MAKINA CORPUS

Introduction à la supervision dans l'Univers Nagios

Version 1.0 Octobre 2011

Regis Leroy

1. Sommaire

2. Makina Corpus

Makina Corpus conçoit, développe et intègre des solutions innovantes s'appuyant exclusivement sur des logiciels libres.

La société a développé une expertise particulière en applications web complexes, dans le domaine des portails, le traitement de données géographiques (SIG) et l'analyse décisionnelle (Business Intelligence).

Makina Corpus intervient sur :

- une expertise technologique de haut niveau ;
- l'intégration d'applications hétérogènes ;
- une offre sur mesure, conçue et développée en interne pour répondre spécifiquement à vos besoins ;
- une réduction du coût d'acquisition logiciel, grâce à l'utilisation de logiciels du monde libre ;
- un service complet clés en main.

2.1. Nos domaines de compétences

Makina Corpus propose des solutions innovantes pour :

- la gestion de contenus (CMS, intranet, extranet, internet) ;
- la géomatique ou SIG (webmapping et clients lourds) ;
- les outils décisionnels (traitement de données, analyses, reporting).

2.2. Notre philosophie

Makina Corpus croit fermement aux valeurs d'ouverture et de partage du logiciel libre et s'implique comme ardent promoteur et contributeur de la communauté du libre.

La valeur ajoutée est forte et concrète pour nos clients : les logiciels libres leur garantissent en effet une totale indépendance par rapport à leurs fournisseurs, réduisent les coûts, et assurent une grande pérennité.

2.3. L'équipe projet et technique : moyens humains

Notre équipe est faite d'hommes et de femmes ayant chacun une individualité. L'entreprise reconnaît et valorise la diversité de tous ces talents, et en exploite au mieux les complémentarités. Dans cette démarche de respect des choix individuels, Makina Corpus a choisi d'être présente sur de nombreuses implantations géographiques et d'accepter le télé-travail.

Pour une nouvelle approche de la relation client-fournisseur, Makina Corpus met en œuvre la méthode agile pragmatique et pro-active **SCRUM**. Les échanges fréquents et programmés entre le client et l'équipe de développement permettent d'identifier et corriger au plus tôt d'éventuels problèmes et surtout de répondre au mieux aux **besoins réels du client**.

2.4. Nos clients

Makina Corpus a réalisé des prestations web pour de nombreuses collectivités territoriales :

Nantes, Angers, Cannes,Cap Atlantique, Lille , Sevran, Challans, La Montagne, Aigrefeuille... Pays des Vals de Saintonge, Arrondissement de Briey, Pays-Haut - Val d'Alzette, Pays du Gard Rhodanien. Conseils Généraux : Vaucluse, Bas-Rhin, Essonne. Conseils Régionaux : Aquitaine, Picardie, Communauté francophone (Belgique)

Dans le domaine des portails web et des intranets, Makina Corpus est également intervenue auprès :

- du Ministère des Affaires Étrangères, de la Préfecture d'Île et Vilaine, de l'Aéroport Nantes-Atlantique et de l'Association des Aéroports Francophones, des Chambres d'agriculture, et de la SAFER de Haute Normandie, de l'Établissement Public Territorial du Fleuve Charente, et de l'Association des Établissements Publics Territoriaux de Bassin, de l'Union Sociale de l'Habitat, de l'INSEE, de l'Agence Nationale des Fréquences, de l'OTAN et des Nations Unies, de la Commission Européenne....

3. Licence

Vous pouvez contribuer à ce document nous signalant les erreurs, en apportant vos remarques, ajouts et commentaires.

Les dessins et schémas sont normalement disponibles avec les sources et utilisables dans le logiciel dia.

Les fichiers source sont disponibles sur https://github.com/regilero/Introduction-a-la-Supervision-dans-l-Univers-Nagios

Retrouvez les livres blancs Makina Corpus et les contributions diverses sur http://makina-corpus.com et http://makina-corpus.org.

4. Réutilisation du contenu de ce livre

Ce livre est soumis à la licence détaillée dans le chapitre 3.Licence.

Cependant, afin de pouvoir réutiliser ce livre blanc dans vos démarches commerciales, nous autorisons :

- la suppression du chapitre 2.Makina Corpus et du chapitre 4.Réutilisation du contenu de ce livre
- l'intégration de l'intégralité du livre blanc (hormis le chapitre 2.Makina Corpus et 4.Réutilisation du contenu de ce livre) dans un modèle de document différent de celui de Makina Corpus.

Il est interdit de retirer les références aux auteurs et à Makina Corpus du chapitre 3.Licence, ou de modifier le contenu livre blanc (hors retrait du chapitre 2.Makina Corpus et chapitre 4.Réutilisation du contenu de ce livre).

5. Auteur

Régis Leroy (aka regilero)<regis.leroy@makina-corpus.com>

Administrateur Système, développeur web, formateur Nagios et Centreon (entre autres) depuis plusieurs années. Son domaine d'intervention principal, les architectures web complexes, l'a amené à former de nombreuses personnes aux outils de supervision libres.

6. Objectifs de ce livre blanc

Nagios est un logiciel Open Source de Supervision qui existe depuis 1999. Depuis quelques années de nombreuses applications complémentaires ou parallèles sont venues se greffer à l'univers de la supervision Nagios. On pourra citer par exemple Centreon, Shinken, Icinga, Nagios XI, Mk_multisite, etc. La communauté Nagios est en effervescence, les projets se multiplient. Certains sont complémentaires, d'autres concurrents. Les grands principes de la supervision régis par Nagios sont cependant toujours présents au cœur de tous ces produits.

En tant que formateur Nagios, je croise très souvent des équipes chargées de la supervision réseau, la principale difficulté que rencontrent ces équipes est très souvent l'absence de vision globale du fonctionnement de la supervision avec Nagios et des différentes possibilités qu'offre l'outil.

Le but de ce document est de fournir en quelques pages les bases théoriques du fonctionnement de la supervision avec les outils du monde Nagios, ce qui comprend :

- le vocabulaire parfois mystérieux ;
- l'importance de la topologie ;

- le fonctionnement des sondes ;
- et une vision d'ensemble des possibilités infinies qu'offre cet outil.

Une fois ces bases acquises, il est beaucoup plus simple de comprendre les discussions des forums, les problèmes auxquels nous sommes parfois confrontés, ou de savoir orienter ses questions afin d'obtenir de l'aide.

7. Nagios?

7.1. Pourquoi Superviser ?

Nagios est un logiciel de Supervision Open Source.

La supervision est un élément clef de l'administration des systèmes et réseaux.

Les grands principes de la supervision se résument en quelques mots. Face aux incidents qui peuvent intervenir dans un système d'information, le but de la supervision est de **prévoir**, d'**alerter**, de **comprendre, d'informer** puis d'**agir**.

La mise en place d'une supervision se fait donc :

- pour prévoir les pannes avant qu'elles ne surviennent ;
- pour détecter les pannes quand elles surviennent et avant qu'elles ne déclenchent des effets en cascade ;
- pour prévenir la bonne personne en fonction des incidents ;
- pour isoler rapidement les causes d'une panne ;
- pour disposer rapidement des informations permettant de résoudre la crise ;
- pour intervenir rapidement au bon endroit et identifier les services impactés par une crise.

Pour cela, il va falloir :

- surveiller plusieurs éléments ;
- établir des graphiques de tendances pour certains d'entre eux ;
- gérer les personnes et les équipements présents ;
- identifier les points critiques pour le système d'information.

L'une des choses que l'on ignore le plus souvent dans la mise en place d'une supervision est que les systèmes de supervisions ne sont pas magiques. Il n'est pas raisonnable d'investir un temps limité avec un stagiaire, de mettre en place un outil de supervision, puis de le laisser vivre sa vie.

La supervision suppose un cycle métier. Il faudra maintenir le système de supervision au fur et à mesure de l'évolution du réseau, des services mis en place dans le SI, des mouvements de personnels.

En terme de méthode, la supervision, une fois l'outil en place, commencera par un recensement des besoins.

Que veut-on ajouter à notre système de supervision?

Quelle est la partie la plus critique et qui pour l'instant n'est pas mise en place?

Il faudra ensuite imaginer les moyens de superviser cette partie, tester cette supervision sur un sous-ensemble du SI, puis la déployer à l'ensemble des machines.

Puis, il faudra le maintenir et valider qu'il fonctionne toujours.

L'un des buts de ce livre blanc est de permettre une meilleure compréhension du fonctionnement des outils libres de supervision afin que les administrateurs puissent facilement intégrer ces derniers dans leur façon de travailler quotidiennement avec le SI, et non comme une grande brique mystérieuse qui envoie des mails qu'on ne lit plus depuis longtemps.

Si vous avez déjà un système de supervision en place ou si vous prévoyez d'en mettre un en place, je vous livre ici une première liste des choses que vous pourriez envisager d'ajouter à votre supervision :

- des brins réseaux qui tombent ;
- des disques qui se remplissent ;
- des services qui ne répondent plus ;
- des processus qui disparaissent ou se multiplient ;
- des machines qui s'emballent ;
- des systèmes DNS et de reverse DNS à jour ;
- des intrusions , des virus ;
- des systèmes qui ne sont pas à jour ;
- des machines qui ne sont pas à l'heure ;
- des tendances (ex.: ce site web prend de plus en plus de mémoire) ;
- des logs ;
- des backups ;
- des boucles complètes (envoyer un message smtp puis le lire en IMAP et en POP).

7.2. Bref historique de Nagios

Nagios est un logiciel Open Source publié sous Licence GNU GPL.

Comme tout logiciel libre, la licence associée à ce logiciel vous offre quatre libertés fondamentales :

- liberté d'étudier les sources ;
- liberté de modifier les sources ;
- liberté de redistribuer les sources ;

- liberté d'utiliser les sources.

Mais il s'agit d'une licence fixée, tout changement doit rester couvert par la licence.

La licence ne traite pas non plus de concepts liés à la gestion de projet, comme la gestion d'une communauté, les marques associées (Nagios est une marque déposée par Ethan Galstad) ou la politique de gestion des contributions et des rapports d'erreurs.

Si vous n'êtes pas un habitué des logiciels libres retenez cependant une chose, dans le monde des logiciels, il n'est pas du tout mal vu de faire des rapports de bugs.

Si vous trouvez un bug, que vous n'aimez pas un principe d'ergonomie, si vous pensez qu'il manque une fonctionnalité, qu'un mot est mal traduit, alors prenez un peu de votre temps et participez à la communauté en râlant (mais faites-le de façon constructive).

La créateur de Nagios se nomme Ethan Galstad. La première version du logiciel se nommait NetSaint et date de 1999.

En 2002 NetSaint devenait Nagios suite à un problème de marque commerciale. Nagios signifiant alors « **Nagios Ain't Gonna Insist on Sainthood** », référence justement au problème de nom sur NetSaint.

La version 1.0 date de Novembre 2002. La version 2.0 de juillet 2006 et la version 3.0 de mars 2008. Mais chacune de ces branches dispose aussi de version intermédiaires régulières. La version stable que vous pourrez trouver au jour où vous lirez ces lignes ne sera pas la 3.0 mais, par exemple, la 3.3.1.

Nous avons déjà cité en introduction quelques nouveaux noms intervenus dans la communauté Nagios depuis 2009. Nous ne détaillerons donc pas dans ce document les cartes stratégiques de différents projets en compétition ni leurs objectifs.

Attaquons le cœur du sujet, les concepts de fonctionnement de Nagios.

8. Les entités manipulées par Nagios

8.1. Hôtes et services

Les deux principales entités rencontrées sur les écrans de supervision de Nagios sont les hôtes *(Hosts* en anglais*)* et services (*services* en anglais). Nagios fait donc une distinction importante entre ce qu'on peut assimiler dans un premier temps à des machines et les services hébergés sur ces machines.

Un service est toujours associé à un hôte, et celui-ci peut bien sur héberger plusieurs services.

Les services regroupent l'essentiel des tâches de supervision de Nagios. Les hôtes, quant à eux, contiennent beaucoup moins de paramètres de configuration.

De plus, Nagios ne s'intéresse à la supervision des hôtes que si la supervision des services rapporte des erreurs. Il est donc courant de négliger l'importance des hôtes dans une installation de supervision. Cependant, un examen attentif de ce que représentent les hôtes pour Nagios permet de soulever de nombreuses questions ayant un impact important pour le fonctionnement et le positionnement du serveur de supervision.

8.2. Les hôtes

8.2.1. Qu'est-ce qu'un hôte?

Pour Nagios, un hôte est principalement une adresse IP. Il est possible et conseillé d'associer à cette adresse IP des noms, y compris des noms DNS.

Les trois principales informations dont vous aurez besoin pour un hôte seront :

- son adresse IP. Exemple: **192.168.0.15** ;
- son nom complet (FQDN: fully qualified domain name). Exemple : **dserv003_rhel5.dmz.example.net** ;
- son nom nomenclaturé. Exemple: **H_LRH5_COMPTA_003**.

L'information principale pour Nagios reste **l'adresse IP**, qui lui permettra de vérifier que cet hôte est actif. C'est l'information que vous utiliserez le plus souvent dans les sondes de vérification, car l'adresse IP fonctionnera même si votre système de DNS interne ne fonctionne plus.

Cette information se stocke dans le champ « **address** » de la configuration du Host.

Le **nom complet** peut être utile afin de retrouver cette machine dans votre gestion de réseau. Tous les serveurs devraient avoir au moins un FQDN, qui pourra être utilisé dans un wiki pour documenter le serveur par exemple.

On peut aussi s'en servir dans une sonde pour vérifier que l'adresse IP indiquée sur le Host est bien celle que le DNS interne affecte à ce FQDN.

On stockera cette information le plus souvent dans le champ « **alias** » de la configuration du Host.

Enfin, le nom « **nomenclaturé** » sera stocké dans le champ « host_name ».

Il s'agit du nom de ce serveur dans Nagios. Cette nomenclature devrait vous aider dans l'optique d'une gestion au sein de Nagios.

Ainsi, dans le nom **H_LRH5_COMPTA_003** :

- le « H_ » indique que cet objet Nagios est un hôte ;
- « LRH5 » indique qu'il s'agit d'une machine Linux avec le système d'exploitation Red Hat Entreprise version 5 ;
- « COMPTA_ » est une première indication fonctionnelle (Serveur dédié à la compta) ;
- enfin « 003 » est lié à mon plan d'adressage réseau classique (dans le FQDN le nom du serveur est dserv003).

Il ne s'agit ici que d'un exemple. Chaque entreprise possède ses propres règles. Le host_name pourrait aussi être la version courte du FQDN (donc ici dserv003). Mais utiliser une nomenclature plus poussée au sein de Nagios peut vous apporter quelques éléments de confort.

Je citerai par exemple les interfaces de configuration de Centreon, les hôtes y sont triés par ordre alphabétique du host_name et regroupés automatiquement en groupes (en terme purement graphique) quand ils partagent le même début de nom.

Ainsi H_LRH5_COMPTA_002, H_LRH5_COMPTA_008 et H_LRH5_COMPTA_008 seraient affichés sous le niveau de regroupement « H_LRH5_COMPTA_ ».

Listons maintenant ce qui à priori pourrait devenir un hôte dans un système de supervision :

- les serveurs **physiques ;**
- les serveurs **virtuels ;**
- les équipements actifs (routeurs, firewall, etc) **;**
- certaines imprimantes **;**
- des postes utilisateurs **;**
- des adresses IP virtuelles **;**
- des adresses IP multiples d'un même Host.

Au niveau théorique un hôte est l'entité qui héberge des services à superviser. Mais il est aussi possible de définir un hôte n'hébergeant aucun service. Il pourrait par exemple s'agir d'un routeur sur lequel aucun service n'est supervisé, mais qui serait un élément physique réel intervenant sur le réseau et qui, s'il devenait inactif deviendrait un problème pour tous les éléments situés en aval.

Vérifier qu'un hôte est actif ou inactif sur le réseau peut donc avoir de l'importance, d'autant que si un hôte héberge un grand nombre de services, il devient inutile de tenter de superviser ces services si l'hôte lui-même ne peut visiblement plus être atteint.

Nagios va donc effectuer des vérifications de supervision au niveau des hôtes. Chaque hôte possède une commande de supervision. Par défaut, cette commande consiste à vérifier que l'hôte répond au **ping**.

➲ *Nagios n'effectue pas de supervision continue des hôtes. Cette supervision n'est effectuée que lorsque la supervision d'un service est en échec. C'est pourquoi si on veut associer des graphiques à une sonde de Ping, il faut souvent créer un service de Ping.*

Nagios ne commence pas par la vérification des hôtes. La supervision est « orientée service ». Nagios vérifie qu'un service est rendu, mais quand celui-ci n'est plus rendu, on peut raisonnablement se demander si cela n'est pas du à un problème plus grave sur l'hôte. Nagios va enclencher le système de supervision de l'hôte, et vérifier que l'échec du service n'est pas du à un accès impossible à l'hôte.

8.2.2. La topologie dans Nagios

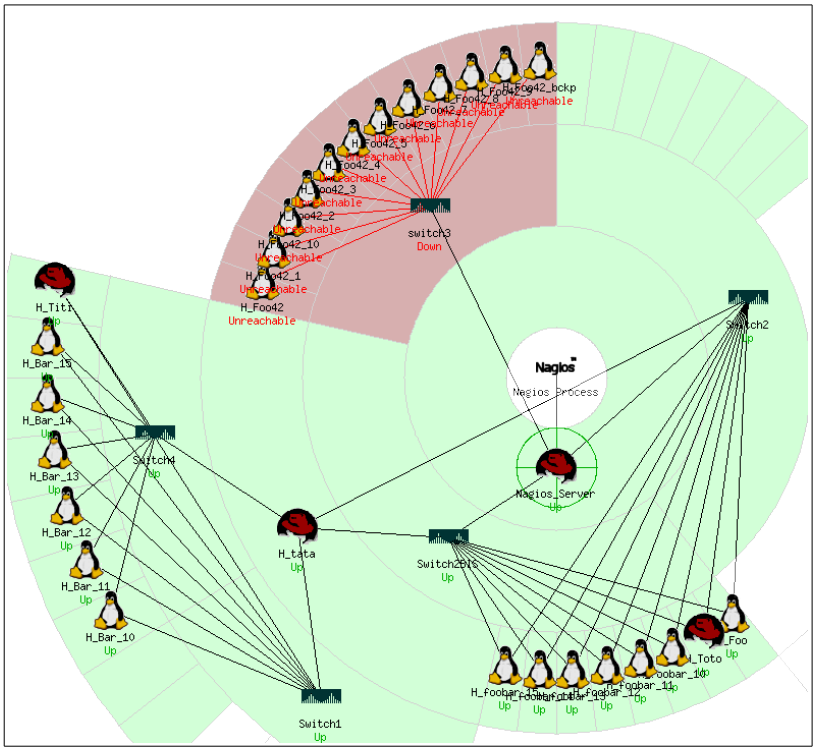
La topologie pourrait se traduire des racines grecques du mot comme la science de l'**étude des lieux**. Pour Nagios la topologie représente la connaissance du réseau physique par le système de supervision.

Le lieu dans lequel la supervision intervient est le **réseau IP** qui lui permet de joindre les hôtes (sur lequel ICMP, TCP ou UDP sont utilisés).

Une topologie IP est bien plus simple qu'une cartographie géographique de la terre. En terme de réseau IP on peut représenter le lieu de chaque machine par rapport au système de supervision

en indiquant simplement, pour chaque hôte, quel est **l'hôte parent dans le chemin qui le relie au serveur de supervision**.

Dans Nagios, la topologie se paramètre donc en renseignant la relation **parent** d'un hôte. Visuellement, elle se manifeste dans la **status map** *(carte de statut)* des hôtes, plusieurs modes de visualisation de cette carte, dont le mode par défaut, se servent de la topologie pour représenter les hôtes en grappes.



Exemple de carte de statut des hôtes

➲ *On remarquera que le centre de la carte de statut est bien le serveur de supervision sur lequel un cercle blanc indique « Nagios ».*

Imaginons un cas d'exemple, vous supervisez plusieurs réseaux, vous possédez plusieurs agences distantes, reliées à internet par l'ADSL, dans lesquelles de nombreux serveurs sont surveillés, et le serveur de supervision est situé dans une de ces agences. Ce serveur de supervision accède donc à une partie des serveurs distants à travers Internet.

Le serveur distant que vous supervisez aura comme parent un switch, qui lui même aura comme parent une box ADSL, elle même fille de votre box ADSL locale, etc. Cette relation parent de chaque machine doit permettre de remonter jusqu'au serveur qui héberge Nagios.

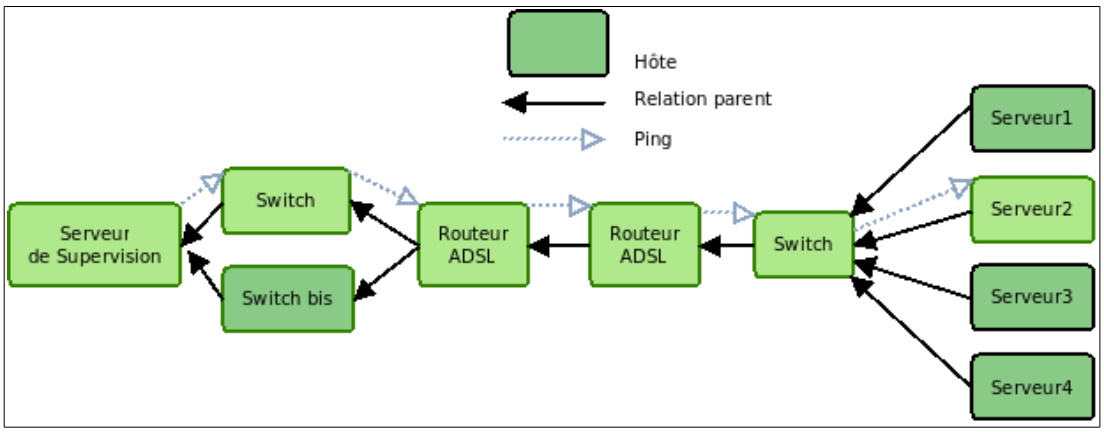


Schéma de relations parent

8.2.3. Network Outage: brin de réseau indisponible

Il est malheureusement courant de négliger de renseigner les informations topologiques dans le système de supervision, chaque hôte est alors perçu par Nagios comme un voisin proche du serveur de supervision qui peut soit être joint (ping) ou être injoignable. Il devient alors inactif.

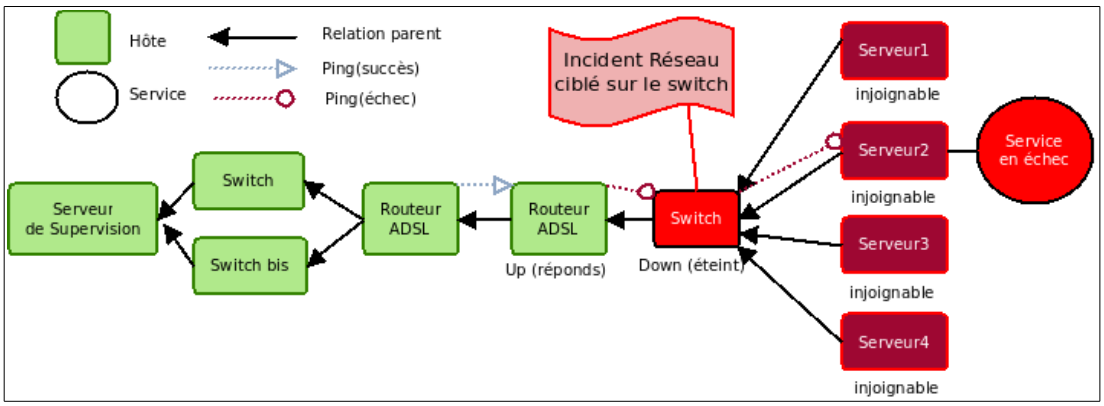
Or Nagios possède un algorithme avancé dans la supervision des hôtes qui lui permet de tirer parti des informations topologiques pour identifier l'hôte clef, le routeur ou le switch par exemple, qui ne répond plus.

Si nous reprenons l'exemple précédent, dans une des agences vous supervisez quatre serveurs, avec sur chacun une dizaine des services. Le switch de l'agence s'éteint.

Lorsque Nagios va tester l'un des quarante services, il va obtenir un échec. Nagios va alors tenter de pinger l'hôte de ce service (un des 4 serveurs). Nouvel échec. La relation parent de cet hôte indique qu'il faut tenter de pinger le switch, échec, il faut alors pinger le routeur ADSL, toujours en échec. La relation parent est remontée jusqu'à trouver un hôte répondant positivement.

L'hôte précédent est alors le fautif.

Ici Nagios va identifier ce switch comme l'hôte fautif. Il va automatiquement indiquer que les quatre serveurs sont **injoignables**.



Incident réseau (network Outage) avec hôtes injoignables

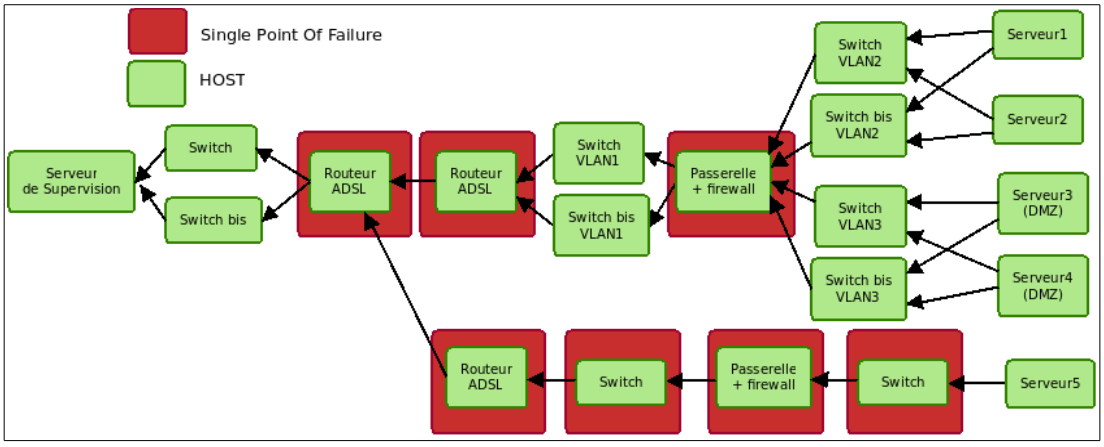
Ce système va permettre à Nagios d'alerter une seule et unique fois, l'alerte sera un **Network Outage** *(Incident Réseau)*. Cette alerte sera ciblée sur la box ADSL.

Sans la topologie, on obtiendrait autant d'alertes que d'hôtes impactés par cet incident. Et surtout, Nagios perdrait énormément de temps à tester des services ou des serveurs situés en aval de ce switch, sans aucune chance de réussite.

Dans le cas du Network Outage, tous les hôtes et services concernés seront bien sur visibles dans les interfaces de Nagios comme n'étant plus en état de fonctionnement normal (hôtes unreacheable et down, services critical), mais le système d'alerte sera ciblé et Nagios ne tentera pas de joindre les serveurs injoignables (unreachable). **Les incidents réseaux sont identifiés à part dans les écrans, souvent en tête, ils sont très importants car ils empêchent la supervision de tous les hôtes et services situés au delà du point d'incident.**

Depuis la version 3 de Nagios, il est possible de **définir plusieurs parents pour un seul hôte**, dans un réseau IP robuste on trouvera en effet de nombreux équipements en double, permettant d'assurer un chemin de remplacement en cas de défaillance. Ainsi, les switchs seront doublés, les routeurs ADSL aussi, il est important de signaler ces chemins alternatifs à Nagios, pour qu'il ne considère pas comme injoignables des hôtes qui sont en fait toujours joignables.

➲ *La topologie visible sur la carte des status va vous permettre d'identifier les **Single Point Of Failure – SPOF** (point unique de défaillance). Il s'agit des noeuds (hôtes) pour lesquels un seul chemin mène au serveur de supervision, dans le cas ou ce nœud s'éteint Nagios ne possède plus d'autres chemins pour joindre les hôtes situés derrière.*



8.2.4. Mais ou est cette carte topologique dans Centreon?

Nagios fournit cette status map dans ses interfaces. Il est par contre très courant que les utilisateurs de Centreon ignorent son existence, simplement parce que par défaut Centreon ne la reprend pas dans ses interfaces de supervision.

Elle existe cependant derrière Centreon : en effet, nagios est une brique indépendante qui tourne « sous » Centreon et peut donc être accédée directement à l'url de Nagios.

Il existe un module complémentaire pour Centreon qui permet de récupérer la status map de Nagios et de l'afficher dans l'onglet Monitoring/StatusMap. Ce module se nomme « Nagios Status Map ». Son auteur original est wistof <wistof@free.fr> et le module à été repris en charge par MERETHIS.

On peut donc le trouver sur la forge Centreon à cette adresse :

http://community.centreon.com/projects/module-nsm. A l'heure actuelle (octobre 2011), il est plutôt conseillé d'utiliser la version de développement, directement depuis le gestionnaire de sources subversion à cette adresse: http://svn.modules.centreon.com/nagios-status-map/trunk/.

8.2.5. Statut des hôtes

A chaque fois que la commande de supervision des hôtes est utilisée, un statut est reporté pour l'hôte.

De façon simplifiée, on pourrait voir ce statut comme un état à deux valeurs, actif ou inactif. OK ou pas. En fait ce statut peut prendre **trois états**, à chaque état est associé un code (qui est la vraie valeur rapportée par la commande de supervision) et son équivalent textuel. On peut y associer un code couleur, qui dépendra cependant de votre **thème**. Le troisième état sert à gérer les hôtes à l'état inconnu car situés au delà d'un incident réseau.

Code	Statut	Code Couleur	Commentaire
0	UP	Vert	Tout va bien, l'hôte répond au ping.
1	DOWN	Rouge	L'hôte est **éteint**, ou bien sa pile IP ne fonctionne plus
2	UNREACHABLE	Orange	L'hôte est **injoignable**. L'un de ses hôtes parents est DOWN et provoque un **Network Outage**.

8.2.6. De la bonne utilisation de la relation parent

Le premier point, le plus important, dans la saisie de la relation parent est qu'il ne faut y indiquer que le parent direct dans le chemin qui mène au serveur Nagios, n'indiquez pas les parents du parent, Nagios se charge de gérer cela pour vous.

On peut voir la relation parent des hôtes comme une gestion des dépendances entre hôtes. Ceci est accentué par la relation multiple depuis Nagios 3. Ainsi on pourrait être tenté de définir comme parent d'un serveur virtuel :

- son serveur de virtualisation ;
- l'éventuel serveur de montages de disques (SAN) ;
- le ou les switchs sur lesquels il possède des connexions.

Mais il faut garder à l'esprit que la relation parent est construite en vue de la détection des Network Outages. Nous sommes dans le monde de la topologie IP des chemins menant à l'hôte depuis Nagios.

➲ *Il faut donc se concentrer sur les relations concernant le réseau IP. Il faut se poser la question: « **Si la pile IP de cet hôte ne fonctionne plus, ce chemin IP vers Nagios sera-t-il coupé ?**». Des notions de dépendances de services existent dans Nagios qui devraient prendre le relais dans la vraie gestion des ressources inter-dépendantes.*

A la lumière de cette question, le serveur de virtualisation est quasi-certainement notre premier et unique parent, sauf s'il s'agit d'un serveur de virtualisation très robuste (en cluster) auquel cas on s'intéressera sans doute plus aux liens vers les switchs. La SAN ne devrait pas à priori être un parent, et pour les switchs sur lesquels des connexions existent il faut bien sûr garder à l'esprit que seules les connexions vers des réseaux permettant de trouver un chemin de la supervision jusqu'à ce serveur sont intéressantes.

8.3. Comment positionner Nagios dans le réseau?

Dans un système de supervision on comprend que la topologie est importante. Le système de supervision deviendra vite non opérationnel s'il est trop éloigné des éléments à superviser et que certains hôtes critiques sont susceptibles d'empêcher toute communication entre le système de supervision et les éléments à superviser.

Toute la topologie est basée sur l'établissement des chemins partant de Nagios et menant aux hôtes. Le placement du serveur de supervision est donc un problème important.

Pour vous aider à décider de l'emplacement de la supervision dans le système d'information voici une liste de questions à se poser, **par ordre d'importance** :

- **diminuer l'impact des Network Outages pour les services critiques** : il faut placer le système de supervision de manière à réduire le nombre d'éléments entre lui et les hôtes hébergeant des services critiques. Réduire les Single Point Of Failure (SPOF) sur ces chemins ;
- **prendre en compte le nombre de barrières à franchir** pour atteindre des réseaux cloisonnés (firewall, VPN, etc.) ;
- **réduire au maximum la taille du plus grand chemin** . Ceci est à pondérer avec l'importance des services situés au bout de ce long chemin ;

- **prendre en compte la sécurité physique du serveur de supervision :** un accès physique au serveur par des personnes non autorisées, le risque de coupure électrique, de panne matérielle (vive la virtualisation et les clusters), etc.
- **prendre en compte la distance entre le serveur de supervision et les équipes de supervision**. Si les équipes de supervision sont incapables de consulter les écrans ou les alertes le système n'est plus opérationnel. Il faut aussi réduire les SPOF sur ces chemins ;
- prendre en compte la **distance entre le serveur de supervision et les serveurs d'envois d'alertes** (mails, sms, service web web2sms, services téléphoniques). Disposez-vous d'un serveur de mail local ? Le serveur de supervision lui-même devrait disposer d'un serveur de mail capable de stocker les messages puis de les relayer au serveur de mail principal.

Sur des réseaux très distribués, il faudra aussi étudier les possibilités de supervision **distribuées** et la **robustesse** du serveur de supervision lui-même.

Des projets comme Centreon et Shinken apportent des solutions à ces problématiques que vous pourrez étudier une fois que vous aurez terminé la lecture de ce document.

8.4. Les services

8.4.1. Qu'est-ce qu'un service?

Pour l'instant nous n'avons parlé que des hôtes. Et pourtant le plus important pour vous comme pour le système de supervision est le service.

Pour Nagios tout peut être considéré comme un service.

Vérifier un service peut consister à vérifier qu'un service attendu par un utilisateur est rendu par une application, ou il peut s'agir de relever des indicateurs de gestion de la mémoire. On peut envisager d'utiliser des services pour générer des alertes en cas de dépassements de seuils (supervision opérationnelle), mais aussi utiliser des services pour établir des statistiques à long terme (supervision prévisionnelle). On peut utiliser des services pour aller chercher de l'information sur des hôtes (mode PULL/actif) ou bien pour recevoir des informations occasionnellement (mode PUSH/passif).

Le service est donc la principale entité de configuration de Nagios. Il s'agit pour le système de supervision d'une tâche qu'il devra le plus souvent planifier dans le temps, et à laquelle il devra associer :

- un comportement de supervision (exécuter une tâche de supervision définie, ou bien simplement attendre de recevoir des informations) ;
- mais aussi des informations sur ce qu'il convient de faire en cas de problèmes (très grossièrement qui prévenir ? Quand ? Et comment ?).

Nagios pourra même aller jusqu'à la supervision pro-active en exécutant des tâches complémentaires en fonction du résultat de la sonde (c.f. les event handlers).

8.4.2. Association des services et des hôtes

On a vu précédemment qu'il existait une autre entité importante, les hôtes. **Un service doit toujours être associé à au moins un hôte**. Quand Nagios effectuera la supervision de ce service, il ne le fera que pour un hôte à la fois.

Cette association peut se faire de deux façons. On peut associer notre service à un **hôte unique**, ou bien l'associer à un **groupe d'hôtes**, de façon à ne le définir qu'une seule fois.

Pour Nagios cependant, ce service sera toujours associé à un hôte unique au moment de sa vérification.

Dans le cas du groupe d'hôtes, la vérification sera planifiée autant de fois qu'il y a d'hôtes dans le groupe.

En cas de problème signalé sur ce service, le sous-système de gestion des problèmes sera enclenché, et tout ce qui pourra être généré par ce sous-système (envoi d'alertes, changements d'états sur les interfaces de supervision, exécution de commandes dépendantes, etc.) pourra être impacté par des configurations du service comme de l'hôte concerné. Cela signifie par exemple que lorsque vous recevez une alerte par mail pour un service, ce mail contient toujours des informations sur l'hôte sur lequel ce service pose problème.

8.4.3. Statut des services

Comme pour les hôtes, les services possèdent un statut. Ce statut est la base des écrans de supervision qui nous renvoient des codes couleurs inspirés des feux de circulation. On imagine donc souvent que ce statut possède trois valeurs (feu vert, feu orange, feu rouge), il en possède en fait cinq.

Code	Statut	Code Couleur	Commentaire
0	OK	Vert	Tout va bien, le service est rendu
1	WARNING	Orange	Attention : le seuil de **warning** est franchi
2	CRITICAL	Rouge	**Critique** : le service n'est plus rendu. Le seuil de niveau critique est franchi, certaines sondes renvoient aussi cette valeur en cas de limite de temps atteinte.
3	UNKNOWN	Gris foncé	**Inconnu** : La sonde de vérification est en erreur. Il peut s'agir d'un mauvais paramétrage, d'une mauvaise utilisation de la sonde ou d'une erreur non gérée par la sonde. Certains service utilisent aussi ce statut en lieu et place du suivant.
4	PENDING	Gris clair	**En attente** : Le service ne possède pas encore assez d'information pour répondre. Il peut s'agir d'un service qui n'a pas encore été exécuté ou d'un service qui a besoin de plusieurs exécutions pour travailler sur des moyennes ou des différences.

Le dernier statut est un statut temporaire, qui ne pose donc pas de problème particulier, le quatrième UNKNOWN est par contre plus problématique. Vous devriez toujours examiner les informations remontées sur les commandes ayant passé un service en UNKNOWN afin de vérifier que vous n'avez pas commis d'erreur dans l'utilisation d'une sonde (ou dans l'écriture d'une sonde. Une sonde SNMP qui se met en UNKNOWN parce que SNMP ne répond pas sur la machine distante ne devrait-elle pas être en CRITICAL plutôt ?).

8.5. Les commandes de supervision

8.5.1. Qu'est-ce qu'une commande de supervision ?

Pour la plupart des services définis dans Nagios, une tâche de supervision doit être effectuée. Seuls les services pour lesquels Nagios attend des informations de l'extérieur ne nécessitent pas de tâche de supervision (les services passifs, en mode push).

Dans le cas courant, Nagios doit donc effectuer une tâche de supervision, mais il ne sait en lui-même effectuer aucune de ces tâches. Il délègue ce travail à des sondes, aussi appelées plugins ou checks. C'est ce qui fait toute la souplesse de cette application de supervision, les tâches de supervision sont infinies, car Nagios ne demande que la saisie d'une commande de supervision que le système d'exploitation se chargera d'effectuer.

Une commande de supervision est une commande au sens « **ligne de commande** ». Il s'agit d'un programme **externe** qui sera appelé par le démon Nagios pour effectuer une tâche de supervision sur un hôte. Cette commande externe recevra un certain nombre de paramètres de Nagios, comme l'adresse IP de l'hôte, ou les niveaux de seuils attendus.

Elle s'exécutera et retournera un résultat à Nagios, il s'agira du nouveau statut de service (ou d'hôte pour les commandes associées à la vérification des hôtes), accompagné de quelques renseignements supplémentaires.

Nagios se charge alors d'interpréter ce résultat au sein du système de supervision, de mettre à jour l'état du service, de planifier la prochaine vérification et de prendre les mesures qui s'imposent (alerter, exécuter d'autres commandes, etc).

Les sondes sont donc des programmes externes à Nagios. Si vous installez Nagios à partir des sources Nagios-core ne contient aucune commande. Il faut installer Nagios-plugins pour bénéficier d'un premier lot de sondes.

La commande de supervision est la définition au sein de Nagios de la ligne de commande qui exécutera une sonde de supervision. Cette ligne de commande comporte un nombre variable de paramètres qui seront renseignés par Nagios au moment ou cette commande devra être exécutée.

Voici un exemple de commande de supervision :

```
$USER1$/check_disk -w $ARG2$ -c $ARG3$ -p $ARG1$
```

Qui lors de l'exécution réelle de la commande deviendra une vraie ligne de commande Unix :

```
/usr/local/nagios/plugins/check_disk -w 80 -c 90 -p /var
```

Voici un deuxième exemple, une commande que nous nommerons :

COMMANDE_HTTP_EXAMPLE_COM (on utilise COMMANDE dans le nom et non « check_http_example_com » parce qu'il s'agit d'une commande et non d'un check, qui lui est un programme externe. La commande **utilise** un check. Elle n'est pas un « check »

Essayez de bien nommer chaque chose, cela vous évitera de vous perdre :

```
$USER1$/check_http -H $ARG1$ -I $HOSTADDRESS$  -u $ARG2$
--useragent=supervision --onredirect=follow --warning=$ARG3$ --critical=$ARG4$
--timeout=8 -s "example.com"
```

Qui donnera (n'essayez pas forcément de comprendre immédiatement ce que fait cette commande, revenez-y plus tard) :

```
/usr/local/nagios/plugins/check_http -H www.example.com -I 192.168.0.15 -u
/index.php --useragent=supervision --onredirect=follow --warning=2 --critical=3
--timeout=8 -s "example.com"
```

Vous pouvez d'ailleurs ouvrir une connexion sur votre serveur de supervision (en ssh par exemple), devenir l'utilisateur Nagios (passez root puis faites un « su – nagios »), puis exécutez vous même cette commande, vous obtiendrez la ligne de texte que le démon Nagios obtient en retour.

En terme de méthode, il s'agit même de la première étape quasiment obligatoire à l'écriture d'une commande Nagios, il faut la tester comme si vous étiez Nagios et que vous deviez l'appliquer à un service et un hôte.

Une commande va en fait pouvoir utiliser 3 types de variables :

- **des variables « globales »**, des constantes internes à Nagios.
  Par exemple $USER1$ qui est une variable contenant le chemin d'accès aux commandes sur le serveur de supervision ('/usr/local/nagios/plugins' dans l'exemple précédent). On retrouve ces arguments dans le fichier « resource » au sein de Nagios ;
- **des variables « dynamiques »,** qui dépendent à un instant « *t* » du service exécutant la commande et de l'hôte sur lequel la commande est exécutée. Par exemple $HOSTADDRESS$ qui est remplacé par 192.168.0.15 ;
- **des variables « argument du service »**, toutes les variables $ARGn$ ou n varie de 1 à 4 dans l'exemple précédent doivent être passées en argument de la commande par le service qui utilise la commande.
  Pour passer ces arguments, on utilise le séparateur « **!** » . Mon service qui utilisera la commande COMMANDE_HTTP_EXAMPLE_COM le fera ainsi :

```
check_command: COMMANDE_HTTP_EXAMPLE_COM!www.example.com!/index.php!2!3
```

On retrouve alors pour $ARG1$ « www.example.com » pour $ARG2$ « /index.php » et pour $ARG3$ et $ARG4$ 2 et 3.

Grâce à cela plusieurs services différents pourraient utiliser la même commande en lui passant des valeurs différentes (seuils de temps de réponse différents, url différentes, etc).

8.5.2. Fonctionnement standard des commandes et sondes

Le fonctionnement des commandes est très lié au fonctionnement des sondes. Toutes les sondes de Nagios doivent avoir le même fonctionnement afin de communiquer simplement leurs informations à Nagios.

Nagios attend donc un résultat sous une forme extrêmement simple :

- un **code de retour**. Tous les programmes exécutés sur une ligne de commande fournissent automatiquement un code de retour ;
- une **ligne de texte**. Même si depuis la version 3 de Nagios, les sondes sont autorisées à retourner plusieurs lignes de texte.

Ce code de retour devra correspondre au code du service (dans le cas d'une commande associée à un service) ou de l'hôte (le cas échéant). Un code 0 (valeur par défaut pour un programme terminant son exécution sans erreur) vaudra OK. Pour indiquer un résultat WARNING ou CRITICAL il faudra renvoyer respectivement 1 ou 2, par exemple avec exit 1 ou exit 2 en shell.

La ligne de texte renvoyée est de son côté composée ainsi :

- le type de la vérification effectuée sous la forme d'une chaîne de caractères simple (comme DISK) ;
- le statut sous sa forme textuelle, si on avait par exemple un code de retour à 2 on doit avoir « CRITICAL » ;
- un tiret « - » ;
- un commentaire libre, souvent utilisé pour donner des informations sur la raison de l'échec ou sur l'état du service. Ce commentaire étant disponible pour l'agent de supervision qui consulte le service dans les interfaces de supervision ;
- éventuellement une barre « | » suivi d'informations statistiques destinées à être traitées par des programmes annexes (pour établir des graphiques par exemple).

On obtiendra par exemple (et sur une seule ligne) pour un résultat de commande de ping :

```
PING OK - 192.168.0.1: rta 97.751ms, lost0% |
rta=97.751ms;200.000;500.000;0;p1=0%;40;80;;
```

Le format de communication de la commande est donc très basique. Cela permet d'écrire des sondes dans tous les langages (C, perl, PHP, python, VB, Java, etc), et de faire passer les résultats à travers des protocoles complexes (SSH, NRPE, etc) sans risque de perte de l'information.

La commande elle même peut être très simple ou extrêmement complexe. Pour Nagios le traitement du résultat sera toujours simple.

Toutes les sondes doivent par contre accepter le paramètre « **--help** »ou « **-h** » et lister toutes les options qu'elles acceptent, avec une explication. Vous pouvez donc toujours exécuter en ligne de commande une sonde et lui demander tout ce qu'elle sait faire. C'est ce qui permet aussi à Centreon un affichage détaillé de toutes les options acceptées par les sondes dans l'éditeur de commande.

En terme de conventions, on retrouvera pour la plupart des commandes des options « **-c** » et « **-w** » pour passer les seuils de warning et de critical, ainsi que « **-t** » pour les timeout. Mais je ne peux que vous inviter vivement à lire l'aide affichée par la sonde avec le « -h ». Grâce à cela, et si vous connaissez un peu le protocole http, vous devriez être en mesure de comprendre ce que la commande COMMANDE_HTTP_EXAMPLE_COM donnée plus haut effectuait.

```
/usr/local/nagios/plugins/check_http --help
```

9. Le fonctionnement de la supervision

9.1.1. Nagios : un ordonnanceur de tâches, un système de notifications et des annexes

Nous venons de voir que les sondes ne sont pas propres à Nagios, elles sont lancées par Nagios, mais il s'agit de processus extérieurs à nagios.

On peut donc voir Nagios comme un démon composé de trois parties principales :

- un ordonnanceur de tâches de supervision (vous pouvez à tout moment consulter la « scheduling queue » dans les écrans de supervision, il s'agit de la liste des tâches prévues, chaque service y est programmé pour un hôte avec une date située, si tout va bien, dans le futur) ;
- système de gestion des alertes (décrit dans le chapitre suivant) ;
- une interface de supervision (en PHP depuis peu, en CGI avant cela).

Et dans les faits, Nagios ne fait effectivement pas beaucoup plus (quelques gestions des reportings, la carte topologique).

Cela correspond à un grand principe de développement, le KISS (Keep It Stupid Simple), faire peu de chose, mais bien les faire, et communiquer facilement avec d'autres briques, en entrée du programme comme en sortie du programme.

Ce principe rend les composants interchangeables. C'est ce qui fait que les sondes Nagios peuvent être utilisées avec d'autres démons de supervision, ou bien qu'on peut alimenter des systèmes de graphiques à partir des résultats obtenus par les sondes. C'est aussi ce qui fait qu'il est possible de répondre à des besoins très complexes comme la supervision de milliers de

services de machines et de systèmes d'exploitation différents en écrivant des programmes relativement simples.

Parmi les fonctionnalités annexes à Nagios que l'on va retrouver dans certaines applications qui lui sont parallèles ou complémentaires , il y a :

- la gestion de la configuration ;
- la gestion des graphiques ;
- la gestion des environnements distribués (avec par exemple dans Shinken l'idée de décomposer Nagios en de multiples programmes indépendants, en poussant le KISS plus loin).

Le KISS est un principe complémentaire d'une autre grande idée des logiciels Open Source qu'Eric S Raymond a synthétisée dans son Livre « La Cathédrale et le Bazar ». Cette idée du bazar signifie qu'il existe autour du cœur de Nagios (ce n'est pas spécifique à Nagios) de nombreux composants qui vont pouvoir apporter des solutions diverses à une même problématique. On ne retrouvera que très rarement une unanimité sur la bonne méthode pour effectuer une tâche.

A titre d'exemple, il existe de nombreuses méthodes pour générer des graphes de type RRD à partir de la supervision. Les bases RRD (Round Robin Databases) sont des fichiers de bases de données tournantes, contenant les points de graphes statistiques qui se déclinent à plusieurs échelles (dernières heures, derniers jours, dernières semaines, dernière année). On les retrouve dans des logiciels comme Cacti ou Munin. Ces bases ont la particularité de comporter un nombre de points maximum fini, donc une taille fixe, ce qui est très utile en terme de supervision.

Dans l'éventail des possibilités offertes pour gérer ce type de graphiques on trouvera des sondes qui vont chercher le statut nagios (Critical, Warning, OK) directement dans des bases RRD générées par cacti, donc dans des données collectées par Cacti. D'autres solutions iront collecter les données à travers des sondes Nagios, puis exploiteront les données statistiques supplémentaires renvoyées par les sondes pour générer des fichiers RRD (comme le fait Centreon, avec en plus un stockage MySQL). Il faut donc comprendre les différentes solutions proposées par le « bazar » pour juger de celle qui sera la plus adaptée à votre situation et qui sera maintenue le plus longtemps. Il y a un risque dans le « bazar », certaines solutions meurent ou disparaissent, mais il y a une interopérabilité qui fait que le service sera toujours rendu, vous ne serez jamais prisonnier d'une solution.

9.1.2. Un service, un hôte, une commande

Les trois principaux objets de configuration dans Nagios sont donc :

- les **hôtes** : vos serveurs et équipements actifs, éventuellement regroupés avec des groupes d'hôtes ;
- les **services** : les services divers et variés que vous allez vouloir vérifier sur ces hôtes et groupes d'hôtes. Vous pouvez regrouper ces services dans des groupes de services, par exemple pour les retrouver plus facilement dans les interfaces de supervision (où en sont tous les services liés à la gestion de tel site web ?) ;
- les **commandes** : quels sont les programmes externes à Nagios que j'utilise pour effectuer mes vérifications ? Avec quels paramètres fixés et quels paramètres variables ?

Un service peut être affecté à plusieurs hôtes, mais il ne possède qu'une seule commande, avec éventuellement des arguments. D'autres services peuvent utiliser cette même commande, avec éventuellement des arguments différents.

Ajoutons que les hôtes autant que les services peuvent hériter des modèles (templates) dans lesquels de nombreux paramètres ont pu être saisis, y compris la commande à utiliser et les arguments à cette commande.

Nagios supporte l'héritage multiple, on peut donc pour un service hériter de plusieurs modèles de services.

Cependant au moment ou cette commande sera exécutée, elle correspondra à un service et à un hôte, c'est ce qu'il vous faut retenir au moment de l'écriture de vos commandes. Ceci par exemple pour éviter de faire passer en argument de la commande l'adresse IP de l'hôte sur lequel lancer la commande, cette adresse vous l'avez à disposition à cet instant « t » dans la variable $HOSTADRESS$.

9.1.3. Ordonnancement des commandes de supervision dans les services

Chaque service possède une commande. Il possède aussi un élément de configuration qui définit à quel intervalle cette commande devrait être effectuée pour l'hôte associé au service.

Ce paramètre se nomme **normal_check_interval** *(intervalle de vérification normal)*. Il s'agit d'un nombre de minutes. Si vous indiquez 5 cela signifie que Nagios planifiera une vérification de ce service pour cet hôte toutes les cinq minutes.

Ce paramètre pourra posséder des valeurs par défaut, et vous pourrez faire varier ce paramètre pour un ensemble de services. Ainsi on peut organiser dans Nagios un ensemble de services à vérifier toutes les cinq minutes, un autre comprenant des services à surveiller une fois toutes les heures, d'autres tous les jours.

Muni de cette information, du nombre d'hôtes, du nombre de services à affecter à chaque hôte, Nagios va essayer de planifier toutes les vérifications à effectuer, puis à chaque exécution de commande il planifiera la prochaine exécution de la commande pour ce service.

Deux autres paramètres très importants vont intervenir dans l'ordonnancement des commandes, il s'agit des paramètres **max_check_attempts** *(nombre maximum de tentatives)* et **retry_check_interval** *(intervalle de vérification en mode 'réessayer')*. En effet, si la commande vérification a retourné autre chose qu'un état OK, Nagios va retenter la commande plusieurs fois (max_check_attempts fois) en espaçant ces nouvelles vérifications avec le paramètre retry_check_interval (en minutes). Sur un service qui était vérifié toutes les 5 minutes, on aura par exemple la possibilité de retenter la vérification 3 fois toutes les deux minutes. Ceci afin de s'assurer que l'état anormal est bien dû à un état du service supervisé et non à une erreur dans l'exécution de la commande.

Enfin, d'autres paramètres avancés peuvent intervenir sur la façon dont Nagios organise la répartition initiale des commandes ou le nombre de commandes à faire tourner en parallèle. Mais l'élément le plus important à retenir est que les vérifications ne seront pas planifiées dès le départ (par exemple, en découpant la journée en tranches de cinq minutes) mais bien au fur et à mesure en fonction du retour de la commande. Ce qui fait qu'un service vérifié toutes les cinq minutes,

pourra aussi subir trois vérifications espacées d'une minute en cas d'erreur, avant de retourner à un intervalle de cinq minutes.

Dans la partie suivante nous verrons les impacts importants de ces trois paramètres sur la gestion des incidents.

9.1.4. Etat des services: HARD ou SOFT?

Si la plupart des utilisateurs occasionnels de Nagios ont remarqué la présence des états CRITICAL, OK et WARNING ; peu nombreux sont ceux qui ont remarqué que les services possèdent aussi un statut HARD ou SOFT.

Le premier (OK,CRITICAL,etc) est le « **Service State** » et le second est le « **Service State Type** ».

Le type Hard ou Soft va venir qualifier l'état actuel du service. On est donc par exemple en WARNING-SOFT ou en WARNING-HARD.

Pour Nagios, le SOFT signifie que l'état est encore incertain,que la sonde a renvoyé une erreur mais qu'il s'agit peut-être d'un problème dans l'exécution de la sonde.

Lorsqu'une sonde renvoie un état qui est différent du dernier état, le type de statut passe à SOFT. On passe alors en mode « **retry** », les paramètres **max_check_attempts** et **retry_check_interval** vont entrer en jeu.

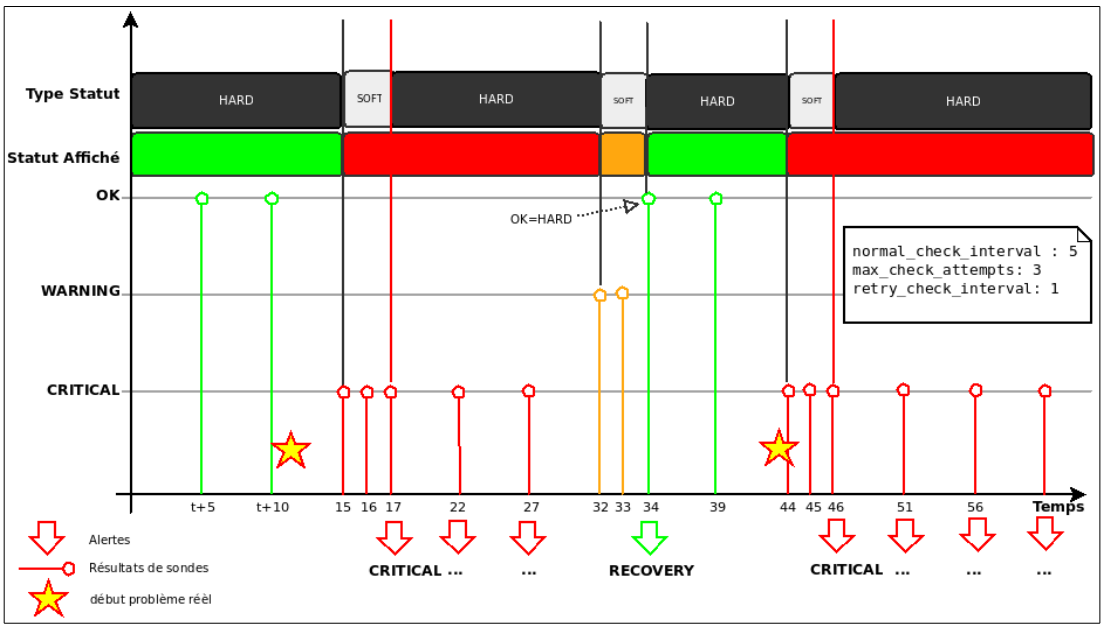
Au lieu d'un espacement habituel (comme 5 minutes), on passe à un espacement des vérifications de retry_check_interval minutes (par exemple 1 minute). Et on va tenter la vérification max_check_attemps fois (par exemple 3 fois).

Si au bout de ces 3 essais, la sonde renvoie toujours le même résultat Nagios va considérer que ce changement de statut est à graver dans le marbre, le type de statut passe à HARD, on reprend un rythme de vérification régulier, et le sous système de notifications entre en ligne de jeu.

- Il n'y aura jamais de notifications avant que le type de statut ne soit passé à HARD. Ce qui n'empêche pas la sonde d'être affichée avec son état SOFT (comme CRITICAL) sur les écrans de supervision, dans les colonnes affichées on voit alors 1/3, 2/3 puis 3/3 s'afficher.

- Un résultat OK est toujours HARD, Nagios considère qu'une sonde ne renvoie jamais OK par erreur ou à cause d'un problème temporairement absent.

Pour comprendre ce qui se passe dans le temps le mieux est d'étudier une frise chronologique d'un service, avec ses changements de statut et de type de statut.



Statut et type de statut: analyse des états de service

Dans le graphique, on voit qu'un événement intervenant juste après la minute 10 et avant la minute 15 ne génère aucune alerte avant la minute 17. Il est cependant déjà répertorié dans les incidents au niveau de l'interface Nagios dès la minute 15.

➲ Quand vous choisissez des valeurs pour normal_check_interval, max_check_attempts et retry_check_interval, n'oubliez jamais que ceci peut entrainer un **temps de décalage de l'alerte** de presque « **normal_check_interval+(retry_check_interval * max_check_attempts)** »

On constate aussi que le passage par un état warning n'ayant pas pu être testé trois fois, aucune alerte de ce changement d'état n'aura été envoyée.

Ce graphique présente quelques flèches d'alertes. Il ne s'agit en fait que de l'activation du sous-système de notifications. Il n'est pas du tout certain que chacune de ces flèches générera une alerte par mail à destination d'un utilisateur. Car le système de notification est très paramétrable.

10. Le sous-système de notifications

10.1. Alerter de façon intelligente

10.1.1. Diminuer le nombre de notifications

Tous les utilisateurs réguliers d'un système de monitoring savent que l'un des principaux problèmes posés par le monitoring est **l'avalanche d'alertes** que celui-ci peut générer. Hors une alerte importante, si elle est noyée dans une grande masse d'alerte sera ignorée :

- soit parce que les utilisateurs auront pris l'habitude de ne pas lire les mails d'alerte, voir de les archiver automatiquement en dehors de leur INBOX ;
- Soit parce que parmi 200 alertes ils ne pourront pas repérer l'alerte important, celle qui est prioritaire.

Quand nous avons étudié la topologie, nous avons vu que la relation parent, permettant de détecter les network outages était déjà pensée dans le but de réduire le nombre d'alertes et de générer une alerte unique et ciblée sur l'équipement réseau fautif.

D'autres éléments vont pouvoir entrer en jeu dans le paramétrage du système de notification afin de réduire les alertes.

Il est possible de suspendre les alertes au niveau global. Il n'y a alors plus aucune alerte générée. A un niveau inférieur, on va pouvoir pour chaque hôte et chaque service décider s'il peut générer des alertes et, le cas échéant quels types d'alertes il génère.

On peut par exemple dire qu'un service donné ne génère des alertes que pour CRITICAL et RECOVERY, jamais pour UNKNOWN et WARNING. Bien sur on aura tout intérêt à définir ces paramètres non pas au niveau d'un service ou d'un hôte en particulier mais plutôt dans un modèle d'hôte ou de service afin que toute la famille profite des mêmes paramétrages.

S'il est très important d'alerter sur les services critiques, il existe certainement certains services non critiques qui ne nécessitent pas de génération d'alertes, n'hésitez pas à suspendre les alertes pour cette famille de services. Profitez-en peut-être aussi pour changer la planification des vérifications sur ces services, une vérification horaire ou quotidienne ne serait-elle pas plus adaptée qu'une vérification toutes les 5 minutes ? Pour vous simplifier la vie, utilisez les modèles et définissez ces paramètres importants dans les modèles.

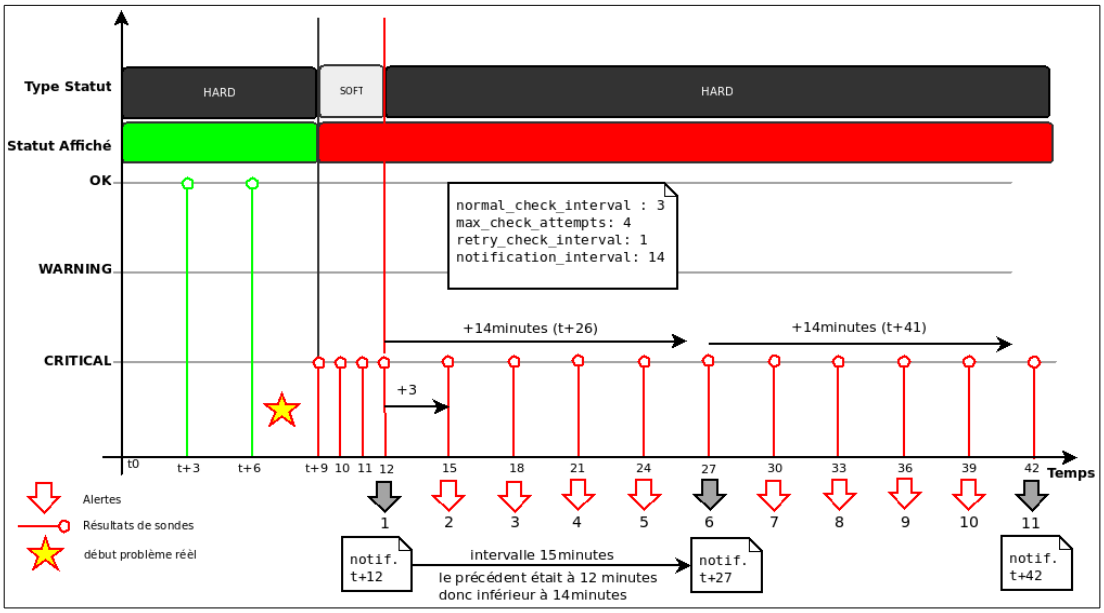
Après l'activation des alertes, il faut aussi définir, soit dans les modèles, soit dans des services ou hôtes, les utilisateurs ou groupes d'utilisateurs destinataires de ces alertes. On peut bien sur envoyer les alertes d'un même service à plusieurs destinataires. On préférera cependant toujours une gestion basée sur des groupes afin de gérer facilement le management des équipes.

10.1.2. Répéter les alertes

Le graphique « Statut et type de statut: analyse des états de service » présenté plus haut montrait plusieurs flèches d'alertes suite au passage en HARD du statut CRITICAL d'un service. A chaque nouvelle alerte théorique, le sous-système de notification se déclenche et va décider de lancer ou non une alerte.

Un nouveau paramètre de configuration important de l'hôte ou du service entre alors en jeu: « **notification_interval** ». Celui-ci permet d'indiquer un temps minimum entre deux notifications. **Une valeur à 0 de ce paramètre signifie qu'on ne veut recevoir l'alerte que la première fois**. C'est le choix le plus courant. On ne reçoit l'info du service en état critique qu'au moment ou Nagios le détecte en HARD.

La prochaine alerte sur ce service sera pour un changement d'état (WARNING ou RECOVERY). Mais si j'indique 14 minutes alors il est possible qu'une fois le délai de 14 minutes dépassé Nagios décide de me ré-envoyer l'alerte. Examinez la frise chronologique ci-dessous :



La répétition des alertes

Sur cette frise on voit qu'un fois le statut passé en HARD au temps t+12 une première alerte est émise. Toutes les 3 minutes (normal_check_interval) Nagios va relancer sa sonde qui va lui renvoyer un état Critique. Au bout de 15 minutes, Nagios vérifiera que l'intervalle de 14 minutes de silence est passé et renverra l'alerte. Un nouvel intervalle de silence est alors démarré.

➲ On remarquera qu'on obtient des notifications toutes les 15 minutes et non toutes les 14 minutes, choisir un intervalle de notification multiple du normal_check_interval peut simplifier la lisibilité (ici 15 minutes au lieu de 14 minutes).

Certaines versions récentes de Nagios3 supportent le paramètre **first_notification_delay** qui est délai supplémentaire associé à l'envoi de la première notification et qui pourrait alors intervenir dans cette frise en décalant l'envoi de la première notification.

10.1.3. Dépendances, Arrêts programmés et Escalades

Toujours dans l'optique de diminuer les notifications, Nagios propose un **système de dépendances entre services**. Pour le moment, nous ne détaillerons pas cette fonctionnalité, je vous laisse consulter la documentation française (en lien dans les annexes de ce document) pour les détails d'implémentation. Il vous suffira pour le moment de savoir que leur but est de cibler la notification à envoyer en indiquant à Nagios que certains services dépendent d'un autre, et que si tous sont en état critique il convient d'envoyer la notification d'erreur du service Maître plutôt que sur ses dépendants.

Vous pouvez aussi indiquer à Nagios des phases d'arrêt prévues (Scheduled Downtime). Si par exemple il est prévu que ce mercredi vous arrêtiez 10 services web et le serveur de base de données associé pour la journée afin d'effectuer des opérations de maintenance, vous pouvez créer une phase d'arrêt programmé pour tous ces services et la stocker dans Nagios.

Au moment ou les services passeront en état critique Nagios, n'enverra pas de notifications tant que cette plage de temps ne sera pas dépassée.

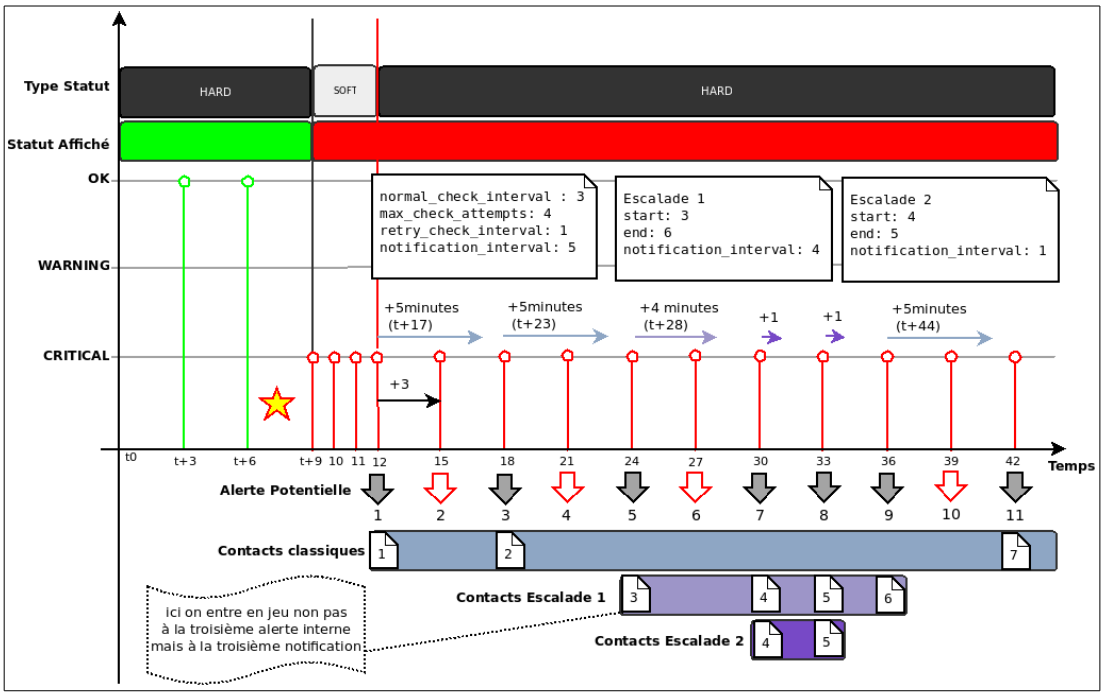
Enfin Nagios propose une notion d'escalade. L'escalade consiste à prévenir des groupes d'utilisateurs différents au fur et à mesure de l'évolution d'un même incident.

Le cas d'utilisation le plus courant est la gestion d'équipes de support en premier puis en second niveau. On peut aussi imaginer prévenir un groupe d'administrateurs spécialisés sur une tâche (comme les DBAs en cas de chute d'un SGBD) puis prévenir l'équipe entière au bout d'une heure de panne, puis prévenir le chef de service au bout de deux heures, etc.

On peut associer un grand nombre d'escalades à un même service. Afin de bien les utiliser, il faut noter quelques particularités dans le façon de définir ces escalades :

- les escalades ne travaillent pas sur le nombre de remontées d'alerte mais sur le nombre d'alertes ayant généré réellement des notifications ;
- une escalade s'exprime ainsi : « A partir de la xième notification, jusqu'à la yième notification sur ce service merci de notifier les groupes de cette escalade » ;
- dès qu'au moins une escalade entre en jeu, les contacts classiques du service ou de l'hôte ne sont plus notifiés, si on veut qu'ils continuent à être notifiés il faut les intégrer dans les contacts de l'escalade ;
- Si une escalade utilise un notification_interval à 0 (une seule alerte) les escalades s'arrêteront après lui, car il n'y aura plus de notifications ;
- les escalades possèdent un intervalle de notification qui peut être différent de celui du service. L'intervalle de notification le plus court est choisi (parmi le service et les n escalades actives à un instant t).

Le schéma suivant peut vous permettre de comprendre ce qui se passe réellement en cas d'escalade:



Exemple d'escalades

Sur cette frise on observe que la deuxième escalade, possédant un intervalle de notification très court à provoqué l'émission d'alertes très rapprochées, y compris pour les contacts de la première escalade qui avaient été définis avec un intervalle de notification assez long. On voit aussi que si les notifications ne sont plus envoyées aux contacts classiques elles sont par contre envoyées à toutes les escalades actives.

Enfin on peut constater un comportement qu'on pourrait presque qualifier de bug. Lorsque la deuxième escalade s'arrête à la 5ème notification, l'intervalle de temps choisi avant la 6ème notification est calculé en prenant l'intervalle de temps le plus court parmi celui du service, de la première escalade, mais aussi de la deuxième escalade alors qu'elle n'est plus active.

➲ *Là encore, choisir des intervalles de temps de notification pour le paramètre « **notification_interval** » qui sont multiples du paramètre « **normal_check_interval** » permet d'améliorer la lisibilité des différentes escalades. Jouer sur les paramètres start et stop est aussi plus clair que de jouer sur des intervalles de notifications différents entre les différents niveaux d'escalade.*

10.1.4. Intégrer la gestion des périodes de temps

Pour les services, les hôtes et les individus, Nagios permet d'indiquer des périodes de temps. Ces périodes vont définir des plages horaires et journalières dans lesquelles les notifications sont autorisées.

Il est alors possible de se servir de ce système pour :

- décider qu'un service n'est critique qu'à certaines plages horaires ;
- piloter des équipes travaillant à des horaires différents ;
- gérer les week-ends en n'envoyant l'alerte qu'à certains contacts dans cette plage de temps (par exemple, un contact possédant un téléphone d'équipe).

Dans la définition des plages temps, nous indiquons les plages de validité horaire pour chaque jour de la semaine (du lundi au dimanche). Il n'est pas possible de définir des choses comme « uniquement le premier lundi du mois ». On peut, par contre, indiquer plusieurs plages horaires par jour, ce qui permet de gérer les nuits en indiquant une plage horaire de minuit à 8:00, puis de 18:00 à 23:59.

Que se passe-t-il si un service à un « notification_interval » à 0 et que la première alerte se situe dans une plage de temps ou les notifications sont inactives ?

En théorie le notification_interval à zéro signifie que vous ne voulez recevoir qu'une seule alerte. Et c'est bien ce qui se passera, la première alerte que vous recevrez sera celle qui sera générée lorsque la sonde remontera un état d'alerte dans la plage de temps autorisée. Toutes les remontées précédentes auront été ignorées en terme d'alerte.

« notification_interval » 0 ne signifie pas que seule la première alerte sera envoyée, cela signifie qu'une seule notification sera envoyée, cette première notification pouvant intervenir assez tard dans la vie de l'alerte.

10.1.5. Choisir le medium de l'alerte

Une fois que Nagios a décidé de vous envoyer une alerte, il faut qu'il choisisse sous quelle forme cette alerte sera envoyée.

Ceci est défini par une commande externe, associée à une commande interne d'envoi d'alerte qui elle-même est associée à l'utilisateur.

Par exemple, la plupart des utilisateurs auront par défaut une commande associée d'envoi d'alerte par mail pour les problèmes de services appelée « **notify-by-email** ».

Cette commande est comme toutes les commandes Nagios, une ligne de commande Unix que Nagios va exécuter en remplaçant à la volée un grand nombre de paramètres par des constantes et des variables propres à l'hôte, au service, et à l'utilisateur. Ainsi, si nous examinons ce notifiy-by-email tel qu'il est fourni par Centreon :

```
/usr/bin/printf "%b" "***** centreon Notification *****\n\nNotification Type:
$NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost: $HOSTALIAS$\nAddress:
$HOSTADDRESS$\nState: $SERVICESTATE$\n\nDate/Time: $DATE$ Additional Info :
$SERVICEOUTPUT$" | /bin/mail -s "** $NOTIFICATIONTYPE$ alert - $HOSTALIAS$/
$SERVICEDESC$ is $SERVICESTATE$ **" $CONTACTEMAIL$
```

Nous pouvons constater qu'il est constitué en fait d'une première commande :

```
/usr/bin/printf "%b" "quelques chose"
```

qui va faire une sorte d'echo d'une chaîne vers la sortie standard (le %b ici signifie que le format de la deuxième chaîne peut contenir des échappements, comme \n pour retour à la ligne). Cette sortie standard sera récupérée en entrée standard pour la commande suivante grâce au pipe Unix :

```
|
```

Cette deuxième commande est la commande d'envoi d'email sur un système Unix :

```
/bin/mail -s "sujet du mail" adresse_email
```

Il est alors possible en remplaçant les caractères « \n » par des retours chariots de voir quelles sont les variables utilisées pour composer le contenu du mail d'alerte :

```
***** centreon Notification *****

Notification Type: $NOTIFICATIONTYPE$

Service: $SERVICEDESC$
Host: $HOSTALIAS$
Address: $HOSTADDRESS$
State: $SERVICESTATE$

Date/Time: $DATE$ Additional Info : $SERVICEOUTPUT$
```

Vous voyez alors qu'il est relativement simple d'éditer cette commande pour la traduire ou y ajouter d'autres informations intéressantes (http://doc.monitoring-fr.org/3_0/html/thebasics-macrolist.html).

Par exemple, les macros $SERVICEACTIONURL$, $SERVICENOTESURL$ et $SERVICENOTES$ peuvent servir à stocker des url externes, comme les pages du wiki où vous stockez la description du service ou de l'hôte, ou les pages de procédures d'urgence.

Ce type d'informations est très utile en cas d'alerte. Ces macros peuvent elles-mêmes contenir les macros $HOSTNAME$ ou $SERVICEDESC$, ce qui peut vous donner des valeurs comme http://monwiki.example.net/hosts/$HOSTNAME/wtf où vous stockez les infos qui dépanneront vos collègues et vous permettront de passer de bonnes vacances.

Nous avons décrit ici la commande d'envoi d'email. Mais on voit bien que Nagios ignore en fait complètement ce que va faire cette commande, on peut donc écrire une infinité de commande, certaines le sont déjà, pour envoyer des alertes vers des système de messagerie instantanées (XMPP, Jabber, ICQ, etc), ou pour envoyer des SMS.

L'envoi de SMS se gère de deux façons :

- soit vous utilisez une passerelle de type service web ou mail-to-sms. En prenant en compte le risque que la communication vers cette plate-forme externe soit indisponible au moment de l'alerte ;
- soit vous installez un téléphone GSM avec un accès GSM (carte SIM) à côté de votre serveur, et vous reliez les deux par un protocole de communication (série, usb, etc). Vous devrez alors chercher un peu sur internet pour trouver des scripts capables de dialoguer avec les différents modèles de téléphones existants.

On retrouve au bout de ce système de notification un principe de flexibilité cher à Nagios : il délègue la tâche à une commande que vous pouvez trouver déjà écrite par d'autres utilisateurs ou que vous pouvez écrire vous-même. Lui ne se charge que de remplacer les variables de la commande par les valeurs dont il dispose.

➲ *Si Nagios doit disposer d'informations complémentaires, comme par exemple l'adresse de l'utilisateur sur un système de messagerie « bizarre », ou bien un numéro, un code,* ***il est toujours possible d'ajouter des macros personnalisées dans Nagios (variables/attributs personnalisés)****, pour les associer aux utilisateurs, aux services ou aux hôtes et y stocker ces informations supplémentaires.*

10.1.6. Et le flapping?

Le flapping est un élément dont vous n'aurez peut-être jamais besoin. Il s'agit d'un système prévu pour réduire le nombre d'alertes émises par des services subissant un grand nombre de changement d'états.

Imaginez que vous ayez un service qui à chaque nouvelle vérification par Nagios change d'état (OK, puis WARNING, puis CRITICAL, puis OK, puis CRITICAL, etc). En terme de notifications, même si vous avez mis un notification_interval à 0, vous allez recevoir une nouvelle notification à chaque changement d'état. Ce qui peut être très gênant.

Qu'est ce qui peut amener un service à changer autant d'état ?

- Un mauvais réglage des seuils : vous avez un sonde de PING par exemple qui alerte en warning à 400ms, et en critique à 500ms, et toute la journée votre ping oscille entre 300ms et 502ms. Il ne vous reste plus qu'à augmenter vos seuils pour régler votre problème.
- Un service supervisé dont l'état est vraiment très instable. A priori on se dit qu'il faudrait régler l'instabilité de ce service.

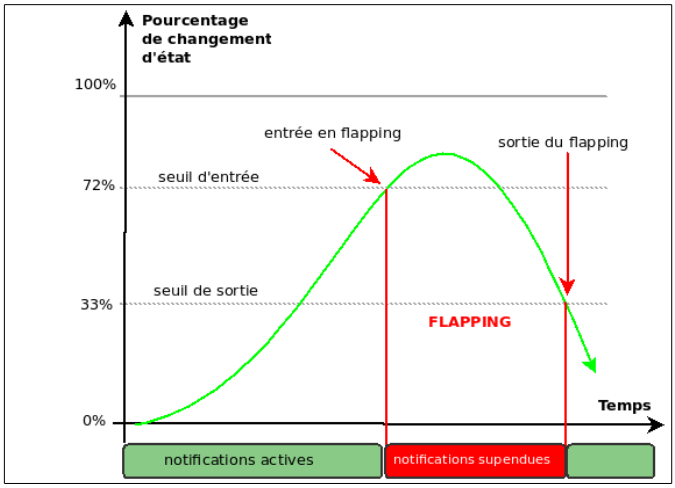
  Le problème est que parfois la connaissance d'un problème n'entraîne pas la mise en œuvre immédiate de la solution (problèmes de budget, de disponibilité, de circuit décisionnaire, etc.). Et en attendant, l'équipe des administrateurs est submergée d'emails issus de la supervision sur ce service.

  La conséquence probable de ceci sera que les alertes seront suspendues sur ce service. Ce qui peut devenir problématique.

Le flapping répond à cette problématique en mesurant **un pourcentage de changement d'état** sur les derniers résultats de mesure et en **suspendant** les alertes quand ce pourcentage de

changement d'état (de « flapping ») est trop fort, puis en les réactivant quand ces changements d'états se font moins fréquents..

La flapping se définit avec deux seuils, un seuil de changement d'état à partir duquel on rentre en état de flapping (notifications suspendues) puis un seuil à partir duquel on retourne au mode normal de notifications.



Entrée et sortie du mode flapping

10.1.7. Acquitter des alertes

Si vous avez déjà examiné attentivement la Vue tactique de Nagios, vous aurez remarqué que les alertes sont recensées différemment selon qu'elles sont acquittées ou non. Acquitter les alertes est un moyen très efficace :

- de signaler à Nagios que certaines sondes sont dans un état différent de OK mais qu'il s'agit d'un problème de paramétrage et non d'une vraie alerte ;
- de signaler à vos collègues que vous prenez en charge la résolution d'un incident ;
- d'empêcher les notifications suivantes sur cette alerte, et notamment d'empêcher les escalades.

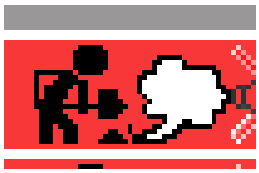
Dans une supervision vivante il existe toujours des services en alerte, en état critique ou unknown et qui sont là depuis un temps assez long, soit parce qu'on a pas encore eu le temps de parfaire la mise en place de nouvelles sondes, soit parce que la mise en place d'un serveur n'est pas encore vraiment terminée.

Il est très important de pouvoir sortir ces sondes des écrans principaux d'alertes, toujours dans l'optique de ne pas noyer l'information utile dans des informations inutiles.

Lors de l'acquittement des alertes, vous pouvez de plus ajouter des commentaires sur le service. Vous pouvez d'ailleurs toujours ajouter des commentaires sur un hôte ou service, y compris sans l'acquitter.

On retrouve dans ces fonctionnalités des outils de supervision opérationnelle extrêmement pratiques.

Dans de nombreux écrans des distinctions très claires seront faites entre les problèmes acquittés et ceux qui ne le sont pas.



Icônes Nagios du thème classique pour l'acquittement et la présence de commentaires

11. Comment Nagios va effectivement superviser ?

A ce stade de votre compréhension ou découverte de Nagios, l'aspect le plus important est maintenant de comprendre comment vont s'effectuer les vérifications. Quelles sont les commandes qui renverront des états de supervision ? Sur quelles machines ces commandes devront tourner pour collecter des informations ? Comme toujours avec Nagios, il n'y a pas qu'une seule méthode et plusieurs solutions sont disponibles.

Nous commencerons par étudier les check actifs, ou le mot **actif** signifie que Nagios décide lui-même d'ordonnancer l'exécution de la vérification.

11.1. Supervision Active Locale

Commençons par la méthode la plus simple et la plus évidente : les **checks actifs locaux**. Cela représente l'ensemble des sondes que Nagios peut exécuter en totalité sur le serveur de supervision pour remonter le statut d'un service ou d'un hôte accessible depuis ce serveur.

On va y retrouver deux familles de vérifications:

- les sondes capables de surveiller l'état du serveur de supervision (Dans quel état sont les disques de ce serveur ? Quelle est la charge du CPU ? Quel est l'état du trafic réseau ? Tel ou tel processus est-il bien présent ? Etc.) ;
- les sondes utilisant tout ce qui est visible depuis ce serveur. Les pings, l'accès à des services à travers TCP/IP (est-ce que le serveur de mail répond sur le port 25 si je l'interroge depuis le serveur de supervision ? Combien de temps faut-il pour obtenir une réponse d'un site web depuis le serveur de supervision ? Etc.).

Votre serveur de supervision est capable de vérifier à partir de lignes de commandes très simples de nombreuses choses qui lui sont accessibles directement. Il n'est par exemple pas nécessaire de disposer d'un client web graphique ou d'un client email graphique pour tester le fonctionnement final d'un site web ou d'un serveur de mail. Le serveur de supervision teste le service proposé sur le réseau comme n'importe quel client local du réseau.

De nombreux checks sont disponibles pour effectuer toutes sortes de vérifications locales, et parmi ceux livrés avec Centreon, ou disponibles sur le site de Nagios ou de Nagios exchange (cf. annexes), un grand nombre sont capables de fonctionner depuis le serveur de supervision.

Il faut alors garder à l'esprit que l'exécution de ces checks représentera une charge en terme de mémoire et de CPU pour le serveur de supervision.

11.2. Supervision Active Distante

Malheureusement, il est impossible de tout vérifier à distance. Il n'existe pas de programme pouvant tourner en local sur un serveur linux capable de vérifier l'état des disques d'un serveur Windows (en dehors des méthodes SNMP que nous verrons dans le point suivant). La deuxième grande famille de vérification pourra donc se nommer **check actifs distants**.

L'idée est alors d'utiliser en local un check dont le travail est de déporter l'exécution d'un second check. Nagios va exécuter une commande locale dont le travail sera d'effectuer une connexion à un hôte distant, de faire exécuter un check sur cette machine distante, puis de rapatrier le code retour et la chaîne de texte de résultat localement afin de la transmettre à Nagios.

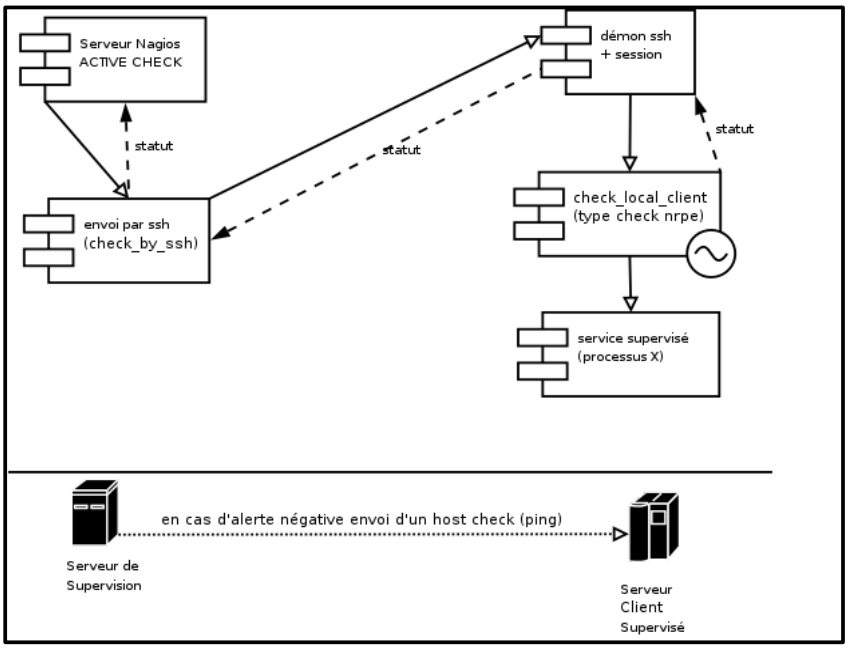
➲ *Ce type de fonctionnement impose un déploiement sur l'hôte distant des checks que l'on veut y exécuter*

Le travail de la sonde locale est alors très léger en terme de charge pour Nagios (ouvrir la communication, transmettre la commande, récupérer le résultat), et l'essentiel de la charge de supervision est alors effectué par le serveur distant.

Les checks capables d'effectuer ce déport de commande sont principalement:

- **check_by_ssh :** à ne pas confondre avec check_ssh qui vérifie simplement qu'un serveur ssh répond sur le port 22. check_by_ssh va exécuter une deuxième commande après avoir ouvert une connexion ssh sur le serveur distant. Il faut que l'hôte distant dispose d'un démon SSH (qui peut exister sous Windows aussi) ;
- **check_nrpe :** NRPE est un protocole TCP/IP chiffré ou non, créé pour Nagios. Il faut que le serveur distant dispose d'un client NRPE (comme par exemple NSCLient++ sous Windows).

Voici les schémas d'exécution avec ces checks déportés :



Check actif distant avec SSH

Voici ce à quoi peut ressembler une commande Nagios avec des check déportés encapsulés dans ces checks particuliers:

```
check_by_ssh -H 192.168.10.15 -i /home/nagios/.ssh/id_dsa -t 8 -C
"/usr/local/nagios/plugins/check_load -w 10,5,5 -c 15,10,10"
```

et

```
check_nrpe -H 192.168.10.15 -p 5666 -t 8 -n -c check_disk -a 30% 20% /dev/sdb1
```

Nous remarquerons que les checks NRPE ont un moyen très particulier de passer des arguments (option -a et espaces).

➲ *Si vous utilisez NRPE avec NSCLient++ sur Windows, vérifiez bien dans le .ini de ce programme, section NRPE, que vous autorisez les arguments.*

11.3. Méthodes Mixtes et avancées

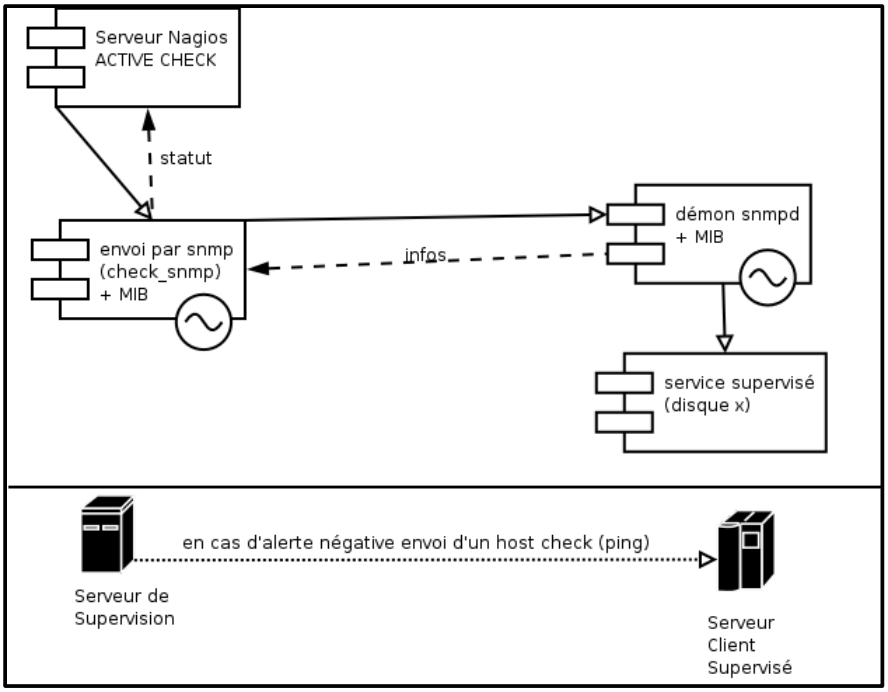
11.3.1. Checks semi-distants, semi-locaux, SNMP et WMI

Il existe aussi des protocoles TCP/IP dédiés à la supervision comme SNMP et WMI. Grâce à ces protocoles, de nombreuses informations qui nécessiteraient des checks actifs distants vont être diffusées à distance à destination du serveur de supervision.

SNMP est utilisable en mode passif comme en mode actif.

Nous détaillerons dans quelques paragraphes ce qu'est SNMP, mais retenez qu'il n'existe pas que les traps SNMP. Vous pouvez utiliser SNMP pour interroger de nombreux équipements (routeurs, imprimantes, serveurs) et de nombreux systèmes d'exploitation (linux, unix, Windows, etc.) sur un langage commun. Ce langage se nomme SNMP MIB II et va permettre de retrouver sans aucun effort, sur tous ces éléments disparates, entre autres :

- des informations sur l'occupation des disques et de la mémoire ;
- des informations détaillées sur les processus ;
- des informations détaillées sur le trafic réseau ;
- des informations sur le load average ;
- etc.



Il est alors utile d'interroger directement le service SNMP distant, d'en extraire un grand nombre d'informations, puis d'effectuer un travail en local permettant d'indiquer des seuils d'alertes.Check actif SNMP

On voit alors que la charge de travail est mixte, d'un côté le démon ou le service SNMP de l'hôte supervisé est mis à contribution pour collecter des informations et nous les transmettre, de l'autre un check local agrège ces informations, et calcule l'état d'une sonde de disque ou de processus pour nous renvoyer un état critique, warning, ok, etc.

Le check SNMP est donc mixte, mais on peut aussi utiliser un check SNMP pour brancher l'exécution d'un script distant lors de l'interrogation d'une adresse SNMP définie (un OID), il est alors possible d'utiliser SNMP aussi comme un moyen de déporter l'exécution d'une sonde distante.

Les fichiers de configuration du démon SNMPD sur Debian détaille de façon assez précise les différents moyens de faire exécuter des scripts par le démon SNMP. Dans la pratique, ce n'est pas l'utilisation principale de SNMP.

SNMP est souvent un domaine assez mal compris, nous détaillerons donc dans une autre partie ce que SNMP recouvre.

11.3.2. Proxy Check

Au niveau des checks distants, nous pouvons citer une utilisation avancée que l'on nome **proxy check** et qui consiste à ajouter un niveau supplémentaire d'encapsulation.

Imaginons que le serveur de supervision A est incapable d'accéder à une machine C, mais qu'il peut accéder par SSH à une machine B, qui elle même peut accéder en SSH à C.

Schématiquement pour faire tourner le check check_dummy sur C il faudrait une commande check_nrpe(check_dummy,C) lancée sur B. Et pour obtenir cela Nagios va lancer sur A check_by_shh(check_nrpe(check_dummy,C),B).

11.3.3. Check Multi, check_cluster, check_dummy

check_dummy est un check très utile pour tester des services qui renvoient toujours un état précis, par exemple pour tester les notifications avec un service qui à chaque exécution de sa commande va recevoir un état CRITICAL. Check_dummy prend en premier argument le code numérique équivalent à OK/WARNING/CRITICAL/UNKNOWN (donc 0, 1, etc.) et en deuxième argument la chaîne de texte à renvoyer. Et le retour de ce check sera cette chaîne de texte avec le code retour indiqué.

check_multi est un check capable de sérialiser l'exécution d'un ensemble de checks en une seule commande, et de ramener l'ensemble des résultats sur plusieurs lignes (ce que Nagios3 autorise dorénavant). Ceci peut permettre de ne faire exécuter et planifier qu'une seule commande à destination d'un hôte et d'y regrouper un ensemble de vérifications. Le code du service sera alors le code le plus critique de l'ensemble des résultats obtenus. Ceci peut aussi servir à regrouper ensemble des services au fonctionnement proche. En fait, ce check peut servir à beaucoup de choses, c'est une sorte de couteau suisse de l'avis même de son auteur.

Dans le même genre de checks très utiles, on trouvera **check_cluster** qui est un check actif local, qui va travailler directement sur les résultats d'autres checks. Ce check va prendre en argument une liste des résultat de sondes effectuées sur des services ou hôtes, en piochant directement dans la valeur connue de ces services ou hôtes pour Nagios, et va afficher un état global à OK/WARNING/CRITICAL en fonction du nombre de ces éléments qui sont en état de répondre. Ainsi un service Web qui dispose de 5 serveurs frontaux dont 2 au moins devraient être allumés (Warning), et qui devient critique quand moins de deux serveurs sont en état de répondre pourra utiliser les résultats de services *S_http_response* lancés sur ces 5 serveurs (et qui n'émettent jamais d'alerte) pour décider d'avertir à partir de 3 serveurs éteints :

```
(\ signifie retour à la ligne qu'il n'est nécessaire de faire dans la vraie
commande))
check_cluster -s -d $SERVICESTATEID:frontal1:S_http_response$, \
$SERVICESTATEID:frontal2:S_http_response$, \
$SERVICESTATEID:frontal3:S_http_response$, \
```

```
$SERVICESTATEID:frontal4:S_http_response$, \
$SERVICESTATEID:frontal5:S_http_response$ -w 3
```

Pour mieux comprendre cette syntaxe vous pouvez tester check_cluster avec les équivalents numériques des résultats des sondes :

```
check_cluster -s -d 0,0,1,2,2,0 -w 2 -c 4
CLUSTER WARNING: Service cluster: 3 ok, 1 warning, 2 critical
```

La syntaxe :

```
$SERVICESTATEID:H-bar:S-foo$
```

permet donc de retrouver l'équivalent numérique de l'état OK/WARNING/CRITICAL du service S-foo sur l'hôte H-bar.

11.4. Supervision Passive

Toutes les façons d'exécuter des sondes et de récolter les résultats examinés jusque là étaient classées comme Actives. Nagios se chargeait de planifier et de lancer les vérifications. Or il existe de nombreux cas où attendre le résultat d'une opération est plus intelligent que d'aller chercher son résultat toutes les 5 minutes. En voici une liste (non exhaustive) :

- savoir si un backup a été réalisé et s'il s'est bien déroulé ;
- détecter une alerte grave dans un log ;
- les TRAPS SNMP. Les services SNMP sont souvent capables d'émettre des TRAPS, il s'agit alors d'une alerte déclenchée depuis le service au moment ou elle survient et envoyée à l'hôte chargé de la supervision.

Par extension, Nagios considère que toutes les saisies effectuées par les opérateurs humains dans l'interface de supervision sont des opérations passives : indiquer un acquittement d'alerte, replanifier l'exécution d'une sonde, forcer la valeur d'une sonde.

➲ *Si vous n'autorisez pas les informations passives pour un service vous ne pourrez donc pas forcer sa valeur à travers l'interface de supervision*

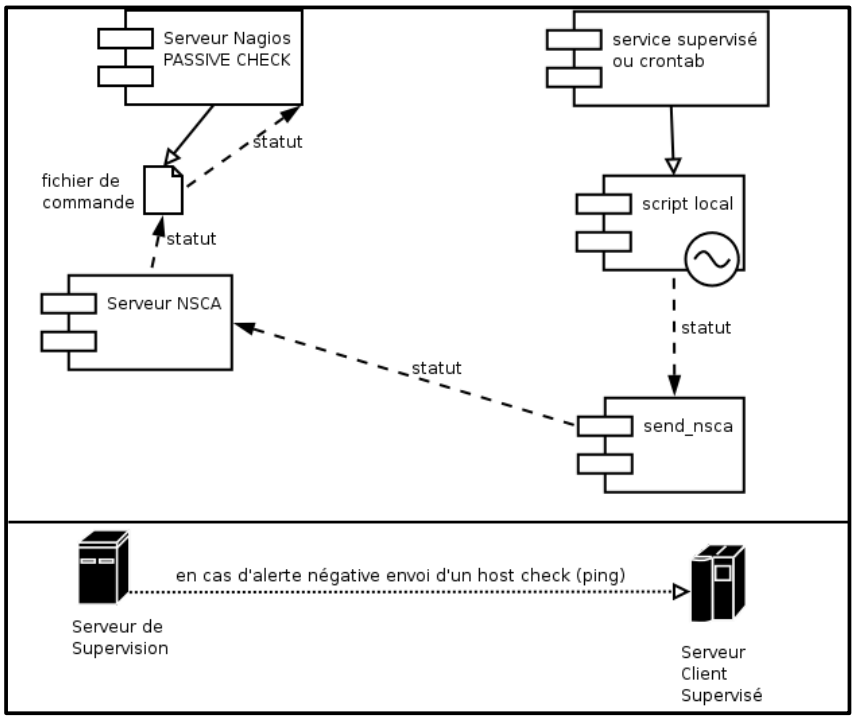
Pour gérer ces informations passives en provenance de l'opérateur, Nagios dispose d'un fichier qu'il traite comme une file d'attente d'informations, le **fichier de commande** (une pile FIFO). Régulièrement, Nagios vérifie le contenu de ce fichier, dépile les informations qu'il contient, et les intègre à sa base de connaissance de la supervision (en fait il s'agit par défaut d'un horrible fichier texte, le fichier de statut, contenant la base de données complète de tous les services et hôtes avec le détail complet de tout ce qui est connu pour chacun). Ce décalage dans la lecture des informations passives explique le message de Nagios quand vous forcez un état d'un service, il

vous indique qu'il peut se produire un décalage dans le temps avant que cette information soit prise en compte. Nous sommes là assez loin d'une interface Web 2.0 fonctionnant en Ajax où le serveur d'application prendrait immédiatement en compte vos ordres.

11.4.1. NSCA

Ce fichier de commande est constitué de lignes de texte assez simples, constituées d'un timestamp, d'un nom d'hôte, de service, et de l'état à associer à ce service ou cet hôte.

Il existe des scripts qui écrivent directement dans ce fichier de commande. Mais il existe surtout un démon nommé **NSCA** (Nagios Service Check Acceptor) dont la tâche principale est d'écrire des commandes dans ce fichier de commande. Ce démon est un service TCP/IP écoutant sur le port 5667 et qui répond à un protocole TCP/IP qui peut être chiffré et qui est nommé NSCA. Ce protocole est dédié à la collecte d'informations passives sur les services et hôtes de Nagios. La façon la plus simple de communiquer avec le démon NSCA est le script **send_nsca** qui est disponible pour Linux et Windows.



Check Passif

Voici par exemple une commande Linux lançant « send_nsca » pour envoyer au serveur de supervision (192.168.10.52) le résultat du service nommé « S_myService » pour l'hôte nommé « H_myHost ». Le résultat sera 0 (OK) avec la chaîne associée « a result string message » :

```
/usr/sbin/send_nsca -H 192.168.10.52 -c /etc/send_nsca.cfg 'H_MyHost'
'S_myService'    0    'a result string message'
```

11.4.2. Fraîcheur, Volatilité

A partir du moment où vous utilisez les services passifs de nouveaux concepts apparaissent.

Ainsi le concept de fraîcheur devient extrêmement important.

Reprenons l'exemple des sauvegardes. Sur votre serveur de backup, dans les procédures de sauvegardes, vous pouvez très certainement ajouter l'exécution de commandes à la fin d'un backup.

On y ajoutera la commande send_nsca envoyant un état OK pour le service Nagios S_BACKUP sur l'hôte qui vient d'être sauvegardé.

Si votre logiciel de backup est bien fait, il vous propose aussi l'exécution de scripts personnalisés en cas de sortie en erreur. Vous pourrez alors en profiter pour envoyer le statut CRITICAL par send_nsca et peut-être aussi un message indiquant cette erreur. Mais que se passera-t-il si le serveur de backup s'éteint, ou si un backup met un temps extrêmement long à se terminer (parce que par exemple il transfère plusieurs tera de données dont il ne devrait pas faire le backup) ?

Au niveau de Nagios, la dernière information connue sur ce service S_BACKUP pour un hôte donné peut commencer à dater. La notion de **fraîcheur** va permettre de dire qu'au bout de, par exemple, **30 heures sans information nouvelle, le service se doit de passer automatiquement à CRITICAL**.

La **volatilité** est un autre concept qui intervient plutôt sur des services du type « vérification de logs ou de gestionnaire d'évènements ». La volatilité d'un service signifie pour Nagios que chaque résultat reçu est indépendant des résultats précédents. Votre service qui reçoit régulièrement des alertes passives en CRITICAL quand une ligne « ERROR:* » est relevée dans un log particulier devrait vous envoyer une nouvelle notification à chaque fois que cet événement se produit. Il s'agit pour chacun des événements d'une alerte qui n'est pas reliée à la précédente (à la différence d'un serveur de base de données qui est tombé depuis une heure, dans 5 minutes la sonde qui dira CRITICAL signalera la continuité de l'alerte).

11.5. Plus de détails sur SNMP

11.5.1. SNMP v1, v2c, v3

SNMP est un protocole de supervision ancien et donc très répandu . La plupart des équipements actifs, comme les switchs et les routeurs supportent SNMP.

Il s'agit d'un protocole TCP/IP ou UDP dédié à la supervision, les ports concernés sont le 161 et le 162 (pour les traps).

Plusieurs versions du protocole existent :

- SNMPV1: première version, pas de chiffrement MIB-I ;
- SNMPV2c: ajout des INFORM (traps avancés) et meilleure gestion des erreurs ; GETBULK, MIB-II. Remarquez le petit « c » : certaines commandes comme snmpwalk veulent « 2c » en numéro de version et non « 2 » ;
- SNMPV3: ajout de chiffrement de communication et des mots de passe de connexion.

Si l'on met de côté la version 3, l'accès aux informations diffusées par SNMP est filtré par la « **communauté** », qui est un mot partagé, « **public** » par défaut. En fait, la communauté public donne accès au niveau « read-only ». Il existe deux autres niveaux, le niveau privé en lecture/écriture (« **private** », communauté private) qui est le plus souvent désactivé (heureusement, vous avez accès à toute la configuration système), et le niveau d'accès « **trap** » pour les traps que nous expliquerons plus avant.

Les principaux défauts de SNMPv3 sont premièrement sa jeunesse : il est très peu répandu, et parfois mal implémenté , souvent non prévu dans les commandes client. Deuxièmement, il refait le travail de SSH ou SSL. SNMP over SSH est encore en phase d'étude, au lieu d'utiliser une surcouche SSL capable de déterminer via l' auto-négociation les protocoles à utiliser, SNMP v3 a choisi de ré-implémenter tous les principes d'une communication sécurisée sans l'auto-négociation.

Son troisième grand défaut est donc qu'il complexifie grandement l'écriture des commandes. Au lieu d'utiliser simplement une communauté « public », vous devrez indiquer :

- un login ;
- un mot de passe ;
- un algorithme de chiffrement du mot de passe ;
- un mot de passe de communication ;
- un algorithme de chiffrement de la communication.

Ceci peut expliquer son usage relativement faible.

SNMP est un monde à part, il possède son propre vocabulaire, parmi les mots les plus importants que vous pourrez rencontrer, citons :

- **SNMP**: Simple Network Management Protocol ;
- **TRAP**: message envoyé par un client SNMP de son propre chef ;

- **NMS**: Network Management Station: reçoit les traps et émet des requêtes SNMP ;
- **MIB**: Management Information Base (bibliothèque de configuration).

11.5.2. La MIB II

Les MIB sont les bibliothèques contenant les éléments interrogeables via SNMP.

Il existe de nombreuses MIB, chaque appareil va pouvoir disposer d'une MIB dans lequel il indiquera ce qu'il souhaite mettre à disposition via SNMP. Mais les MIB les plus importantes sont les MIB standardisées. Puisqu'on y retrouve grosso modo les mêmes informations quelque soit l'équipement (donc autant un windows XP SP2 qu'un routeur Cisco ou une distribution Linux Unbuntu dernier cri)

La plupart des équipements respectent la MIB-II qui permet de retrouver de nombreuses informations, comme :

- la liste des processus (avec informations mémoire et CPU) ;
- les débits réseaux ;
- le load average ;
- l'utilisation mémoire ;
- le contact du système, l'uptime ;
- l'utilisation de l'espace disque ;
- les tables de routages.

Voici des adresses numériques indiquées dans la MIB II :

.1.3.6.1.2.1.1 system : uptime, contact, nom

.1.3.6.1.2.1.2 interfaces: interfaces réseau

.1.3.6.1.2.1.4 ip: ip routing, etc

.1.3.6.1.2.1.5 icmp: icmp errors, discards... (icmp c'est par exemple le ping)

.1.3.6.1.2.1.6/7 tcp ou udp: états des connexions tcp ou udp

On remarquera que ces arborescences sont chiffrées, on les nomme **OID**. Les MIB fournissent une explication textuelle de ces chiffres et des informations que l'on pourra trouver en face de ces chiffres. Sans la MIB rien ne vous interdit de consulter un OID, il est simplement humainement plus complexe de comprendre la réponse 42 donnée en face de 1.3.6.1.4.1.9.42.5252.6.3.6.25.102.2 (par exemple).

L'OID .1.3.6.1 peut donc aussi s'écrire iso.org.dod.internet : iso(1).org(3).dod(6).internet(1)

Certains équipements ou constructeurs fournissent donc leurs propres MIB définissant une branche dans l'arbre des OID, avec un numéro officiel défini auprès de l'IANA (Internet Assigned Numbers Authority) . On retrouvera par exemple CISCO à cette branche de l'arbre:

iso.org.dod.internet.private.entreprise.cisco = 1.3.6.1.4.1.9

9 est donc le numéro de CISCO. Tout ce qui sera écrit sous 1.3.6.1.4.1.9 est sous la responsabilité de CISCO qui publie certainement un très grand nombre de MIB expliquant le détail de l'arborescence. Microsoft dispose lui aussi d'un très grand arbre d'OID défini dans des MIB

Microsoft. Ceci ne vous empêche pas du tout d'interroger l'arbre de la MIB II sur une machine Microsoft ou CISCO (.1.3.6.1.2.1) pour y trouver les informations SNMP standard.

Les MIB elles-mêmes peuvent être lues par un être humain (pas trop normal comme être humain quand même), La syntaxe est ASN.1. (comme dans les schémas Active Directory ou LDAP) et permet de définir les types d'objets gérés (chaînes, entiers, listes, etc) et leurs associations aux OID.

11.5.3. Les traps et INFORM SNMP

Les traps SNMP sont définis dans les MIB. Ils sont lancés à travers le réseau, à destination du ou des serveurs de supervision.

Les traps peuvent être en **TCP** ou en **UDP**, l'UDP est très largement préférable car il est beaucoup plus léger pour le réseau. Voici par exemple un cas concret :

- un routeur détecte une congestion réseau ;
- il envoie un TRAP signalant cet état ;
- si le trap est en UDP, il s'agit d'une trame unique sur le réseau, qui arrivera ou non ;
- si le trap est en TCP la connexion va utiliser beaucoup de ressources réseau, d'autant que celui-ci est saturé, SYN;SYN/ACK; SYN/SYN/ACK, jamais de HACK en retour ;
- le routeur ne ferait qu'aggraver son problème de réseau.

Il faut donc voir le trap SNMP comme une bouteille à la mer qui pourra se perdre.

SNMPv2 propose les INFORM. Un INFORM est un TRAP qui attend un ACKNOWLEDGE (acquittement) et qui est relancé au bout d'un certain temps si aucun ACKNOWLEDGE n'a été reçu. TCP/IP est donc vraiment non nécessaire .

11.5.4. Outils SNMP

Les outils SNMP les plus connus sont écrits en PERL, il s'agit des SNMPTools :

- **snmpget** va permettre de récupérer une information avec une clef définie ;
- **snmpwalk** va permettre de récupérer un ensemble de clefs.

Il existe aussi un démon **snmptrapd** capable de capturer des traps (le plus souvent installé avec Centreon, qui intègre des services de gestion des traps).

Quand vous voulez tester une sonde SNMP, vous devriez toujours commencer par vous connecter en SSH sur le serveur de supervision, devenir l'utilisateur Nagios et tenter un snmpwalk à destination de l'hôte supervisé (ici 10.44.144.72 avec la communauté public) :

```
snmpwalk -v 2c -c public 10.44.144.72 .1
```

Remarquez le point devant le 1 dans cette ligne de commande, cela signifie que l'on va essayer de descendre l'arbre SNMP depuis sa racine qui est « **.1** ». Vous devriez obtenir en réponse une longue (très longue) liste d'informations, certaines traduites, d'autres uniquement chiffrées (s'il s'agit de MIB inconnues du système).

La commande snmpwalk pour ne lister que le contenu de la MIB II est :

```
snmpwalk -v 2c -c public 10.44.144.72 .1.3.6.1.2.1
```

Si vous obtenez une absence de réponse, la communication n'est pas établie (donc les sondes SNMP de Nagios ne marcheront pas), parmi les causes qui peuvent expliquer cela citons :

- un hôte **éteint** (testez le ping) ;
- un **firewall** qui bloque le trafic SNMP vers cet hôte ;
- **absence** de service ou de démon SNMP sur la machine distante ;
- le démon ou service SNMP distant **n'écoute** pas sur le réseau mais uniquement sur le localhost (c'est le cas par défaut sur debian, voir /etc/default/snmpd) ;
- la machine distante ne reconnaît pas le serveur de supervision comme une machine **autorisée** à se connecter (configuration du service ou démon) ;
- vous utilisez SNMP **v3**? Lisez le man de snmpwalk et testez toutes les options :-).

```
(-a PROTOCOL -A PASSPHRASE -e ENGINE-ID -E ENGINE-ID -l LEVEL -n CONTEXT -u
USER-NAME -x PROTOCOL -X PASSPHRASE -Z BOOTS,TIME)
```

11.6. Supervision pro-active

Enfin il existe dans le monde de la supervision un éden utopique qui se nomme « Supervision pro-active ». Dans ce monde, le système de supervision ne se contente pas de vérifier que tout se passe bien, et de signaler les problèmes, il intervient sur les problèmes afin de les régler sans intervention humaine.

De tels robots peuvent, par exemple, servir à rebooter un routeur en se servant de la fonction eject d'un lecteur CD :

http://thedailywtf.com/Articles/ITAPPMONROBOT.aspx

Plus sérieusement, la supervision pro-active consiste effectivement à agir automatiquement lors de la détection d'un problème. Dans Nagios, ceci se fait grâce aux **Events Handlers** (gestionnaires d'évènements).

Comme beaucoup de choses dans nagios, les gestionnaires d'évènements sont des scripts externes à Nagios. Il est donc possible de s'en servir pour faire à peu près tout et n'importe quoi. On pourrait, par exemple, les utiliser pour analyser de façon statistique le travail effectué par Nagios. Mais la tâche principale de ces event handlers est le plus souvent d'aller redémarrer des services tombés.

➲ *Il est très important de nommer les services ayant des event handlers de façon spécifique et de recenser les event handlers existants. Il est en effet très désagréable de constater qu'un robot redémarre des services que vous avez éteints manuellement pour effectuer des*

opérations de maintenance. L'existence des robots et l'étendue de leur tâche doit donc être connue de l'équipe d'administration.

Pour comprendre le fonctionnement des event-handlers, il n'y a quasiment qu'un seul concept à intégrer :

« Un event handler associé à un service sera exécuté après la commande du service à chaque fois, quelque soit le retour de cette commande »

Il ne faut donc surtout pas y coder un script de relance de service basique, il pourrait alors s'exécuter toutes les 5 minutes.

Des scripts event-handler d'exemple en Bash sont fournis avec Nagios et peuvent être étudiés. Ils peuvent être écrits dans à peu près n'importe quel langage et s'appuyer sur des clefs SSH, des OID SNMP, des commandes WMI ou des éjections de lecteur CD pour agir.

Le point important est qu'ils doivent accepter en paramètres des informations sur l'état de service et de l'hôte (CRITICAL/OK/WARNING? HARD ou SOFT? Tentative 2 sur 5 ? Adresse IP de l'hôte ?) et agir en conséquence.

Au niveau du paramétrage, nous trouverons souvent des commandes qui passent justement ces informations au script d'event handler sous cette forme :

```
$USER1$/eventhandlers/moneventhandler $SERVICESTATE$ $SERVICESTATETYPE$
$SERVICEATTEMPT$ $HOSTADDRESS$
```

A titre personnel, j'utilise le plus souvent SSH pour exécuter ce type de relance à distance.

Grâce aux clefs SSH, il est possible de générer une clef pour l'utilisateur Nagios pour chaque tâche différente. Puis d'associer la signature de cette clef à une commande unique au niveau du serveur visé et autorisé uniquement depuis le serveur de supervision.

On obtient alors une solution robuste en terme de sécurité.

Voici à quoi devrait ressembler le fichier ~/.ssh/authorized_keys2 de l'utilisateur utilisé sur l'hôte distant (il n'y a pas de retour à la ligne, la longue suite de lettres est la signature publique d'une clef SSH de l'utilisateur Nagios nommée id_dsa_relance_apache et stockée dans le HOME/.ssh/ de cet utilisateur Nagios sur le serveur de supervision) :

```
 #cat /root/.ssh/authorized_keys2
command="/etc/init.d/apache2 restart",from="10.1.11.25",no-x11-forwarding ssh-
dss
AAAAB3NzaC1kc3MAAACBAP5300tehtIGKafosm/qqbYaTYBtjEIUSM7mIngJDodAewBcPUPxJi6nL/2
7QF2x+4b/oECrMt2sxRRLXBU7EPdR+jJAAAAFQAAIAdUPy0sRfH1iLNmAF2FWqMNTLeewft6x1+4SNI
imhu0oYGoKsGvc6M+W+LRaeci0ia4389EDrEuDQPUIdJDBWd7cPytDNWxvkm5B+Easnuj6MpinkVvCe
FJ5XL79lIkSETv2QiY+lbRhpL2zW76mIDBJXOZcdAhX2BAGQSiRsbr1iKunZUfgZBKwAAAIA7oz2Hn+
Za3Ph0V+5xYAfx0lsk77gPeXio/StHbRhxV9QAQcJO0g6J1Ar0QNIMMBD4vGDIzCYwPFc/mY7piJsaJ
lGLL9WgOGC7c0perLg7o3vdxVdBnfyf+MHR85myIsBgQMVNdQPCwPahG03+CHNva91lJO0FYP1+2OYG
```

```
DNCOriGPOPJhkcgR4t5dkbttVL1sNZRnKoPzK8F33HMrkUanjJvlS45x/NkVtFIkPr21WpR944XTZIN
vmwuhU8/ODNRewNMl/3oJ3b9A== nagios@supervisor
```

12. Ressources Complémentaires

Cette introduction à Nagios vous aura permis, je l'espère, de mieux comprendre certains principes de la supervision. Certains détails ne sont qu'abordés (comme les dépendances). Et certains autres n'ont mêmes pas été abordés, comme les options de tuning ou l'utilisation des bases SQL avec NDO ou la supervision distribuée.

Une fois les grands principes connus, vous devriez cependant pouvoir naviguer parmi l'immensité des ressources disponibles et retrouver les éléments nécessaires.

12.1. Documentation Nagios

La documentation de référence est incontournable et détaille de façon très précise les différentes options.

http://nagios.sourceforge.net/docs/nagioscore/3/en/

On pourra en trouver une traduction française sur :

http://doc.monitoring-fr.org/

http://doc.monitoring-fr.org/3_0/html/index.html

12.2. Livres

- (fr) « **Nagios au coeur de la supervision Open Source** » « De l'installation à l'optimisation ».Auteur Olivier Jan. Éditeur ENI-édition.ISBN:978-2-7460-4632-0
- (fr)« **Nagios 3 pour la supervision et la métrologie** » « Déploiement, configuration et optimisation » Auteur Jean Gabès. Éditeur Eyrolles ISBN-10:2212124732 ISBN-13: 978-2212124736. Notez que Jean Gabès est l'auteur de Shinken.
- (en)« **Nagios: System And Network Monitoring** » Auteur Wolf Barth.Editeur: No Starch Press,US. ISBN-10 :1593270704 ISBN-13:978-1593270704

12.3. Sites Web

Sites Nagios officiels :

http://www.nagios.org

http://exchange.nagios.org/ (communauté, partage de sondes et autres)

Sites Français :

http://www.monitoring-fr.org/

http://wiki.monitoring-fr.org/

http://forums.monitoring-fr.org/

http://blog.nicolargo.com/

http://www.cfsl-asso.org/

Projets Annexes :

http://www.centreon.com/

http://forge.centreon.com/

http://www.shinken-monitoring.org/

https://www.icinga.org/

http://www.thruk.org/

http://mathias-kettner.de/checkmk_multisite.html

http://fannagioscd.sourceforge.net/wordpress/ (FAN)