_dns*). Pour les grappes d'htes, assurez-vous d'avoir configuré les définitions d'htes pour chaque membre de la grappe (vous aurez aussi besoin de définir au moins un service é superviser pour chacun des htes).
Important: Vous allez sans doute vouloir désactiver les notifications pour les éléments individuels de la grappe (définitions d'htes ou de services). Bien qu'aucune notification ne soit envoyée é propos des éléments individuels, vous aurez encore un affichage visuel de l'état individuel de l'hte ou du service dans le CGI d'état. Cela sera utile é l'avenir pour retrouver exactement la source de problémes é l'intérieur de la grappe.

Superviser la grappe dans son ensemble peut ’tre effectué en utilisant les résultats de la grappe d'éléments précédemment mis en cache. Bien que vous ayez la possibilité de revérifier tous les éléments de la grappe pour déterminer son état, pourquoi gaspiller de la bande passante et des ressources, alors que vous avez déjé le résultat dans le cache ? Les résultats des éléments de la grappe mis en cache peuvent ’tre trouvés dans le journal des états (en supposant que vous supervisiez chaque élément). Le plugin *check_cluster2* a été développé spécialement pour vérifier les états des htes et services mis en cache dans le journal des états.
Important : Bien que vous n'ayez pas activé les notifications pour les éléments individuels de la grappe, vous voudrez les activer pour son contrle d'ensemble.

Utilisation du plugin check_cluster2

Le plugin check_cluster2 est étudié pour vérifier l'état général d'une grappe d'htes ou de services. Il fonctionne en vérifiant, pour chaque élément de la grappe d'htes ou de services, l'information relative é chaque état et mise en cache dans le fichier journal des états.

Le plugin check_cluster2 peut ’tre trouvé dans le répertoire "contrib" des plugins Nagios officiels : http://sourceforge.net/projects/nagiosplug/ [Anglais].

Superviser les grappes de services

Mettons que vous ayez trois serveurs DNS en redondance sur votre réseau. Vous aurez tout d'abord besoin de superviser chaque serveur DNS individuellement avant de vérifier la grappe.
Je présume que vous avez donc déjé 3 services séparés (tous nommés "Service DNS") associés é vos htes (nommés "host1", "host2", "host3").

Afin de superviser ces services comme une grappe, vous avez besoin de créer un nouveau service pour la grappe.

Avant cela, vous devrez avoir défini la commande liée é la surveillance de la grappe. Vous avez donc défini la commande *check_service_cluster* [NdT : cluster = grappe] de la maniére suivante :

```
define command{
        command_name    check_service_cluster
        command_line    /usr/local/nagios/libexec/check_cluster2 --service -l $
        }
```

Vous avez maintenant besoin de créer le service "grappe" et utiliser la commande *check_service_cluster* que vous venez de créer comme commande de vérification. L'exemple ci-dessous détaille comment le faire. Il va générer un état CRITICAL si 2 services ou plus de la grappe ne sont pas en état OK, et un état WARNING, si seulement 1 service n'est pas en état OK. Si tous les services sont en état OK, le service retournera également OK.

```
define service{

        check_command   check_service_cluster!"DNS Cluster"!1!2!$SERVICESTATEID

        }
```

Il est important de noter que nous passons é la macro $ARG4$ une liste d'éléments - séparateur virgule - de macro d'état de service é la volée. Ce point est important : Nagios va remplir ces macros avec l'état du service (en numérique plutt qu'en texte) des membres de la grappe.

Superviser les grappes d'htes

La supervision des grappes d'htes ressemble beaucoup é celle des grappes de services. Evidemment, la plus grande différence est que les membres de la grappe sont des htes et non des services. Afin de superviser l'état d'une grappe d'htes, vous devez définir un service qui utilise le plugin *check_cluster2*. Le service *ne doit pas*

’tre associé é l'un des htes de la grappe, car cela causerait des problémes relatifs aux notifications de la grappe si cet hte était hors service. Une bonne idée serait d'associer le service é l'hte sur lequel Nagios tourne. Car, si l'hte sur lequel tourne Nagios tombe, alors Nagios ne fonctionne plus et vous ne pouvez plus rien superviser (sauf si vous avez prévu une supervision redondante d'htes)

Supposons que vous avez défini la commande *check_host_cluster* de la maniére suivante :

```
define command{
        command_name check_host_cluster
        command_line /usr/local/nagios/libexec/check_cluster2 --service -l $ARG
        }
```

Supposons que vous ayez trois htes (nommés "host1", "host2", "host3") dans votre grappe. Si vous voulez que Nagios vous envoie un état WARNING si au moins un hte de la grappe est dans un état différent de OK, ou un état CRITICAL si au moins deux sont dans un état différent de OK, le service é définir pour superviser la grappe d'hte devra ressembler é :

```
define service{

        check_command   check_host_cluster!"Super Host Cluster"!1!2!$HOSTSTATEI

        }
```

Il est important de noter que nous passons é la macro $ARG4$ une liste d'éléments - séparateur virgule - de macro d'état d'hte é la volée. Ce point est important : Nagios va remplir ces macros avec l'état du service (en numérique plutt qu'en texte) des membres de la grappe.

Et voilé ! Nagios contrlera périodiquement l'état de la grappe d'htes, et vous enverra des notifications, quand son état sera dégradé (en supposant que vous ayez autorisé les notifications pour le service). Notez que pour les définitions d'htes de chaque membre de la grappe, vous préférerez vraisemblablement ne pas avoir de notifications lorsque l'hte sera hors service. Souvenez-vous que vous ne vous souciez pas tant de l'état de chaque hte que de celui de la grappe. Suivant la configuration de votre réseau et ce que vous souhaitez accomplir, vous voudrez peut-’tre laisser les notifications s'effectuer pour les états "inaccessible" dans les définitions d'htes.

Dépendances d'htes et de services

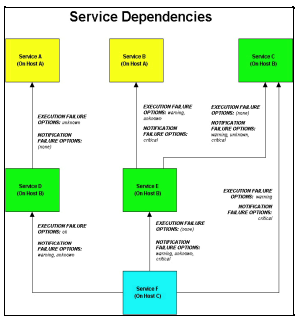
Introduction

Les dépendances d'htes et de services sont une fonctionnalité *avancée* qui vous permet de contrler le comportement des htes et des services selon l'état d'un ou plusieurs autres htes ou services. Je vais expliquer comment les dépendances fonctionnent, ainsi que les différences entre les dépendances d'htes ou de services.

Aperéu des dépendances de services

L'image ci-dessous montre un exemple de diagramme de dépendances de services. Voici quelques points é noter :

1. Un service peut ’tre dépendant d'un ou plusieurs autres services
2. Un service peut ’tre dépendant de services qui ne sont pas associés au m’me hte
3. Les dépendances de service ne se sont pas héritées (é moins que ce ne soit explicitement spécifié)
4. Les dépendances de service permettent de supprimer l'exécution de services et de notifications de service selon différents critéres (états OK, WARNING, UNKNOWN, et/ou CRITICAL)



Définition de dépendances de services

Tout d'abord, les bases. Vous créez des dépendances de service en ajoutant des définitions de dépendance de service dans votre (vos) fichier(s) de configuration des objets. Dans chaque définition, vous spécifiez le service *dépendant*, le service *dont il dépend*, et la condition (s'il y a lieu) qui provoque l'échec des dépendances d'exécution et de notification (ces notions sont décrites plus loin).

Vous pouvez créer plusieurs dépendances pour un m’me service, mais il vous faut une définition de dépendance de service séparée pour chaque dépendance créée.

Dans l'exemple ci-dessus, les définitions de dépendance du *Service F* sur l'*hte C* seraient écrites comme ceci :

```
define servicedependency{
        host_name                       Host B
        service_description             Service D
        dependent_host_name             Host C
        dependent_service_description   Service F
        execution_failure_criteria      o
        notification_failure_criteria   w,u
```

```
        }

define servicedependency{
        host_name                       Host B
        service_description             Service E
        dependent_host_name             Host C
        dependent_service_description   Service F
        execution_failure_criteria      n
        notification_failure_criteria   w,u,c
        }

define servicedependency{
        host_name                       Host B
        service_description             Service C
        dependent_host_name             Host C
        dependent_service_description   Service F
        execution_failure_criteria      w
        notification_failure_criteria   c
        }
```

Les autres définitions de dépendances décrites dans l'image précédente s'écriraient comme suit :

```
define servicedependency{
        host_name                       Host A
        service_description             Service A
        dependent_host_name             Host B
        dependent_service_description   Service D
        execution_failure_criteria      u
        notification_failure_criteria   n
        }

define servicedependency{
        host_name                       Host A
        service_description             Service B
        dependent_host_name             Host B
        dependent_service_description   Service E
        execution_failure_criteria      w,u
        notification_failure_criteria   c
        }

define servicedependency{
        host_name                       Host B
        service_description             Service C
        dependent_host_name             Host B
        dependent_service_description   Service E
        execution_failure_criteria      n
        notification_failure_criteria   w,u,c
        }
```

Comment les dépendances d'un service sont testées

Avant que Nagios n'exécute un contrle de service ou n'envoie des notifications concernant un service, il vérifiera si le service comporte des dépendances. Si ce n'est pas le cas, le contrle est exécuté ou la notification est envoyée comme en temps normal. Si le service *a bien* une ou plusieurs dépendances, Nagios vérifiera chacune de la maniére suivante :

1. Nagios récupére l'état courant* du service *dont il dépend*.
2. Nagios compare l'état courant du service *dont il dépend* aux options d'échec soit d'exécution soit de notification dans la définition de dépendance (selon ce qui adapté).
3. Si l'état courant du service *dont il dépend* correspond é une des options d'échec, la dépendance est réputée avoir échoué et Nagios sortira de la boucle de vérification des dépendances.
4. Si l'état courant du service *dont il dépend* ne correspond é aucune des options d'échec de la dépendance, la dépendance est réputée avoir réussi et Nagios continuera avec la prochaine dépendance.

Ce cycle continue jusqu'é ce que toutes les dépendances du service aient été vérifiées, ou jusqu'é ce qu'une dépendance échoue.

*Il est important de noter que par défaut, Nagios utilisera l'état hard courant du (des) service(s) dont il dépend lors de ses vérifications de dépendance. Si vous voulez que Nagios utilise l'état le plus récent des services (que ce soit un état soft ou hard), activez l'option soft_service_dependencies.

Dépendances d'exécution

Les dépendances d'exécution permettent de limiter les vérifications de service *actives*. Les vérifications de service passives ne sont pas affectées par les dépendances d'exécution.

Si *tous* les tests de dépendance d'exécution du service *réussissent*, Nagios exécute le contrle du service comme é l'accoutumée. Si ne serait-ce qu'une dépendance d'exécution du service échoue, Nagios arr'tera temporairement l'exécution des contrles pour ce service (dépendant). Par la suite, les tests des dépendances d'exécution du service vont réussir. Alors, Nagios recommencera les contrles de ce service de maniére normale. Pour plus d'informations sur l'algorithme d'ordonnancement des contrles , lisez ceci.

Dans l'exemple ci-dessus, les dépendances d'exécution du **Service E** échoueraient si le **Service B** est dans un état WARNING ou UNKNOWN. Si c'était le cas, le contrle de service ne serait pas réalisé et serait ordonnancé pour une future exécution (potentielle).

Dépendances de notification

Si *tous* les tests de dépendance de notification du service *réussissent*, Nagios enverra les notifications pour ce service comme é l'accoutumée. Si, ne serait-ce qu'une dépendance de notification du service échoue, Nagios arr'tera temporairement l'émission de notifications pour ce service (dépendant). Plus tard, les tests des dépendances de notifications du service vont réussir. Alors, Nagios recommencera é envoyer des notifications pour ce service de maniére normale. Pour plus d'informations sur l'algorithme de notification, lisez ceci.

Dans l'exemple ci-dessus, les dépendances de notification du **Service F** échoueraient si le **Service C** est dans un état CRITICAL, *et/ou* si le **Service D** est dans un état WARNING ou UNKNOWN, *et/ou* si le **Service E** est dans un état WARNING, UNKNOWN, ou CRITICAL. Si c'était le cas, les notifications pour ce service ne

seraient pas envoyées.

Héritage de dépendance

Comme je l'ai déjé dit, par défaut les dépendances de service *ne sont pas* héritées. Dans l'exemple ci-dessus, vous pouvez voir que le Service F est dépendant du Service E. Toutefois, il n'hérite pas automatiquement des dépendances du Service E sur le Service B et le Service C. Pour rendre le Service F dépendant du Service C, nous avons d ajouter une autre définition de dépendance. Il n'y a pas de définition de dépendance pour le Service B, donc le Service F *n'est pas* dépendant du Service B.

Si vous *voulez* rendre les dépendances de service héritables, utilisez le paramétre inherits_parent dans la définition de la dépendance du service. Quand ce paramétre est activé, il indique que la dépendance hérite des dépendances *du service dont elle dépend* (également appelé le service matre). En d'autres termes, si le service matre dépend d'autres services et qu'une de ces dépendances est en échec, la dépendance sera aussi en échec.

Dans l'exemple ci-dessus, imaginez que vous vouliez ajouter une nouvelle dépendance au service F qui le rende dépendant du service A. Vous pourriez créer une nouvelle définition de dépendance qui indique le service F comme le service *dépendant* et le service A comme le service *matre* (c'est-é-dire le service *dont il est dépendant*). Vous pourriez également modifier la définition de dépendance des services D et F de la maniére suivante :

```
define servicedependency{
        host_name                       Host B
        service_description             Service D
        dependent_host_name             Host C
        dependent_service_description   Service F
        execution_failure_criteria      o
        notification_failure_criteria   n
        inherits_parent         1
        }
```

Comme le paramétre *inherits_parent* est activé, la dépendance entre les services A et D sera testée quand la dépendance entre les services F et D le sera.

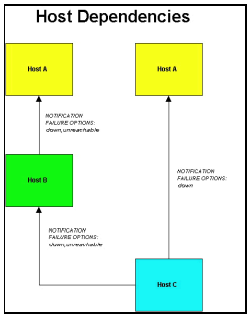
Les dépendances peuvent avoir de multiples niveaux d'héritage. Si la définition de dépendance entre A et D avait le paramétre *inherits_parent* activé, et que le service A était dépendant d'un autre service (appelons-le service G), le service F serait dépendant des services D, A, et G (et le serait potentiellement avec chacun selon des critéres différents).

Dépendances d'htes

Comme vous vous y attendez probablement, les dépendances d'htes fonctionnent d'une maniére similaire é celles de services. La principale différence est que ce sont des htes et pas des services. Une autre différence est que la dépendance d'hte ne sert qu'é supprimer des notifications d'htes et non pas des controles d'htes.

ATTENTION ! Ne confondez pas les dépendance d'htes avec les relations parent/enfant. Vous utiliserez les relations parent/enfant (définies avec le paramétre *parents* dans la définition d'hte) dans la plupart des cas, plutt que des dépendances d'htes.

L'image ci-dessous est un exemple de diagramme logique de dépendances d'htes:



Dans l'image ci-dessus, les définitions de dépendances pour l'hte C devraient ’tre définies ainsi:

```
define hostdependency{
        host_name                       Host A
        dependent_host_name             Host C
        notification_failure_criteria   d
        }

define hostdependency{
        host_name                       Host B
        dependent_host_name             Host C
        notification_failure_criteria   d,u
        }
```

Comme pour les dépendances de services, les dépendances d'htes ne sont pas héritées. Dans l'exemple de cette image, vous pouvez voir que l'hte C n'hérite pas des dépendances de l'hte B. Pour que C soit dépendant de A, une autre définition d'hte doit ’tre précisée.

Les dépendances de notifications d'htes marchent d'une maniére similaire é celles de services. Si *toutes* les notifications de dépendance d'un hte *réussissent*, Nagios enverra les notifications comme é l'accoutumée. Si ne serait-ce qu'une de ces dépendances échoue, Nagios supprimera temporairement toutes les notifications pour cet hte (dépendant). Par la suite, les dépendances réussiront é nouveau. Nagios recommencera alors é envoyer les notifications de maniére habituelle. Vous trouverez ici plus d'informations sur la l'algorithme de notification.

NOTE : Les dépendances d'exécution d'htes fonctionnent de la m’me maniére que les dépendances d'éxécution de services. Toutefois, elle ne sont prises en compte que lors des *contrles d'hte ordonnancés réguliérement*. Les contrles d'htes é la demande ne sont pas concernés par les dépendances d'htes.

Suivi précis des changements d'état

Introduction

Ce type de fonctionnalité ne sera probablement pas utilisée par beaucoup d'entre vous. Quand elle est activée, elle permet d'enregistrer des changements dans le contrle d'un service ou d'un hte, m’me si l'état de celui-ci ne change pas. Nagios va alors surveiller plus particuliérement ce service ou cet hte et enregistrer tout changement. Comme vous allez le constater, ceci peut ’tre trés utile plus tard, lors d'une analyse de vos fichiers de logs.

Comment cela marche t il ?

Dans des conditions de fonctionnement normales, le résultat de la surveillance d'un hte ou d'un service n'est enregistré que lorsqu'il a changé d'état depuis le dernier contrle. Il y a quelques exceptions é cette régle, mais c'est comme cela que cela se passe la plupart du temps.

Si vous activez ce type de contrle pour un ou plusieurs états d'un hte ou d'un service en particulier, Nagios enregistrera dans ses journaux toute différence entre le contrle actuel et le précédent. Examinez l'exemple suivant, sur 8 tests consécutifs d'un service :

Contrle du Service # Etat du Service :		Message issu é la fin du Contrle:
:		
x	OK	RAID array optimal
x+1	OK	RAID array optimal
x+2	WARNING	RAID array degraded (1 drive bad, 1 hot spare rebuilding)
x+3	CRITICAL	RAID array degraded (2 drives bad, 1 host spare online, 1 hot spare rebuilding)
x+4	CRIICAL	RAID array degraded (3 drives bad, 2 hot spares online)
x+5	CRITICAL	RAID array failed
x+6	CRITICAL	RAID array failed
x+7	CRITICAL	RAID array failed

Cette séquence étant donnée, vous devriez seulement voir deux entrées dans vos journaux, concernant cette catastrophe. La premiére arrivera é X+2 quand le service basculera de l'état OK é l'état WARNING. Le deuxiéme arrivera (trop tard), au moment du passage de WARNING é CRITICAL.

Vous pouriez avoir envie, pour une raison quelconque, d'avoir un historique complet de cet accident dans vos journaux. Peut ’tre pour expliquer é votre patron comment tout cela est arrivé soudainement, ou aller vous en jeter un ou deux au bar du coin, pour en rire plutt qu'en pleurer, ou encore .

Ceci dit, si le suivi précis avait été activé pour les états CRITICAL, les états x+4 et x+5 logged auraient été enregistrés en plus de x+2 et x+3. Pourquoi ? parce que dans ce cas-lé, Nagios aurait examiné les message émis pour vérifier s'ils différaient des précédents. Si le message émis change alors que l'état ne change pas, le message sera quand m’me enregistré.

Un exemple similaire peut ’tre donné avec un service qui contrle un serveur web. Si le plugin check_http retourne d'abord un WARNING sur une erreur 404, puis ensuite des WARNING é cause d'un modéle manquant sur la page, vous pouvez avoir envie de le savoir. Si vous n'avez pas activé le suivi précis, seul le

premier WARNING (celui de l'erreur 404) sera enregistré dans les logs et vous n'aurez aucune idée (en analysant les logs archivés) que les WARNING suivants ne sont pas dus é une erreur 404, mais plutt é un modéle absent de la page web retournée.

Dois je activer le suivi précis?

Tout d'abord, vous devez décider si vous avez réellement besoin d'examiner vos logs pour trouver la cause d'un probléme. Vous pouvez décider de l'activer pour quelques services ou htes, mais pas pour tous. Vous pouvez aussi décider que vous ne surveillerez que quelques états d'htes ou de services, mais pas tous. Par exemple, surveiller les états WARNING et CRITICAL d'un service, et pas les états OK ou UNKNOWN.

Cette décision dépend principalement du plugin que vous allez utiliser. Si le plugin retourne toujours le m’me texte/message pour un état particulier, il n'y a aucune raison de se fatiguer é activer ce type de contrle.

Comment l'activer ?

Vous pouvez activé le suivi précis des services et des htes en utilisant la directive *stalking_options* dans les définitions d'htes et de services.

Inconvénients

Vous devez ’tre conscients du fait qu'activer ce type de contrle améne quelques inconvénients. Ils sont relatifs aux fonctions d'enregistrement trouvées dans les différents CGIs (histogramme, résumé des alertes, etc.). Comme le suivi précis va apporter des entrées supplémentaires dans les journaux, les données retournées montreront un accroissement sensible du nombre d'alertes.

D'une maniére générale, je déconseille d'activer ce service *sans* avoir mené auparavant une réflexion profonde sur le sujet. Mais, bien entendu, c'est lé pour servir si vous en avez besoin.

Données de performance

Introduction

Nagios est conéu pour permettre aux plugins de renvoyer des données de performance optionnelles en plus des données d'état habituelles, ainsi que pour permettre de transférer ces données de performance é des applications tierces qui les traiteront. Vous trouverez ci-dessous une description des différents types de données de performance, ainsi que des informations sur la faéon de les traiter

Types de données de performance

Nagios peut fournir de base deux catégories de données de performance :

1. **Données de performance du contrle**
2. **Données de performance du plugin**

Les données de performance du contrle sont des données internes liées é l'exécution proprement dite d'un contrle d'hte ou de service. Cela inclut entre autres la latence de contrle de service (c.-é-d. le "retard" par rapport au moment o le contrle du service était ordonnancé) et le nombre de secondes nécessaires é l'exécution du contrle d'hte ou de service. Ce type de données de performance est disponible pour tous les contrles réalisés. Les macros $HOSTEXECUTIONTIME$ et $SERVICEEXECUTIONTIME$ contiennent le nombre de secondes nécessaires é l'exécution d'un contrle d'hte ou de service, et les macros $HOSTLATENCY$ et $SERVICELATENCY$ peuvent servir é déterminer le "retard" qu'a pris le lancement d'un contrle régulier d'hte ou de service par rapport au moment o il était ordonnancé.

Les données de performance du plugin sont des données externes propres au plugin utilisé pour réaliser le contrle d'hte ou de service. Elles peuvent par exemple comporter le pourcentage de perte de paquets, l'espace disque disponible, la charge du processeur, le nombre d'utilisateurs courant, etc. - é la base tout type de mesure que le plugin réalise lors de son exécution. Les données de performance du plugin sont optionnelles et peuvent ne pas ’tre fournies par tous les plugins. Au moment de la rédaction de ce document, aucun plugin ne retourne de donénes de performance [NdT: ce n'est plus vrai aujourd'hui], mais ils devraient prochainement le faire. Les données de performance du plugin (lorsqu'elle sont disponibles) sont accessibles é travers les macros $HOSTPERFDATA$ et $SERVICEPERFDATA$. Vous trouverez ci-dessous plus d'informations sur la faéon dont les plugins peuvent renvoyer é Nagios des données de performance qui soient exploitables via les macros $HOSTPERFDATA$ et $SERVICEPERFDATA$.

Génération des données de performance par les plugins

En temps normal les plugins renvoient un seule ligne de texte indiquant l'état d'une quelconque donnée mesurable. Par exemple, le plugin check_ping peut renvoyer une ligne de texte comme suit :

```
PING ok - Packet loss = 0%, RTA = 0.80 ms
```

Avec ce type de valeur, toute la ligne de texte est accessible via les macros $HOSTOUTPUT$ ou $SERVICEOUTPUT$ (selon que le plugin a été utilisé pour un contrle d'hte ou de service).

Pour simplifier le transfert de données de performance du plugin é Nagios, la spécification des plugins a été augmentée. Si un plugin souhaite renvoyer des données de performance é Nagios, il affiche la ligne de texte

habituelle, suivie du caractére "pipe" (|), puis une chane de caractéres contenant une ou plusieurs mesures de performance. Prenons par exemple le plugin check_ping et supposons qu'il ait été modifié pour renvoyer le pourcentage de paquets perdus et le temps moyen de retour [NdT : average round trip time] comme données de performance. La chane retournée par le plugin pourrait ressembler é ceci :

```
PING ok - Packet loss = 0%, RTA = 0.80 ms | percent_packet_loss=0, rta=0.80
```

Quand Nagios rencontre ce format de données affichées par un plugin, il éclate le texte en deux parties : tout ce qui précéde le caractére "pipe" est considéré comme étant l'affichage "normal" du plugin et tout ce qui suit le caractére "pipe" est considéré comme des données de performance du plugin. L'affichage "normal" est stocké dans les macros $HOSTOUTPUT$ ou $SERVICEOUTPUT$, et les données de performance optionnelles sont stockées dans les macros $HOSTPERFDATA$ ou $SERVICEPERFDATA$. Dans notre exemple, les macros $HOSTOUTPUT$ ou $SERVICEOUTPUT$ contiendraient "*PING ok - Packet loss = 0%, RTA = 0.80 ms*" (sans les guillemets) et les macros $HOSTPERFDATA$ ou $SERVICEPERFDATA$ contiendraient "*percent_packet_loss=0, rta=0.80*" (sans les guillemets).

Format de l'affichage des données de performance

Le démon Nagios ne traite pas directement les données de performance, si bien qu'il ne préoccupe pas du formatage des données de performance. Il n'y pas de limitations intrinséques concernant le format ou le contenu des données de performance. Cependant, si vous utilisez un addon externe pour traiter les données de performance (p.ex. PerfParse), l'addon peut attendre un format particulier de la part du plugin. Vérifiez la documentation accompagnant l'addon pour plus d'information. De m'me, assurez-vous de lire le "Guide du développeur de plugin" sur SourceForge (http://nagiosplug.sourceforge.net/) si vous souhaitez écrire des plugins.

Activation du traitement des données de performance

Pour traiter les données de performance mises é disposition par Nagios et les plugins, vous devez :

1. Activer le paramétre process_performance_data.
2. Configurer Nagios pour que les données de performance soient écrites dans des fichiers et/ou traitées en exécutant des commandes.

Ecriture des données de performance dans des fichiers

Vous pouvez configurer Nagios pour qu'il écrive toutes les données de performance d'hte ou de service dans des fichiers grôce aux paramétres host_perfdata_file et service_perfdata_file. Vous pouvez contrler comment les données sont écrites dans ces fichiers avec les paramétres host_perfdata_file_template et service_perfdata_file_template. De plus, vous pouvez faire en sorte que Nagios exécute réguliérement des commandes pour traiter les données de performance présentes dans les fichiers grôce aux paramétres host_perfdata_file_processing_command et service_perfdata_file_processing_command.

Traitement des données de performance par des commandes

Vous pouvez configurer Nagios pour qu'il traite les données de performance d'hte et de service en exécutant des commandes grôce aux paramétres host_perfdata_command ou service_perfdata_command. Voici un exemple de commande qui écrit simplement les données de performance de service dans un fichier :

```
define command{
        command_name    process-service-perfdata
        command_line    /bin/echo -e "$LASTSERVICECHECK$\t$HOSTNAME$\t$SERVICED
        }
```

Arr’t planifié

Introduction

Nagios vous permet de planifier des périodes d'arr’t pour les htes et les services que vous supervisez. Ceci est utile au cas o vous avez prévu d'arr’ter un serveur pour maintenance, etc. Quand un hte ou un service est dans une période d'arr’t planifié, les notifications pour cet hte ou ce service ne sont pas envoyées.

Fichier des arr’ts planifiés

Les arr’ts planifiés des htes et services sont stockés dans le fichier des arr’ts planifiés défini par une directive dans le fichier de configuration principal.

Mémorisation des arr’ts planifiés

Les arr’ts planifiés sont mémorisés, m’me si Nagios est relancé. Au démarrage, Nagios va lire le fichier des arr’ts planifiés, supprimer toute entrée ancienne ou invalide, et planifier les arr’ts pour les htes et services concernés.

Planifier l'arr’t

Vous pouvez planifier l'arr’t des htes et des services é travers le CGI d'informations complémentaires (en visualisant les informations soit d'un hte soit d'un service). Cliquez le lien "Schedule downtime for this host/service" pour planifier l'arr’t.

Une fois l'arr’t d'un hte ou d'un service planifié, Nagios ajoutera un commentaire é cet hte/service indiquant que son arr’t est prévu durant la période que vous avez indiqué. Quand la période est passée, Nagios supprime automatiquement le commentaire ajouté. Pas mal, non ?

Arr’ts planifiés fixes ou variables

Il y a deux types d'arr’ts planifiés: les "fixes" et les "variables". Quand vous positionnez un arr’t é travers l'interface web, on vous demande s'il est fixe [fixed] ou variable [flexible]. Voici leur définition:

"Fixe" démarre et s'arr’te é l'heure exacte programmée. Bon, celle-lé était facile.

"Variable" est adapté é des arr’ts dont vous savez qu'ils vont durer X minutes (ou heures), mais ne savez pas exactement quand cela va redémarrer. Quand vous choisissez "variable", Nagios va démarrer automatiquement au bon moment, quelque part entre l'heure de départ et l'heure de fin que vous avez spécifiées. L'arr’t durera aussi longtemps que la durée spécifiée. Ceci suppose que l'hte ou service dont vous avez programmé l'arr’t s'arr’te (ou devienne inaccessible) ou entre dans un état non-OK, quelque part entre votre heure de départ et d'arr’t.

L'heure é laquelle l'hte ou le service aura un probléme sera pour Nagios, l'heure de démarrage de l'arr’t planifié. Il durera ensuite aussi longtemps que vous l'avez spécifié, m’me si l'hte ou le service revient dans un état normal avant la fin de l'arr’t planifié.

Ceci pour bonne raison : comme nous le savons tous, vous pouvez penser qu'un probléme est résolu complétement (et redémarrer le serveur), au moins 10 fois avant qu'il ne se décide é vraiment marcher. Bien prévu n'est-ce pas ? :-)

Arr’ts déclenchés sur événement

Lorsque vous planifiez l'arr’t d'un hte ou d'un service, vous avez la possibilité de déclencher l'arr’t sur événement. L'arr’t planifié est alors déclenché lorsqu'un autre service ou hte démarre son arr’t planifié. Cette fonctionnalité est extr’mement pratique lorsque l'arr’t planifié d'un grand nombre de services ou d'htes dépend d'un hte ou d'un services particulier.
Par exemple, si vous planifiez un arr’t variable sur un hte, vous voudrez certainement déclencher l'arr’t planifié de tous ses "fils".

Comment l'arr’t planifié affecte les notifications

Quand un hte ou un service arrive dans la période d'arr’t planifié, Nagios n'autorisera pas l'émission des notifications pour cet hte ou ce service. La suppression des notifications se fait par l'ajout d'un filtre é la logique de notification. Vous *ne verrez pas* d'icne dans les CGI indiquant la désactivation des notifications pour cet hte/service. Quand la période d'arr’t planifié sera passée, Nagios autorisera l'émission des notifications pour cet hte ou service comme il le ferait normalement.

Chevauchement d'arr’ts planifiés

J'aime appeler ceci la loi de Murphy [**NdT** : "Si quelque chose ne doit pas fonctionner, il ne fonctionnera pas"]. Vous voyez de quoi je veux parler !! Vous arr’tez un serveur pour réaliser une mise é jour matérielle "de routine", pour réaliser plus tard que les drivers du S.E. ne fonctionnent pas, que le contrleur RAID a fondu, ou que l'image disque a raté et rend vos disques originaux inutilisables. La morale de l'histoire est qu'une intervention de routine sur un serveur est susceptible de prendre trois ou quatre fois plus longtemps que ce que vous aviez prévu

Prenons le scenario suivant :

1. Vous planifiez un arr’t de l'hte A de 19h30 é 21h30 le lundi
2. Vous arr’tez le serveur vers 19h45 lundi soir pour une mise é jour des disques durs
3. Aprés avoir perdu une heure et demie é vous battre avec des erreurs SCSI et des incompatibilités de drivers, vous arrivez finalement é booter la machine
4. A 21h15 vous vous apercevez qu'une de vos partitions a été cachée ou semble n'exister nulle part sur le disque
5. Sachant que la nuit va ’tre longue, vous retournez planifier un arr’t supplémentaire pour l'hte A de 21h20 lundi é 1h30 mardi matin.

Si vous planifiez des périodes d'arr’t qui se chevauchent pour un hte ou un service (dans ce ca, les périodes étaient 19h30-21h30 et 21h20-1h30), Nagios attendra que la derniére période d'arr’t planifié soit écoulée avant d'autoriser l'émission des notifications pour cet hte ou ce service. Dans cet exemple, les notifications seraient supprimées pour l'hte A jusqu'é 1h30 mardi matin.

Utilisation de l'interpréteur Perl intégré

Introduction

Stephen Davies a créé du code permettant de compiler Nagios avec un interpréteur Perl intégré. Ceci peut ’tre intéressant si vous utilisez massivement des plugins écrits en Perl.

Stanley Hopcroft a beaucoup travaillé sur l'interpréteur Perl intégré et a commenté les avantages et désavantages de son utilisation. Il a également donné plusieurs indices utiles pour écrire des plugins Perl qui fonctionnent correctement avec l'interpréteur intégré. La majeure partie de cette documentation provient de ses commentaires.

Notez que "ePN", tel qu'il est utilisé dans cette documentation, fait référence é Perl intégré é Nagios [embedded Perl Nagios], ou si vous préférez, Nagios compilé avec un interpréteur Perl intégré.

Avantages

Parmi les avantages de ePN [embedded Perl Nagios] on compte :

- Nagios passera beaucoup moins de temps é exécuter vos plugins Perl car il n'a plus besoin de créer un sous-processus [fork] pour lancer le plugin (en chargeant é chaque fois l'interpréteur Perl). Il exécute maintenant votre plugin grôce é un appel de bibliothéque.
- Il réduit grandement l'impact sur le systéme des plugins Perl et/ou vous permet de lancer plus de contrles en plugin Perl que ce dont vous seriez capables autrement. En d'autres termes, vous ’tes moins tenté d'écrire vos plugins en d'autres langages comme C/C++, ou Expect/TCL, qui sont généralement considérés comme ayant un cycle de développement plus long que Perl (m’me s'ils tournent a peu prés dix fois plus vite TCL étant une exception).
- Si vous n''tes pas un développeur C, vous pouvez quand m’me faire beaucoup de choses avec Nagios en laissant Perl faire le gros du travail, sans que Nagios ne soit trop ralenti. Ceci dit, notez que ePN n'accélérera pas votre plugin (une fois té le temps de chargement de l'interpréteur). Si vous voulez des plugins plus rapides, alors tournez-vous vers les XSUB Perl (XS), ou C *aprés* vous ’tre assuré que votre Perl est propre et que votre algorithme est correct (l'apport de Benchmark.pm *n'a pas de prix* pour comparer les performances des éléments de langage Perl).
- L'utilisation de ePN est un excellent moyen d'en apprendre plus sur Perl.

Désavantages

Les désavantages de ePN [embedded Perl Nagios] ressemblent beaucoup é ceux du mod_perl d'Apache (i.e. Apache avec un interpréteur Perl intégré) par rapport é l'Apache standard :

- Un programme Perl qui fonctionne *parfaitement* avec Nagios standard peut *ne pas* fonctionner avec ePN. Il vous faudra peut-’tre modifier vos plugins pour qu'ils tournent.
- Les plugins Perl sont plus difficiles é déboguer sous ePN qu'avec un Nagios standard.
- Votre ePN aura une plus grande taille (encombrement en mémoire) qu'un Nagios standard.
- Certaines constructions [constructs] Perl ne peuvent pas ’tre utilisées, ou peuvent se comporter différemment de ce é quoi vous vous attendiez.
- Il vous faudra connatre "plus d'une faéon de le faire", et peut-’tre choisir celle qui semble la moins attirante ou évidente.

- Il vous faudra une meilleure connaissance de Perl (mais rien de bien ésotérique ou concernant la structure interne de Perl sauf si vous utilisez les XSUBS).

Public visé

- Les développeurs Perl moyens ; ceux qui connaissent la puissance des fonctionnalités du langage sans en connatre les rouages internes.
- Ceux qui ont un point de vue utilitaire plutt qu'une profonde compréhension [sic].
- Si vous aimez les objets Perl, la gestion de noms, les structures de données, et le débogueur, cela suffit sans doute.

Ce que vous devez faire quand vous écrivez un plugin Perl (ePN ou pas)

- Générez **toujours** un affichage
- Utilisez "use utils" et importez ce qui en est exporté ($TIMEOUT %ERRORS &print_revision &support)
- Jetez un il sur la faéon dont sont écrits les plugins Perl standard, comme par exemple :

    - Quittez toujours avec une valeur $ERRORS{CRITICAL}, $ERRORS{OK}, etc.
    - Utilisez getopt pour lire les paramétres de la ligne de commande
    - Gérez les dépassements de délai
    - Appelez print_usage (que vous fournissez) quand il n'y a pas de paramétres é la commande
    - Utilisez des noms de paramétres standard (par exemple H 'host', V 'version')

Ce que vous devez faire quand vous écrivez un plugin Perl pour ePN

1. <DATA> ne peut pas ’tre utilisé ; utilisez ici des documents é la place, par exemple :

```
my $data = <<DATA;

portmapper 100000

portmap 100000

sunrpc 100000

rpcbind 100000

rstatd 100001

rstat 100001

rup 100001

..
```

```
DATA


%prognum = map { my($a, $b) = split; ($a, $b) } split(/\n/, $data) ;
```

2. Les blocks BEGIN ne fonctionneront pas comme vous l'attendez. Il vaut mieux les éviter.
3. Assurez-vous de la parfaite propreté du code é la compilation, i.e.

    - utilisez use strict
    - utilisez perl -w (les autres paramétres (notamment T) ne sont d'aucune aide)
    - utilisez perl -c
4. Evitez les variables de portée lexicale (my) déclarées globalement comme moyen de passer des **variables** aux fonctions. En fait ceci est **fatal** si la fonction est appelée par le plugin plus d'une fois lorsque le contrle est exécuté. Ces fonctions se comportent comme des encapsulations [closures] qui verrouillent les variables lexicales globales sur leur premiére valeur lors des appels suivants é la fonction. Si toutefois votre variable globale est en lecture seule (une structure de données complexe par exemple), ce n'est pas un probléme. Ce que Bekman recommande en remplacement est une des solutions suivantes :

    - faites une fonction anonyme et appelez-la é travers une référence au code, par exemple :

```
remplacez ceci                par


my $x = 1 ;                   my $x = 1 ;
sub a { .. Process $x  }   $a_cr = sub {  Process $x  } ;
.                             .
.                             .
a ;                           &$a_cr ;
$x = 2                        $x = 2 ;
a ;                           &$a_cr ;


# les encapsulations anonymes reprennent __toujours__ la valeur lexi
```

    - mettez la variable globale et la fonction qui l'utilise dans leur propre paquetage [package] (comme objet ou module)
    - passez les variables aux fonctions comme références ou alias (\$lex_var ou $_[n])
    - remplacez les variables lexicales par des variables globales au paquetage et excluez les des objections faites par "use strict" en déclarant "use vars qw(global1 global2 ..)"
5. Sachez o trouvez plus d'informations.

Vous pouvez obtenir des informations utiles des indices habituels (les livres O'Reilly, plus "Object Oriented Perl" de Damien Conways) mais pour les bonnes réponses dans ce contexte commencez par le guide du mod_perl de Stas Bekman sur http://perl.apache.org/guide/.

Ce document merveilleux au format livre n'a strictement rien é voir avec Nagios, mais tout é voir avec l'écriture de programmes pour l'interpréteur Perl intégré é Apache (i.e. le mod_perl de Doug MacEachern).

La page "man" perlembed est essentielle pour le contexte et les encouragements.

Si l'on considére que Lincoln Stein et Doug MacEachern savent deux-trois choses sur Perl et l'intégration de Perl, leur livre "Writing Apache Modules with Perl and C" vaut certainement d'’tre lu.

6. Sachez que votre plugin peut retourner d'étranges valeurs avec ePN, et que cela est probablement d au point 4 ci-dessus.
7. Soyez pr’t é déboguer en :

    - ayant un ePN de test
    - ajoutant des instructions print é votre plugin pour afficher la valeur des variables sur STDERR (STDOUT ne peut pas ’tre utilisé)
    - ajoutant des instructions print é p1.pl pour afficher ce qu'ePN pense qu'est votre plugin avant d'essayer de le lancer (vi)
    - lanéant l'ePN en avant-plan (probablement en conjonction avec les recommandations précédentes)
    - utilisant le module "Deparse" sur votre plugin pour voir comment l'analyseur syntaxique l'a optimisé, et ce que l'interpréteur reéoit réellement (voir "Constants in Perl" de Sean M. Burke, The Perl Journal, automne 2001)

    perl -MO::Deparse <votre_programme>
8. Sachez qu'ePN transforme votre plugin lui aussi, et si tout le reste a échoué essayez de déboguer la version transformée.

    Comme vous pouvez le constater ci-dessous p1.pl réécrit votre plugin comme une fonction appelé 'hndlr' dans le paquetage nommé "Embed::<quelque-chose-ayant-rapport-avec-le-nom-de-fichier-de-votre-plugin>".

    Votre plugin attend peut-’tre des paramétres de la ligne de commande dans @ARGV, donc pl.pl assigne également @_ é @ARGV.

    Ceci é son tour est évalué et si "eval" remonte une erreur (qu'elle soit syntaxique ou d'exécution), le plugin est jeté dehors.

    La copie d'écran suivante montre comment un ePN de test a transformé le plugin *check_rpc* avant d'essayer de l'exécuter. Seule une petite partie du code du plugin est montrée ici, car nous ne nous intéressons qu'aux transformations que l'ePN lui fait subir). Les transformations sont affichées en rouge :

```
package main;

use subs 'CORE::GLOBAL::exit';
```

```
                sub CORE::GLOBAL::exit { die "ExitTrap: $_[0] (Embed::chec
                package Embed::check_5frpc; sub hndlr { shift(@_);
                @ARGV=@_;
```

```
#! /usr/bin/perl -w
#
# check_rpc plugin for nagios
#
# usage:
#    check_rpc host service
#
# Check if an rpc serice is registered and running
# using rpcinfo - $proto $host $prognum 2>&1 |";
#
# Use these hosts.cfg entries as examples
#
# command[check_nfs]=/some/path/libexec/check_rpc $HOSTADDRESS$ nfs
# service[check_nfs]=NFS;24x7;3;5;5;unix-admin;60;24x7;1;1;1;;check_rpc
#
# initial version: 3 May 2000 by Truongchinh Nguyen and Karl DeBisschop
# current status: $Revision: 1.8.2.4
#
# Copyright Notice: GPL
#
 le reste du plugin ci-dessous
```

9. Ne pas utiliser "use diagnostics" dans un plugin lancé par votre ePN de production. Je pense qu'il force la valeur de retour é CRITICAL dans tous les plugins Perl.

10. Envisagez l'utilisation d'un mini Perl intégré pour vérifier votre plugin. Cela ne suffit pas é valider votre plugin avec l'ePN, mais si le plugin échoue é ce test il échouera également avec l'ePN. Un exemple de mini ePN est inclus dans le répertoire *contrib/* de la distribution de Nagios é cette fin. Placez-vous dans le répertoire contrib/ et tapez "make mini_epn" pour le compiler. Il doit ’tre exécuté depuis le répertoire o se trouve p1.pl (ce fichier est distribué avec Nagios).

Compilation de Nagios avec l'interpréteur Perl intégré

Bien, vous pouvez respirer maintenant. Alors, vous voulez *toujours* compiler Nagios avec l'interpréteur Perl intégré ? ;-)

Si vous voulez compiler Nagios avec l'interpréteur Perl intégré il vous faut relancer le script de configuration (configure) avec le paramétre **--enable-embedded-perl**. Si vous voulez que l'interpréteur Perl intégré utilise un cache interne pour les scripts compilés, ajoutez également le paramétre **--with-perlcache**. Par exemple :

```
./configure --enable-embedded-perl --with-perlcache autres paramétres
```

Surveillance adaptative

Introduction

Nagios permet de changer certaines commandes, attributs de vérification sur les htes et les services durant l'exécution. Je me référerai é ce dispositif comme ô la surveillance adaptative é. Ces dispositifs de surveillance adaptatifs de Nagios ne seront probablement pas trés utiles pour 99% des utilisateurs, mais ils permettent de faire des choses intéressantes.

Qu'est-ce qui peut ’tre changé ?

Les attributs suivants pour les services peuvent ’tre modifiés en cours d'exécution :

- La commande de vérification [check command] et ses arguments.
- La commande du gestionnaire d'événements [event handler] et ses arguments.
- L'intervalle de vérification [check interval]
- L'intervalle de vérification sur probléme [retry check interval]
- Le nombre maximum de vérification [max check attempts]

Les attributs suivants pour les htes peuvent ’tre modifiés en cours d'exécution :

- La commande de vérification [check command] et ses arguments.
- La commande du gestionnaire d'événements [event handler] et ses arguments.
- L'intervalle de vérification [check interval]
- Le nombre maximum de vérification [max check attempts]

Les attributs globaux suivants peuvent ’tre modifiés en cours d'exécution :

- Le gestionnaire global des événements relatifs aux htes [global_host_event_handler] et ses arguments.
- Le gestionnaire global des événements relatifs aux services [global_service_event_handler] et ses arguments.

Commandes externes pour la surveillance adaptative

Afin de changer les attributs globaux ou spécifiques é un hte ou un service durant l'exécution, vous devez soumettre la commande externe appropriée é Nagios par l'intermédiaire du fichier de commande externe. La table ci dessous liste les différents attributs qui peuvent ’tre changés durant l'exécution, avec la commande externe permettant ce changement.

NOTE : Lorsque les commandes de vérification [check command] ou du gestionnaire d'événements [event handler] sont modifiés, il est important de noter que ces commandes doivent avoir été configurées en utilisant des définitions de commande avant que Nagios n'ai été démarré. Si une commande non configurée est entrée, elle sera ignorée. Les arguments sont fournis avec la commande en utilisant le séparateur "!" entre la commande et les arguments, et entre les arguments.
Les informations sur le traitement des arguments durant l'exécution sont disponibles sur la page macros.

é

Attribut	Commande externe	Notes
Commande de vérification du service [Service check command]	CHANGE_SVC_CHECK_COMMAND:*nom_commande*	Modifie la commande courante de vérification du service é *nom_commande* .
Commande du gestionnaire d'événements du service [Service event handler]	CHANGE_SVC_EVENT_HANDLER:*nom_commande*	Modifie la commande courante du gestionnaire d'événements du service é *nom_commande*
Intervalle de vérification du service [Service check interval]	CHANGE_NORMAL_SVC_CHECK_INTERVAL:*intervalle*	Modifie l'intervalle courant de vérification du service é *intervalle*.
Intervalle de vérification du service sur probléme [Service check retry interval]	CHANGE_RETRY_SVC_CHECK_INTERVAL:*intervalle*	Modifie l'intervalle courant de vérification du service sur probléme é *intervalle*.
Nombre maximum de vérification du service [Max service check attempts]	CHANGE_MAX_SVC_CHECK_ATTEMPTS:*tentatives*	Modifie le nombre courant maximum de vérification du service é *tentatives*.
Commande de vérification de l'hte [Host check command]	CHANGE_HOST_CHECK_COMMAND:*nom_commande*	Modifie la commande courante de vérification de l'hte du service é *nom_commande*
Commande du gestionnaire d'événements de l'hte [Host event handler]	CHANGE_HOST_EVENT_HANDLER:*nom_commande*	Modifie la commande courante du gestionnaire d'événements de l'hte é *nom_commande*
	CHANGE_NORMAL_HOST_CHECK_INTERVAL:*intervalle*	

Intervalle de vérification de l'hte [Host check interval]		Modifie l'intervalle courant de vérification de l'hte é *intervalle*.
Nombre maximum de vérification de l'hte [Max host check attempts]	CHANGE_MAX_HOST_CHECK_ATTEMPTS:*tentatives*	Modifie le nombre courant maximum de vérification de l'hte é *tentatives*.
Commande du gestionnaire d'événements globale des htes [Global host event handler]	CHANGE_GLOBAL_HOST_EVENT_HANDLER;*nom_commande*	Modifie la valeure de global host event handler command é *nom_commande*.
Commande du gestionnaire d'événements globale des services [Global service event handler]	CHANGE_GLOBAL_SVC_EVENT_HANDLER;*nom_commande*	Modifie la valeure de global service event handler command é *nom_commande*.

Configuration des objets par héritage via les modéles

Introduction

Cette partie de la documentation tente d'expliquer la configuration des objets par héritage et comment ils sont utiliser dans la configuration des objets basé sur des modéles.

Une de mes motivations majeures en ajoutant un support pour la configuration des objets via des modéles était de fournir l'héritage de propriétés depuis d'autres définitions d'objets. L'héritage de propriétés d'objets s'accomplit de maniére récursive lors de la lecture des fichiers de configurations par Nagios.

Si vous avez encore des doutes sur les principes de récursion ou d'héritage aprés la lecture de ce document, regardez les exemples de configuration dans le paquetage. Si cela ne vous suffit pas, envoyer un email avec une description détaillée de votre probléme sur la liste de diffusion *nagios-users*[NdT : liste de diffusion en anglais, pour un support en franéais vous pouvez aller sur http://forums.opsyx.com/].

Les concepts de base

Il y a trois variables concernant le principe de récursion et d'héritage qui sont disponibles dans toutes définitions d'objets. Elles sont indiquées en rouge dans ce qui suit

```
define someobjecttype{
        object-specific variables
                        name template_name
                        name_of_template_to_use
                        register      [0/1]
        }
```

La premiére variable est *name*. C'est simplement un nom pour faire référence au "modéle" dans d'autres définitions d'objets qui, dans ce cas, héritent des propriétés/variables des objets. Le nom du modéle doit ’tre unique dans tous les objets du m’me type, vous ne pouvez pas avoir deux ou plus de définitions d'hte qui ont "hosttemplate" comme nom de modéles.

La seconde variable est *use*. C'est la variable qui permet de spécifier le nom du modéle duquel vous voulez hériter les propriétés/variables. Le nom que vous spécifiez pour cette variable doit ’tre défini dans un autre modéle d'objet (en utilisant la variable *name*).

La troisiéme variable est *register*. Cette variable est utilisée pour indiquer si oui ou non la définition de l'objet devrait ’tre "enregistrée" avec Nagios. Par défaut, toutes les définitions d'objet sont enregistrées. Si vous utilisez une définition partielle d'objet comme modéle, vous souhaiterz éviter l'enregistrer (un exemple est fourni par la suite). Les valeurs sont: 0 = NE PAS enregistrer la définition de l'objet, 1 = enregistrer la définition (la valeur par défaut). Cette variable n'est pas héritée; chaque définition d'objet (m’me partielle) utilisée comme template doit explicitement mettre la variable *register* é *0*. Ceci permet d'éviter de spécifier la directive *register* é *1* pour chaque objet qui doit ’tre enregistré [NdT : je comprends la phrase mais j'arrive pas é trouver une bonne traduction ! :(].

Variables locales versus variables héritées

Une des choses importante é comprendre avec l'héritage est que les variables locales d'un objet sont toujours prioritaires par rapport é celles définies dans le modéle. Regardez l'exemple suivant contenant deux définitions d'htes (l'ensemble des variables nécessaires au bon fonctionnement n'a pas été présenté).

```
define host{
        host_name               bighost1
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
                        name                    hosttemplate1
        }

define host{
        host_name               bighost2
        max_check_attempts      3
                        use                     hosttemplate1
        }
```

Vous remarquerez que la définition de l'hte *bighost1* a été définie avec *hosttemplate1* comme nom de modéle. La définition de l'hte *bighost2* utilise la définition de *bighost1* comme modéle d'objet. Une fois que Nagios a analysé ces données, la définition de l'hte *bighost2* devrait ’tre équivalente é la définition suivante :

```
define host{
        host_name               bighost2
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      3
        }
```

Vous pouvez voir que les variables *check_command* et *notification_options* sont héritées depuis le modéle d'objet (o l'hte *bighost1* est défini). Cependant, les variables *host_name* et *max_check_attempts* ne sont pas héritées depuis le modéle parce qu'elles ont été définies localement. Souvenez-vous, les variables définies localement surchargent les variables qui normalement devraient ’tre héritées depuis le modéle. Cela devrait ’tre un concept assez simple é comprendre.

L'héritage en chane

Les objets peuvent hériter de propriétés/variables é travers plusieurs niveaux de modéles d'objets. Regardez l'exemple suivant :

```
define host{
        host_name               bighost1
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
        name                  hosttemplate1
        }
```

```
define host{
        host_name               bighost2
        max_check_attempts      3
        use                     hosttemplate1
                        name                    hosttemplate2
        }

define host{
        host_name               bighost3
        use                     hosttemplate2
        }
```

Vous pouvez remarquer que la définition de l'hte *bighost3* hérite de variables depuis la définition d'hte *bighost2*, laquelle hérite de variables depuis la définition d'hte *bighost1*. Une fois que Nagios a analysé ces données, les défintions des htes devraient ’tre équivalentes é celle-ci :

```
define host{
        host_name               bighost1
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
        }

define host{
        host_name               bighost2
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      3
        }

define host{
        host_name               bighost3
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      3
        }
```

Il n'y a aucune limite d'héritage sur la "profondeur" de l'héritage, mais vous voudrez probablement vous limiter vous-m’me é quelques niveaux pour une gestion plus saine.

Utiliser des définitions incompléte d'hte comme modéle

Il est possible d'utiliser des définitions incompléte d'objets comme modéle. Par "définition incompléte", j'entends que toutes les variables nécessaires é un objet n'ont pas été fournies dans la définition. Capeut sembler bizarre d'utiliser une définition incompléte comme modéle, pourtant c'est en effet ce que je vous recommande d'utiliser. Pourquoi ? Et bien, cela peut ’tre vu comme un ensemble de valeurs par défaut utilisées par toutes les définitions d'objets. Prenez l'exemple qui suit :

```
define host{
        check_command           check-host-alive
```

```
        notification_options    d,u,r
        max_check_attempts      5
        name                    generichosttemplate
                        register                        0
        }

define host{
        host_name               bighost1
        address                 192.168.1.3
        use                     generichosthosttemplate
        }

define host{
        host_name               bighost2
        address                 192.168.1.4
                use                     generichosthosttemplate
        }
```

Remarquez que la premiére définition est incompléte car il manque la variable *host_name* nécessaire. Nous n'avons pas besoin de fournir le nom d'hte car nous voulons simplement utiliser cette définition comme un modéle générique. Dans le but d'éviter que cette définition soit enregistrée par Nagios comme celle d'un hte traditionnel, nous affectons 0 é la variable *register*.

Les définitions des htes *bighost1* et *bighost2* héritent des variables définies dans l'hte générique. La seule variable que nous avons choisie de surcharger est la variable *addres*s afin que les deux htes aient exactement les m’mes propiétés en dehors de leurs variables *host_name* et *address*. Une fois que Nagios a analysé ces données, les définitions des htes devraient ’tre équivalentes é ceci :

```
define host{
        host_name               bighost1
        address                 192.168.1.3
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
        }

define host{
        host_name               bighost2
        address                 192.168.1.4
        check_command           check-host-alive
        notification_options    d,u,r
        max_check_attempts      5
        }
```

Enfin, l'utilisation de modéle pour les variables par défaut fait gagner un temps précieux. Cela vous évitera aussi de vous casse la t’te si vous voulez changer des valeurs par défaut pour un grand nombre d'htes.

Trucs et astuces "gain de temps" dans les définitions de modéles d'objets

ou
"Comment ne pas p’ter les plombs"

Introduction

Cette documentation va essayer de vous expliquer comment vous pouvez exploiter les fonctionnalités (quelque peu) cachées des définitions d'objets basées sur des modéles, ne serait-ce que pour "ne pas p’ter les plombs". Comment, me demanderez vous ? Plusieurs types d'objets permettent de spécifier de multiples noms d'htes et/ou de groupes d'htes dans les définitions, vous permettant de "copier" la défintion de l'objet dans de multiples htes ou services. Je vais parcourir séparement chaque type d'objet permettant cette fonctionnalité. Pour commencer, les types d'objets concernés sont les suivants :

- Services
- Escalades de service
- Dépendances de services
- Escalades d'hte
- Dépendances d'htes
- Groupes d'htes

Les types d'objets qui ne sont pas listés ci-dessus (c.-é-d. les périodes, les commandes) ne supportent pas les fonctionnalités que je vais vous décrire maintenant.

Correspondance avec des expressions réguliéres

Les exemples que je donne ci-dessous utilisent une correspondance "standard" avec les noms des objets. Si vous le souhaitez, vous pouvez activer la correspondance des noms d'objets avec des expressions réguliéres grôce au paramétre de configuration use_regexp_matching. Par défaut, les expressions réguliéres ne sont utilisées que dans les noms d'objets contenant les caractéres de remplacement ***** et **?**. Si vous souhaitez voir les expressions réguliéres appliquées sur tous les noms d'objets (sans tenir compte du fait qu'ils contiennent ou non les caractéres de remplacement * et ?), activez le paramétre de configuration use_true_regexp_matching.

Les expressions réguliéres peuvent ’tre utilisées dans n'importe quel champ des exemples ci-dessous (noms d'htes, noms de groupe d'htes, noms de service, et noms de groupes de service).

NOTE : Attention lorsque vous activez les expressions réguliéres - vous aurez peut-’tre é modifier votre fichier de configuration, pour éviter que des paramétres que vous ne souhaitez pas voir interprétés comme des expressions réguliéres ne le soient ! Les problémes devraient ’tre mis en évidence lorsque vous vérifiez votre configuration.

Définitions de service

Htes multiples : Si vous désirez créer des services identiques, assignés é plusieurs htes, vous pouvez spécifier des htes multiples dans la directive *host_name* comme suit:

```
define service{
        host_name               HOST1,HOST2,HOST3,,HOSTN

        service_description     SOMESERVICE
        other service directives
        }
```

La définition ci-dessus va créer un service appellé *SOMESERVICE* sur les htes *HOST1* é *HOSTN*. Toutes les instances du service *SOMESERVICE* seront identiques (c.-é-d. auront la m'me commande de contrle, le m'me nombre maximum d'essais, la m'me période de notification, etc.).

Tous les htes situés dans de multiples groupes d'htes : Si vous voulez créer des services identiques, assignés é tous les htes d'un ou de plusieurs groupes d'htes, vous pouvez le faire en une seule définition de service. Comment ? Le paramétre *hostgroup_name* vous permet de spécifier un ou plusieurs groupes d'htes pour lesquels le service doit 'tre créé :

```
define service{
                HOSTGROUP1,HOSTGROUP2,,HOSTGROUPN
                SOMESERVICE
                other service directives
        }
```

La définition ci-dessus va créer un service appellé *SOMESERVICE* sur tous les htes membre des groupes d'htes *HOSTGROUP1* é *HOSTGROUPN*. Toutes les instance du service *SOMESERVICE* seront identiques (c.-é-d. auront la m'me commande de contrle, le m'me nombre maximum d'essais, la m'me période de notification, etc.).

Tous les htes : Si vous voulez créer des services identiques, assignés é tous les htes de vos fichiers de configurations, vous pouvez utiliser un caractére de remplacement dans le paramétre *host_name* comme suit:

```
define service{
                service_description     SOMESERVICE
                other service directives
        }
```

La définition ci-dessus va créer un service appellé *SOMESERVICE* sur **tous les htes** définis dans vos fichiers de configurations. Toutes les instance du service *SOMESERVICE* seront identiques (c.-é-d. auront la m'me commande de contrle, le m'me nombre maximum d'essais, la m'me période de notification, etc.).

Définitions d'escalade de service

Htes multiples : Si vous désirez créer des escalades de service pour des services portant le m'me nom/description, et qui sont assignés é plusieurs htes, vous pouvez spécifier des htes multiples dans le paramétre *host_name* comme suit:

```
define serviceescalation{
                host_name       HOST1,HOST2,HOST3,,HOSTN
                service_description SOMESERVICE
                other escalation directives
        }
```

La définition ci-dessus va créer une escalade pour les services appelés *SOMESERVICE* sur les htes *HOST1* é *HOSTN*. Toutes les instance de l'escalade de service seront identiques (c.-é-d. auront le m'me groupe de contact, le m'me intervalle de notification, etc.).

Tous les htes de multiples groupes d'htes : Si vous désirez créer une escalade de service pour des services portant le m'me nom/description, et qui sont assignés é tous les htes dans un ou plusieurs groupes d'htes, vous pouvez spécifier dans le paramétre *hostgroup_name* :

```
define serviceescalation{
        hostgroup_name  HOSTGROUP1,HOSTGROUP2,,HOSTGROUPN
        service_description SOMESERVICE
        other escalation directives
                }
```

La définition ci-dessus va créer une escalade pour les services appelés *SOMESERVICE* sur tous les htes membres des groupes *HOSTGROUP1* é *HOSTGROUPN*. Toutes les instance de l'escalade de service seront identiques (c.-é-d. auront le m'me groupe de contact, le m'me intervalle de notification, etc.).

Tous les htes : Si vous désirez créer une escalade de service identique pour des services portant le m'me nom/description, et qui sont assignés é tous les htes définis dans vos fichiers de configuration, vous pouvez utiliser un caractére de remplacement dans la directive *host_name* de la maniére suivante :

```
        define serviceescalation{
                        host_name *
                        service_description SOMESERVICE
                        other escalation directives
                }
```

La définition ci-dessus va créer une escalade pour tous les services appelés *SOMESERVICE* de **tous les htes** définis dans vos fichiers de configuration. Toutes les instance de l'escalade de service seront identiques (c.-é-d. auront le m'me groupe de contact, le m'me intervalle de notification, etc.).

Tous les services d'un m'me hte : Si vous voulez créer une escalade de service pour tous les services assignés é un hte précis, vous pouvez utiliser un caractére de remplacement dans le paramétre *service_description* de la maniére suivante :

```
define serviceescalation{
        host_name       HOST1
        service_description *
        other escalation directives
                }
```

La définition ci-dessus va créer une escalade pour **tous** les services de l'hte *HOST1*. Toutes les instance de l'escalade de service seront identiques (c.-é-d. auront le m'me groupe de contact, le m'me intervalle de notification, etc.).

Si vous vous sentez l'esprit particuliérement aventureux, vous pouvez spécifier un caractére de remplacement é la fois dans les paramétres *host_name* et *service_description*. Cela créera une escalade de service pour **tous les services** définis dans vos fichiers de configuration.

Services multiples du m'me hte : Si vous souhaitez créer une escalade de service pour plusieurs services assignés é un hte précis, vous pouvez spécfier plusieurs descriptions de service dans le paramétre *service_description* de la maniére suivante :

```
define serviceescalation{
        host_name HOST1
        service_description SERVICE1,SERVICE2,,SERVICEN
        other escalation directives             }
```

La définition ci-dessus va créer une escalade pour les services *SERVICE1* é *SERVICEN* de l'hte *HOST1*. Toutes les instance de l'escalade de service seront identiques (c.-é-d. auront le m'me groupe de contact, le m'me intervalle de notification, etc.).

Tous les Services de multiples groupes de services : Si vous voulez créer des escalades de service pour tous les services appartenant é un ou plusieurs groupes de services, vous pouvez utiliser le paramétre *servicegroup_name* comme suit :

```
define serviceescalation{
        servicegroup_name SERVICEGROUP1,SERVICEGROUP2,,SERVICEGROUPN
        other escalation directives
                }
```

La définition ci-dessus va créer une escalade pour tous les services membres des groupes de services *SERVICEGROUP1* é *SERVICEGROUPN*. Toutes les instance de l'escalade de service seront identiques (c.-é-d. auront le m'me groupe de contact, le m'me intervalle de notification, etc.).

Définitions de dépendance de service

Htes multiples : Si vous souhaitez créer des dépendances de service pour des services portant le m'me nom/description, et qui sont assignés é plusieurs htes, vous pouvez spécifier de multiples htes dans les paramétres *host_name* et/ou *dependent_host_name* comme suit :

```
define servicedependency{
        host_name HOST1,HOST2
        service_description SERVICE1
        dependent_host_name HOST3,HOST4
        dependent_service_description SERVICE2
        other dependency directives
                }
```

Dans l'exemple ci-dessus, le service *SERVICE2* des htes *HOST3* et *HOST4* sera dépendant du service *SERVICE1* des htes *HOST1* et *HOST2*. Toutes les instances de la dépendance de service seront identiques, mis é part les noms d'htes (c.-é-d. auront le m'me critére d'échec de notification, etc.).

Tous les htes de multiples groupes d'htes : Si vous voulez créer des dépendances de service pour des services portant le m'me nom/description, et qui sont assignés é tous les htes d'un ou plusieurs groupes d'htes, vous pouvez utiliser le paramétre *hostgroup_name* et/ou *dependent_hostgroup_name* comme suit :

```
define servicedependency{
        hostgroup_name HOSTGROUP1,HOSTGROUP2
```

```
        service_description SERVICE1
        dependent_hostgroup_name HOSTGROUP3,HOSTGROUP4
        dependent_service_description SERVICE2
        other dependency directives
                }
```

Dans l'exemple ci-dessus, le service *SERVICE2* de tous les htes membres des groupes d'htes *HOSTGROUP3* et *HOSTGROUP4* seront dépendant du service *SERVICE1* de tous les htes membres des groupes d'htes *HOSTGROUP1* et *HOSTGROUP2*. Si l'on suppose qu'il y a cinq htes dans chacun des groupes d'htes, cette définition équivaut é créer 100 définitions simples de dépendance de service ! Toutes les instances de la dépendance de service seront identiques, mis é part les noms d'htes (c.-é-d. auront le m'me critére d'échec de notification, etc.).

Tous les services d'un m'me hte : Si vous voulez créer des dépendances de service pour tous les services assignés é un hte précis, vous pouvez utiliser un caractére de remplacement dans les paramétres *service_description* et/ou *dependent_service_description* comme suit :

```
define servicedependency{
        host_name HOST1
        service_description *
        dependent_host_name HOST2
        dependent_service_description *
        other dependency directives
                }
```

Dans l'exemple ci-dessus, **tous les services** de l'hte *HOST2* seront dépendants de **tous les services** de l'hte *HOST1*. Toutes les instances de la dépendance de service seront identiques (c.-é-d. auront le m'me critére d'échec de notification, etc.).

Services muiltiples d'un m'me hte : Si vous voulez créer des dépendances de service pour plusieurs services assignés é un hte précis, vous pouvez spécifier plusieurs descriptions de service dans les paramétres *service_description* et/ou *dependent_service_description* comme suit :

```
define servicedependency{
        host_name HOST1
        service_description SERVICE1,SERVICE2,,SERVICEN
        dependent_host_name HOST2
        dependent_service_description SERVICE1,SERVICE2,,SERVICEN
        other dependency directives
                }
```

Tous les services de multiples groupes de service : Si vous voulez créer des dépendances de service pour tous les services appartenant é un ou plusieurs groupes de services, vous pouvez utiliser les paramétres *servicegroup_name* et/ou *dependent_servicegroup_name* comme suit :

```
define servicedependency{
        servicegroup_name SERVICEGROUP1,SERVICEGROUP2,,SERVICEGROUPN
        dependent_servicegroup_name SERVICEGROUP3,SERVICEGROUP4,SERVICEGROUPN
        other escalation directives
                }
```

Définitions d'escalade d'hte

Htes multiples : Si vous souhaitez créer des escalades d'hte pour plusieurs htes, vos pouvez spécifier plusieurs htes dans le paramétre *host_name* comme suit :

```
define hostescalation{
        host_name HOST1,HOST2,HOST3,,HOSTN
        other escalation directives
                }
```

La définition ci-dessus créera une escalade d'hte pour les htes *HOST1* é *HOSTN*. Toutes les instances de l'escalade d'hte seront identiques (c.-é-d. auront les m'mes groupes de contacts, le m'me intervalle de notification, etc.).

Tous les htes de multiples groupes d'htes : Si vous voulez créer des escalades d'hte pour tous les htes d'un ou plusieurs groupes d'htes, vos pouvez utiliser le paramétre *hostgroup_name* comme suit :

```
define hostescalation{
        hostgroup_name HOSTGROUP1,HOSTGROUP2,,HOSTGROUPN
        other escalation directives
                }
```

La définition ci-dessus créera une escalade d'hte sur tous les htes membres des groupes d'htes *HOSTGROUP1* é *HOSTGROUPN*. Toutes les instances de l'escalade d'hte seront identiques (c.-é-d. auront les m'mes groupes de contacts, le m'me intervalle de notification, etc.).

Tous les htes : Si vous voulez créer des escalades d'hte identiques pour tous les htes définis dans vos fichiers de configuration, vous pouvez utiliser un caractére de remplacement dans le paramétre *host_name* comme suit :

```
define hostescalation{
                        host_name *
                        other escalation directives
                }
```

La définition ci-dessus créera une escalade d'hte sur **tous les htes** définis dans vos fichiers de configuration. Toutes les instances de l'escalade d'hte seront identiques (c.-é-d. auront les m'mes groupes de contacts, le m'me intervalle de notification, etc.).

Définitions de dépendance d'hte

Htes multiples : Si vous voulez créer des dépendances d'hte pour plusieurs htes, vous pouvez spécifier plusieurs htes dans les paramétres *host_name* et/ou *dependent_host_name* comme suit :

```
define hostdependency{
        host_name HOST1,HOST2
        dependent_host_name HOST3,HOST4,HOST5
        other dependency directives
                }
```

La définition ci-dessus équivaut é créer six dépendances d'hte distinctes. Dans cet exemple, les htes *HOST3*, *HOST4* et *HOST5* seront dépendants é la fois d'*HOST1* et de *HOST2*. Toutes les instances de la dépendance d'hte seront identiques, é l'exception des noms d'htes (c.-é-d. auront le m'me critére d'échec de notification, etc.).

Tous les htes de multiples groupes d'htes : Si vous voulez créer des dépendances d'hte pour tous les htes d'un ou plusieurs groupes d'htes, vous pouvez utiliser les paramétres *hostgroup_name* et/ou *dependent_hostgroup_name* comme suit :

```
define hostdependency{
        hostgroup_name HOSTGROUP1,HOSTGROUP2
        dependent_hostgroup_name HOSTGROUP3,HOSTGROUP4
        other dependency directives
                }
```

Dans l'exemple ci-dessus, tous les htes des groupes d'htes *HOSTGROUP3* et *HOSTGROUP4* seront dépendants de tous les htes des groupes d'htes *HOSTGROUP1* et *HOSTGROUP2*. Toutes les instances de la dépendance d'hte seront identiques, é l'exception des noms d'htes (c.-é-d. auront le m'me critére d'échec de notification, etc.).

Groupes d'htes

Tous les htes : Si vous souhaitez créer un groupe d'hte contenant tous les htes définis dans vos fichiers de configuration, vous pouvez utiliser un caractére de remplacement dans le paramétre *members* de la maniére suivante :

```
define hostgroup{
        hostgroup_name HOSTGROUP1
        members *
        other hostgroup directives
                }
```

La définition ci-dessus créera un groupe d'htes appelé *HOSTGROUP1* contenant **tous les htes** définis dans vos fichiers de configuration.

Intégration de UCD-SNMP (NET-SNMP)

Note : Nagios n'a pas été conéu pour remplacer une application compléte d'administration SNMP comme HP OpenView ou OpenNMS [Anglais]. Toutefois, vous pouvez faire en sorte que les interruptions [NdT : traps] SNMP reéus par un hte de votre réseau génénent des alertes dans Nagios. Voici comment

Introduction

Cet exemple explique comment générer facilement dans Nagios des alertes qui correspondent é des interruptions SNMP reéues par le service UCD-SNMP [Anglais] *snmptrapd*. Ces indications présupposent que l'hte qui reéoit les interruptions SNMP n'est pas celui sur lequel s'exécute Nagios. Si votre machine de supervision est la m'me que celle qui reéoit les interruptions SNMP, vous devrez adapter les exemples que je fournis. De plus, je présuppose que vous avez installé le service nsca sur votre machine de supervision et le client nsca (*send_nsca*) sur la machine qui reéoit les interruptions SNMP.

Dans cet exemple, je décrirai comment configurer Nagios pour générer des alertes é partir des interruptions SNMP reéues concernant les travaux de sauvegarde ArcServe qui tournent sur mes serveurs Novell. Je voulais ’tre notifié en cas d'erreur de la sauvegarde, ce qui a trés bien fonctionné pour moi. Vous devrez adapter les exemples é vos propres besoins.

Logiciels complémentaires

Traduire des interruptions SNMP en événements Nagios peut ’tre fastidieux. Si vous voulez vous simplifier les choses, intéressez-vous au projet SNMP Trap Translator d'Alex Burger disponible sur http://www.snmptt.org [Anglais] qui, combiné é Net-SNMP, constitue un systéme d'administration des interruptions plus évolué. La documentation de snmptt détaille la procédure d'intégration é Nagios.

Définition du service

Vous devez d'abord définir un service dans votre fichier de configuration des objets, pour les interruptions SNMP (dans notre exemple, je définis un service pour les travaux de sauvegarde ArcServe). Si l'on suppose que l'hte dont proviennent les alertes s'appelle **novellserver**, la définition du service ressemblerait é ceci :

```
define service{
        host_name                       novellserver
        service_description             ArcServe Backup
        is_volatile                     1
        active_checks_enabled           0
        passive_checks_enabled          1
        max_check_attempts              1
        contact_groups                  novell-backup-admins
        notification_interval           120
        notification_period             24x7
        notification_options            w,u,c,r
        check_command                   check_none
        }
```

Il est important de noter que le paramétre *volatile* est activé pour ce service. Nous l'activons car nous voulons qu'une notification soit générée pour chaque alerte reéue. Notez également que les contrles actifs sont désactivés pour ce service, et les contrles passifs activés. Cela signifie que le service ne sera jamais contrlé activement - toutes les informations d'alerte devront ’tre envoyées passivement par le *client nsca* qui s'exécute sur l'hte d'administration SNMP (dans mon exemple, celui-ci s'appelle **firestorm**).

Configuration SNMP d'ArcServe et de Novell

Pour qu'ArcServe (et mon serveur Novell) envoient des interruptions SNMP é mon hte d'administration, j'ai d mener les actions suivantes :

1. Modifier le travail autopilot d'ArcServe pour qu'il envoie des interruptions SNMP en cas d'échec, de réussite, etc. des travaux
2. Editer SYS:\ETC\TRAPTARG.CFG et y ajouter l'adresse IP de mon hte d'administration (celui qui reéoit les interruptions SNMP)
3. Charger SNMP.NLM
4. Charger ALERT.NLM pour permettre l'envoi effectif des interruptions SNMP

Configuration de l'hte d'administration SNMP

Sur mon hte Linux d'administration SNMP (**firestorm**), j'ai installé le logiciel UCD-SNMP [Anglais] (NET-SNMP). Une fois le logiciel installé, il m'a fallu :

1. Installer les MIB ArcServe (comprises dans le CD d'installation d'ArcServe).
2. Editer le fichier de configuration de snmptrapd *(/etc/snmp/snmptrapd.conf)* pour définir un gestionnaire d'interruption [NdT : trap handler] pour les alertes ArcServe. Le détail est ci-dessous.
3. Démarrer le service *snmptrapd* pour recevoir les interruptions SNMP entrantes.

Pour que le service *snmptrapd* route les interruptions SNMP d'ArcServe é notre hte Nagios, nous devons définir un gestionnaire d'interruption dans le fichier */etc/snmp/snmptrapd.conf*. Dans mon cas, le fichier de configuration ressemble é ceci :

```
############################
# ArcServe SNMP Traps
############################

# Tape format failures
traphandle ARCserve-Alarm-MIB::arcServetrap9 /usr/local/nagios/libexec/eventhan

# Failure to read tape header
traphandle ARCserve-Alarm-MIB::arcServetrap10 /usr/local/nagios/libexec/eventha

# Failure to position tape
traphandle ARCserve-Alarm-MIB::arcServetrap11 /usr/local/nagios/libexec/eventha

# Cancelled jobs
traphandle ARCserve-Alarm-MIB::arcServetrap12 /usr/local/nagios/libexec/eventha

# Successful jobs
traphandle ARCserve-Alarm-MIB::arcServetrap13 /usr/local/nagios/libexec/eventha

# Imcomplete jobs
traphandle ARCserve-Alarm-MIB::arcServetrap14 /usr/local/nagios/libexec/eventha

# Job failures
traphandle ARCserve-Alarm-MIB::arcServetrap15 /usr/local/nagios/libexec/eventha
```

Cet exemple présuppose que vous avez un répertoire */usr/local/nagios/libexec/eventhandlers/* sur votre hte d'administration SNMP et que le script *handle-arcserve-trap* s'y trouve. Vous pouvez modifier ces paramétres selon vos besoins. Quoiqu'il en soit, le script *handle-arcserve-trap* de mon hte d'administration ressemble é ceci :

```
#!/bin/sh

# Arguments:
#  $1 = trap type

        # First line passed from snmptrapd is FQDN of host that sent the trap
     read host
```

```
# Given a FQDN, get the short name of the host as it is setup in Nagios
hostname="unknown"
case $host in
    novellserver.mylocaldomain.com)
        hostname="novellserver"
        ;;
    nt.mylocaldomain.com)
        hostname="ntserver"
        ;;
    esac

# Get severity level (OK, WARNING, UNKNOWN, or CRITICAL) and plugin output
state=-1
output="No output"
case "$1" in

    # failed to format tape - critical
    11)
        output="Critical: Failed to format tape"
        state=2
        ;;

    # failed to read tape header - critical
    10)
        output="Critical: Failed to read tape header"
        state=2
        ;;

    # failed to position tape - critical
    11)
        output="Critical: Failed to position tape"
        state=2
        ;;

    # backup cancelled - warning
    12)
        output="Warning: ArcServe backup operation cancelled"
        state=1
        ;;

    # backup success - ok
    13)
        output="Ok: ArcServe backup operation successful"
        state=0
        ;;

    # backup incomplete - warning
    14)
        output="Warning: ArcServe backup operation incomplete"
        state=1
```

```
            ;;

        # backup failure - critical
        15)
            output="Critical: ArcServe backup operation failed"
            state=2
            ;;
    esac


    # Submit passive check result to monitoring host
    /usr/local/nagios/libexec/eventhandlers/submit_check_result $hostname "ArcS

exit 0
```

Notez que le script *handle-arcserve-trap* appelle le script *submit_check_result* pour envoyer effectivement l'alerte é l'hte de supervision. Si votre hte de supervision s'appelle **monitor**, le script *submit_check_result* ressemblerait é ceci (modifiez-le pour préciser l'emplacement correct du programme *send_nsca* sur votre hte d'administration):

```
#!/bin/sh

# Arguments
#       $1 = name of host in service definition
#       $2 = name/description of service in service definition
#       $3 = return code
#       $4 = output

/bin/echo -e "$1\t$2\t$3\t$4\n" | /usr/local/nagios/bin/send_nsca monitor -c /u
```

Pour terminer

Vous avez maintenant configuré tout ce qui doit l''tre, il ne reste plus qu'é redémarrer Nagios sur votre serveur de supervision. C'est fini ! Vous devriez recevoir des alertes dans Nagios é chaque fois qu'un travail échoue, réussit, etc.

Intégration d'un TCP Wrapper

Introduction

Cet exemple explique comment générer aisément des alertes dans Nagios pour des connexions rejetées par un TCP wrapper (encapsuleur TCP). Ces explications supposent que l'hte pour lequel vous générez ces alertes (i.e. l'hte sur lequel vous utilisez le TCP wrappers) n'est pas le m’me hte que celui sur lequel Nagios est installé. Si vous souhaitez générer des alertes sur l'hte Nagios, vous aurez besoin de faire quelques modifications é l'exemple que je vous propose. Je suppose également que vous avez installé le daemon nsca sur la machine de surpervision et le client nsca (*send_nsca*) sur la machine qui génére les alertes TCP wrappers.

Définition du Service

Tout d'abord vous devez définir un service dans votre fichier de configuration des objets pour les alertes du TCP wrapper. En supposant que l'hte émettant les alertes s'appelle **firestorm**, un exemple de définition pourrait ressembler é quelque chose comme éa :

```
define service{
        host_name                       firestorm
        service_description             TCP Wrappers
        is_volatile                     1
        active_checks_enabled           0
        passive_checks_enabled          1
        max_check_attempts              1
        contact_groups                  security-admins
        notification_interval           120
        notification_period             24x7
        notification_options            w,u,c,r
        check_command                   check_none
        }
```

Il est important de noter que le service é l'option *volatile* activée. Cette option est activée parce que nous voulons qu'une notification soit générée pour toutes les alertes survenant. Notez également que les contrles actifs sont désactivés pour ce service, alors que les contrles passifs sont activés. Ceci signifie que ce service ne sera jamais contrlé activement - toutes les alertes seront envoyées passivement par le *client nsca* de l'hte **firestorm**.

Configuration du TCP Wrapper

Maintenant, il faut modifier le fichier */etc/hosts.deny* sur la machine **firestorm**. Pour que l'encapsuleur TCP envoie une alerte é chaque connexion refusée, vous devez ajouter une ligne de ce type:

```
ALL: ALL: RFC931: twist (/usr/local/nagios/libexec/eventhandlers/handle_tcp_wra
```

Cette ligne suppose qu'il existe un script appellé *handle_tcp_wrapper* dans le répertoire */usr/local/nagios/libexec/eventhandlers/* sur **firestorm**. Le répertoire et le nom du script peuvent ’tre changés comme vous le voulez.

Ecriture du Script

La derniére chose é faire est d'écrire le script *handle_tcp_wrapper* sur **firestorm** qui enverra les alertes é l'hte de supervision. Cela donnera quelque chose qui ressemblera é éa :

```
#!/bin/sh

/usr/local/nagios/libexec/eventhandlers/submit_check_result firestorm "TCP Wrap
```

Notez que le script *handle_tcp_wrapper* appelle le script *submit_check_result* pour envoyer des alertes é l'hte chargé de supervision. Supposons que votre hte de supervision s'appelle **monitor**, le script *submit check_result* pourrait ressembler é ceci (vous devrez éventuellement modifier le script pour spécifier l'emplacement du programme *send_nsca* sur **firestorm**):

```
#!/bin/sh

# Arguments
#       $1 = name of host in service definition
#       $2 = name/description of service in service definition
#       $3 = return code
#       $4 = output

/bin/echo -e "$1\t$2\t$3\t$4\n" | /usr/local/nagios/bin/send_nsca monitor -c /u
```

Finition

Maintenant que vous avez configure tout ce dont vous avez besoin, vous devez redémarrer le processus *inetd* sur **firestorm** et redémarrer Nagios sur votre serveur de supervision. C'est tout ! Quand le TCP wrapper sur **firestorm** refusera une connexion, vous devriez recevoir des alertes via Nagios. Cela ressemblera é éa :

```
Denied sshd2-sdn-ar-002mnminnP321.dialsprint.net
```

Securisé Nagios

Introduction

Ce paragraphe est un rapide survol des éléments que vous devez conserver é l'esprit au moment d'installer Nagios, afin que l'installation soit sécurisée. Ce document est nouveau et si quelqu'un a des éléments é y ajouter, qu'il m'envoie un courrier é l'adresse nagios@nagios.org [NdT : mail é envoyer en anglais].

Ne pas lancer Nagios en tant que Root!

Il n'est pas nécessaire d''tre root pour faire lancer Nagios. M'me si vous lancez Nagios au démarrage é l'aide d'un init script, vous pouvez l'obliger é abandonner des priviléges aprés le lancement et é fonctionner avec les droits d'un autre utilisateur/groupe en utilisant les directives nagios_user et nagios_group dans le fichier principal de configuration.

Si vous avez besoin d'exécuter le gestionnaire d'événements ou des plugins en tant que root, essayez d'utiliser sudo.

Autoriser les commandes externes seulement si c'est nécessaire

Par défaut, les commandes externes sont désactivées. Cela a pour but d'emp'cher un administrateur de lancer Nagios et de laisser involontairement l'interface de commande ouverte é la disposition "d'autres" Si vous pensez avoir besoin des gestionnaires d'événements ou d'exécuter des commandes é partir de l'interface web, il faudra autoriser l'usage des commandes externes. Si vous pensez n''tre dans aucun des cas précédents, je vous conseille de désactiver les commandes externes.

Donner les autorisations adaptées au fichier des commandes externes

Si vous permettez l'utilisation des commandes externes, assurez-vous que le répertoire */usr/local/nagios/var/rw* dispose des permissions adéquates. Vous souhaitez que seulement l'utilisateur Nagios (habituellement *nagios*) et l'utilisateur du serveur web(habituellement *nobody*) aient l'autorisation d'écrire dans le fichier de commandes. Si vous avez installer Nagios sur une machine dédiée et qu'elle n'a pas de comptes d'utilisateurs, cela doit 'tre suffisant.

Si vous avez installé Nagios sur une machine multi-utilisateur publique, permettre l'accés au fichier de commande via le serveur web peut causer des problémes de sécurité. Aprés tout, je ne pense pas que vous souhaitiez permettre é tout le monde de contrler nagios. ans ce cas, je vous recommande de n'accorder l'accés au fichier de commande qu'é l'utilisateur nagios et d'utiliser quelque chose comme CGIWrap pour exécuter les CGIs comme l'utilisateur *nagios* plutt que l'utilisateur *nobody*.

Les instructions pour configurer les autorisations é donner au fichier de commandes externes se trouvent ici.

Authentification requise pour les CGIs

Je vous recommande fortement de protéger l'accés aux CGI par une authentification. Ceci fait, lisez la documentation sur les droits par défaut dont disposent les contacts autorisés et n'accordez qu'exceptionnellement une autorisation é des contacts spécifiques. Les instructions pour la configuration des droits se trouvent ici. Si vous désactivez l'authentification préalable é l'accés aux CGI par la commande use_authentication dans le fichier de configuration des CGI, le CGI de commande refusera d'écrire la moindre commande dans le fichier des commandes externes. De toutes faéons, vous ne souhaitez pas que tout le monde puisse contrler vtre Nagios ?

Utiliser les chemins complets dans la définition des commandes

Lorsque vous définissez des commandes, assurez-vous d'avoir bien précisé le *chemin d'accés complet* de tous les scripts ou binaires que vous exécutez.

Cacher les informations sensibles é l'aide des macros $USERn$

Les CGI parcourent le fichier de configuration principal et le(s) fichier(s) de configuration des objets, n'y laissez donc pas d'informations sensibles (noms d'utilisateur, mots de passe, etc). Si vous avez besoin de préciser un mot de passe ou un nom d'utilisateur dans une définition de commande, utilisez une macro $USERn$ pour le cacher. Les macros $USERn$ sont définies dans un ou plusieurs fichiers de ressources. Les CGIs ne parcourant pas les fichiers de ressources, vous pouvez leur donner des permissions beaucoup plus restrictives (600 ou 660). Consultez le fichier *resource.cfg* dans la distribution de base de Nagios, il vous donnera un exemple de d?finition de la macro $USERn$.

Eliminer les caractéres dangereux des macros

Utilisez la directive illegal_macro_output_chars pour filtrer les caractéres dangereux des chanes issues des macros $OUTPUT$ et $PERFDATA$ avant qu'elles ne soient utilisées pour des notifications, etc. Des caractéres dangereux peuvent ’tre tout caractére qui peut ’tre interprété par un shell, ouvrant ainsi un trou de sécurité. Un bon exemple est la présence de la quote inverse (`) dans $OUTPUT$ et/ou $PERFDATA$, qui pourrait permettre é un attaquant d'exécuter un commande arbitraire comme l'utilisateur nagios (une autre bonne raison pour ne pas exécuter Nagios comme l'utilisateur root).

Régler Nagios pour des performances maximales

Introduction

Maintenant que vous avez pu installer et lancer Nagios, vous voulez savoir comment le régler plus finement Voici quelques points é prendre en compte pour optimiser Nagios. Faites moi savoir si vous en trouvez d'autres

Trucs et astuces d'optimisation :

1. **Utilisez des changements d'état agrégés**. En activant la consolidation des changements d'état (grôce au paramétre aggregate_status_updates), vous réduirez considérablement la charge sur votre hte de supervision, car il n'essaiera pas constamment de mettre é jour le journal des états. Ceci est particuliérement vrai si vous supervisez un grand nombre de services. La principale contrepartie lorsque vous agrégez les changements d'état est que les modifications d'état des htes et des services ne sont pas immédiatement répliquées dans le fichier d'état. Ceci peut vous poser probléme, ou pas.
2. **Utilisez un disque virtuel [NdT : ramdisk] pour conserver les données d'état**. Si vous utilisez le journal des états standard et que vous n'utilisez *pas* l'agrégation des changements d'état, pensez é mettre le répertoire o le journal des états est stocké sur un disque virtuel en mémoire. Cela accélérera pas mal les choses (é la fois pour le programme principal et les CGI) parce que cela évite beaucoup d'interruptions et d'accés au disque.
3. **Vérifiez la latence des services pour déterminer la meilleure valeur pour le nombre maximal de contrles en paralléle**. Nagios peut restreindre le nombre maximal de contrles de service exécutés en paralléle é la valeur que vous spécifiez dans le paramétre max_concurrent_checks. Cela vous permet de gérer la charge que Nagios impose é votre hte de supervision, mais cela peut aussi ralentir le traitement. Si vous notez des latences importantes (> 10 ou 15 secondes) pour la majorité de vos contrles de service (via le CGI d'informations complémentaires), vous privez sans doute Nagios des contrles dont il a besoin. Ce n'est pas la faute de Nagios - c'est la vtre. Dans des conditions idéales, tous les contrles de service ont une latence de 0, ce qui signifie qu'ils sont exécutés au moment précis o ils ont été ordonnancés. Ceci dit, il est normal que certains contrles aient de petites latences. Je recommanderais de doubler la valeur que propose Nagios pour le nombre minimal de contrles en paralléle, fournie lorsque Nagios est lancé avec le paramétre **-s**. Continuez é augmenter cette valeur tant que la latence moyenne pour vos services reste assez basse. Vous trouverez plus d'informations sur l'ordonnancement des contrles de service ici.
4. **Utilisez des contrles passifs é chaque fois que c'est possible**. La surcharge induite par le traitement des résultats des contrles passifs de service est bien moindre que celle des contrles actifs "normaux", donc prenez cette information en compte si vous supervisez de nombreux services. Notez que les contrles passifs de service ne sont réellement utiles que si une application externe réalise une partie de la supervision ou produit des rapports ; si c'est Nagios qui réalise tout le travail, ceci ne changera rien.
5. **Evitez l'utilisation des plugins interprétés.** L'utilisation de plugins compilés (C/C++, etc.) réduira significativement la charge de votre hte de supervision par rapport aux scripts interprétés (Perl, etc.). Si les scripts Perl ou autres sont faciles é écrire et fonctionnent bien, le fait qu'ils soient compilés/interprétés é chaque exécution peut augmenter considérablement la charge de votre hte de supervision lorsque vous avez de nombreux contrles de service. Si vous souhaitez utiliser des plugins Perl, essayez de les compiler en vrais exécutables grôce é perlcc(1) (un utilitaire qui fait partie de la distribution Perl standard) ou essayez de compiler Nagios avec un interpréteur Perl intégré (voir ci-dessous).
6. **Utilisez l'interpréteur Perl intégré.** Si vous utilisez de nombreux scripts Perl pour les contrles de service, etc., vous vous apercevrez sans doute qu'en compilant Nagios avec un interpréteur Perl

intégré vous accélérez les traitements. Pour compiler l'interpréteur Perl intégré, vous devez ajouter le paramétre *--enable-embedded-perl* au script de configuration [NdT : ./configure] avant de compiler Nagios. De m'me, si vous ajoutez le paramétre *--with-perlcache*, la version compilée de tous les scripts Perl exécutés par l'interpréteur Perl intégré sera mise en cache pour réutilisation.

7. **Optimisez les commandes de contrle d'hte.** Si vous contrlez l'état des htes avec le plugin check_ping, vous vous apercevrez que ces contrles se font bien plus vite en les éclatant. Plutt que de spécifier une valeur de 1 pour le paramétre *max_attempts* dans la définition de l'hte et de dire au plugin check_ping d'envoyer 10 paquets ICMP é l'hte, il est bien plus rapide de passer max_attempts é 10 et de n'envoyer qu'un paquet ICMP é chaque fois. Ceci est d au fait que Nagios peut souvent déterminer l'état d'un hte aprés n'avoir exécuté le plugin qu'une fois, il vaut donc mieux que le premier contrle soit le plus rapide possible. Cette méthode présente des inconvénients dans certaines situations (c.-é-d. que les htes lents é répondre peuvent 'tre considérés comme hors service), mais vous aurez des contrles d'hte plus rapides si vous l'utilisez. Vous pouvez aussi utiliser un plugin plus rapide (c.-é-d. check_fping) dans le paramétre *host_check_command* plutt que check_ping.
8. **Ne planifiez pas une vérification réguliére des htes**. Il ne faut PAS planifier des vérifications réguliéres d'htes é moins que ce ne soit absolument nécessaire. Il n'y a pas beaucoup de raisons de faire cela, car les vérifications d'htes sont effectuées é la demande lorsque c'est nécessaire. Pour désactiver la vérification réguliére d'un hte, mettez 0 é la directive *check_interval* dans la définition d'hte. Si vous avez besoin de vérifier réguliérement les htes, essayez de mettre un intervalle de vérification plus grand et vérifiez que vos vérifications sont optimisées (voir plus haut).
9. **N'utilisez pas le contrle agressif des htes.** Sauf si Nagios a du mal é identifier les rétablissements d'hte, je recommanderais de *ne pas* activer le paramétre use_aggressive_host_checking. Quand cette option est désactivée, les contrles s'exécutent beaucoup plus vite, accélérant le traitement des résultats de contrles de service. Cependant, les rétablissements d'htes peuvent 'tre manqués en certaines circonstances lorsque l'option est désactivée. Par exemple, si un hte se rétablit et que tous les services associés é cet hte restent dans un état non-OK (et ne "bagotent" pas entre différents états non-OK), Nagios peut ne pas voir que l'hte s'est rétabli. Certains utilisateurs peuvent avoir besoin d'activer cette option, mais ce n'est pas le cas pour la majorité, et je recommanderais de *ne pas* l'utiliser si vous n'en avez pas expressément besoin
10. **Augmentez l'intervalle de vérification des commandes externes.** Si vous gérez beaucoup de commandes externes (p. ex. des vérifications passives dans une supervision distribuée), vous devrez probablement affecter **-1** au paramétre command_check_interval. Cela forcera Nagios é vérifier les commandes externes aussi souvent que possible. C'est important parce que la plupart des systémes ont une petite taille de tampon pour les tubes de redirections [NdT : pipe] (c.-é-d. 4 Ko). Si Nagios ne lit pas les données depuis le tube le plus rapidement possible, l'application qui écrit dans le fichier de commandes externes (p. ex. le démon NSCA) se bloquera et attendra jusqu'é ce qu'il y ait assez d'espace libre dans le tube pour écrire ses données.
11. **Optimisez le matériel pour des performances maximales.** La configuration matérielle va affecter directement les performances de votre systéme d'exploitation, et donc celles de Nagios. L'amélioration principale que vous puissiez réaliser concerne les disques durs. La vitesse du processeur et la mémoire affectent bien évidemment les performances, mais les accés disque seront votre goulet d'étranglement le plus fréquent. Ne stockez pas les plugins, le journal des états, etc. sur des disques lents (c.-é-d. des vieux disques IDE ou des montages NFS). Si vous en avez, utilisez des disques UltraSCSI ou des disques IDE rapides. Une remarque importante pour les utilisateurs de IDE/Linux : bien des installations de Linux n'essaient pas d'optimiser les accés au disque. Si vous ne changez pas les paramétres d'accés au disque (en utilisant un utilitaire comme **hdparam**), vous perdrez **beaucoup** des fonctionnalités améliorant la vitesse des nouveaux disques IDE.

Utilisation de l'outil Nagiostats

Introduction

Un outil appelé *nagiostats* est inclus dans la distribution de Nagios. Il est compilé et installé avec le démon principal de Nagios.

L'outil nagiostat vous permet d'obtenir diverses informations sur le processus Nagios. Vous pouvez obtenir les informations soit dans un format compatible avec MRTG soit dans un format intelligible.

Instructions d'utilisation

Vous pouvez lancer l'outil *nagiostats* avec l'option **--help** pour avoir les informations d'utilisation :

```
[nagios@lanman ~]# /usr/local/nagios/bin/nagiostats --help

Nagios Stats 2.0a1
Copyright (c) 2003 Ethan Galstad (nagios@nagios.org)
Last Modified: 11-18-2003
License: GPL

Usage: /usr/local/nagios/bin/nagiostats [options]

Startup:
 -V, --version      display program version information and exit.
 -L, --license      display license information and exit.
 -h, --help         display usage information and exit.

Input file:
 -c, --config=FILE  specifies location of main Nagios config file.

Output:
 -m, --mrtg         display output in MRTG compatible format.
 -d, --data=VARS    comma-seperated list of variables to output in MRTG
                    (or compatible) format.  See possible values below.
                    Percentages are rounded, times are in milliseconds.

MRTG DATA VARIABLES (-d option):
 NUMSERVICES        total number of services.
 NUMHOSTS           total number of services.
 NUMSVCOK           number of services OK.
 NUMSVCWARN         number of services WARNING.
 NUMSVCUNKN         number of services UNKNOWN.
 NUMSVCCRIT         number of services CRITICAL.
 NUMSVCPROB         number of service problems (WARNING, UNKNOWN or CRITIAL).
 NUMHSTUP           number of hosts UP.
 NUMHSTDOWN         number of hosts DOWN.
 NUMHSTUNR          number of hosts UNREACHABLE.
 NUMHSTPROB         number of host problems (DOWN or UNREACHABLE).
```

```
 xxxACTSVCLAT       MIN/MAX/AVG active service check latency (ms).
 xxxACTSVCEXT       MIN/MAX/AVG active service check execution time (ms).
 xxxACTSVCPSC       MIN/MAX/AVG active service check % state change.
 xxxPSVSVCPSC       MIN/MAX/AVG passive service check % state change.
 xxxSVCPSC          MIN/MAX/AVG service check % state change.
 xxxACTHSTLAT       MIN/MAX/AVG active host check latency (ms).
 xxxACTHSTEXT       MIN/MAX/AVG active host check execution time (ms).
 xxxACTHSTPSC       MIN/MAX/AVG active host check % state change.
 xxxPSVHSTPSC       MIN/MAX/AVG passive host check % state change.
 xxxHSTPSC          MIN/MAX/AVG host check % state change.
 NUMACTHSTCHKxM    number of active host checks in last 1/5/15/60 minutes.
 NUMPSVHSTCHKxM    number of passive host checks in last 1/5/15/60 minutes.
 NUMACTSVCCHKxM    number of active service checks in last 1/5/15/60 minutes.
 NUMPSVSVCCHKxM    number of passive service checks in last 1/5/15/60 minutes.

 Note: Replace x's in MRTG variable names with 'MIN', 'MAX', 'AVG', or the
       the appropriate number (i.e. '1', '5', '15', or '60').

[nagios@lanman ~]#
```

Affichage intelligible

Pour une utilisation normale, lancez l'outil *nagiostat*, en spécifiant uniquement le chemin du fichier de configuration en argument, comme suit :

```
[nagios@lanman ~]# /usr/local/nagios/bin/nagiostats -c /usr/local/nagios/etc/na

Nagios Stats 2.0a1
Copyright (c) 2003 Ethan Galstad (nagios@nagios.org)
Last Modified: 11-18-2003
License: GPL

CURRENT STATUS DATA
------------------------------------------------------
Status File:                            /usr/local/nagios/var/status.dat
Status File Age:                        0d 0h 0m 13s
Status File Version:                    2.0-very-pre-alpha

Program Running Time:                   14d 17h 19m 13s

Total Services:                         32
Services Checked:                       32
Services Scheduled:                     29
Active Service Checks:                  29
Passive Service Checks:                 3
Total Service State Change:             0.000 / 65.530 / 2.930 %
Active Service Latency:                 0.048 / 14.837 / 1.035 %
Active Service Execution Time:          0.076 / 60.006 / 4.301 sec
Active Service State Change:            0.000 / 10.530 / 0.762 %
Active Services Last 1/5/15/60 min:     1 / 13 / 29 / 29
```

```
Passive Service State Change:         0.000 / 65.530 / 23.883 %
Passive Services Last 1/5/15/60 min:  0 / 0 / 0 / 0
Services Ok/Warn/Unk/Crit:            23 / 5 / 1 / 3
Services Flapping:                    1
Services In Downtime:                 0

Total Hosts:                          9
Hosts Checked:                        9
Hosts Scheduled:                      9
Active Host Checks:                   9
Passive Host Checks:                  0
Total Host State Change:              0.000 / 28.420 / 4.034 %
Active Host Latency:                  0.000 / 15.741 / 5.443 %
Active Host Execution Time:           1.022 / 10.032 / 3.047 sec
Active Host State Change:             0.000 / 28.420 / 4.034 %
Active Hosts Last 1/5/15/60 min:      0 / 8 / 9 / 9
Passive Host State Change:            0.000 / 0.000 / 0.000 %
Passive Hosts Last 1/5/15/60 min:     0 / 0 / 0 / 0
Hosts Up/Down/Unreach:                7 / 1 / 1
Hosts Flapping:                       0
Hosts In Downtime:                    0


[nagios@lanman ~]#
```

Comme vous pouvez le constater, l'outil affiche un bon nombre de mesures concernant le processus Nagios. Les mesures qui ont des valeurs multiples sont (sauf mention contraire) les valeurs min, max et moyenne pour cette mesure particuliére.

Intégration é MRTG

Vous pouvez utiliser l'outil *nagiostat* pour afficher diverses mesures de Nagios avec MRTG (ou tout autre programme compatible). Pour cela, lancez *nagiostat* en utilisant les arguments **--mrtg** et **--data**. L'option **--data** est utilisée pour spécifier que les statistiques sont destinées é ’tre représentées graphiquement. Les valeurs possibles pour l'option **--data** sont listées en lanéant l'outil *nagiostat* avec l'option**--help**.

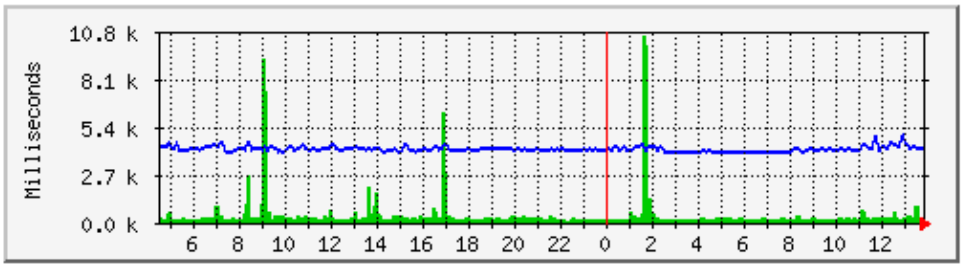
Voici un extrait de fichier de configuration MRTG utilisé par l'outil *nagiostat* pour représenter graphiquement le temps moyen de latence et d'exécution d'un service.

```
# Service Latency and Execution Time
Target[nagios-a]: `/usr/local/nagios/bin/nagiostats --mrtg --data=AVGACTSVCLAT,
MaxBytes[nagios-a]: 100000
Title[nagios-a]: Average Service Check Latency and Execution Time
PageTop[nagios-a]: <H1>Average Service Check Latency and Execution Time</H1>
Options[nagios-a]: growright,gauge,nopercent
YLegend[nagios-a]: Milliseconds
ShortLegend[nagios-a]:  
LegendI[nagios-a]:  Latency:
LegendO[nagios-a]:  Execution Time:
Legend1[nagios-a]: Latency
```

```
Legend2[nagios-a]: Execution Time
Legend3[nagios-a]: Maximal 5 Minute Latency
Legend4[nagios-a]: Maximal 5 Minute Execution Time
```

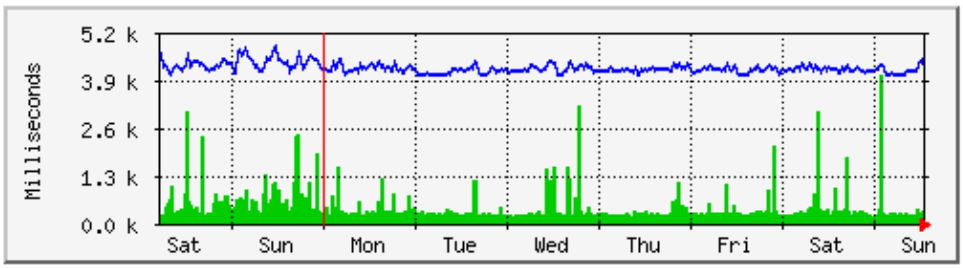
Les graphes MRTG générés par l'extrait de configuration ci-dessus ressemblent é ceci :

`Daily' Graph (5 Minute Average)



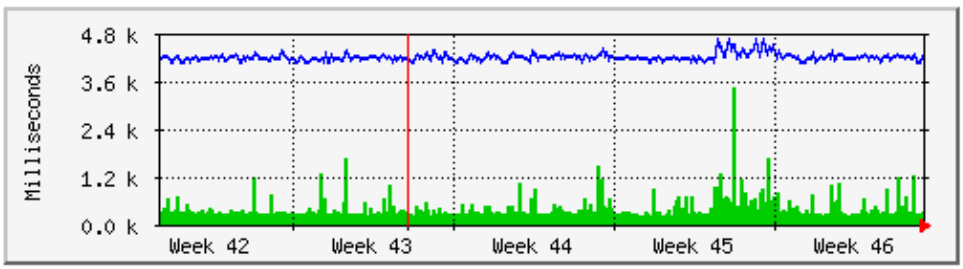
Max Latency: 10.6 k Average Latency: 480.0 Current Latency: 342.0
Max Execution Time:4993.0 Average Execution Time:4223.0 Current Execution Time:4238.0

`Weekly' Graph (30 Minute Average)



Max Latency:4103.0 Average Latency: 470.0 Current Latency: 368.0
Max Execution Time:4847.0 Average Execution Time:4259.0 Current Execution Time:4596.0

`Monthly' Graph (2 Hour Average)



Max Latency:3503.0 Average Latency: 426.0 Current Latency: 353.0
Max Execution Time:4699.0 Average Execution Time:4227.0 Current Execution Time:4309.0

Utilisation des macros dans les commandes

Macros

Une des fonctionalités disponibles dans Nagios est la possibilité d'utiliser des macros [NdT: macro-commandes] dans les définitions de commandes. Avant d'exécuter une commande, Nagios remplacera toutes les macros dans celle-ci par les valeurs correspondantes. Cela vous permet de définir quelques commandes génériques que vous manipulerez é votre guise.

Substitution des macros

Avant qu'une commande (contrles d'hte et de service, notifications, gestionnaires d'événements, etc.) soit exécutée, Nagios va remplacer toutes les macros qu'il trouvera dans la définition de la commande par les valeurs correspondantes.

Quand vous utilisez des macros d'hte et de service dans les définitions de commandes, elles font référence aux valeurs de l'hte ou du service pour lequel s'exécute la commande. Prenons un exemple. Supposons que nous ayons une définition d'hte et une commande *check_ping* définies de la maniére suivante :

```
define host{
        host_name       linuxbox
        address         192.168.1.2
        check_command   check_ping

        }

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

La ligne de commande finale, aprés substitution, qui sera exécutée pour le contrle de l'hte ressemblera é ceci :

```
        /usr/local/nagios/libexec/check_ping -H 192.168.1.2 -w 100.0,90% -c 200
```

Vous pouvez aussi passer des paramétres aux commandes, ce qui est pratique si vous voulez avoir des définitions de commandes plus génériques. Les paramétres sont spécifiés dans la définition de l'objet (c.-é-d. de l'hte ou du service), en les séparant du nom de la commande par des points d'exclamation (!) comme ceci :

```
define service{
        host_name               linuxbox
        service_description     PING

        check_command           check_ping!200.0,80%!400.0,40%

        }
```

Dans l'exemple ci-dessus, la commande de contrle du service a deux paramétres (auxquels on fait référence par les macros $ARGn$). La macro $ARG1$ vaudra "200.0,80%" et $ARG2$ vaudra "400.0,40%" (sans les

guillemets). Supposons que nous utilisons la définition d'hte ci-dessus et une commande *check_ping* définie comme ceci :

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

La ligne de commande finale é exécuter pour le contrle du service sera :

```
        /usr/local/nagios/libexec/check_ping -H 192.168.1.2 -w 200.0,80% -c 400
```

Macros é la demande

Normalement, quand vous utilisez des macros d'hte ou de service dans les définitions de commandes, elles font référence aux valeurs de l'hte ou du service pour lequel s'exécute la commande. Par exemple, si la commande de contrle d'un hte s'exécute pour l'hte appelé "linuxbox", toutes les macros d'hte listées dans le tableau ci-aprés font référence aux valeurs de cet hte ("linuxbox").

Si vous souhaitez faire référence aux valeurs d'un autre hte ou service dans une commande (pour lequel ne s'exécute pas la commande), vous pouvez utiliser ce qu'on appellle des macros "é la demande". Les macros é la demande ressemblent aux macros normales, mis é part qu'elle contiennent l'identifiant de l'hte ou du service o aller chercher leur valeur. Voici la syntaxe de base pour les macros é la demande :

- $HOSTMACRO:*host_name*$
- $SERVICEMACRO:*host_name*:*service_description*$

Notez que le nom de la macro est séparé de l'identifiant de l'hte ou du service par deux points (:). Pour les macros de service é la demande, l'identifiant du service est composé d'un nom d'hte et d'une description de service - qui sont séparés par deux point (:) également.

Voici des exemples de macros d'hte et de service é la demande :

```
$HOSTDOWNTIME:myhost$
$SERVICESTATEID:novellserver:DS Database$
```

Nettoyage des macros

Les valeurs de certaines macros sont dépouillées des métacaractéres du shell potentiellement dangereux, avant d''tre remplacées dans la commande é exécuter. Les caractéres qui seront supprimés dans les macros sont définis par le paramétre illegal_macro_output_chars. Ce nettoyage s'applique aux macros suivantes :

1. $HOSTOUTPUT$
2. $HOSTPERFDATA$
3. $HOSTACKAUTHOR$
4. $HOSTACKCOMMENT$
5. $SERVICEOUTPUT$
6. $SERVICEPERFDATA$

7. $SERVICEACKAUTHOR$
8. $SERVICEACKCOMMENT$

Macros et variables d'environnement

A partir de la version 2.0 de Nagios, la plupart des macros sont disponibles sous forme de variables d'environnement. Cela signifie que les scripts lancés par Nagios (c.-é-d. les commandes de contrle de service et d'hte, les commandes de notification, etc.) peuvent faire référence é ces macros directement en tant que variables d'environnement standard. Pour des raisons de sécurité et de bon sens, $USERn$ et les macros d'hte et de service é la demande ne sont pas disponibles comme variables d'environnement. Les variables d'environnement qui contiennent des macros sont nommées comme les macros correspondantes (listées ci-dessous), avec le préfixe "NAGIOS_". Par exemple, la macro $HOSTNAME$ est disponible é travers la variable d'environnement nommée "NAGIOS_HOSTNAME".

Validité des macros

Bien que toutes les macros puissent ’tre utilisées dans toutes les commandes que vous définissez, elles ne sont pas toutes "valables" pour une commande en particulier. Par exemple, certaines macros ne sont valables que dans les commandes de notification de service, alors que d'autres ne sont valables que dans les commandes de contrle d'hte. Il y a dix types de commandes que Nagios identifie et traite différemment. Ce sont les suivantes :

1. Les contrles de service
2. Les notifications de service
3. Les contrles d'hte
4. Les notifications d'hte
5. Les gestionnaires d'événements de service et/ou le gestionnaire global d'événements de service
6. Les gestionnaires d'événements d'hte et/ou le gestionnaire global d'événements d'hte
7. La commande OCSP
8. La commande OCHP
9. Les commandes de données de performance de service
10. Les commandes de données de performance d'hte

La tableau ci-dessous liste toutes les macros actuellement disponibles dans Nagios, accompagnées d'une bréve description et des types de commandes pour lesquels elles sont valables. Si une macro est utilisée dans une commande o elle n'est pas valable, elle est remplacée par une chane vide. Notez que les macros sont entiérement en majuscules et sont comprises entre des caractéres **$**.

Tableau de disponibilité des macros

Légende :

Non	la macro n'est pas disponible
Oui	La macro est disponible

Nom de la macro	Contrles de service	Notifications de service	Contrles d'hte	Notifications d'hte	Gestionnaires d'événements de service,

					Gestionnaire global d'événements de service, commandes OCSP	
Macros d'hte : 3						
$HOSTNAME$	Oui	Oui	Oui	Oui	Oui	
$HOSTALIAS$	Oui	Oui	Oui	Oui	Oui	
$HOSTADDRESS$	Oui	Oui	Oui	Oui	Oui	
$HOSTSTATE$	Oui	Oui	Oui 1	Oui	Oui	
$HOSTSTATEID$	Oui	Oui	Oui 1	Oui	Oui	
$HOSTSTATETYPE$	Oui	Oui	Oui 1	Oui	Oui	
$HOSTATTEMPT$	Oui	Oui	Oui	Oui	Oui	
$HOSTLATENCY$	Oui	Oui	Oui	Oui	Oui	
$HOSTEXECUTIONTIME$	Oui	Oui	Oui 1	Oui	Oui	
$HOSTDURATION$	Oui	Oui	Oui	Oui	Oui	
$HOSTDURATIONSEC$	Oui	Oui	Oui	Oui	Oui	
$HOSTDOWNTIME$	Oui	Oui	Oui	Oui	Oui	
$HOSTPERCENTCHANGE$	Oui	Oui	Oui	Oui	Oui	
$HOSTGROUPNAME$	Oui	Oui	Oui	Oui	Oui	
$HOSTGROUPALIAS$	Oui	Oui	Oui	Oui	Oui	
$LASTHOSTCHECK$	Oui	Oui	Oui	Oui	Oui	
$LASTHOSTSTATECHANGE$	Oui	Oui	Oui	Oui	Oui	
$LASTHOSTUP$	Oui	Oui	Oui	Oui	Oui	
$LASTHOSTDOWN$	Oui	Oui	Oui	Oui	Oui	
$LASTHOSTUNREACHABLE$	Oui	Oui	Oui	Oui	Oui	
$HOSTOUTPUT$	Oui	Oui	Oui 1	Oui	Oui	
$HOSTPERFDATA$	Oui	Oui	Oui 1	Oui	Oui	
$HOSTCHECKCOMMAND$	Oui	Oui	Oui	Oui	Oui	
$HOSTACKAUTHOR$	Non	Non	Non	Oui	Non	
$HOSTACKCOMMENT$	Non	Non	Non	Oui	Non	
$HOSTACTIONURL$	Oui	Oui	Oui	Oui	Oui	
$HOSTNOTESURL$	Oui	Oui	Oui	Oui	Oui	
$HOSTNOTES$	Oui	Oui	Oui	Oui	Oui	
Macros de service :						
$SERVICEDESC$	Oui	Oui	Non	Non	Oui	
$SERVICESTATE$	Oui 2	Oui	Non	Non	Oui	
$SERVICESTATEID$	Oui 2	Oui	Non	Non	Oui	
$SERVICESTATETYPE$	Oui	Oui	Non	Non	Oui	
$SERVICEATTEMPT$	Oui	Oui	Non	Non	Oui	
$SERVICELATENCY$	Oui	Oui	Non	Non	Oui	
$SERVICEEXECUTIONTIME$	Oui 2	Oui	Non	Non	Oui	

$SERVICEDURATION$	Oui	Oui	Non	Non	Oui	N
$SERVICEDURATIONSEC$	Oui	Oui	Non	Non	Oui	N
$SERVICEDOWNTIME$	Oui	Oui	Non	Non	Oui	N
$SERVICEPERCENTCHANGE$	Oui	Oui	Non	Non	Oui	N
$SERVICEGROUPNAME$	Oui	Oui	Non	Non	Oui	N
$SERVICEGROUPALIAS$	Oui	Oui	Non	Non	Oui	N
$LASTSERVICECHECK$	Oui	Oui	Non	Non	Oui	N
$LASTSERVICESTATECHANGE$	Oui	Oui	Non	Non	Oui	N
$LASTSERVICEOK$	Oui	Oui	Non	Non	Oui	N
$LASTSERVICEWARNING$	Oui	Oui	Non	Non	Oui	N
$LASTSERVICEUNKNOWN$	Oui	Oui	Non	Non	Oui	N
$LASTSERVICECRITICAL$	Oui	Oui	Non	Non	Oui	N
$SERVICEOUTPUT$	Oui [2]	Oui	Non	Non	Oui	N
$SERVICEPERFDATA$	Oui [2]	Oui	Non	Non	Oui	N
$SERVICECHECKCOMMAND$	Oui	Oui	Non	Non	Oui	N
$SERVICEACKAUTHOR$	Non	Oui	Non	Non	Non	N
$SERVICEACKCOMMENT$	Non	Oui	Non	Non	Non	N
$SERVICEACTIONURL$	Oui	Oui	Non	Non	Oui	N
$SERVICENOTESURL$	Oui	Oui	Non	Non	Oui	N
$SERVICENOTES$	Oui	Oui	Non	Non	Oui	N
Macros récapitulatives :						
$TOTALHOSTSUP$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALHOSTSDOWN$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALHOSTSUNREACHABLE$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALHOSTSDOWNUNHANDLED$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALHOSTSUNREACHABLEUNHANDLED$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALHOSTPROBLEMS$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALHOSTPROBLEMSUNHANDLED$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALSERVICESOK$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALSERVICESWARNING$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALSERVICESCRITICAL$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALSERVICESUNKNOWN$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALSERVICESWARNINGUNHANDLED$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALSERVICESCRITICALUNHANDLED$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALSERVICESUNKNOWNUNHANDLED$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALSERVICEPROBLEMS$	Oui	Oui [4]	Oui	Oui [4]	Oui	O
$TOTALSERVICEPROBLEMSUNHANDLED$	Oui	Oui[4]	Oui	Oui[4]	Oui	O
Macros de notification :						
$NOTIFICATIONTYPE$	Non	Oui	Non	Oui	Non	N
$NOTIFICATIONNUMBER$	Non	Oui	Non	Oui	Non	N
Macros de contact :						
$CONTACTNAME$	Non	Oui	Non	Oui	Non	N

$CONTACTALIAS$	Non	Oui	Non	Oui	Non	N
$CONTACTEMAIL$	Non	Oui	Non	Oui	Non	N
$CONTACTPAGER$	Non	Oui	Non	Oui	Non	N
$CONTACTADDRESSn$	Non	Oui	Non	Oui	Non	N
Macros de date :						
$LONGDATETIME$	Oui	Oui	Oui	Oui	Oui	O
$SHORTDATETIME$	Oui	Oui	Oui	Oui	Oui	O
$DATE$	Oui	Oui	Oui	Oui	Oui	O
$TIME$	Oui	Oui	Oui	Oui	Oui	O
$TIMET$	Oui	Oui	Oui	Oui	Oui	O
Macros de fichier :						
$MAINCONFIGFILE$	Oui	Oui	Oui	Oui	Oui	O
$STATUSDATAFILE$	Oui	Oui	Oui	Oui	Oui	O
$COMMENTDATAFILE$	Oui	Oui	Oui	Oui	Oui	O
$DOWNTIMEDATAFILE$	Oui	Oui	Oui	Oui	Oui	O
$RETENTIONDATAFILE$	Oui	Oui	Oui	Oui	Oui	O
$OBJECTCACHEFILE$	Oui	Oui	Oui	Oui	Oui	O
$TEMPFILE$	Oui	Oui	Oui	Oui	Oui	O
$LOGFILE$	Oui	Oui	Oui	Oui	Oui	O
$RESOURCEFILE$	Oui	Oui	Oui	Oui	Oui	O
$COMMANDFILE$	Oui	Oui	Oui	Oui	Oui	O
$HOSTPERFDATAFILE$	Oui	Oui	Oui	Oui	Oui	O
$SERVICEPERFDATAFILE$	Oui	Oui	Oui	Oui	Oui	O
Macros diverses :						
$PROCESSSTARTTIME$	Oui	Oui	Oui	Oui	Oui	O
$ADMINEMAIL$	Oui	Oui	Oui	Oui	Oui	O
$ADMINPAGER$	Oui	Oui	Oui	Oui	Oui	O
$ARGn$	Oui	Oui	Oui	Oui	Oui	O
$USERn$	Oui	Oui	Oui	Oui	Oui	O

Description des macros

Macros d'hte : 3	
$HOSTNAME$	Nom court de l'hte (par exemple "biglinuxbox"). Cette valeur provient du paramétre *host_name* de la définition de l'hte.
$HOSTALIAS$	Nom long/description de l'hte. Cette valeur provient du paramétre *alias* de la définition de l'hte.
$HOSTADDRESS$	Adresse de l'hte. Cette valeur provient du paramétre *address* de la définition de l'hte.
$HOSTSTATE$	Une chane représentant l'état actuel de l'hte ("UP", "DOWN", ou "UNREACHABLE").
$HOSTSTATEID$	

	Un nombre correspondant é l'état actuel de l'hte : 0=UP, 1=DOWN, 2=UNREACHABLE.
$HOSTSTATETYPE$	Une chane indiquant le type d'état actuel du contrle d'hte ("HARD" ou "SOFT"). L'état est SOFT lorsque le contrle a renvoyé un état non-OK (non-UP) et va ’tre réessayé. L'état est HARD lorsque le contrle d'hte a été tenté le nombre de fois maximum défini.
$HOSTATTEMPT$	Le nombre d'essais de contrle d'hte actuel. Par exemple, si c'est la seconde fois que l'hte est contrlé, ce sera le nombre 2. Le nombre actuel d'essai n'est réellement utile que pour l'écriture de gestionnaires d'événements d'hte pour des états "SOFT", qui agiraient de maniére spécifique selon le nombre de tentatives.
$HOSTLATENCY$	Un nombre (réel) indiquant le retard en secondes par rapport au moment o un contrle d'hte *ordonnancé* devait ’tre exécuté. Par exemple, si un contrle était ordonnancé é 03:14:15 et qu'il n'a été exécuté qu'é 03:14:17, la latence de ce contrle est de 2.0 secondes. Les contrles d'htes é la demande ont une latence de zéro secondes.
$HOSTEXECUTIONTIME$	un nombre (réel) indiquant le nombre de secondes qu'a pris l'exécution du contrle d'hte (c.-é-d. le temps pendant lequel le contrle s'est exécuté).
$HOSTDURATION$	Une chane indiquant le temps qu'a passé l'hte dans sont état actuel. Le format est "XXh YYm ZZs", pour les heures, les minutes et les secondes.
$HOSTDURATIONSEC$	Un nombre indiquant le nombre de secondes qu'a passé l'hte dans sont état actuel.
$HOSTDOWNTIME$	Un nombre indiquant la "profondeur d'arr’t planifié" pour l'hte. Si cet hte est actuellement dans une période d'arr’t planifié, la valeur sera supérieure é zéro. Si l'hte n'est pas en cours d'arr’t planifié, la valeur sera zéro.
$HOSTPERCENTCHANGE$	Un nombre (réel) indiquant le pourcentage de changement d'état subit par l'hte. Le pourcentage de changement d'état est utilisé par l'algorithme de détection d'oscillation.
$HOSTGROUPNAME$	le nom court du groupe d'htes auquel appartient cet hte. Cette valeur provient du paramétre *hostgroup_name* de la définition de groupe d'htes. Si l'hte fait partie de plus d'un groupe d'hte cette macro contiendra le nom d'un seul de ceux-ci.
$HOSTGROUPALIAS$	Le nom long/alias du groupe d'htes auquel appartient cet hte. Cette valeur provient du paramétre *alias* de la définition de groupe d'htes. Si l'hte fait partie de plus d'un groupe d'hte cette macro contiendra l'alias d'un seul de ceux-ci.

$LASTHOSTCHECK$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o le dernier contrle d'hte a eu lieu.
$LASTHOSTSTATECHANGE$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o l'état de l'hte a changé pour la derniére fois.
$LASTHOSTUP$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o l'hte a été vu dans l'état UP pour la derniére fois.
$LASTHOSTDOWN$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o l'hte a été vu dans l'état DOWN pour la derniére fois.
$LASTHOSTUNREACHABLE$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o l'hte a été vu dans l'état UNREACHABLE pour la derniére fois.
$HOSTOUTPUT$	Le texte en sortie du dernier contrle d'hte (p. ex. "Ping OK").
$HOSTPERFDATA$	Cette macro contient les données de performance qui ont éventuellement été renvoyées par le dernier contrle d'hte.
$HOSTCHECKCOMMAND$	Cette macro contient le nom de la commande (ainsi que les paramétres qui ont pu lui ’tre passés) utilisée lors du contrle d'hte.
$HOSTACKAUTHOR$	Une chane contenant le nom de l'utilisateur qui a acquitté le probléme de l'hte. Cette macro n'est valable que dans les notifications dont la macro $NOTIFICATIONTYPE$ vaut "ACKNOWLEDGEMENT".
$HOSTACKCOMMENT$	Une chane contenant le commentaire associé é l'acquittement saisi par l'utilisateur qui a acquitté le probléme de l'hte. Cette macro n'est valable que dans les notifications dont la macro $NOTIFICATIONTYPE$ vaut "ACKNOWLEDGEMENT".
$HOSTACTIONURL$	URL d'action associée é l'hte. Cette valeur provient du paramétre *action_url* de la définition d'informations complémentaires de l'hte.
$HOSTNOTESURL$	URL de notes associée é l'hte. Cette valeur provient du paramétre *notes_url* de la définition d'informations complémentaires de l'hte.
$HOSTNOTES$	Notes associées é l'hte. Cette valeur provient du paramétre *notes* de la définition d'informations complémentaires de l'hte.
Macros de service :	
$SERVICEDESC$	Le nom long/description du service (p. ex. "Main Website"). Cette valeur provient du paramétre *description* de la défintion du service.

$SERVICESTATE$	Une chane indiquant l'état actuel du service ("OK", "WARNING", "UNKNOWN", ou "CRITICAL").
$SERVICESTATEID$	Un nombre qui correspond é l'état actuel du service : 0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN.
$SERVICESTATETYPE$	Une chane indiquant le type d'état actuel du contrle de service ("HARD" ou "SOFT"). L'état est SOFT quand le contrle de service renvoie un état non-OK state et va ’tre réessayé. L'état est HARD quand le contrle de service a été tenté le nombre de fois maximum défini.
$SERVICEATTEMPT$	Le nombre actuel d'essais de contrle du service. Par exemple, si c'est la seconde fois que le service est contrlé, ce sera le nombre deux. Le nombre actuel d'essai n'est réellement utile que pour l'écriture de gestionnaires d'événements d'hte pour des états "SOFT", qui agiraient de maniére spécifique selon le nombre de tentatives.
$SERVICELATENCY$	Un nombre (réel) indiquant le retard en secondes par rapport au moment o un contrle de service ordonnancé devait ’tre exécuté. Par exemple, si un contrle était ordonnancé é 03:14:15 et qu'il n'a été exécuté qu'é 03:14:17, la latence de ce contrle est de 2.0 secondes.
$SERVICEEXECUTIONTIME$	un nombre (réel) indiquant le nombre de secondes qu'a pris l'exécution du contrle de service (c.-é-d. le temps pendant lequel le contrle s'est exécuté).
$SERVICEDURATION$	Une chane indiquant le temps qu'a passé le service dans sont état actuel. Le format est "XXh YYm ZZs", pour les heures, les minutes et les secondes.
$SERVICEDURATIONSEC$	Un nombre indiquant le nombre de secondes qu'a passé le service dans sont état actuel.
$SERVICEDOWNTIME$	Un nombre indiquant la "profondeur d'arr’t planifié" pour le service. Si ce service est actuellement dans une période d'arr’t planifié, la valeur sera supérieure é zéro. Si le service n'est pas en cours d'arr’t planifié, la valeur sera zéro.
$SERVICEPERCENTCHANGE$	Un nombre (réel) indiquant le pourcentage de changement d'état subit par le service. Le pourcentage de changement d'état est utilisé par l'algorithme de détection d'oscillation.
$SERVICEGROUPNAME$	Le nom court du groupe de services auquel appartient cet service. Cette valeur provient du paramétre *servicegroup_name* de la définition de groupe de services. Si le service fait partie de plus d'un groupe de service cette macro contiendra le nom d'un seul de ceux-ci.
$SERVICEGROUPALIAS$	

	Le nom long/alias du groupe de services auquel appartient ce service. Cette valeur provient du paramétre *alias* de la définition de groupe de services. Si le service fait partie de plus d'un groupe de service cette macro contiendra l'alias d'un seul de ceux-ci.
$LASTSERVICECHECK$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o le dernier contrle de service a eu lieu.
$LASTSERVICESTATECHANGE$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o l'état du service a changé pour la derniére fois.
$LASTSERVICEOK$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o le service a été vu dans l'état OK pour la derniére fois.
$LASTSERVICEWARNING$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o le service a été vu dans l'état WARNING pour la derniére fois.
$LASTSERVICEUNKNOWN$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o le service a été vu dans l'état UNKNOWN pour la derniére fois.
$LASTSERVICECRITICAL$	C'est le moment au format time_t (secondes écoulées depuis l'époque UNIX) o le service a été vu dans l'état CRITICAL pour la derniére fois.
$SERVICEOUTPUT$	Le texte en sortie du dernier contrle de service (p. ex. "Ping OK").
$SERVICEPERFDATA$	Cette macro contient les données de performance qui ont pu ’tre renvoyées par le dernier contrle de service.
$SERVICECHECKCOMMAND$	Cette macro contient le nom de la commande (ainsi que les paramétres qui ont pu lui ’tre passés) utilisée lors du contrle de service.
$SERVICEACKAUTHOR$	Une chane contenant le nom de l'utilisateur qui a acquitté le probléme du service. Cette macro n'est valable que dans les notifications dont la macro $NOTIFICATIONTYPE$ vaut "ACKNOWLEDGEMENT".
$SERVICEACKCOMMENT$	Une chane contenant le commentaire associé é l'acquittement saisi par l'utilisateur qui a acquitté le probléme du service. Cette macro n'est valable que dans les notifications dont la macro $NOTIFICATIONTYPE$ vaut "ACKNOWLEDGEMENT".
$SERVICEACTIONURL$	URL d'action associée au service. Cette valeur provient du paramétre *action_url* de la définition d'informations complémentaires du service.
$SERVICENOTESURL$	URL de notes associée au service. Cette valeur provient du paramétre *notes_url* de la définition

	d'informations complémentaires du service.
$SERVICENOTES$	Notes associées au service. Cette valeur provient du paramétre *notes* de la définition d'informations complémentaires du service.
Macros de notification :	
$NOTIFICATIONTYPE$	Une chane représentant le type de notification envoyée ("PROBLEM", "RECOVERY", "ACKNOWLEDGEMENT", "FLAPPINGSTART" ou "FLAPPINGSTOP").
$NOTIFICATIONNUMBER$	Le nombre actuel de notifications émises pour le service ou l'hte. Le nombre de notifications augmente de un (1) é chaque nouvelle notification envoyée pour l'hte ou le service (exception faite des acquittements). Le nombre de notifications est remis é zéro quand l'hte ou le service revient é la normale (*aprés* que la notification de retour é la normal ait été envoyée). Les acquittements n'incrémentent pas le nombre de notifications.
Macros récapitulatives :	
$TOTALHOSTSUP$	Cette macro donne le nombre total d'htes qui sont actuellement dans l'état UP.
$TOTALHOSTSDOWN$	Cette macro donne le nombre total d'htes qui sont actuellement dans l'état DOWN.
$TOTALHOSTSUNREACHABLE$	Cette macro donne le nombre total d'htes qui sont actuellement dans l'état UNREACHABLE.
$TOTALHOSTSDOWNUNHANDLED$	Cette macro donne le nombre total d'htes qui sont actuellement dans l'état DOWN et qui ne sont pas en cours de traitement. Un probléme d'hte est dit non traité s'il n'est pas acquitté, ni en cours d'arr't planifié, et dont les contrles sont activés.
$TOTALHOSTSUNREACHABLEUNHANDLED$	Cette macro donne le nombre total d'htes qui sont actuellement dans l'état UNREACHABLE et qui ne sont pas en cours de traitement. Un probléme d'hte est dit non traité s'il n'est pas acquitté, ni en cours d'arr't planifié, et dont les contrles sont activés.
$TOTALHOSTPROBLEMS$	Cette macro donne le nombre total d'htes qui sont actuellement dans l'état DOWN ou UNREACHABLE.
$TOTALHOSTPROBLEMSUNHANDLED$	Cette macro donne le nombre total d'htes qui sont actuellement dans l'état DOWN ou UNREACHABLE et qui ne sont pas en cours de traitement. Un probléme d'hte est dit non traité s'il n'est pas acquitté, ni en cours d'arr't planifié, et dont les contrles sont activés.
$TOTALSERVICESOK$	Cette macro donne le nombre total de services qui sont actuellement dans l'état OK.
$TOTALSERVICESWARNING$	

	Cette macro donne le nombre total de services qui sont actuellement dans l'état WARNING.
$TOTALSERVICESCRITICAL$	Cette macro donne le nombre total de services qui sont actuellement dans l'état CRITICAL.
$TOTALSERVICESUNKNOWN$	Cette macro donne le nombre total de services qui sont actuellement dans l'état UNKNOWN.
$TOTALSERVICESWARNINGUNHANDLED$	Cette macro donne le nombre total de services qui sont actuellement dans l'état WARNING et qui ne sont pas en cours de traitement. Un probléme de service est dit non traité s'il n'est pas acquitté, ni en cours d'arr't planifié, et dont les contrles sont activés.
$TOTALSERVICESCRITICALUNHANDLED$	Cette macro donne le nombre total de services qui sont actuellement dans l'état CRITICAL et qui ne sont pas en cours de traitement. Un probléme de service est dit non traité s'il n'est pas acquitté, ni en cours d'arr't planifié, et dont les contrles sont activés.
$TOTALSERVICESUNKNOWNUNHANDLED$	Cette macro donne le nombre total de services qui sont actuellement dans l'état UNKNOWN et qui ne sont pas en cours de traitement. Un probléme de service est dit non traité s'il n'est pas acquitté, ni en cours d'arr't planifié, et dont les contrles sont activés.
$TOTALSERVICEPROBLEMS$	Cette macro donne le nombre total de services qui sont actuellement dans l'état WARNING, CRITICAL, ou UNKNOWN.
$TOTALSERVICEPROBLEMSUNHANDLED$	Cette macro donne le nombre total de services qui sont actuellement dans l'état WARNING, CRITICAL, ou UNKNOWN et qui ne sont pas en cours de traitement. Un probléme de service est dit non traité s'il n'est pas acquitté, ni en cours d'arr't planifié, et dont les contrles sont activés.
Macros de contact :	
$CONTACTNAME$	Nom court du contact (p. ex. "jdoe") qui est notifié pour un probléme d'hte ou de service. Cette valeur provient du paramétre *contact_name* de la définition de contact.
$CONTACTALIAS$	Nom long/description du contact (p. ex. "John Doe") qui est notifié. Cette valeur provient du paramétre *alias* de la définition de contact.
$CONTACTEMAIL$	Adresse email du contact qui est notifié. Cette valeur provient du paramétre *email* de la définition de contact.
$CONTACTPAGER$	Numéro/adresse du Pager du contact qui est notifié. Cette valeur provient du paramétre *pager* de la définition de contact.
$CONTACTADDRESSn$	

	Adresse du contact qui est notifié. Chaque contact peut avoir six adresses différentes (en plus de son adresse email et de son numéro de pager). Les macros correspondant é ces adresses sont $CONTACTADDRESS1$ - $CONTACTADDRESS6$. Cette valeur provient du paramétre *addressx* de la définition de contact.
Macros de date :	
$LONGDATETIME$	Date et heure actuelles (p. ex. *Fri Oct 13 00:30:28 CDT 2000*). Le format de la date est défini par le paramétre date_format.
$SHORTDATETIME$	Date et heure actuelles (p. ex. *10-13-2000 00:30:28*). Le format de la date est défini par le paramétre date_format.
$DATE$	Date du jour (p. ex. *10-13-2000*). Le format de la date est défini par le paramétre date_format.
$TIME$	Heure actuelle (p. ex. *00:30:28*).
$TIMET$	Moment actuel au format time_t (secondes écoulées depuis l'époque UNIX).
Macros de fichier :	
$MAINCONFIGFILE$	Emplacement du fichier de configuration principal.
$STATUSDATAFILE$	Emplacement du journal des états.
$COMMENTDATAFILE$	Emplacement du fichier de commentaire.
$DOWNTIMEDATAFILE$	Emplacement du fichier des arr'ts planifiés.
$RETENTIONDATAFILE$	Emplacement du fichier de mémorisation des états.
$OBJECTCACHEFILE$	Emplacement du fichier de cache des objets.
$TEMPFILE$	Emplacement du fichier temporaire.
$LOGFILE$	Emplacement du fichier journal.
$RESOURCEFILE$	Emplacement du fichier de ressources.
$COMMANDFILE$	Emplacement du fichier de commandes externes.
$HOSTPERFDATAFILE$	Emplacement du fichier de performance des htes (s'il est défini).
$SERVICEPERFDATAFILE$	Emplacement du fichier de performance des services (s'il est défini).
Macros diverses :	
$PROCESSSTARTTIME$	Moment au format time_t (secondes écoulées depuis l'époque UNIX) o le processus Nagios a été (re)démarré pour la derniére fois. Vous pouvez en déduire le nombre de secondes écoulées depuis le démarrage (ou redémarrage) de Nagios, en soustrayant $PROCESSSTARTTIME$ de $TIMET$.
$ADMINEMAIL$	Adresse email globale de l'administrateur. Cette valeur provient du paramétre admin_email.
$ADMINPAGER$	Numéro/adresse global du pager de l'administrateur. Cette valeur provient du

	paramétre admin_pager.
$ARGn$	Le *n*iéme paramétre passé é la commande (notification, gestionnaire d'événement, contrle de service, etc.). Nagios gére jusqu'é 32 macros de paramétre (de $ARG1$ é $ARG32$).
$USERn$	La *n*iéme macro définie par l'utilisateur. Les macros utilisateur peuvent ’tre définies dans un ou plusieurs fichiers de ressources. Nagios gére jusqu'é 32 macros utilisateur (de $USER1$ é $USER32$).

Notes

[1] Ces macros ne sont pas valables pour l'hte auquel elles sont associées lorsque cet hte est en cours de contrle (c.-é-d. qu'elles n'ont pas de sens, car elles ne sont pas encore déterminées).

[2] Ces macros ne sont pas valables pour le service auquel elles sont associées lorsque ce service est en cours de contrle (c.-é-d. qu'elles n'ont pas de sens, car elles ne sont pas encore déterminées).

[3] Quand des macros d'hte sont utilisées dans des commandes relatives é un service (c.-é-d. les notifications de service, les gestionnaires d'événements, etc.) elles font référence é l'hte auquel est associé le service.

[4] Quand les macros récapitulatives d'hte et de service sont utilisées dans des commandes de notification, les totaux sont filtrés pour ne prendre en compte que les htes et les services autorisés é ce contact (c.-é-d. les htes et les services configurés pour lui envoyer des notifications).

Information sur les CGI

Introduction

Voici une bréve description des CGI fournis avec Nagios, ainsi que des autorisations requises pour accéder et utiliser ces CGI.

Important : Par défaut, les CGI ne fonctionnent que si vous vous ’tes authentifié auprés du serveur web et que vous ’tes autorisé é accéder aux informations demandées. Pour plus d'informations sur la configuration des autorisations de votre serveur web et des CGI, lisez les sections Installer l'interface web et Authentification et autorisations dans les CGI.

Index

CGI d'état

status.cgi

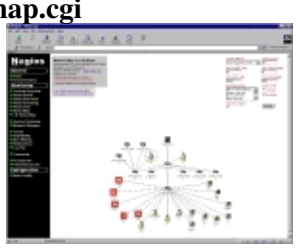
C'est le CGI le plus important de Nagios. Il vous permet de voir l'état de tous les htes et de tous les services supervisés. Le CGI d'état peut produire deux types principaux d'affichage - un aperéu de l'état de tous les groupes d'htes (ou d'un groupe particulier) et une vue détaillée de tous les services (ou de ceux associés é un hte particulier). Vous pouvez associer de jolies icnes aux htes en utilisant les options d'informations complémentaires sur les htes du fichier de configuration.

Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir tous les htes **et** tous les services.
- Si vous ’tes autorisé pour tous les services vous pouvez voir tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez voir tous les htes et tous les services dont vous ’tes un contact.

CGI de cartographie des états

statusmap.cgi



Ce CGI crée une carte de tous les htes que vous avez défini dans votre réseau. Il utilise la bibliothéque gd de Thomas Boutell (version 1.6.3 ou plus) pour créer une image au format PNG de l'agencement de votre réseau. Les coordonnées utilisées pour dessiner chaque hte (ainsi que les jolies icnes facultatives) sont retirées des options d'informations complémentaires sur les htes du fichier de configuration. Si vous ne pouvez pas accéder é ce CGI, ou si vous avez des erreurs lors de sa compilation ou de son exécution, lisez cette FAQ.

Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir tous les htes.
- Si vous ’tes un *contact authentifié* vous pouvez voir les htes dont vous ’tes un contact.

Note : Les utilisateurs qui ne sont pas autorisés é voir un hte particulier verront un noeud nommé *unknown* é la place. Je suis conscient qu'ils ne devraient *rien voir du tout*, mais éa n'a aucun sens de générer la carte si vous ne pouvez pas voir les dépendances entre les htes

CGI d'interface WAP

statuswml.cgi

Ce CGI gére l'interface WAP d'accés aux informations sur l'état du réseau. Si vous avez un appareil WAP (i.e. un téléphone portable compatible Internet), vous pouvez accéder aux informations d'état alors que vous ’tes en déplacement. Les vues disponibles comprennent le résumé par groupe d'htes,

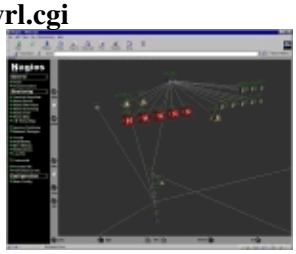
l'aperéu par groupe d'htes, le détail d'un hte, le détail d'un service, tous les problémes, et les problémes non pris en compte. Vous pouvez de plus désactiver les notifications et les contrles et acquitter des problémes depuis votre portable. Plutt cool, non ?

Autorisation requises :

- Si vous ’tes autorisé pour les informations systéme vous pouvez voir les informations sur le processus de Nagios.
- Si vous ’tes autorisé pour tous les htes vous pouvez voir les données d'état de tous les htes **et** services.
- Si vous ’tes autorisé pour tous les services vous pouvez voir les données d'état de tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez voir les données d'état de tous les htes et services dont vous ’tes un contact.

CGI du monde des états (VRML)

statuswrl.cgi



Ce CGI crée une modélisation en 3D utilisant le langage VRML de tous les htes définis dans votre réseau. Les coordonnées utilisées pour dessiner les htes (ainsi que les jolies textures) sont définies en utilisant les options d'informations complémentaires sur les htes du fichier de configuration des CGI. Il vous faudra un navigateur incluant le composant VRML (comme Cortona) pour pouvoir visualiser ce modéle.

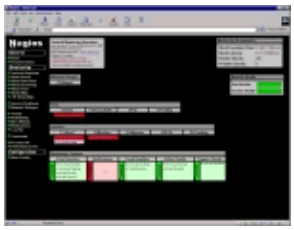
Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir tous les htes.
- Si vous ’tes un *contact authentifié* vous pouvez voir les htes dont vous ’tes un contact.

Note : Les utilisateurs qui ne sont pas autorisés é voir un hte particulier verront un noeud nommé *unknown* é la place. Je suis conscient qu'ils ne devraient *rien voir du tout*, mais éa n'a aucun sens de générer la carte si vous ne pouvez pas voir les dépendances entre les htes

CGI d'aperéu tactique

tac.cgi



Ce CGI vous donne une vue générale de toute l'activité de supervision du réseau. Il vous permet de repérer rapidement les indisponibilités du réseau, l'état des htes et des services. Il distingue les problémes qui ont été traités d'une faéon ou d'une autre (i.e. qui ont été acquittés, dont les notifications sont désactivées, etc.) et ceux qui n'ont pas été traités, et qui donc méritent attention. Trés utile si vous avez beaucoup d'htes/services é superviser et que vous ne voulez consulter qu'un seul écran pour ’tre averti des problémes.

Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir tous les htes **et** services.
- Si vous ’tes autorisé pour tous les services vous pouvez voir tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez voir tous les htes et services dont vous ’tes un contact.

CGI d'indisponibilité du réseau

outages.cgi

Ce CGI produit une liste des htes "é problémes" de votre réseau qui sont responsables des ruptures de lien. Ceci est particuliérement utile sur les grands réseaux pour identifier rapidement la cause d'un probléme. Les htes sont triés selon la gravité de la rupture dont ils sont responsables. Pour plus d'informations sur le fonctionnement de ce CGI, voyez ici

Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir tous les htes.
- Si vous ’tes un *contact authentifié* vous pouvez voir les htes dont vous ’tes un contact.

CGI de configuration

config.cgi

Ce CGI permet de visualiser les définitions d'objets (c.a.d htes, groupes d'htes, contacts, groupes de contacts, périodes, services et commandes) tels que spécifiés dans votre (vos) fichier(s) de configuration des objets.

Autorisation requises :

- Vous devez avoir l'autorisation d'accés aux informations de configuration pour voir les définitions de contacts, groupes de contacts, groupes d'htes, périodes, et commandes. Vous verrez aussi toutes les définitions d'htes et de services.

CGI de commande

cmd.cgi

Ce CGI permet d'envoyer des commandes au processus de Nagios. Bien que ce CGI accepte plusieurs arguments, mieux vaut ne pas s'y essayer : la plupart changent selon les révisions de Nagios. Utilisez plutt les informations complémentaires des CGI comme point de départ pour envoyer des commandes.

Autorisation requises :

- Vous devez ’tre autorisé pour les commandes systéme pour envoyer des commandes affectant le processus de Nagios (redémarrage, arr’t, changement de mode, etc.).
- Si vous ’tes autorisé pour toutes les commandes aux htes vous pouvez envoyer des commandes é tous les htes **et** é tous les services.
- Si vous ’tes autorisé pour toutes les commandes aux services vous pouvez envoyer des commandes é tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez envoyer des commandes é tous les htes et services dont vous ’tes un contact.

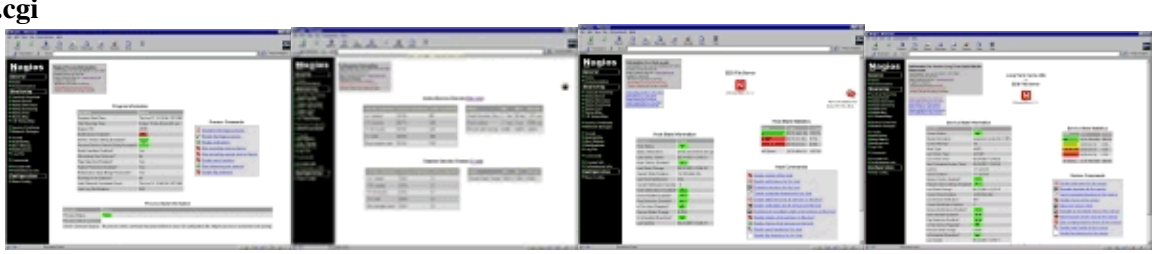
Notes :

- Si vous avez choisi de ne pas utiliser l'authentification dans les CGI, ce CGI ne permettra é *personne* d'envoyer des commandes é Nagios, et ce dans votre propre intér’t. Je suggére de plus dans ce cas de supprimer le CGI.

- Pour que ce CGI envoie effectivement ses commandes é Nagios, vous devrez mettre les bons droits d'accés aux fichiers et aux répertoires ainsi que le décrit cette FAQ.

CGI d'informations complémentaires

extinfo.cgi



Ce CGI permet de voir les informations relatives au processus de Nagios, aux statistiques sur les htes et les services, aux commentaires sur les htes et les services, et plus encore. C'est également un point d'entrée pour envoyer des commandes é Nagios via le CGI de commande. Bien que ce CGI accepte plusieurs arguments, mieux vaut ne pas s'y essayer : la plupart changent selon les révisions de Nagios. Vous pouvez accéder é ce CGI en cliquant sur les liens appelés 'Network Health' [**NdT**: Santé/Etat du réseau] et 'Process Information' [**NdT**: Information sur le processus Nagios] de la barre latérale de navigation, ou en cliquant sur un lien hte ou service dans l'affichage du CGI d'état.

Autorisation requises :

- Vous devez ’tre autorisé pour les informations systéme pour voir les informations sur le processus de Nagios.
- Si vous ’tes autorisé pour tous les htes vous pouvez voir les informations complémentaires de tous les htes **et** tous les services.
- Si vous ’tes autorisé pour tous les services vous pouvez voir les informations complémentaires de tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez voir les informations complémentaires de tous les htes et de tous les services dont vous ’tes un contact.

CGI du fichier journal

showlog.cgi

Ce CGI affiche le fichier journal. Si vous avez activé la rotation du journal, vous pouvez voir le contenu des journaux archivés en utilisant les liens de navigation situés en haut de la page.

Autorisation requises :

- Vous devez ’tre autorisé pour les informations systéme pour voir le fichier journal.

CGI d'historique d'alerte

history.cgi

Ce CGI affiche l'historique des problémes relatifs soit é un hte particulier soit é tous les htes. L'affichage est un sous-ensemble de ce que produit le CGI du fichier journal. Vous pouvez filtrer l'affichage pour n'obtenir que certains types de problémes (i.e. alertes hard et/ou soft, les différents types d'alertes sur les services et les htes, tous les types d'alertes, etc.). Si vous avez activé la rotation du journal, vous pouvez voir l'historique contenu dans les journaux archivés en utilisant les liens de navigation situés en haut de la page.

Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir l'historique de tous les htes **et** de tous les services.
- Si vous ’tes autorisé pour tous les services vous pouvez voir l'historique de tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez voir l'historique de tous les htes et tous les services dont vous ’tes un contact.

CGI des notifications

notifications.cgi

Ce CGI affiche les notifications envoyées aux différents contacts d'un hte ou d'un service. L'affichage est un sous-ensemble de ce que produit le CGI du fichier journal. Vous pouvez filtrer l'affichage pour n'obtenir que certains types de problémes (i.e. notifications relatives aux services, aux htes, envoyées é des contacts spécifiques, etc). Si vous avez activé la rotation du journal, vous pouvez voir les notifications contenues dans les journaux archivés en utilisant les liens de navigation situés en haut de la page.

Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir les notifications relatives é tous les htes **et** tous les services.
- Si vous ’tes autorisé pour tous les services vous pouvez voir les notifications relatives é tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez voir les notifications relatives é tous les htes et tous les services dont vous ’tes un contact.

CGI des tendances

trends.cgi

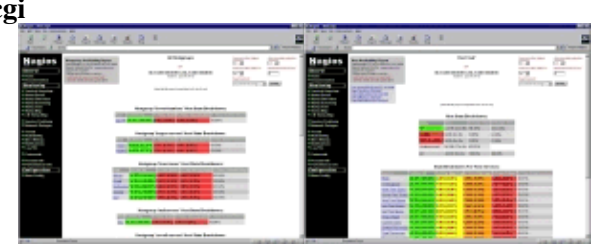
Ce CGI crée un graphique des états des htes et des services sur une période donnée. Pour que ce CGI soit pertinent, vous devez activer la rotation du journal et archiver les journaux dans le répertoire défini par la variable log_archive_path. Ce CGI utilise la bibliothéque gd de Thomas Boutell (version 1.6.3 ou plus) pour créer l'image des tendances. Si vous ne pouvez pas accéder é ce CGI, ou si vous avez des erreurs lors de sa compilation ou de son exécution, lisez cette FAQ.

Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir les tendances de tous les htes **et** de tous les services.
- Si vous ’tes autorisé pour tous les services vous pouvez voir les tendances de tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez voir les tendances de tous les htes et tous les services dont vous ’tes un contact.

CGI de rapport de disponibilité

avail.cgi



Ce CGI permet de créer un rapport sur la disponibilité des htes et des services sur une période donnée. Pour que ce CGI soit pertinent, vous devez activer la rotation du journal et archiver les journaux dans le répertoire défini par la variable log_archive_path.

Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir la disponibilité de tous les htes **et** services.
- Si vous ’tes autorisé pour tous les services vous pouvez voir la disponibilité de tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez voir la disponibilité de tous les services et htes dont vous ’tes un contact.

CGI d'histogramme des alertes

histogram.cgi

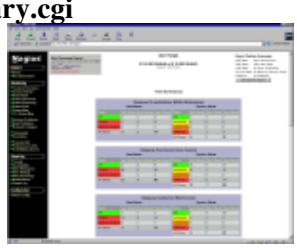
Ce CGI permet d'afficher l'histogramme de disponibilité d'htes et services sur une période de temps. Pour que ce CGI soit pertinent, vous devez activer la rotation du journal et archiver les journaux dans le répertoire défini par la variable log_archive_path. Ce CGI utilise la bibliothéque gd de Thomas Boutell (version 1.6.3 ou plus) pour créer l'image des tendances. Si vous ne pouvez pas accéder é ce CGI, ou si vous avez des erreurs lors de sa compilation ou de son exécution, lisez cette FAQ.

Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir la disponibilité de tous les htes **et** services.
- Si vous ’tes autorisé pour tous les services vous pouvez voir la disponibilité de tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez voir la disponibilité de tous les services et htes dont vous ’tes un contact

CGI du récapitulatif des alertes

summary.cgi



Ce CGI fournit des rapports synthétiques sur les alertes concernant les htes et services, ainsi que le nombre total d'alertes, les services/htes générant le plus d'alertes, etc.

Autorisation requises :

- Si vous ’tes autorisé pour tous les htes vous pouvez voir la disponibilité de tous les htes **et** services.
- Si vous ’tes autorisé pour tous les services vous pouvez voir la disponibilité de tous les services.
- Si vous ’tes un *contact authentifié* vous pouvez voir la disponibilité de tous les services et htes dont vous ’tes un contact.

Personnalisation de l'en-t’te et du pied de page des CGI

Introduction

Si vous ’tes amené é faire des installations personnalisées de Nagios, vous pourriez avoir envie d'avoir un en-t’te et/ou un pied de page personnalisés é l'affichage des résultats des CGI. Ceci est particuliérement utile pour l'utilisateur final concernant l'affichage des informations de contact du support, etc.

Il est important de tenir compte du fait que, é moins d''tre exécutables, les en-t’tes et pieds de page personnalisés ne sont **pas** exécutés de quelle maniére que ce soit avant d''tre affichés. Le contenu du fichier d'en-t’te et de pied de page est simplement lu et affiché au résultat de l'exécution de la CGI. Cela signifie qu'ils ne peuvent contenir que des informations qu'un navigateur web peut comprendre (HTML, JavaScript, etc.).

Si les en-t’tes et pieds de page personnalisés sont exécutables, alors les fichiers sont exécutés et leur sortie est affichée é l'utilisateur : ils doivent donc afficher du code HTML valide. Cette fonctionnalité permet d'utiliser vos propres CGI pour ajouter des données é l'affichage de Nagios. Ceci a été utilisé pour insérer des graphiques de rrdtool en utilisant ddraw et des menu de commande dans la fen’tre d'affichage de Nagios. Ces en-t’tes et pieds de page exécutables disposent du m’me environnement que les CGI natives : vous pouvez donc disposer des informations sur la requ’te HTTP, sur l'utilisateur authentifié etc pour afficher les informations adéquates.

Comment éa marche?

Vous pouvez inclure les en-t’tes et pieds de page personnalisés dans le résultat des CGI en fournissant des fichiers HTML avec un nom approprié dans le sous-répertoire *ssi/* du répertoire HTML de Nagios (p.ex. */usr/local/nagios/share/ssi*).

Les en-t’tes personnalisés sont inclus immédiatement aprés la balise **<BODY>** du résultat de l'exécution de la CGI. Pareillement, les pieds de page personnalisés sont inclus avant la balise de fermeture **</BODY>**.

Il y a deux types d'en-t’tes et de pied de page personnalisés:

- **En-t’tes/pieds de page globaux**. Ces fichiers devraient ’tre nommés *common-header.ssi* et *common-footer.ssi*, respectivement. Si ces fichiers existent, ils seront inclus dans le résultat de l'exécution de toutes les CGI.
- **En-t’tes/pieds de page de CGI spécifiques**. Ces fichiers devraient ’tre nommés dans le format *NOMCGI-header.ssi* et *NOMCGI-footer.ssi*, o *CGINAME* est le nom physique du CGI sans l'extension .cgi. Par exemple, l'en-t’te et le pied de page pour le CGI résumé des alertes [alert summary] (summary.cgi) devrait ’tre appelé *summary-header.ssi* et *summary-footer.ssi*, respectivement.

L'utilisation des en-t’tes et pied de pages personnalisés sont optionnels. Utilisez-les si vous le désirez. De m’me pour les en-t’tes et pieds de page spécifiques.