HES-SO // MASTER

Projet d'approfondissement,
orientation Technologies de l'information et de la communication (TIC)

PLUGIN POUR SHINKEN

rédigé par
LIONEL SCHAUB

Sous la direction de
Prof. Gérald Litzistorf

de la MRU TIC de hepia

8 Juin 2012

MSE | MASTER OF SCIENCE IN ENGINEERING

Plugin pour Shinken

Responsable	Litzistorf Gerald
MRU	TIC / hepia
Orientation	TIC
Axes technologiques concernés	TIC / Ingénierie logicielle TIC / Systèmes d'information et multimédia TIC / Réseaux d'entreprise et sécurité IT
Résumé	La haute disponibilité d'un système d'information d'entreprise exige compétences humaines et outils appropriés. Au quotidien un superviseur, capable d'afficher en temps réel l'état des composants (serveur, proxy, firewall, ?) ainsi que l'historique des événements, facilite le diagnostic en cas de panne et produit des rapports destinés au management sur la qualité de service. Cette étude permettra au laboratoire, qui utilise actuellement Nagios, de migrer vers le produit Shinken Elle est proposée conjointement par deux des membres du projet Shinken, David Hannequin et Jean Gabès, et a comme objectif final d'obtenir un système de supervision hautement disponible et mettant en avant les possibilités de corrélations de l'outil afin d'aider à l'aide de corrections des problèmes sources et ainsi, améliorer la disponibilité du système d'information.
Cahier des charges	Ce projet d'approfondissement (PA) comprend les étapes suivantes : - Étudier le système Nagios en production au laboratoire et son système de plugin - Étudier le plugin Nagios check_mk - Configurer Shinken pour remplacer le Nagios existant - Proposer le cadre applicatif et l'architecture du plugin Shinken à développer - Développer et tester ce plugin - Produire le rapport final au format pdf et les fichiers utiles sur CD Sous réserve de modification en cours de projet
Connaissances préalables	
Mots-clés	TIC Développement software; TIC Programmation répartie; TIC Système d'information; TIC Systèmes distribués; TIC Sécurité

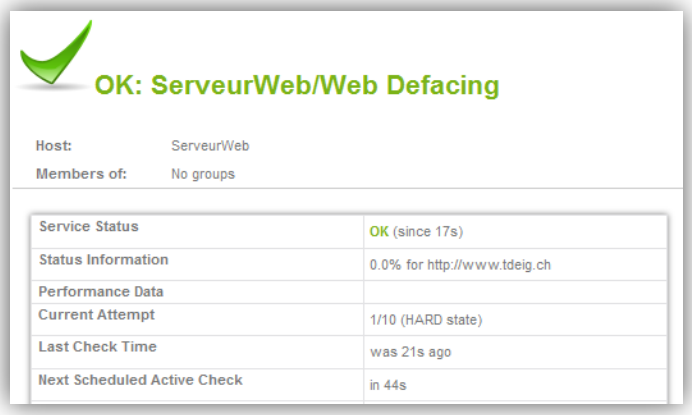
MSE | MASTER OF SCIENCE IN ENGINEERING

Printemps 2012

Projet d'Approfondissement

Résumé: Plugin pour Shinken

Le but de ce travail est, dans un premier temps, d'effectuer une étude des outils de supervision de systèmes informatiques et plus particulièrement de l'outil Shinken. Puis dans un second temps, de développer un plugin pour Shinken.



Ce travail s'est déroulé sur quatorze semaines et a été réalisé en quatre étapes:

- Choisir une distribution Linux, installer Shinken et définir un scénario simple.
- Mettre en pratique ce scénario et effectuer un rapport technique sur l'architecture et le fonctionnement de Shinken.
- Effectuer un état de l'art des solutions de monitoring.
- Développer un plugin pour Shinken.

L'analyse des outils de supervision commence par une recherche des besoins en mesure de disponibilité.

L'étude de Shinken comprend une analyse des processus et des connexions réseau utilisées par les différents daemons. Cette étude comprend également une analyse des possibilités de configuration de Shinken.

Durant les premières semaines, un scénario simple a été conçu dans le but de tester l'architecture et le bon fonctionnement de Shinken. Une attention particulière a été apportée à l'analyse des différents délais présents dans l'ordonnancement et l'exécution des tests.

Le plugin développé doit être capable de détecter si un site web a été victime d'un web defacing. Ce plugin doit, par conséquent, proposer une méthode efficace de comparaison de pages web. Il doit également être capable de détecter une modification malveillante qui modifie radicalement l'apparence de la page web tout en modifiant très peu le code source.

Étudiant: Schaub Lionel

Table des matières

1. Avant-Propos

1.1. Règles typographiques

Ce document a été rédigé avec la police d'écriture Calibri 10pt, en suivant les règles typographiques suivantes:

- Les liens vers des pages web sont en bleu souligné (http://example.com).
- Les liens vers des emplacements dans ce document sont soulignés (Règles typographiques).
- Les légendes sont en police Calibri 9pt gras et de couleur bordeaux (**Légende**).
- Les notes de bas de page sont en police Calibri 10pt.
- Certains mots importants sont en gras (**important**).
- Le contenu des fichiers texte est en police Courier New 9pt sur fond gris clair (`Fichier texte`).
- Les commandes sont en police Courier New 10pt (`Commande`).
- Les chemins pointant vers de fichiers ou des dossiers sont en italique (*/tmp/mon-fichier*).
- Les couleurs des captures d'écran de la console ont été inversés pour obtenir du noir sur blanc et ainsi améliorer la lisibilité et éviter un gaspillage inutile d'encre/tonner.

1.2. Lexique

Ce lexique définit certains termes utilisés dans ce document:

- **Supervision**: En informatique la supervision est la surveillance du bon fonctionnement d'un système, d'un service ou d'une activité.
- **Outil de supervision**: Un outil de supervision est un programme permettant d'accomplir une ou plusieurs tâches utiles dans le cadre d'une supervision informatique.
- **Solution de supervision**: Une solution de supervision regroupe un ou plusieurs outils de supervision dans le but de former un ensemble répondant aux attentes en matière de supervision.
- **Machine**: Dans ce document, le terme machine correspond à tout périphérique informatique connecté au réseau (routeurs, serveurs, ordinateurs de bureau, machines virtuelles, …).
- **Test**: Dans le contexte de Shinken ou de Nagios, un test correspond à l'exécution d'un plugin.

1.3. Remerciements

Je tiens à remercier M. Gérald Litzistorf pour m'avoir suivi dans ce travail. Je tiens tout particulièrement à le remercier pour l'aide qu'il m'a apporté dans la rédaction de ce document.

Je remercie aussi M. Sébastien Pasche pour avoir donné l'idée d'effectuer ce PA (Projet d'Approfondissement) sur Shinken.

Je remercie également M. Jean Gabes (créateur de Shinken) pour avoir pris le temps de répondre à mes questions concernant les sources de Shinken.

Je tiens aussi à remercier toutes les personnes qui ont participé à la relecture de ce document.

2. Présentation

Le but de ce travail est, dans un premier temps, d'installer et d'étudier le système de monitoring open source Shinken [1] puis de le comparer à Nagios[2] et aux autres systèmes de monitoring.

La seconde partie consiste à développer un plugin pour Shinken utilisant peu de ressources et pouvant profiter de l'évolutivité de l'architecture de Shinken.

L'énoncé est également disponible sur le site web du laboratoire de transmission de données à l'adresse suivante : http://www.tdeig.ch/shinken/Plugin_pour_Shinken.pdf

3. Cahier des charges

Le cahier des charges effectif (ci-dessous) diffère du cahier des charges original présent dans l'énoncé. Ces changements résultent d'adaptations effectuées au début du projet avec mon professeur responsable.

Le cahier des charges effectif est le suivant:

- Choisir une distribution Linux, installer Shinken (avec une configuration minimale) et définir un scénario simple.
- Mettre en pratique ce scénario et effectuer un rapport technique sur l'architecture et le fonctionnement de Shinken.
- Effectuer un état de l'art des solutions de monitoring.
- Développer un plugin pour Shinken.

Certaines parties importantes de Shinken telles que la haute disponibilité et la répartition de charge ne sont pas traitées dans cette étude.

4. Besoins opérationnels de mesure de disponibilité

Les besoins opérationnels dans le domaine de la mesure de disponibilité varient beaucoup en fonction du corps de métier. En effet, un administrateur des systèmes IT verra le taux de disponibilité comme une qualité de service, tandis que le directeur financier verra le taux d'indisponibilité comme une perte de gain.

4.1. Affichages personnalisés

Il est courant de dire que l'administrateur système veut voir uniquement les machines qui ont un problème, tandis que le directeur veut voir que toutes les machines fonctionnent. On remarque que la manière dont les produits présentent l'information doit être dépendante du type d'utilisateur.

[1] Site web officiel de Shinken : http://www.shinken-monitoring.org/
[2] Site web officiel de Nagios : http://www.nagios.org/

4.2. Ordonnancement des tests

Un autre aspect des besoins opérationnels réside dans la configuration de l'ordonnancement des tests.

Le premier paramètre de configuration des tests qui vient à l'esprit est l'intervalle entre deux vérifications.

Un autre paramètre important est le nombre de tests consécutifs retournant une erreur à effectuer avant de considérer la machine comme étant dans un état critique. On peut prendre l'exemple simple d'une machine qui ne répond pas à une requête ICMP Request. Faut-il immédiatement la considérer comme présentant un problème ou effectuer un second test de vérification ? La quasi-totalité des solutions de supervision sur le marché permettent de configurer ces paramètres.

Un autre paramètre important est la possibilité de définir des plages horaires où la machine peut subir un incident sans être considérée comme ayant un problème. Ceci permet, par exemple, d'éviter de réveiller à trois heures du matin le technicien de piquet pour aller redémarrer un service qui est utilisé uniquement la journée en interne.

Il est également important de pouvoir définir une période de maintenance. Durant cette période la machine ne sera pas considérée comme ayant un problème.

4.3. Seuils

Il peut être intéressant de définir des seuils. Shinken utilise (comme Nagios) le code de retour des plugins pour déterminer le statut d'une machine ou d'un service.
Il y a quatre codes possibles :

- 0: Ok
- 1: Warning
- 2: Critical
- 3: Unknown.

On peut, par exemple, configurer le plugin check_ping pour effectuer trois requêtes ping à la suite et si le temps de réponse est supérieur à 3 secondes ou si un ping est resté sans réponse il retourne Warning. En revanche, si plus de 1 ping est resté sans réponse ou si le temps d'attente dépasse 5 secondes check_ping retournera Critical. La ligne de commande pour le scénario ci-dessus est la suivante:

```
check_ping -H $HOSTADDRESS$ -w 1,50 -c 2,70% -p 3.
```

Le chapitre 6.1.1. fournit plus de détails sur les seuils et les notions temporelles. Le chapitre 12. fournit plus d'informations sur la configuration de check_ping.

4.4. Facteur business

Un nouveau facteur commence à s'imposer dans le monde de la supervision. Il s'agit de l'importance (appelée parfois criticité) d'une machine pour le business. Prenons l'exemple d'une entreprise de vente en ligne où l'un de leurs quatre serveurs DNS cesse de fonctionner. Cet incident n'a pas d'effet sur le business de l'entreprise. Si, par contre, leur unique serveur web ne fonctionne plus, c'est presque toute l'activité commerciale qui s'arrête. De cet indicent peut résulter une perte financière non négligeable ainsi qu'une insatisfaction des clients.

4.5. Notifications

Il existe également un besoin d'être notifié. Lorsqu'un service ne fonctionne plus, il n'est pas nécessaire de prévenir toute l'équipe IT, seules les personnes concernées doivent être averties par email ou par SMS. Le

moment de l'envoi de la notification est également important. On peut imaginer qu'un technicien doit être informé le plus rapidement possible, tandis que le responsable IT aimerai peut être recevoir à la fin de la journée un rapport journalier des différents incidents qui se sont produits pendant la journée.

Il est aussi important de gérer les notifications en fonction de la durée du problème. Si un technicien reçoit une alerte importante et qu'une heure plus tard le problème est toujours présent, il est intéressant de pouvoir envoyer l'alerte à son supérieur ou à un technicien de niveau supérieur.

4.6. Affichage en temps réel

Un technicien chargé de la surveillance d'un réseau IT aura besoin d'un affichage en temps réel de l'état du réseau pour lui permettre de visualiser certaines informations telle que l'évolution de la charge des serveurs ou des liens formant le réseau IP.

4.7. Démarche proactive

L'historique est un outil souvent nécessaire pour trouver la cause d'un incident ou tout simplement pour faire des statistiques et prévoir les investissements futurs pour le maintien de la stabilité du réseau. On peut donc faire du Risk Management et de la statistique pour déterminer la probabilité qu'un problème survienne, l'impact que ce problème peut avoir sur l'infrastructure IT et sur le business.

4.8. SLA – Service Level Agreement

L'historique est également nécessaire pour prouver et mesurer le taux de disponibilité d'un serveur. Dans des contrats de SLA un taux de disponibilité inférieur à un taux souvent très élevé (par exemple 99.999%) se traduit par une compensation financière au bénéfice du client. L'historique permet, dans ce cas, d'éviter qu'un client tente d'abuser le prestataire de service en lui faisant croire que son serveur était indisponible alors que ce n'était pas le cas. L'historique permet également de calculer correctement la part financière qui doit être redistribuée au client en cas d'indisponibilité.

4.9. Besoins annexes

La mesure de disponibilité est une activité importante dans la gestion d'un parc informatique, néanmoins elle ne peut pas être utilisée seule. D'autres outils doivent être utilisés en association à un système de mesure de disponibilité. Voici une liste non exhaustive de systèmes annexes:

- Système de gestion de base de données (SGBD) pour la sauvegarde des données de supervision (incident, SLA, …).
- Système d'inventaire du matériel et/ou du logiciel.
- Système de suivi de problèmes[3].
- Système de génération de rapports.
- Système de mesure de charge[4] et de performances.

[3] Wikipedia : http://fr.wikipedia.org/wiki/Système_de_suivi_de_problèmes
[4] Des détails sur la mesure de charge sont disponibles dans mon travail de Bachelor : http://www.tdeig.ch/kvm/schaub_M.pdf

5. État de l'art

5.1. Solutions de supervision

La supervision est un sujet qui est devenu une source de préoccupation lorsque les entreprises ont commencé à avoir plusieurs serveurs avec chacun plusieurs services.

Les solutions de supervision sont apparues suite à un besoin d'automatiser les vérifications du bon fonctionnement des différents services informatiques (serveur web, connexion à la base de données, etc.).

La plupart des logiciels de supervision open source que l'on trouve actuellement sont basés sur le très célèbre Nagios. Comme le cœur de Nagios est principalement maintenu par une seule personne (Ethan Galstad), les propositions de modification de code sont rarement traitées. C'est une des raisons au nombre important de logiciels basés sur Nagios.

Cet état de l'art présente treize produits dont huit open sources et cinq propriétaires.

Pour la majorité des outils gratuits présentés ci-dessous, il existe une version avec un abonnement de support et/ou de services.

5.1.1. Nagios

Nagios est une application permettant de surveiller l'état des hôtes d'un réseau et des différents services présents sur ces derniers. Il est codé en C sous licence GPL et est disponible sur les systèmes Linux et *nix.

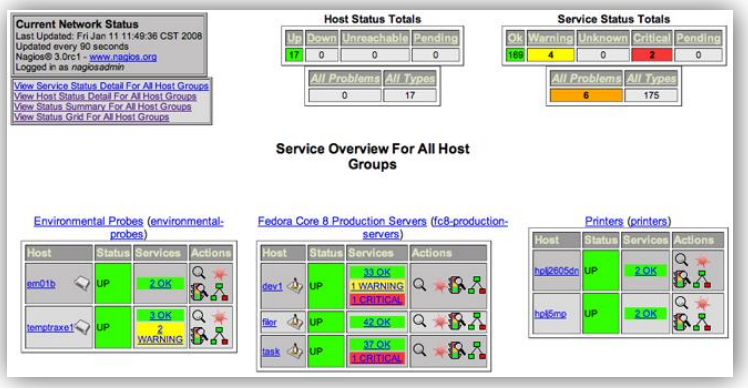


Figure 1 - Capture d'écran de Nagios
(source: http://www.nagios.com/products/nagioscore/screenshots)

Nagios est aujourd'hui le système le plus utilisé en entreprise. Malgré sa grande popularité, beaucoup d'utilisateurs n'en sont pas pleinement satisfaits. Certains développeurs ont tenté de proposer des modifications du code source à Ethan Galstad. Ce dernier étant peu réactif et acceptant rarement les propositions d'améliorations, des développeurs ont commencé à créer des forks[5] de Nagios. C'est de là que sont nés la plupart des nombreux fork de Nagios.

Nagios est divisé en trois parties :

- Moteur : Ordonnance les tâches de supervision (test du fonctionnement du serveur web, etc.).
- Interface web : Affiche l'état du réseau et les anomalies.
- Les plugins : Chaque plugin est un logiciel indépendant chargé de faire une vérification puis de transmettre le résultat à Nagios.

Site officiel : http://www.nagios.org

[5] Fork sur Wikipédia : http://fr.wikipedia.org/wiki/Fork_(développement_logiciel)

5.1.2. Shinken

Shinken est une application récente, gratuite et dont le développement est très actif. Elle a vu le jour en 2009. Son code est sous licence AGPL. Shinken peut fonctionner sur toutes les plateformes supportant le logiciel Python.

C'est une réimplémentation de Nagios en langage Python qui ajoute de nouvelles fonctionnalités orientées performance. Parmi ces fonctionnalités on peut noter la haute disponibilité et la possibilité d'être distribué. Pour réaliser ces améliorations le système a été découpé en plusieurs parties (appelées daemon dans le reste de ce document).

Ces différents daemons peuvent être répartis sur plusieurs machines.

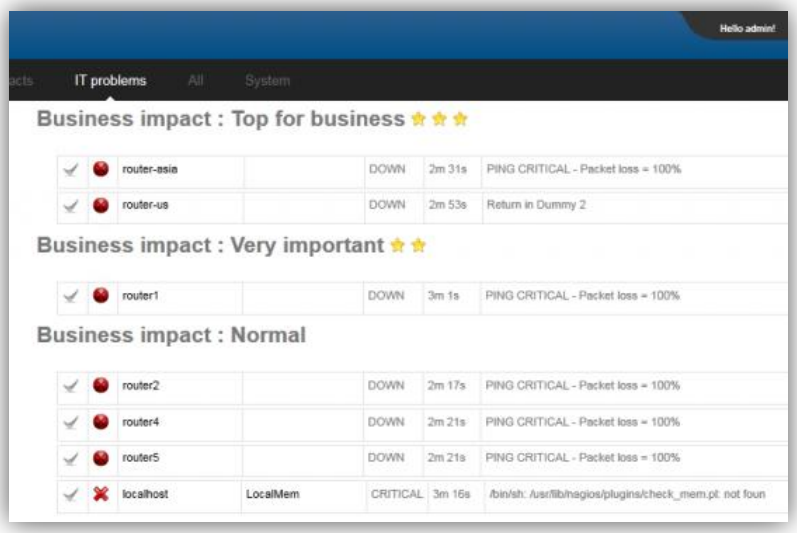


Figure 2 - Capture d'écran de Shinken
(source: http://www.shinken-monitoring.org/screenshots/)

Le passage de Nagios à Shinken se fait de manière aisée car Shinken utilise le même format de fichiers de configuration que Nagios. Il est uniquement nécessaire d'adapter les fichiers de configuration spécifiques à Shinken si l'on ne souhaite pas garder les paramètres par défaut.

Shinken intègre le facteur business (business impact sur la figure ci-dessus). Le déclenchement des alertes et les notifications sont paramétrables de manière précise. Il dispose d'un système de détection de flapping[6] permettant d'éviter des tempêtes d'alertes.

Plus d'informations sur le projet Shinken sont disponibles à cette adresse :
http://www.shinken-monitoring.org/project/

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

[6] Une machine est en état de flapping lorsque son état change constamment.

5.1.3. Centreon

Centreon est un logiciel de supervision dérivé de Nagios. Il est sous licence GPLv2.

On peut le voir comme une interface web pour Nagios, plus évoluée que l'interface de base de Nagios. L'interface web contient tout ce que contient l'interface de base de Nagios tout en étant plus conviviale. Elle ajoute également de nombreux graphiques générés à l'aide de l'outil RRDTool.

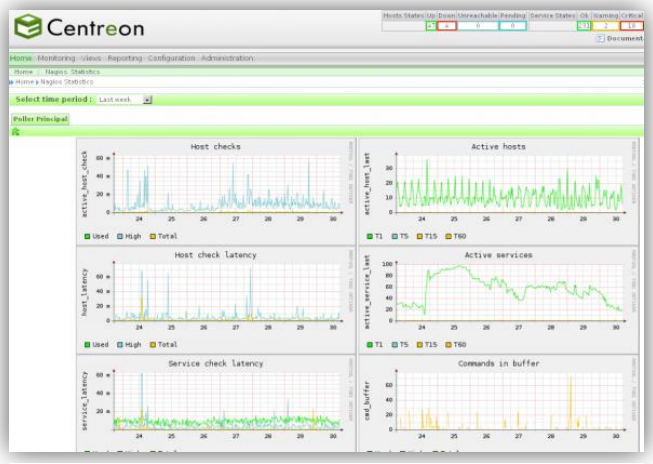


Figure 3 - Capture d'écran de Centreon (source: http://www.centreon.com/)

Il permet de générer des rapports sur les incidents. Les données sont stockées dans une base de données MySQL. L'interface web est multiutilisateur et gère des listes de contrôle d'accès pouvant être paramétrées de manière très précise.

Centreon permet de créer une architecture distribuée pour répartir la charge et/ou effectuer une répartition sur différents sites géographiques.

Bien qu'il fonctionne par défaut avec Nagios, il est possible d'utiliser Centreon avec Shinken.

Site officiel : http://www.centreon.com/

5.1.4. MRTG

MRTG (qui signifie Multi Router Traffic Grapher) est un logiciel sous licence GPL qui utilise le protocole SNMP pour communiquer avec les équipements compatibles. Il crée des graphiques avec les informations obtenues via SNMP.

Il est compatible avec tous les systèmes supportant Perl.

L'auteur de MRTG a également conçu RRDTool. RRDTool (**R**ound-**R**obin **D**atabase **Tool**) est un logiciel qui implémente les fonctionnalités de génération de graphique de MRTG ainsi qu'une base de données cyclique. RRDTool est utilisé dans d'autres outils de supervision telle que Cacti.

Site officiel : http://oss.oetiker.ch/mrtg/

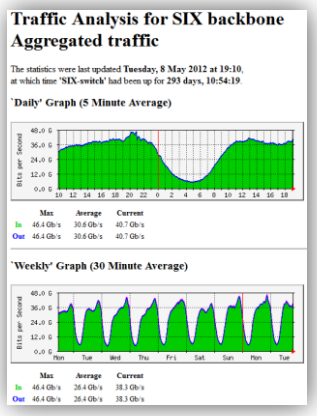


Figure 4 - Capture d'écran de MRTG

5.1.5. Cacti

Cacti est un logiciel open source (sous licence GPL) de supervision de réseau. Il est basé sur le système de base de données RRDTool. Son interface est principalement peuplée de graphiques indiquant l'état du réseau.

Il gère nativement le SNMP et peut facilement être intégré à d'autres sources de données telles que Nagios ou Shinken.

Cacti est souvent considéré comme le successeur de MRTG (lui aussi basé sur RRDTool).

Cacti dispose d'un poller qui se charge de tester le bon fonctionnement de services à des intervalles prédéterminés. Ce poller existe en deux versions, l'une codée en PHP pour les petites architectures et une autre codée en C pour les architectures plus grandes.

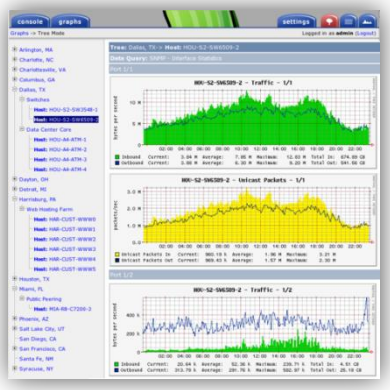


Figure 5 - Capture d'écran de Cacti

Site officiel : http://www.cacti.net/

5.1.6. Zabbix

Zabbix est une solution de supervision gratuite facile à mettre en place. Il est sous licence GPLv2. Une licence commerciale est également disponible pour les entreprises désirant intégrer Zabbix à leur produit propriétaire.

Actuellement cette solution ne contient pas de mécanisme performant pour éviter des tempêtes d'alertes. Zabbix contient un système d'affichage en temps réel de l'état des machines surveillées. Il contient également de nombreux graphiques permettant de visualiser l'évolution au cours du temps de l'infrastructure surveillée. Il dispose d'un système de gestion de SLA. Il fournit une gestion fine des droits d'accès des utilisateurs.

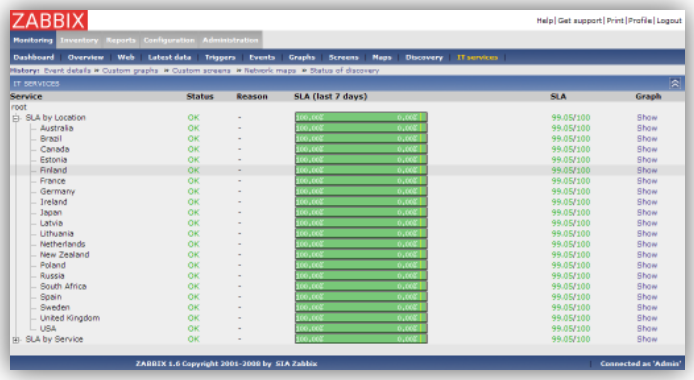


Figure 6 - Capture d'écran de Zabbix
(source: http://www.zabbix.com/screenshots.php)

L'architecture de Zabbix peut être décomposée en trois parties:

- Le serveur de stockage de données (MySQL, PostgreSQL ou Oracle)
- L'interface web de gestion
- Le serveur de traitement

Zabbix dispose d'un système de répartition de charge et de haute disponibilité. Les trois parties de Zabbix peuvent être sur des machines différentes. Le serveur de traitement peut également avoir des "proxy Zabbix". Un proxy Zabbix est une machine supplémentaire qui peut être utilisée pour traiter une partie des données lorsque la charge de la machine principale devient trop élevée.

Site officiel : http://www.zabbix.com

5.1.7. Icinga

Icinga est un fork de Nagios. Il a été créé par des développeurs qui étaient actifs sur le projet Nagios et qui n'appréciaient pas le manque de réactivité du développeur principal de Nagios ainsi que sa volonté de ne pas rendre tous les plugins libres.

Icinga est, comme Nagios, disponible uniquement sur les plateformes Linux et *nix. Il est écrit en C et sous licence GPL. Il est rétro-compatible avec la configuration et les plugins Nagios.

Icinga dispose d'un système de génération de rapports et de détection des oscillations d'état (flapping).

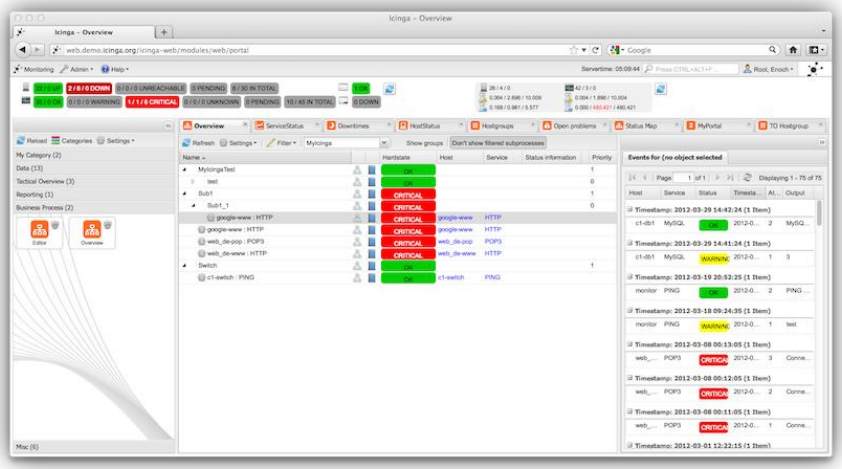


Figure 7 - Capture d'écran de Icinga
(source: https://www.icinga.org/screenshots/)

Il est disponible en deux versions[7]:

- Icinga Classic
- Icinga New Web

La version Icinga New Web dispose d'un système de gestion de SLA.

Site officiel : http://www.icinga.org/

5.1.8. EyesOfNetwork

EyesOfNetwork est un pack comprenant plusieurs logiciels liés à la supervision. Il est sous licence GPLv2 et est distribué sous forme d'une version minimaliste et personnalisée de CentOS.

Il dispose d'une interface web unifiée qui permet aux différents acteurs (administrateurs, techniciens, …) d'un système d'information d'accéder à une version de l'interface personnalisée à leur corps de métier.

Il comprend notamment:

- Nagios
- Cacti
- EONWEB : Interface Web unifiée du produit
- Get-Info: Système de génération de documents

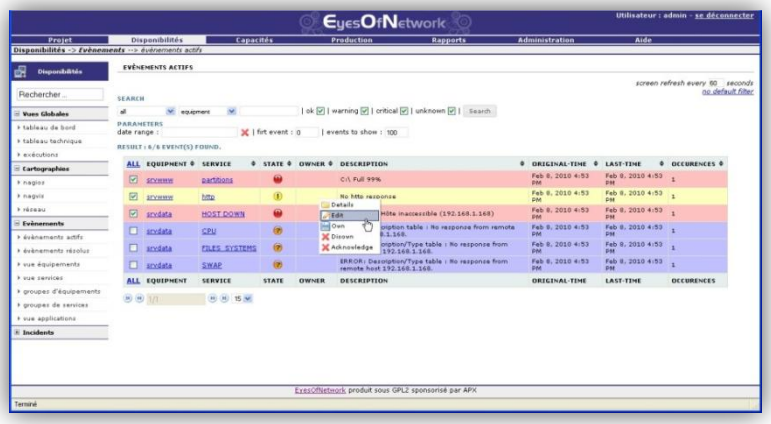


Figure 8 - Capture d'écran de EyesOfNetwork

[7] Comparaison des deux versions de Icinga avec Nagios : https://www.icinga.org/nagios/feature-comparison/

Toutes les informations récoltées sont stockées dans une base de données MySQL ou BERKELEY.

Dans la dernière version d'EyesOfNetwork (v3.0), Shinken (v0.6.5) a été intégré en Beta.

Site officiel : http://www.eyesofnetwork.com/

5.1.9. BoardVisor

BoardVisor est une solution propriétaire éditée par la société IDCWARE. BoardVisor se veut simple d'utilisation, par conséquent il n'y a aucun fichier de configuration à éditer, tout se fait via une interface web.

La solution est réalisée sur mesure, seuls les modules nécessaires au client sont intégrés. Elle est livrée clefs en main, l'installation est effectuée par IDCWARE et doit être totalement fonctionnelle lors de sa livraison.

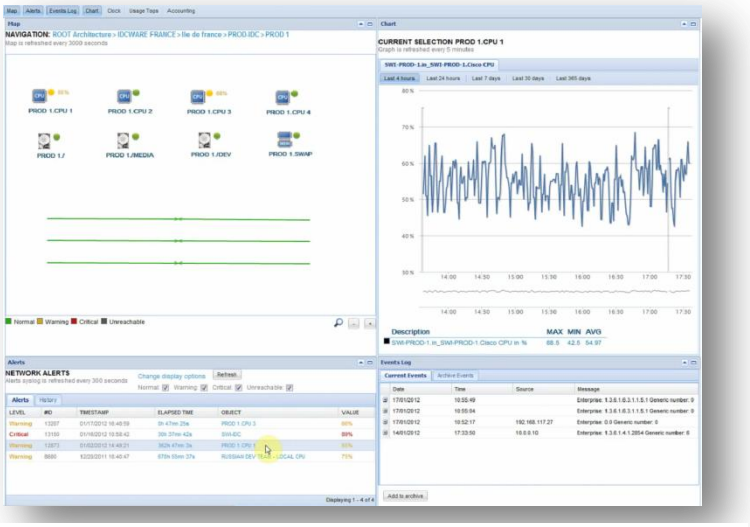


Figure 9 - Capture d'écran de BoardVisor

BoardVisor est divisé en trois parties:

- Les Gatherers : Ce sont des sondes-agents qui sont chargées de collecter les données.
- Le Harvester : Il traite les informations reçues des Gatherers, stocke les informations dans une base de données et met à jour le GUI grâce à un mécanisme push (il envoie l'information à l'interface web sans qu'elle ne l'ait demandée). Les Gatherers peuvent être distribués sur plusieurs machines.
- Le GUI[8] : C'est l'interface web de gestion. Il affiche de manière intuitive les informations en provenance du Harvester.

BoardVisor contient une fonction de génération de rapports. Il gère également l'inventaire du matériel et des logiciels.

Site officiel : http://www.idcware.com/la-solution/

[8] GUI - Graphical User Interface

5.1.10. NetCrunch

NetCrunch est un logiciel propriétaire édité par la société AdRem Software. Son prix est de 1'500 $ pour 50 périphériques surveillés.

Figure 10 - Capture d'écran de NetCrunch
(source: http://fr.wikipedia.org/wiki/NetCrunch)

Il fonctionne selon l'architecture client-serveur. Il y a le serveur NetCrunch et le client (console d'administration) qui est un logiciel avec une interface graphique se connectant au serveur pour afficher les données. Un client web est également disponible.

Il fonctionne sans agent et peut utiliser des sources d'informations telles que SNMP, les journaux d'évènement de Windows et les serveurs Syslog sous Linux. Il contient un système de module permettant d'étendre le système. NetCrunch 6 contient plus de 2'000 modules préinstallés.

NetCrunch intègre des fonctionnalités de gestion de la disponibilité, de gestion de la performance des serveurs et de génération de rapports. Il dispose également d'un système de découverte automatique de la topologie du réseau.

Il peut aussi être utilisé pour faire un inventaire du matériel et des logiciels.

Site officiel : http://www.adremsoft.fr/netcrunch/

5.1.11. Traverse

Traverse est une solution propriétaire éditée par Zyrion. Une édition gratuite de démonstration est disponible. Le prix de l'édition standard varie avec la taille du réseau surveillé et commence à 50'000 $.

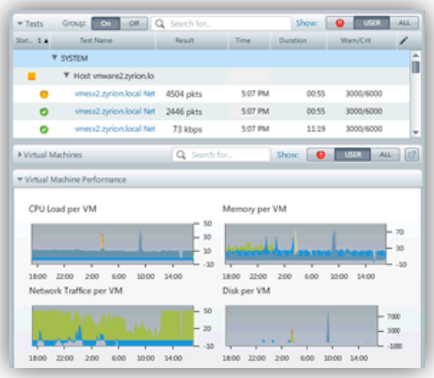


Figure 11 - Capture d'écran de Traverse
(source: www.zyrion.com/products/)

Traverse intègre des fonctionnalités de gestion de SLA, de performance du réseau et de gestion de parcs de machines virtuelles et de Clouds. Il gère les environnements virtualisés VMware, XenServer, KVM et Hyper-V.

Il propose des vues représentant la topologie physique du réseau et des vues business permettant de voir en temps réel la performance du service informatique.

Site officiel : http://www.zyrion.com/products/

5.1.12. Tivoli Monitoring

Tivoli Monitoring est une solution propriétaire développée par la société IBM.

Il fait partie de la suite de logiciel Tivoli. Depuis la version 6.1 il ne repose plus sur le framework Tivoli, il utilise le moteur d'une solution concurrente (rachetée par IBM) nommée Omegamon (éditée à l'origine par Candle Corp).

Il est composé des éléments suivants :

- Tivoli Data Warehouse (TDW) :
  Sauvegarde les données de supervision dans une base de données SQL.
- Tivoli Entreprise Monitoring Server (TEMS) :
  Il lance les tests sans agent et communique avec les agents puis il envoie les données à sauvegarder au TDW.
- Tivoli Entreprise Portal Server (TEPS) :
  Permet d'accéder à distance au TDW grâce à une interface web.
- Tivoli Common Reporting (TCR) :
  Fournit des rapports en fonction des données présentes dans le TDW.

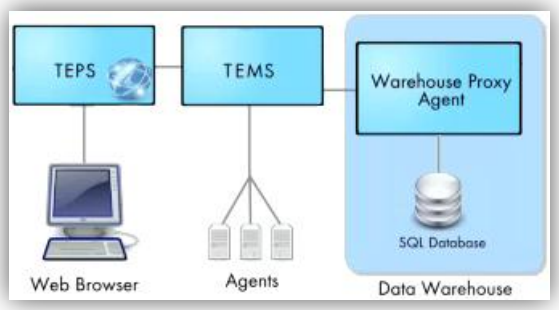


Figure 12 - Schéma de la solution Tivoli Monitoring (source: www.ibm.com)

Tivoli Monitoring fournit des agents permettant de récupérer des informations sur l'état des machines. Ces agents sont prévus pour continuer à fonctionner lorsque le réseau est interrompu ou dans le cas où le serveur de supervision ne fonctionne plus. Si un tel cas se produit les agents distribués sur les machines vont sauvegarder les informations et les transmettre au serveur de supervision dès que ce dernier est à nouveau joignable. Tivoli Monitoring est aussi utilisable sans agent (agentless).

Site officiel: http://www-142.ibm.com/software/products/fr/fr/tivomoni

5.1.13. OpenView

OpenView est un logiciel propriétaire développé par la société HP. En 2007, il a été renommé HP Network Management Center.

La suite HP OpenView contient une multitude de logiciels. Parmi ces logiciels, on trouve notamment:

- HP OpenView Network Node Manager (OV NNM) : Chargé de surveiller le réseau et les services réseau à l'aide du protocole SNMP.
- HP Operations Manager (OM) : Surveille les systèmes qui utilisent des agents. Il existe un agent pour les systèmes Windows (OMW) et un agent pour les systèmes Unix (OMU).
- HP OpenView Performance (OVP*) : Logiciels permettant de surveiller les performances de différents équipements.
- HP OpenView Storage : Permettant de gérer le stockage des informations de supervision.

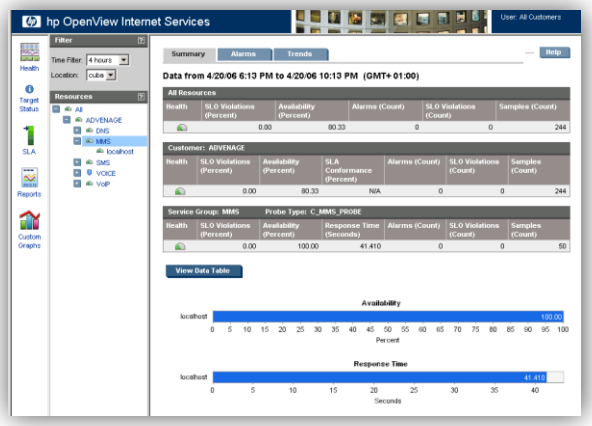


Figure 13 - Capture d'écran de HP OpenView Internet Services

- HP OpenView Smart Plug-ins (SPI) : Plugins permettant de superviser une base de données, un système VMware, un système SAP, etc.
- HP Software Business Availability Center (BAC) : Fournit des services de gestion de SLA, de gestion d'utilisateurs, etc.

Site officiel: http://www8.hp.com/us/en/software/enterprise-software.html

5.2. Comparatif

Nom	Licence	Langage	Système d'exploitation	Commentaires	Prix
Shinken	AGPL	Python	Linux, Windows, ...	Compatible Nagios	Gratuit
Nagios	GPL	C	Linux, Unix		Gratuit
Centreon	GPLv2	PHP/Perl/C	Linux, Unix	Utilise Nagios, RRDTool	Gratuit
Cacti	GPL	PHP/Shell/C	Linux, Unix	Basé sur RRDTool	Gratuit
Zabbix	GPLv2	PHP/C	Linux, Mac, Unix		Gratuit
Icinga	GPL	PHP/C	Linux	Fork de Nagios	Gratuit
EyesOfNetwork	GPLv2	Multi	OS dédié (CentOS)	Package contenant Nagios	Gratuit
MRTG	GPL	Perl	Linux, Windows, ...		Gratuit
NetCrunch	Propriétaire	N/A	Windows		> 1'500 $
BoardVisor	Propriétaire	N/A	OS dédié		Payant
Traverse	Propriétaire	N/A	Linux, Windows		> 50'000 $
Tivoli	Propriétaire	N/A	Linux, Windows		> 750 $
Open View	Propriétaire	N/A	Windows, HP-UX		Payant

Tableau 1 - Comparaison des solutions de supervision

5.3. Comparaison de Nagios et de Shinken

Shinken étant une réimplémentation de Nagios, il est intéressant de comparer leur état de développement. Est-ce que Shinken a réimplémenté toutes les fonctionnalités de Nagios ? Quelles sont les fonctionnalités phares de Shinken ?

Parmi les fonctionnalités phares de Shinken on peut relever:

- Scalabilité et haute disponibilité.
- Utilisable sur toutes les plateformes supportées par Python.
- Notion d'importance pour le business.
- Possibilité de mettre un Poller dans la DMZ et un Poller dans le LAN pour éviter une configuration complexe et risquée du pare-feu.

Nagios possède néanmoins certaines fonctionnalités qui ne sont pas disponibles dans Shinken:

- Pearl embarqué.
- Modules binaires.
- Replanification automatique des tests.

Il est intéressant de comparer l'évolution du nombre de lignes de code au cours du temps.

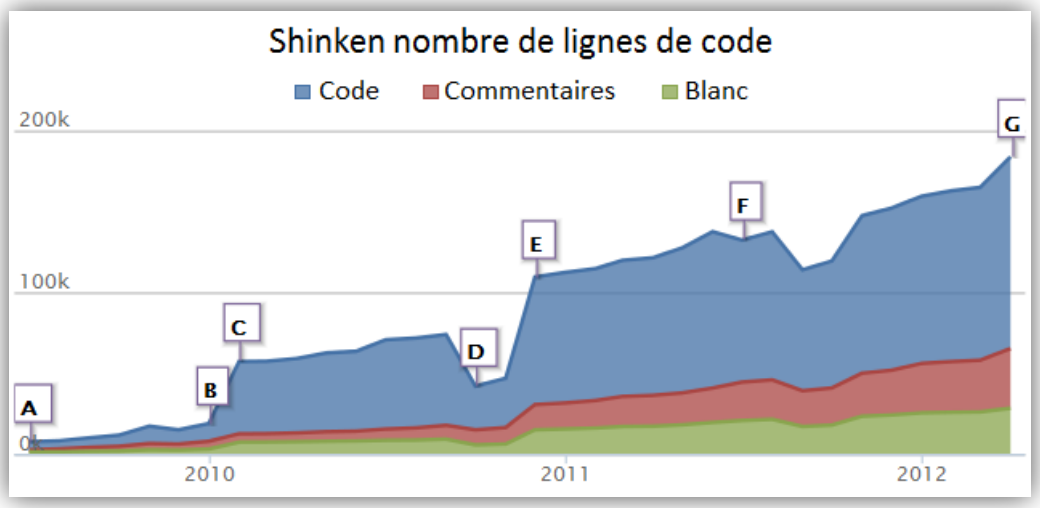


Figure 14 - Shinken - Nombre de lignes de code

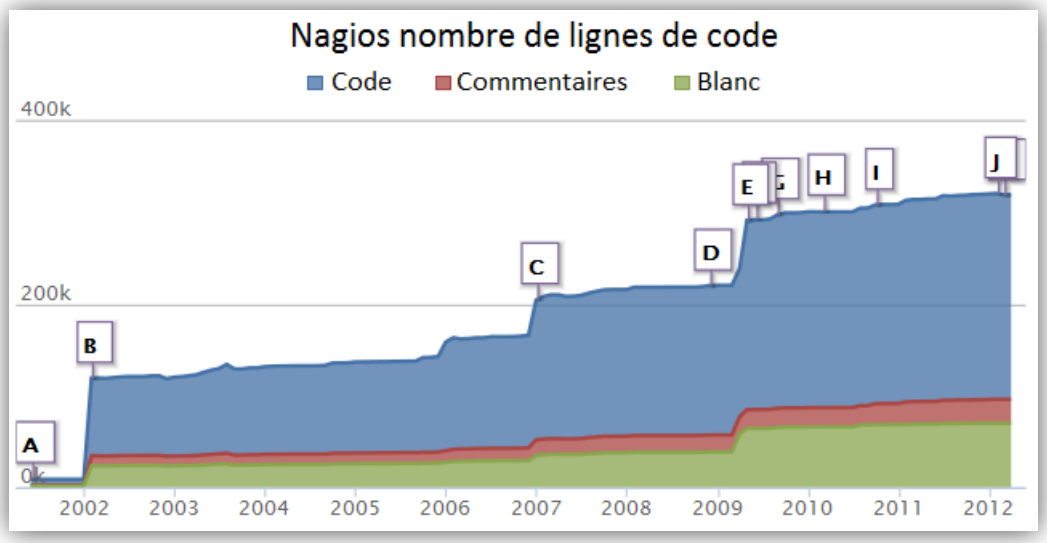


Figure 15 - Nagios - Nombre de lignes de code

On remarque sur les schémas ci-dessus (fournis par www.ohloh.net) que Shinken est nettement plus jeune que Nagios. On remarque aussi les méthodes de développement des deux logiciels. En effet Nagios se développe peu et s'est développé par à-coups. Shinken, quant à lui, se développe rapidement (200'000 lignes de codes en 3 ans). Son code est parfois refactoré et des éléments inutiles sont supprimés. La baisse brutale du nombre de lignes de code en septembre 2010 correspond à la suppression des fichiers XML contenant l'aide au profit du Wiki (http://www.shinken-monitoring.org/wiki).

6. Scénarios de test

6.1. Test ICMP (Ping) d'une machine

Ce scénario très simple consiste à tester le fonctionnement du système en configurant un test ICMP (ping) sur un hôte du réseau. Ce test doit s'effectuer périodiquement à un intervalle d'une minute.

Pour mettre en pratique ce scénario il faut disposer d'un ordinateur (ou d'une machine virtuelle) allumé. En débranchant le câble réseau, on peut tester le passage de l'état UP à l'état DOWN. A ce moment le serveur Shinken doit envoyer un email de notification.
Quelques instants plus tard on rebranche le câble et la machine doit repasser à l'état UP puis envoyer un second email de notification.

Les changements d'états peuvent également être observés sur l'interface web de Shinken (WeUI).

Grâce à ce scénario, on teste le bon fonctionnement de toute l'architecture de Shinken.

La configuration et le test de ce scénario sont détaillés dans le chapitre Mise en pratique du scénario.

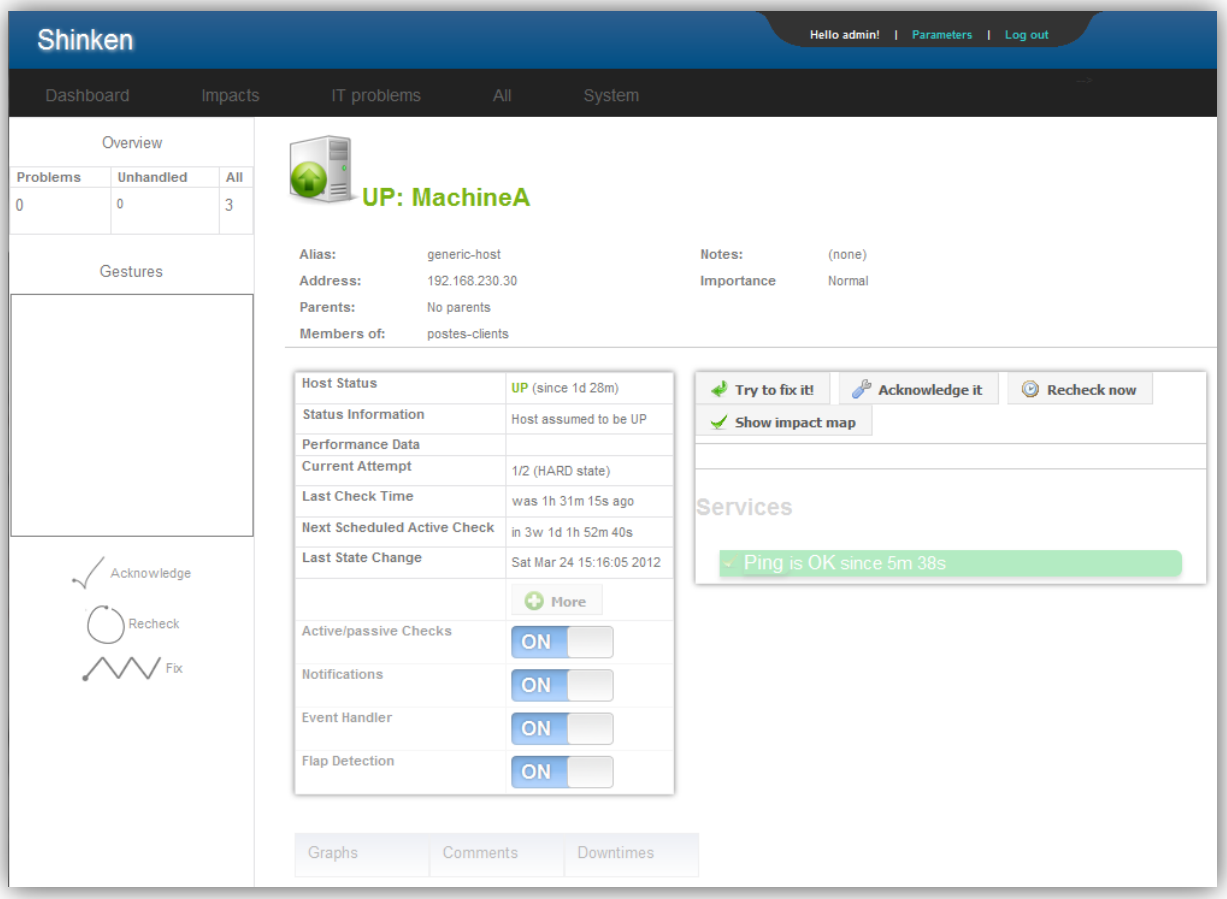


Figure 16 - WebUI - MachineA - Ping Ok

6.1.1. Notions temporelles

Lorsque l'on utilise des outils de monitoring il est important de garder à l'esprit les notions temporelles du système. En effet, si l'on configure un test ICMP pour qu'il s'effectue toutes les minutes, il faut garder à l'esprit qu'il peut se produire plus d'une minute avant qu'une machine indisponible soit détectée comme telle.

Si on prend le pire des cas, il faudra attendre une minute plus le temps d'attente maximal (5 sec.) pour que Shinken détecte le problème. Dans le meilleur des cas il faut le temps d'attente maximal (5 sec.) pour qu'un problème soit détecté par Shinken. Nous avons donc un temps de détection qui se situe entre 5 secondes et 65 secondes.

A cela vient s'ajouter le temps d'envoi d'un email et l'intervalle de vérification des emails configurée sur le client de messagerie du technicien. Cet intervalle ne s'applique pas si la messagerie fonctionne en mode Push.

Le diagramme en flèche ci-dessous présente les délais, le meilleur des cas et le pire des cas pour le scénario décrit ci-dessus. Pour simplifier la figure, elle ne montre pas l'intervalle de vérification des emails. On considère donc être dans le cas d'une messagerie configurée en mode Push.

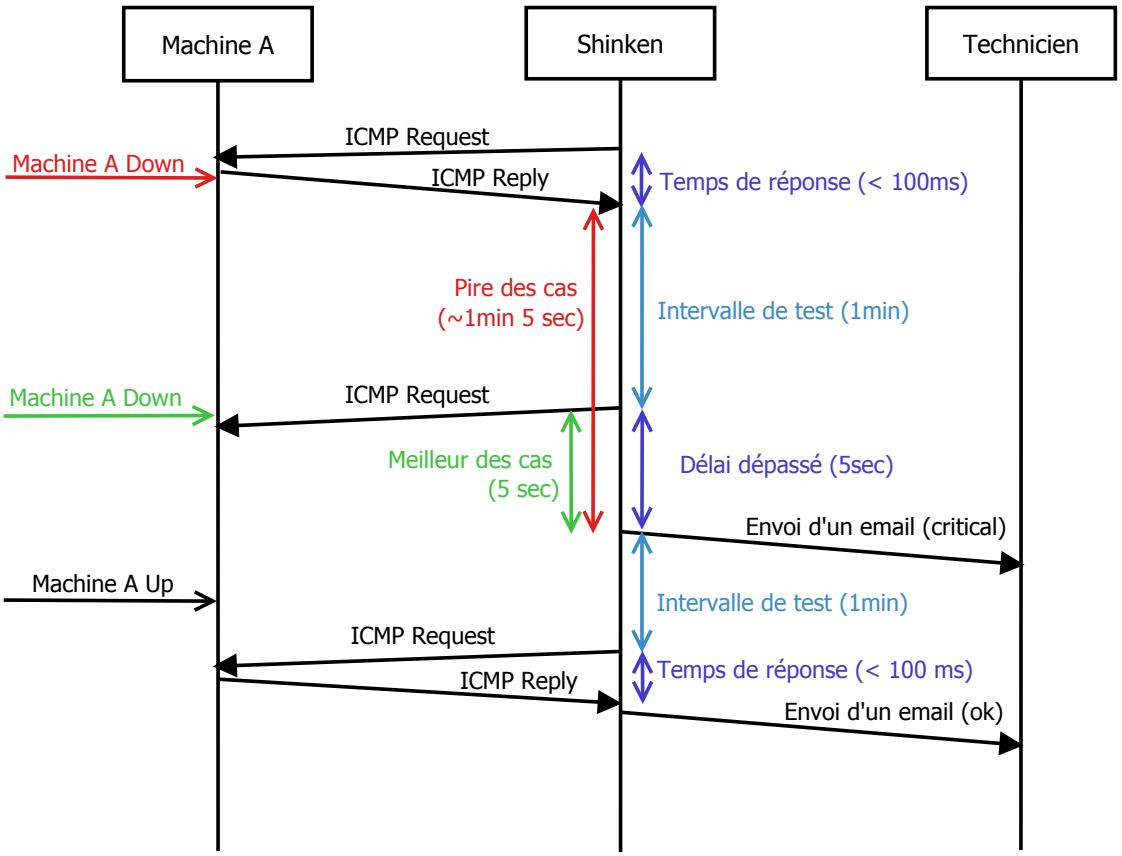


Figure 17 - Diagramme de séquence d'un test ICMP simple

Dans le diagramme ci-dessus on remarque que le décompte de temps commence dès que le test est effectué. Pour déterminer si Shinken commençait le décompte de temps après avoir lancé le test ou après que le test se soit terminé, un plugin a été développé. Voir l'annexe Plugin check_ordonnanceur.

Il faut aussi garder à l'esprit que si la machine ou le réseau subit des mini coupures d'une ou deux secondes il y a peu de chances que ces coupures soient détectées. Le diagramme ci-dessous décrit ce problème.

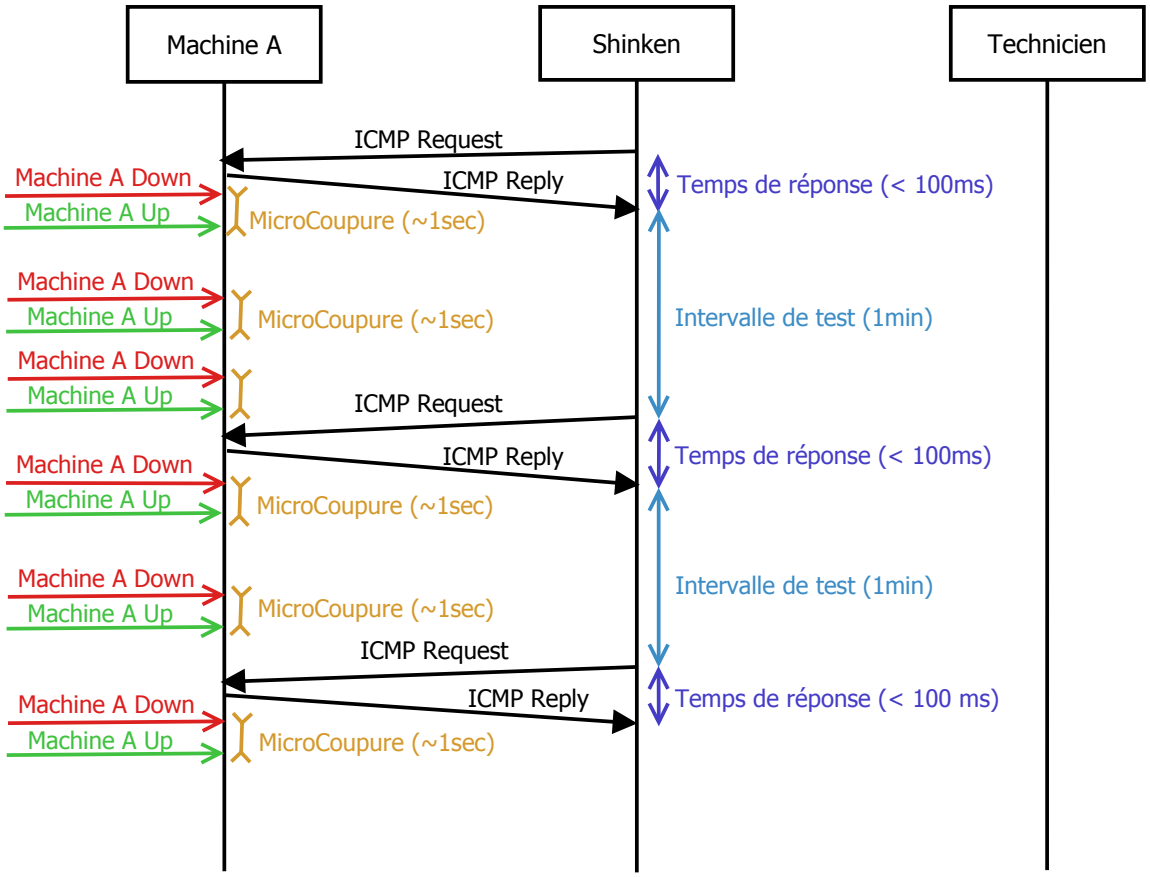


Figure 18 - Diagramme de séquence d'un test ICMP simple avec microcoupures

7. Choix de l'architecture matérielle et du système d'exploitation

Pour des raisons de simplicité j'ai choisi d'installer Shinken et une machine Linux (Ubuntu) dans des machines virtuelles.

Le logiciel de virtualisation utilisé est VMware Player 3.1.3.
La version de Shinken utilisée est la 1.0, sortie une semaine après le début du projet.
La machine Linux est un Ubuntu 10.04.3 LTS 64-bits.

Shinken est installé sur une distribution CentOS 6.2 car c'est la distribution la plus utilisée en entreprise. Elle est également utilisée par les développeurs de Shinken sur les machines de test et les machines en production.

Une machine virtuelle utilisable sous VMware Player avec Shinken 1.0 préinstallé sous CentOS 6.2 est fournie en annexe. Les procédures pour installer et configurer CentOS 6.2 et Shinken 1.0 sont disponibles en annexes.

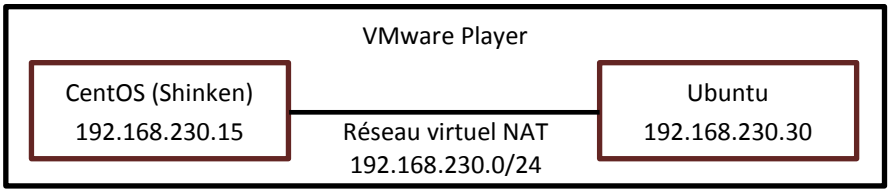


Figure 19 - Schéma bloc de l'architecture

8. Architecture de Shinken

Shinken est un outil de supervision découpé en plusieurs parties. Le cœur du système correspond aux six daemons. Il y a également les modules qui permettent d'ajouter des fonctionnalités au cœur de Shinken et des plugins qui permettent d'effectuer les tests.

8.1. Daemons (Services)

Chaque daemon (excepté l'Arbiter) peut avoir plusieurs instances réparties sur plusieurs machines pour permettre à Shinken de supporter une plus grande charge et également fournir de la haute disponibilité.

8.1.1. Arbiter

L'Arbiter est le seul daemon qui doit être unique. Il est chargé de gérer tous les autres. Il lit les fichiers de configuration puis envoie les parties utiles aux autres daemons. Il est également chargé de s'occuper de la haute disponibilité, si un deamon devient indisponible, il doit transférer la configuration de ce dernier à un daemon disponible. L'Arbiter est exécuté avec l'utilisateur root.

Il peut être étendu grâce à des modules. Il existe, par exemple, un module pour recevoir les résultats de tests passifs NSCA.

8.1.2. Scheduler

Le Scheduler doit ordonnancer les tests. Il détermine à quel moment les tests doivent s'effectuer mais il ne lance pas le check (rôle du Poller). Il récupère ensuite les résultats des tests (check) et les transmets au Reactionner et au Broker si nécessaire.

Il peut être étendu grâce à des modules. Il y a peu d'intérêt à créer des modules pour le Scheduler, les seuls modules présents sont des modules pour implémenter la rétention de statut. Néanmoins la rétention de statut est une fonctionnalité qui n'est pas utile avec l'architecture de Shinken.

8.1.3. Poller

Le Poller doit lancer les plugins tels que check_ping en fonction des instructions du Scheduler. Lorsque le plugin se termine il renvoie le résultat au Scheduler.

Il est possible de configurer un Poller dans la DMZ et un Poller dans le LAN pour éviter une configuration compliquée et dangereuse du firewall.

En effet, si on installe une machine Shinken dans la DMZ et que l'on souhaite superviser des machines dans le LAN, il faudra créer dans le pare-feu une règle pour chaque test que l'on souhaite effectuer dans le LAN.
Si l'on place un Poller sur une machine dans le LAN, il est possible de lui faire exécuter les tests des machines qui sont dans le LAN. Avec cette configuration il faut seulement ouvrir les ports nécessaires à la communication entre la machine Poller du LAN et la machine Shinken de la DMZ.
Cette configuration s'effectue grâce au paramètre *poller_tags*[9].

Il peut être étendu grâce à des modules.

[9] Page du wiki officiel expliquant comment effectuer cette configuration : http://www.shinken-monitoring.org/wiki/setup_dmz_monitoring

8.1.4. Broker

Le rôle du Broker est d'exporter les données en provenance du Scheduler. Il a besoin de modules pour exporter les données dans différents emplacements.

Il y a, par exemple, un module pour exporter les données dans une base de données MySQL.

L'interface web de Shinken correspond au module WebUI du Broker.

8.1.5. Reactionner

Le Reactionner se charge des notifications selon les instructions du Scheduler. Il envoie, par exemple, les emails d'alerte.

Il peut être étendu grâce à des modules. Il existe un module nommé AndroidSMS qui permet d'envoyer des SMS de notification lorsqu'il s'exécute sur un téléphone mobile sous Android.

8.1.6. Receiver

Le Receiver est le seul daemon qui est facultatif. Son rôle est de réceptionner les tests passifs et de les stocker temporairement dans un tampon. C'est l'Arbiter qui se charge de récupérer les résultats. Il a besoin de modules pour communiquer avec les différents protocoles. Par défaut, il y a un module Nagios NSCA qui gère le système de plugin passif NSCA de Nagios. Un test passif est un test qui n'est pas ordonnancé par Shinken. Dans le cas d'un système NSCA les données sont envoyées par l'agent send_nsca.

8.2. Modules

Les modules sont des extensions des daemons. Certains daemons tels que le Broker et le Receiver fonctionnent uniquement avec des modules.

Les modules sont présents dans le dossier */usr/local/shinken/shinken/modules/*.

Ces derniers hérite de la classe BaseModule.

```
class Android_reactionner(BaseModule):
```

Figure 20 - Module AndroidSMS - héritage BaseModule

Chaque module doit également définir des propriétés.

```
properties = {
    'daemons' : ['reactionner'],
    'type' : 'android_sms',
    'external' : False,
    # To be a real worker module, you must set this
    'worker_capable' : True,
}
```

Figure 21 - Modules AndroidSMS - définition des propriétés

Dans les propriétés ci-dessus on remarque le daemon auquel il est associé et le nom du module. Ce module se nomme donc android_sms et est un module pour le Reactionner.

8.3. Plugins et agents

Les plugins sont des logiciels exécutables. Ils sont chargés d'effectuer un test sur une machine et renvoyer un résultat. Elles sont exécutées par le Poller. Comme Shinken utilise les mêmes spécifications que Nagios pour les plugins, les plugins Nagios sont compatibles avec Shinken et inversement.

Les plugins sont présentes dans le répertoire */usr/local/shinken/etc/libexec/*.

8.3.1. Langages de programmation

Ces commandes peuvent être programmées dans n'importe quel langage de programmation. Dans le cas d'un langage compilé il faut que l'exécutable soit compatible pour la plateforme sur laquelle il va être exécuté. Dans le cas d'un langage interprété il faut que l'interpréteur soit installé sur la machine et que le Shebang[10] soit présent au début du script. Dans le cas de Shinken il est pratique d'utiliser le langage (interprété) Python car Shinken étant codé en Python il est certain que Python est présent sur le système. Actuellement, la plupart des plugins de Shinken viennent de Nagios et sont codés en C.

8.3.2. Spécifications

La seule contrainte est l'obligation pour ces commandes de définir leur code de retour par une valeur spécifique au résultat de la commande. Les commandes, dans la plupart des cas, prennent en argument le nom d'hôte ou l'adresse IP de la machine à tester. Les codes de retour possibles sont les suivants :

- 0 : Ok
- 1 : Warning
- 2 : Critical
- 3 : Unknown

Le statut Unknown (inconnu) est souvent utilisé lorsque les arguments de la commande sont erronés. Un texte peut également être écrit sur la sortie standard pour fournir une information supplémentaire. Néanmoins seul le code de retour sera traité dans la logique de Shinken.

Les spécifications complètes des plugins sont disponibles ici :

http://www.shinken-monitoring.org/wiki/official/development-pluginapi

8.3.3. Agents

Un agent est un logiciel qui s'exécute sur les machines que l'on souhaite tester. Il permet de récupérer de l'information directement sur la machine puis la transmettre à un plugin de Shinken lorsqu'elle en fait la demande. Les agents doivent êtres compilés pour les architectures (type de processeur et système d'exploitation) des machines qui doivent être testées. Ceci peut être gênant dans le cas où le parc de machines est très hétérogène.

9. Documentation de Shinken

La documentation de Shinken est disponible à l'adresse http://www.shinken-monitoring.org/wiki .
Elle est, dans l'ensemble, bien faite. Il arrive cependant que certaines informations soient erronées.
Depuis la version 1.0 la documentation s'est grandement améliorée.
On peut librement effectuer des corrections en créant un compte.

[10] Page Wikipédia concernant le Shebang: http://fr.wikipedia.org/wiki/Shebang

10. Analyse du fonctionnement de Shinken

10.1. Processus ouverts

Dans la capture ci-dessous on voit les processus Shinken ainsi que l'utilisateur sous lequel ils s'exécutent. On voit aussi le fichier de configuration associé au deamon.

```
[root@shinken ~]# ps -Af | grep shinken
shinken   1309     1  python /usr/local/shinken/bin/shinken-scheduler -d -c /usr/local/shinken/etc/schedulerd.ini
shinken   1320  1309  python /usr/local/shinken/bin/shinken-scheduler -d -c /usr/local/shinken/etc/schedulerd.ini
shinken   1322     1  python /usr/local/shinken/bin/shinken-poller -d -c /usr/local/shinken/etc/pollerd.ini
shinken   1333  1322  python /usr/local/shinken/bin/shinken-poller -d -c /usr/local/shinken/etc/pollerd.ini
shinken   1338     1  python /usr/local/shinken/bin/shinken-reactionner -d -c /usr/local/shinken/etc/reactionnerd.ini
shinken   1348  1338  python /usr/local/shinken/bin/shinken-reactionner -d -c /usr/local/shinken/etc/reactionnerd.ini
shinken   1361     1  python /usr/local/shinken/bin/shinken-broker -d -c /usr/local/shinken/etc/brokerd.ini
shinken   1369  1361  python /usr/local/shinken/bin/shinken-broker -d -c /usr/local/shinken/etc/brokerd.ini
shinken   1374     1  python /usr/local/shinken/bin/shinken-receiver -d -c /usr/local/shinken/etc/receiverd.ini
shinken   1384  1374  python /usr/local/shinken/bin/shinken-receiver -d -c /usr/local/shinken/etc/receiverd.ini
root      1394     1  python /usr/local/shinken/bin/shinken-arbiter -d -c /usr/local/shinken/etc/nagios.cfg -c /usr/local/shinken/etc/
shinken-specific.cfg
root      1395  1394  python /usr/local/shinken/bin/shinken-arbiter -d -c /usr/local/shinken/etc/nagios.cfg -c /usr/local/shinken/etc/
shinken-specific.cfg
root      1402  1394  python /usr/local/shinken/bin/shinken-arbiter -d -c /usr/local/shinken/etc/nagios.cfg -c /usr/local/shinken/etc/
shinken-specific.cfg
shinken   1408  1322  python /usr/local/shinken/bin/shinken-poller -d -c /usr/local/shinken/etc/pollerd.ini
shinken   1418  1361  python /usr/local/shinken/bin/shinken-broker -d -c /usr/local/shinken/etc/brokerd.ini
shinken   1427  1361  python /usr/local/shinken/bin/shinken-broker -d -c /usr/local/shinken/etc/brokerd.ini
shinken   1445  1322  python /usr/local/shinken/bin/shinken-poller -d -c /usr/local/shinken/etc/pollerd.ini
shinken   1658  1338  python /usr/local/shinken/bin/shinken-reactionner -d -c /usr/local/shinken/etc/reactionnerd.ini
```

Figure 22 - Processus Shinken et fichier de configuration associé

Shinken est écrit en Python. Comme Python est un langage interprété, tous les processus sont des processus Python. En observant la ligne de commande qui a été utilisée pour lancer le processus, on peut voir le programme python qui a été lancé.

Avec la configuration par défaut il y a (processus) :

- 2 Schedulers
- 3 Pollers
- 3 Reactionners
- 4 Brokers
- 2 Recievers
- 3 Arbiters

La raison pour laquelle il y a plusieurs processus par daemon provient des modules des différents daemons. Les processus supplémentaires sont en réalité des processus fils. Avec la configuration défaut (minimale) il y a un processus père par daemon.

Tous les processus sont exécutés sous l'utilisateur shinken excepté les processus de l'Arbiter qui sont exécutés en tant que root.

Dans la capture ci-dessous j'ai ajouté le paramètre -H qui permet d'afficher la hiérarchie des processus. Cette hierarchie peut aussi être obtenue en regardant les colonnes PID et PID du père.

```
[root@shinken ~]# ps -AHf | grep shinken
root      1939  1262  0 15:45 tty1     00:00:00       grep shinken
shinken   1309     1  0 13:41 ?        00:00:29   python /usr/local/shinken/bin/shinken-scheduler
shinken   1320  1309  0 13:41 ?        00:00:00     python /usr/local/shinken/bin/shinken-scheduler
shinken   1322     1  0 13:41 ?        00:00:24   python /usr/local/shinken/bin/shinken-poller
shinken   1333  1322  0 13:41 ?        00:00:06     python /usr/local/shinken/bin/shinken-poller
shinken   1408  1322  0 13:41 ?        00:00:01     python /usr/local/shinken/bin/shinken-poller
shinken   1445  1322  0 13:41 ?        00:00:01     python /usr/local/shinken/bin/shinken-poller
shinken   1338     1  0 13:41 ?        00:00:22   python /usr/local/shinken/bin/shinken-reactionner
shinken   1348  1338  0 13:41 ?        00:00:04     python /usr/local/shinken/bin/shinken-reactionner
shinken   1903  1338  0 15:42 ?        00:00:00     python /usr/local/shinken/bin/shinken-reactionner
shinken   1361     1  0 13:41 ?        00:00:26   python /usr/local/shinken/bin/shinken-broker
shinken   1369  1361  0 13:41 ?        00:00:42     python /usr/local/shinken/bin/shinken-broker
shinken   1418  1361  0 13:41 ?        00:00:20     python /usr/local/shinken/bin/shinken-broker
shinken   1427  1361  0 13:41 ?        00:00:01     python /usr/local/shinken/bin/shinken-broker
shinken   1374     1  0 13:41 ?        00:00:09   python /usr/local/shinken/bin/shinken-receiver
shinken   1384  1374  0 13:41 ?        00:00:00     python /usr/local/shinken/bin/shinken-receiver
root      1394     1  0 13:41 ?        00:00:37   python /usr/local/shinken/bin/shinken-arbiter
root      1395  1394  0 13:41 ?        00:00:01     python /usr/local/shinken/bin/shinken-arbiter
root      1402  1394  0 13:41 ?        00:00:00     python /usr/local/shinken/bin/shinken-arbiter
```

Figure 23 – Hiérarchie des processus Shinken

10.2. Ports ouverts

En utilisant l'outil `netstat` on peut voir quel logiciel écoute sur quel port.

```
[root@shinken ~]# netstat -lp | grep "LISTEN .*python"
tcp        0      0 *:50000                     *:*                         LISTEN      1418/python
tcp        0      0 *:7767                      *:*                         LISTEN      1427/python
tcp        0      0 *:7768                      *:*                         LISTEN      1309/python
tcp        0      0 *:7769                      *:*                         LISTEN      1338/python
tcp        0      0 localhost:7770              *:*                         LISTEN      1394/python
tcp        0      0 *:7771                      *:*                         LISTEN      1322/python
tcp        0      0 *:7772                      *:*                         LISTEN      1361/python
tcp        0      0 *:7773                      *:*                         LISTEN      1374/python
```

Figure 24 - Liste des ports en écoute par Shinken

La capture ci-dessus n'affiche pas suffisamment de détails, on ne peut pas savoir quel daemon de Shinken est associé à quel port. On ne sait pas non plus si les processus Python sont des processus Shinken.

En se référant à la liste des processus, on peut déterminer le daemon concerné en comparant les PIDs.

Daemon	Port
Broker	50000
Broker (module WebUI)	7767
Scheduler	7768
Reactionner	7769
Arbiter	7770
Poller	7771
Broker	7772
Receiver	7773

Tableau 2 - Ports et Deamons

J'ai effectué une recherche avec `grep` pour savoir dans quels fichiers et à quel endroit se trouve la configuration de ces ports.

```
[root@shinken etc]# grep -E -R -n "(50000)|(7767)|(7768)|(7769)|(7770)|(7771)|(7772)|(7773)" ./*
./brokerd.ini:15:# port=7772
./pollerd.ini:15:# port=7771
./reactionnerd.ini:7:port=7769
./receiverd.ini:7:port=7773
./schedulerd.ini:5:port=7768
./shinken-specific.cfg:27:  port 7767
./shinken-specific.cfg:127:  port 50000
./shinken-specific.cfg:316:  port 7771
./shinken-specific.cfg:345:  port 7769
./shinken-specific.cfg:353:  port 7772
./shinken-specific.cfg:367:  port 7770
./shinken-specific.cfg:382:  port 7768
./shinken-specific.cfg:388:  port 7773
```

Figure 25 - Recherche des ports dans les fichiers de configuration

Nom du fichier	Ligne	Port
brokerd.ini	15	7772
pollerd.ini	15	7771
reactionnerd.ini	7	7769
receiverd.ini	7	7773
schedulerd.ini	5	7768
shinken-specific.cfg	27	7767
shinken-specific.cfg	127	50000
shinken-specific.cfg	316	7771
shinken-specific.cfg	345	7769
shinken-specific.cfg	353	7772
shinken-specific.cfg	367	7770
shinken-specific.cfg	382	7768
shinken-specific.cfg	388	7773

Tableau 3 - Configuration des ports de communication

10.3. Communications réseau

Shinken utilise la bibliothèque Pyro (PYthon Remote Objects)[11] pour permettre aux deamons de communiquer entre eux. Les connexions réseaux ne sont donc pas gérées directement par Shinken.

Pour analyser les connexions réseau entre les deamons, les commandes netstat[12] et ps[13] ont été utilisées pour récupérer des informations sur les processus de Shinken et leurs connexions établies ou en écoute. En corrélant ces données il a été possible de créer le graphique ci-dessous.

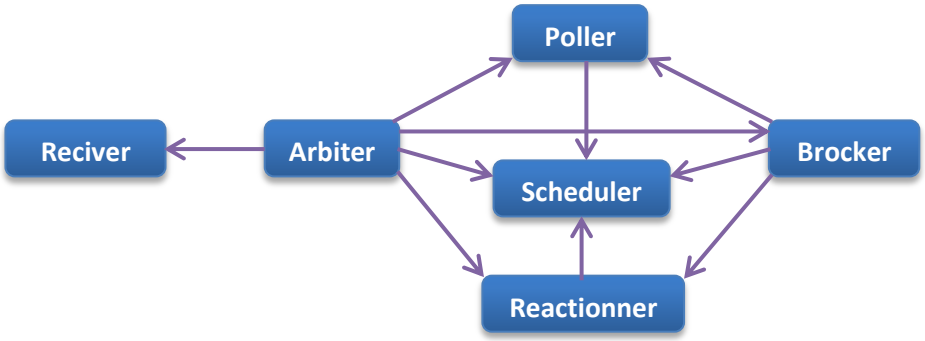


Figure 26 - Schéma réalisé à partir de l'analyse des connexions réseaux entre les daemons

[11] Site officiel de Pyro : http://packages.python.org/Pyro4/
[12] Man page de netstat : http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man8/netstat.8.html
[13] Man page de ps : http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man1/ps.1.html

11. Fichiers de configuration

Les fichiers de configurations de Shinken se trouvent dans le répertoire ***/usr/local/shinken/etc/*** .

On peut distinguer deux types de fichiers de configuration. Il y a les fichiers de configuration qui sont quasiment identiques à ceux de Nagios et ceux qui sont spécifiques à Shinken.

11.1. Spécifiques à Shinken

Les fichiers de configuration spécifiques à Shinken sont au format INI et portent l'extension du même nom. La seule exception est le fichier shinken-specific.cfg qui est un fichier CFG. Ce dernier est lu par l'Arbiter et contient la configuration des daemons et la définition des modules. Il y a un fichier de configuration au format INI par daemon. Ces fichiers sont utilisés pour configurer les paramètres d'exécution et de configuration réseau des deamons.
On y trouve:

- L'utilisateur avec lequel le processus va être lancé (shinken par défaut).
- Le répertoire de travail.
- Le numéro de port et l'adresse IP de l'interface qu'il doit écouter.
- Des paramètres de certificats pour permettre la communication dans un canal SSL.
- L'activation des logs.

Ces paramètres correspondent aux paramètres minimaux pour que l'Arbiter puisse se connecter aux autres daemons et leur transmettre leur configuration définie dans shinken-specific.cfg.

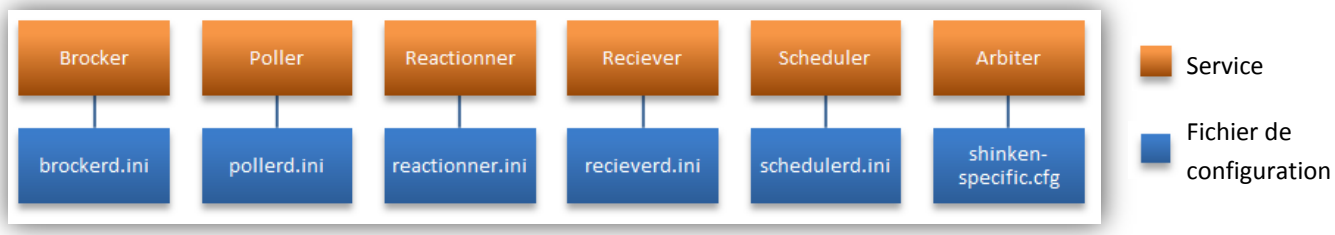


Figure 27 - Fichiers de configuration des daemons Shinken

Dans le cas où Shinken est exécuté sous Windows, le nom de ces fichiers de configuration est suffixé par "-windows".

11.2. Semblables à Nagios

Les autres fichiers de configuration sont des fichiers de configuration au format CFG.

Ces fichiers sont quasiment identiques à ceux que l'on trouve dans Nagios dans le but de pouvoir effectuer une transition de Nagios vers Shinken (et inversement) sans devoir réécrire la configuration.

C'est dans ces fichiers qu'est contenue la configuration des hôtes, des tests, des contacts, etc.

Ces fichiers sont tous inclus dans le fichier nagios.cfg qui est lu par l'Arbiter. L'Arbiter se charge ensuite de répartir la configuration vers les différents daemons.

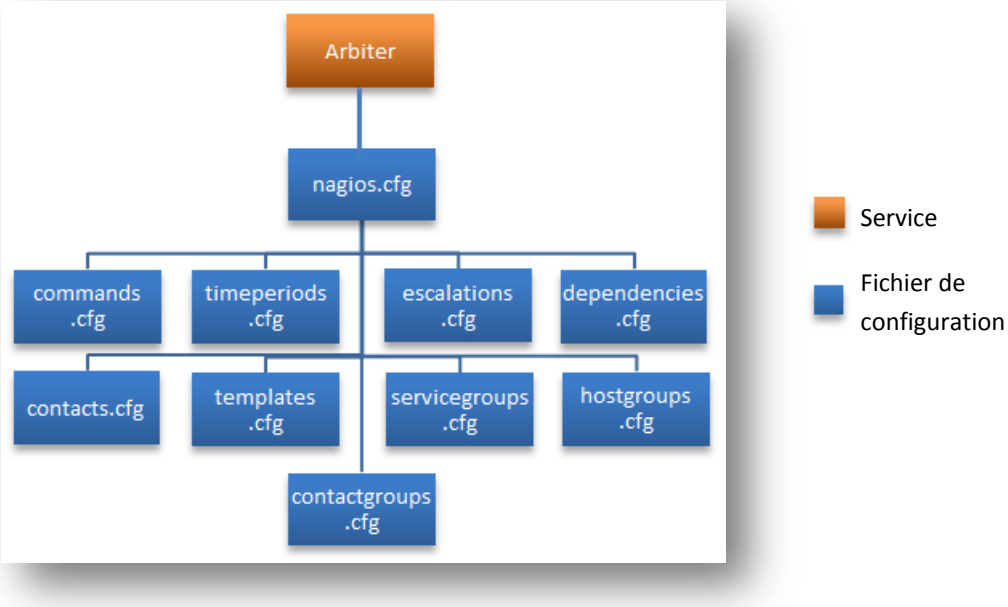


Figure 28 - Fichiers de configuration de l'infrastructure

Le tableau ci-dessous liste les différents fichiers de configuration et décrit leur utilité. Il est néanmoins possible d'écrire la configuration dans n'importe quel fichier tant qu'il est inclus par le fichier nagios.cfg. Il est également possible de mettre toute la configuration dans le fichier nagios.cfg. La structure de ces fichiers est héritée de Nagios et permet de mieux répartir la configuration et de la rendre plus claire.

Les définitions de service dans les fichiers de configuration correspondent aux définitions des tests. Le nom service provient de Nagios.

Nom du fichier	Description
nagios.cfg	Ce fichier inclut tous les autres fichiers de configuration et contient quelques paramètres généraux.
commands.cfg	Il contient la définition des commandes. Il fait la liaison entre les exécutables du répertoire libexec et le nom de la commande utilisée dans la configuration des tests.
timeperiods.cfg	Ce fichier contient la définition des périodes temporelles telles que la semaine de travail ou encore les jours de congés. Ces périodes temporelles peuvent être utilisées pour configurer les périodes où des notifications ne doivent pas être générées.
escalations.cfg	Ce fichier est présent pour des raisons de compatibilité avec Nagios. Cette configuration doit se faire dans le fichier shinken-specific.cfg. La configuration des escalades[14] permet de définir des niveaux de contact et de contacter le niveau supérieur après n notifications ou après un temps t. La fonction temporelle est disponible uniquement sur Shinken.
dependencies.cfg	Ce fichier permet de définir des dépendances entre un hôte et un service ou un autre hôte. Il est préférable de définir la propriété service_dependencies pour les services.
contacts.cfg	Ce fichier contient la définition des contacts.
contactgroups.cfg	Ce fichier contient la définition des groupes de contacts.
templates.cfg	Ce fichier contient les templates qui permettent de définir les paramètres par défaut pour les contacts, les hôtes et les services.

[14] Escalades temporelles dans Shinken : http://www.shinken-monitoring.org/news/escalation-based-on-time-for-sla/

servicesgroups.cfg	Ce fichier contient la définition des groupes de services.
hostgroups.cfg	Ce fichier contient la définition des groupes d'hôtes.
hosts	Ce répertoire est prévu pour contenir des fichiers CFG définissant les hôtes.

Tableau 4 - Description des fichiers de configuuration

11.3. Vérification des fichiers de configuration

Shinken vérifie les fichiers de configuration lorsqu'on le redémarre. Néanmoins il affiche uniquement les erreurs de syntaxe.

La vérification enregistre des données plus détaillées dans le fichier */tmp/shinken_checkconfig_result*. Ce fichier donne des informations telles que les fichiers lus, les paramètres non reconnus qui sont présents dans les fichiers de configuration.

12. Mise en pratique du scénario

Pour configurer le scénario, trois fichiers ont été modifiés et un fichier a été créé. La configuration par défaut des daemons n'a pas été modifiée car elle correspond déjà à une configuration minimale.

12.1. Ajout d'un contact

Les contacts sont normalement ajoutés dans le fichier *etc/contacts.cfg*. Le contact créé se nomme admin. Il faut configurer l'adresse email pour recevoir les emails de notifications. Le champ `use` permet de se baser sur un template qui contient les valeurs de configurations par défaut. Pour plus de détails voir le chapitre Fichiers de configuration.

```
[root@shinken shinken]# less etc/contacts.cfg
define contact{
        use                                     generic-contact
        contact_name                            admin
        email                                   lionel.schaub@master.hes-so.ch
        pager                                   0600000000    ; contact phone number
        password                                admin
        is_admin                                1
}
```

Figure 29 - Fichier de configuration des contacts

12.2. Ajout d'un groupe de contact

Le groupe de contacts admins a été créé avec comme seul utilisateur l'utilisateur admin créé précédemment. Les groupes sont généralement ajoutés dans le fichier *etc/contactgroups.cfg*.

```
[root@shinken shinken]# less etc/contactgroups.cfg
define contactgroup{
        contactgroup_name       admins
        alias     admins
        members   admin
}
```

Figure 30 - Fichier de configuration des groupes de contacts

12.3. Ajout d'une machine (host)

Un répertoire nommé *etc/hosts/* est prévu pour stocker les fichiers de configuration des machines. Le fichier *machinea.cfg* a été créé avec la définition de la machine nommée MachineA. On remarque dans ce fichier qu'il hérite du template generic-host. On remarque que le groupe de contacts associé à cette machine est admins.

```
[root@shinken shinken]# less etc/hosts/machinea.cfg
define host{
        use                     generic-host
        contact_groups          admins
        host_name               MachineA
        address                 192.168.230.30
        icon_set                server
        }

define service{
        use                     generic-service
        host_name               MachineA
        service_description     Ping
        check_command           check_ping
        normal_check_interval   1
        retry_check_interval    1
        initial_state           u
        }
```

Figure 31 - Fichier de configuration de la MachineA

12.4. Recherche et configuration d'une commande

Les définitions des commandes se trouvent dans le fichier de configuration *etc/commands.cfg*. On y trouve la commande check_ping qui appelle le plugin check_ping avec un certain nombre d'arguments. Ces arguments signifient que le plugin va retourner Critical si la requête n'a pas reçu de réponse après cinq secondes et Warning si la requête a obtenu une réponse après trois secondes et avant cinq secondes. Elle indique également qu'une seule requête ICMP sera effectuée.

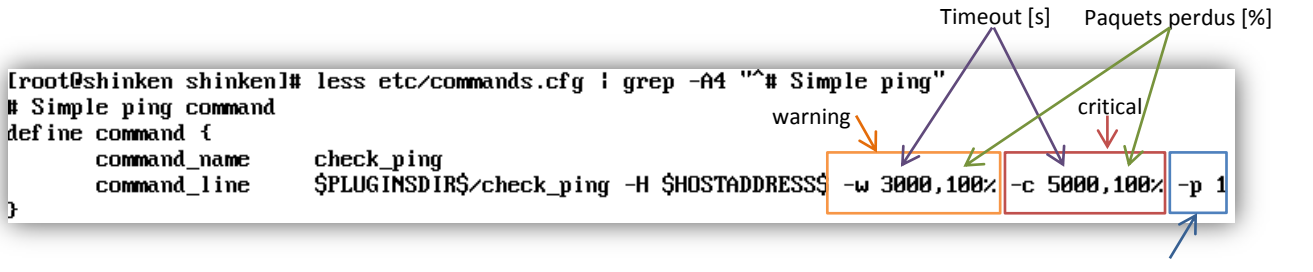


Figure 32 - Fichier de configuration des plugins

12.5. Ajout d'un service à une machine

Le service a été configuré dans le même fichier que la machine. Le paramètre host_name fait la liaison entre le service et la machine (host). Le paramètre check_command fait la liaison entre le service et la commande auquel il est associé. Le paramètre normal_check_interval permet de définir l'intervalle entre deux tests (en min.) et le paramètre retry_check_interval permet de définir l'intervalle entre deux tests (check) lorsque le dernier test a retourné Critical. Et pour finir le paramètre initial_state définit l'état initial (avant que le premier test n'ait été effectué) à Unknown.

```
[root@shinken shinken]# less etc/hosts/machinea.cfg
define host{
        use                     generic-host
        contact_groups          admins
        host_name               MachineA
        address                 192.168.230.30
        icon_set                server
        }

define service{
        use                     generic-service
        host_name               MachineA
        service_description     Ping
        check_command           check_ping
        normal_check_interval   1
        retry_check_interval    1
        initial_state           u
        }
```

Figure 33 - Fichier de configuration du test de la MachineA

12.6. Ajout d'un groupe de machines

Ces derniers sont généralement créés dans le fichier *etc/hostgroups.cfg*. Un groupe nommé postes-clients a été créé avec comme seule machine la MachineA créée précédemment.

```
[root@shinken shinken]# less etc/hostgroups.cfg
define hostgroup{
        hostgroup_name postes-clients
        alias          postes-clients
        members        MachineA
}
```

Figure 34 - Fichier de configuration des groupes de machines

12.7. Redémarrage de Shinken

Pour que la configuration soit prise en compte il faut redémarrer Shinken :

```
/etc/init.d/shinken restart
```

13. Développement des Plugins

Dans le cadre de ce projet deux plugins ont été développés. Ce chapitre décrit les différentes étapes du développement de ces deux plugins. Les informations sur le système de plugin de Shinken sont disponibles dans le chapitre Plugins et agents de ce document.

13.1. Vérification de l'espace disque

Le plugin qui devait être développé devait permettre de vérifier l'espace disque restant d'une machine Linux distante. Avant de commencer le développement j'ai vérifié si un plugin similaire existait et j'ai trouvé le plugin *check_disk* parmi les plugins provenant de Nagios.

Ce plugin permet de vérifier l'espace disponible sur un disque dur local ou une partition locale. Comme le but est de vérifier l'espace disponible d'une machine distante il faut trouver un moyen d'exécuter ce script à distance. En me basant sur un travail[15] effectué précédemment au laboratoire de transmission de données par M. Lescourt Adrien, j'ai découvert le plugin *check_by_ssh* qui permet d'exécuter une commande distante grâce à SSH et qui retourne ensuite le code de retour et le message retourné par la commande distante.

En combinant ces deux plugins il est possible de vérifier l'espace disque restant d'une machine distante. Vu que la fonctionnalité désirée peut être obtenue en combinant les plugins *check_disk* et *check_by_ssh*, c'est cette solution qui a été mise en place plutôt que de développer un plugin.

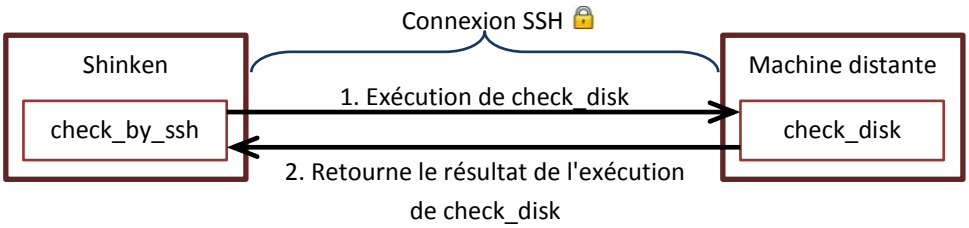


Figure 35 - Schéma du fonctionnement de la vérification de l'espace disque par Shinken

Pour commencer, le plugin check_ssh a été testé localement sur la machine cible (machine distante). Une fois cette étape terminée, un serveur SSH a été installé et configuré sur la machine cible puis la connexion SSH depuis la machine contenant Shinken a été testée. Ensuite, le plugin check_by_ssh a été testé manuellement depuis la ligne de commande de la machine contenant Shinken. Finalement, Shinken a été configuré pour exécuter ce test sur la machine distante.

13.1.1. Mise en application

Tout d'abord, il faut s'assurer que la machine sur laquelle on veut exécuter le test dispose d'un serveur SSH actif. La procédure pour installer un serveur SSH est disponible dans l'annexe Installer un serveur SSH sur Ubuntu.

Sur la machine Shinken il faut se connecter sous le compte shinken : `su shinken`

Sur la machine Shinken il faut générer les clefs à l'aide de la commande `ssh-keygen` (accepter les paramètres par défaut).

[15] Lien vers le travail de M. Lescourt : http://www.tdeig.ch/shinken/Adrien_Lescourt_PluginNagios.pdf

Une fois la paire de clefs créée il faut placer la clef publique sur le serveur SSH. Sur la machine Shinken lancer la commande `ssh-copy-id utilisateur@adresse-ip`

Il faut maintenant approuver le serveur SSH. La méthode la plus simple est d'essayer de se connecter au serveur SSH grâce à la commande `ssh utilisateur@adresse-ip`

L'empreinte de la clef du serveur SSH va s'afficher et un message demandera de la confirmer. Il faut vérifier que l'empreinte correspond à celle du serveur SSH et, dans le cas échéant, confirmer la clef.

Il faut copier le plugin check_disk dans la machine distante. Une solution simple consiste à utiliser SCP[16] (copie de fichier via SSH) :
`scp libexec/check_disk utilisateur@adress-ip:/usr/local/shinken/libexec/`

Se déconnecter du compte shinken : `exit`

Configurer une commande dans le fichier *etc/commands.cfg* :

```
define command{
        command_name    check_ssh_disk
        command_line    $USER1$/check_by_ssh -H $HOSTADDRESS$ --command="$USER1$/check_disk
--warning=10% --critical=5% --path=/media/testVolume" --logname=utilisateur
}
```

Figure 36 - Commande check_ssh_disk

La commande définie ci-dessus vérifie, via SSH, l'espace disque de la partition montée dans le répertoire /media/testVolume. Si l'espace restant est inférieur à 5% la commande renverra "Critical" et si l'espace restant est inférieur à 10% et supérieur à 5% la commande renverra "Warning".

La dernière étape consiste à créer un service associé à la machine distante. Le service défini dans la capture ci-dessous est configuré pour effectuer le test toutes les 10 minutes et toutes les minutes lorsque l'état vaut "Critical". Ceci permet de vérifier rapidement que l'état est revenu à Ok lorsque l'on a libéré de l'espace disque.

```
define service{
        use                     generic-service
        host_name               MachineC
        service_description     Occupation du disque dur
        check_command           check_ssh_disk
        normal_check_interval   10
        retry_check_interval    1
        initial_state           u
}
```

Figure 37 - Service check_ssh_disk

Pour prendre en compte la nouvelle configuration il faut redémarrer Shinken :
`/etc/init.d/shinken restart`

[16] Transférer des fichiers avec SCP : http://cc.in2p3.fr/docenligne/134/fr

13.1.2. Tests effectués

Pour tester le bon fonctionnement de la configuration effectuée ci-dessus, trois tests ont étés effectués.

13.1.2.1. Test du statut Ok

Des fichiers ont été placés sur la machine supervisée de sorte à ce que l'espace disque restant soit d'environ 85%. Au prochain test, Shinken affichait bien que l'état de la machine était "Ok".

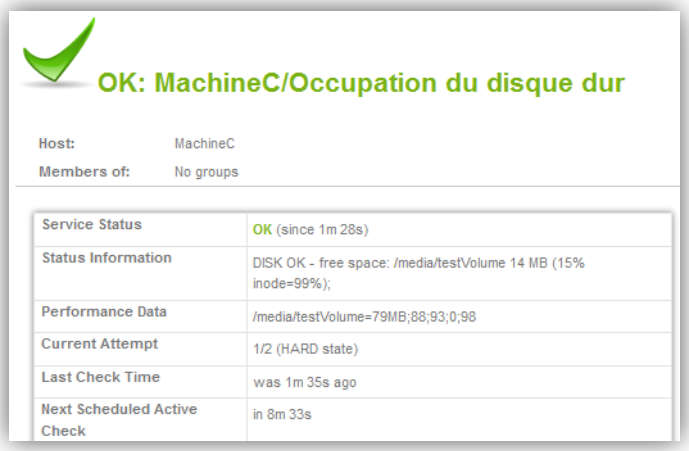


Figure 38 - check_ssh_disk Ok

13.1.2.2. Test du statut Warning

Pour ce test un fichier a été ajouté sur la machine supervisée pour que le taux d'utilisation du disque soit de 91%. Quelques minutes plus tard, Shinken affichait la machine comme étant dans l'état "Warning".

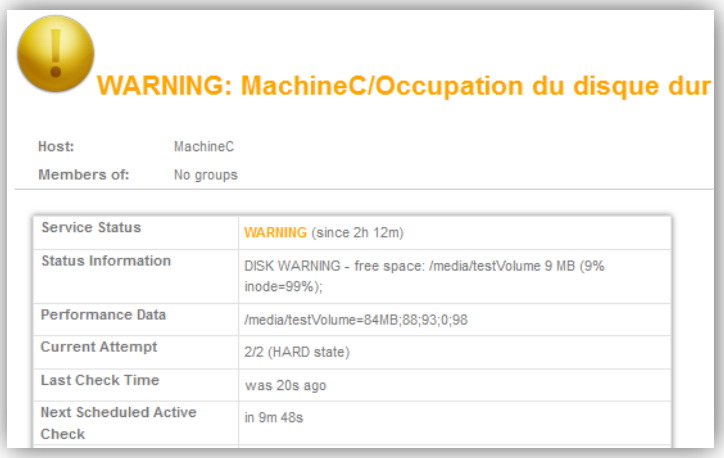


Figure 39 - check_ssh_disk Warning

13.1.2.3. Test du statut Critical

Pour tester l'état "Critical" le disque dur de la machine supervisée a été rempli à 96%. Après que le test a été effectué, Shinken affichait la machine comme étant dans un état "Critical".

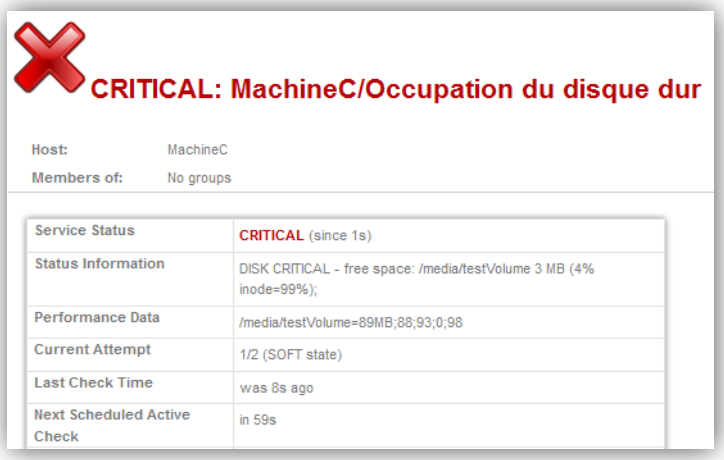


Figure 40 - check_ssh_disk Critical

Dans la capture ci-dessus on voit que le dernier test a été lancé il y a 8 secondes et qu'il s'est terminé il y a 1 seconde. On peut donc déterminer que le test s'est exécuté pendant 7 secondes. On remarque également que le prochain test est planifié dans 59 secondes soit une minute après la fin du dernier test. Ce test valide donc aussi la configuration effectuée avec la propriété *retry_check_interval*.

13.2. Web Defacing

13.2.1. Définition

Certains hackers cherchent des failles sur des sites web et, lorsqu'ils en trouvent une, exploitent la faille dans le but de modifier la page d'accueil sans avoir obtenu l'autorisation du propriétaire du site. Cet acte est nommé Web Defacing. La page web modifiée contient généralement un message relativement grand qui indique que le site web a été "défacé".

13.2.2. Exemple

La capture ci-dessous montre la page d'accueil du site web IBM developerWorks après qu'il ait été "défacé".

Figure 41 - Page web du site http://www.ibm.com/developerworks/ en Juillet 2010

13.2.3. Choix du plugin

Le plugin que j'ai choisi de développer doit permettre de détecter si un site web a été "défacé".

Lors de la recherche d'une idée de plugin, la principale difficulté fut d'en trouver un qui n'existait pas encore. En effet, il existe une grande quantité de plugins développés à l'origine pour Nagios et qui fonctionnent donc sur Shinken.

13.2.4. Choix du langage de programmation

Le langage de programmation que j'ai choisi est Python.

Ce langage nécessite que l'interpréteur Python soit installé sur la machine. Ceci est une fausse contrainte car si on utilise Shinken il a lui-même besoin de l'interpréteur Python, donc si le plugin est utilisé avec Shinken on est certain que l'interpréteur sera installé. De plus, l'interpréteur Python est installé sur la quasi-totalité des distributions Linux.

Un programme Python peut être exécuté sur toutes les plateformes pour lesquels un interpréteur Python est disponible (Linux, Windows, MacOSX, Android, etc.).

J'ai également choisi Python pour respecter la philosophie de Shinken ainsi que pour sa bibliothèque standard qui contient beaucoup de fonctions prêtes à l'emploi.

13.2.5. Comparaison de deux pages web

Une comparaison de deux pages web peut être vue comme une comparaison de deux fichiers texte. La comparaison de deux fichiers texte n'est pas une opération triviale.

13.2.5.1. Problématique

En effet, si on utilise un algorithme naïf, il va parcourir tous les caractères du fichier tout en comparant si le caractère à la même position dans l'autre fichier est identique. Cet algorithme pose problème car, si l'on ajoute un espace au milieu du fichier, tous les caractères suivants seront décalés et l'algorithme retournera un taux de modification d'environ 50% alors que seul un caractère a été ajouté.

13.2.5.2. Algorithme de recherche de la plus longue sous-séquence commune

Pour éviter ce problème, il faut créer un algorithme qui, pour chaque différence de caractère détectée, va tenter de retrouver la suite de la chaîne plus loin dans les fichiers. Si l'on prend l'exemple précédent l'algorithme va détecter que l'espace ajouté dans le deuxième fichier ne correspond pas au caractère du premier fichier. Il va donc avancer d'un caractère dans le premier fichier et vérifier si le caractère correspond à un espace. Puis il va continuer d'avancer dans le fichier jusqu'à arriver à la fin du fichier. Ensuite il va revenir au point où il a détecté la différence, puis il va avancer dans le deuxième fichier. A ce moment-là, il va sauter l'espace qui a été ajouté et il va continuer de comparer les fichiers avec un décalage d'un caractère dans le deuxième fichier. Ces étapes sont effectuées à chaque fois qu'une différence est trouvée.

Cet algorithme s'appelle *l'algorithme de recherche de la plus longue sous-séquence commune*. Son défaut majeur réside dans sa complexité algorithmique qui vaut dans le pire des cas $O(n^2)$ (n étant le nombre de caractères) et dans le meilleur des cas (fichiers identiques) O(n). Néanmoins, il a l'avantage d'être très facile à implémenter en utilisant la récursivité.

```
def algo(self, str1, str2, p1=0, p2=0):
    if p1 >= len(str1) or p2 >= len(str2):
        return 0
    if str1[p1] == str2[p2]:
        return 1 + self.algo(str1, str2, p1+1, p2+1)
    else:
        return max( [self.algo(str1, str2, p1+1, p2), self.algo(str1, str2, p1, p2+1)] )
```

Figure 42 - Algorithme de recherche de la plus longue sous-séquence commune en Python

Cet algorithme a été implémenté mais il n'est pas utilisé dans le plugin car le temps de calcul est trop élevé et augmente exponentiellement en fonction du nombre de caractères.

13.2.5.3. Comparaison des lignes du fichier

Le plugin utilise une classe de la bibliothèque Python standard appelée *difflib.SequenceMatcher*. Cette classe fournit deux méthodes intéressantes:

- ratio(): Cette fonction calcule le pourcentage de similarité entre deux fichiers texte. Elle implémente l'algorithme de recherche de la plus longue sous-séquence commune en l'appliquant sur les lignes plutôt que sur les caractères. Cette fonction a également une complexité en $O(n^2)$, néanmoins dans ce cas, n correspond au nombre de lignes et l'algorithme s'exécute dans un temps raisonnable (moins de 5 sec.) pour une page web de taille standard.
- quick_ration(): Cette fonction calcule également le pourcentage de similarité mais de manière plus rapide. Elle utilise le même algorithme que la fonction ratio mais ajoute une méthode heuristique[17] pour éviter d'effectuer certaines parties du calcul et ainsi réduire le temps d'exécution. Cette méthode a le défaut de fournir un résultat non-exact qui ne peut être qu'inférieur ou égal au résultat exact. Les tests effectués montrent que l'utilisation de cette fonction n'est pas adaptée pour la comparaison de pages web car les résultats obtenus sont peuvent être plus de quatre fois inférieurs au résultat exact, tout particulièrement lorsque le pourcentage de modification de la page est faible. Néanmoins, si le besoin en rapidité d'exécution est important et que le pourcentage de modification toléré est supérieur à 10%, il peut être acceptable d'utiliser cette fonction.

Cette méthode de comparaison est implémentée dans la classe *LineComparator*. Un flag permet de choisir la version rapide ou la version précise.

13.2.5.4. Comparaison de la taille des pages

Comparer la taille des pages est une méthode très simple et très efficace dans le cas où les modifications effectuées sont soit majoritairement des suppressions, soit majoritairement des insertions.

Cette méthode de comparaison est implémentée dans la classe *FileSizeComparator*.

13.2.5.5. Comparaison utilisant un parseur

Comme la méthode standard de comparaison des fichiers a une complexité exponentielle et que son temps d'exécution peut être trop élevé, il a fallu trouver une méthode plus efficace.

[17] La méthode heuristique est expliquée ici : http://docs.python.org/library/difflib.html

L'idée retenue consiste à utiliser un parseur HTML pour extraire des éléments de la page web et ensuite les comparer. Le parseur utilisé se nomme *Beautiful Soup*[18]. Il est écrit en Python et fonctionne par conséquent sur toutes les plateformes disposant d'un interpréteur Python. Un parseur HTML parcourt la page une seule fois, par conséquent sa complexité est en O(n).

Beautiful Soup n'est pas installé par défaut. Une procédure d'installation est disponible dans l'annexe Installer *Beautiful Soup* de ce document.

13.2.5.5.1. Comparaison des textes visibles

Le parseur a été utilisé pour extraire une liste des textes de la page web. Cette liste est comparée avec la liste de la page web précédemment téléchargée. La comparaison s'effectue en parcourant une des deux listes et en vérifiant pour chaque élément si un élément identique existe dans l'autre liste. Si c'est le cas, cet élément est supprimé des deux listes. Les éléments restants dans les deux listes correspondent aux différences. En effectuant le rapport du nombre de différences sur le nombre d'éléments dans les deux listes, on obtient le pourcentage de différences. Cette méthode ne prend pas en compte l'ordre d'apparition des éléments dans le code source de la page web.

13.2.5.5.2. Comparaison de la structure de la page web

Le parseur a également été utilisé pour extraire la structure de la page web (l'arbre DOM). Cet arbre a été aplati pour qu'il puisse être comparé sous la forme d'une liste de textes. Pour aplatir l'arbre chaque balise est ajoutée à la liste et est précédée du nom de toutes les balises dans lesquelle la balise courante est contenue.

Si on considère le code HTML suivant:

```
<html>
    <head>
        <title>Ma page web</title>
    </head>
    <body>
        <div>
            Le contenu de ma page web.
        </div>
    </body>
</html>
```

Code 1 - Code HTML

La structure de ce code sera aplatie de la manière suivante:

```
<html>
<html><head>
<html><head><title>
<html><body>
<html><body><div>
```

[18] Beautiful Soup : http://www.crummy.com/software/BeautifulSoup/

13.2.6. Structure du plugin

Le plugin est découpé en six classes.

Il y a quatre classes qui implémentent les différentes méthodes de comparaison. Ces quatre classes héritent de la classe abstraite *Comparator* et doivent toutes posséder une fonction *get_diff()* qui prend en entrée deux chaînes de caractères correspondant aux deux versions de la page à comparer. Cette fonction doit retourner le pourcentage de modification. La classe *LineComparator* prend un paramètre supplémentaire qui permet de choisir la méthode rapide ou la méthode précise. La classe *HtmlDomComparator* prend un paramètre supplémentaire permettant de définir si les attributs des balises sont inclus ou exclus lors de la comparaison.

La classe *CheckWebDefacing* se charge de récupérer le contenu de la page web grâce à la class *urllib2* de la bibliothèque standard de Python. Le contenu de la page web est ensuite stocké dans le répertoire */tmp/check_web_defacing*. Si la page web n'est pas présente dans ce répertoire, le plugin va créer un fichier contenant la page web puis retourner 3 (Unknown). Le nom du fichier correspond à l'URL encodée en *base64*[19] afin d'éviter des problèmes avec des caractères spéciaux. Si la page web est présente dans le répertoire, elle sera comparée avec la page qui vient d'être téléchargée puis elle sera remplacée par la page qui vient d'être téléchargée. La classe *CheckWebDefacing* s'occupe également de lancer les différentes méthodes de comparaison en fonction des différents arguments passés au plugin. Si plusieurs méthodes de comparaison ont été choisies, le résultat retourné sera une moyenne des résultats de chaque méthode.

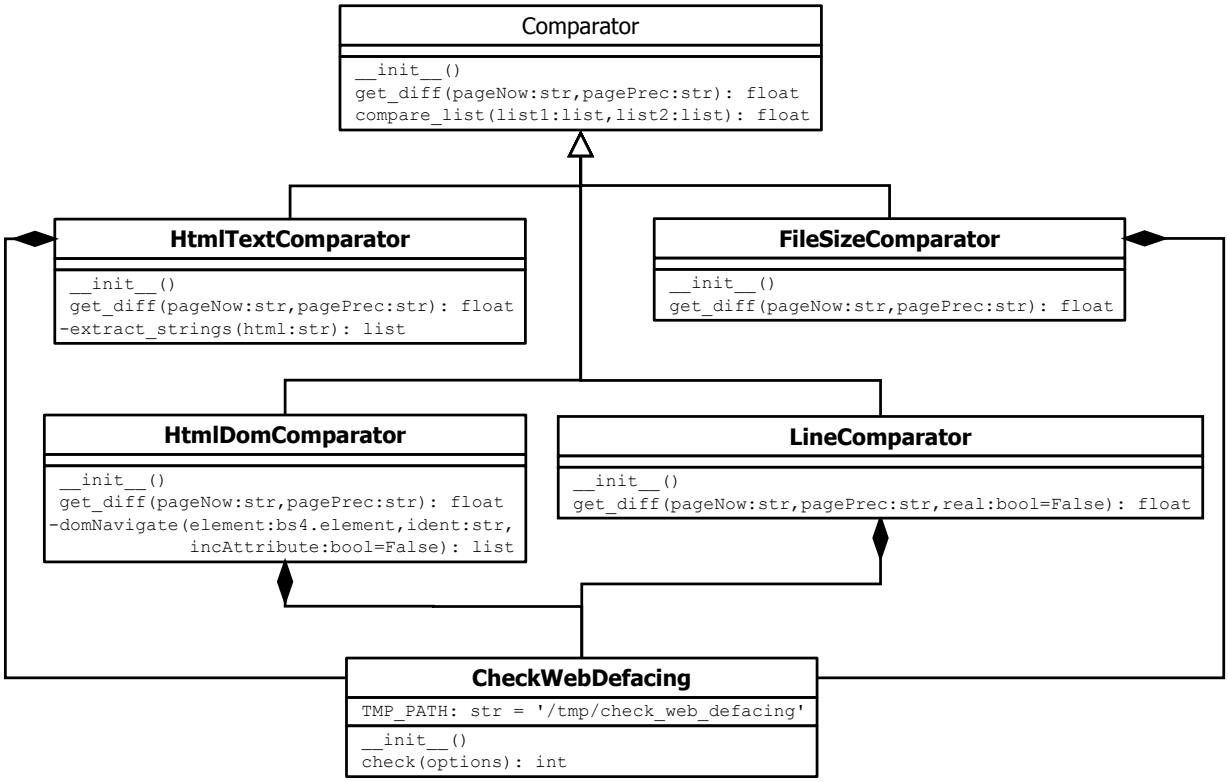


Figure 43 - Diagramme de classe du plugin check_web_defacing

[19] Encodage base64 sur Wikipédia : http://fr.wikipedia.org/wiki/Base64

13.2.7. Configuration du plugin dans Shinken

Pour configurer le plugin dans Shinken, le plugin a été ajouté dans le répertoire *libexec*.

Une commande a été ajoutée au fichier commands.cfg.

```
define command{
        command_name    check_web_defacing
        command_line    $USER1$/check_web_defacing.py -w 1.5 -c 2  -q -l -s -t -d -a  -u $ARG1$
}
```

Figure 44 - Commande check_web_defacing

Puis un host correspondant à un serveur web a été ajouté.

```
define host{
        use             generic-host
        contact_groups  admins
        host_name       ServeurWeb
        address         192.168.230.60
        icon_set        server

}


define service{
        use                     generic-service
        host_name               ServeurWeb
        service_description     Web Defacing
        check_command           check_web_defacing!http://www.tdeig.ch
        normal_check_interval   1
        retry_check_interval    1
        initial_state           u
        max_check_attempts      10
}
```

Figure 45 - Host & Service check_web_defacing

Pour appliquer la nouvelle configuration Shinken doit être relancé: `/etc/init.d/shinken restart`

13.2.8. Tests effectués

Des tests ont été effectués durant toute la durée du développement. Chaque nouvelle fonctionnalité a été testée au fur et à mesure de leur développement.

A la fin du développement des tests ont été effectué pour déterminer les types de modification le plugin est capable de détecter et les type qu'il n'est pas capable de détecter. Ces tests ont été effectués en se mettant dans la peau d'une personne malveillante qui cherche à contourner le mécanisme de détection du plugin.

Le tableau ci-dessous fournit le résultat de ces tests.
V : détecté X : non détecté O: détecté en configurant des pourcentages très faibles

Comparateur ->	FileSize	Line	HtmlText	HtmlDom
Remplacement du code HTML par un nouveau code	V	V	V	V
Remplacement du code HTML par un nouveau code et ajout de caractère de bourrage pour que taille de la nouvelle page web corresponde à l'ancienne.	X	V	V	V
Mettre le code HTML existant en commentaire et ajout d'un peu de code.	O	O	V	V
Ajout d'un script JavaScript qui modifie la page.	O	O	X	O
Modification d'un script JavaScript existant pour lui faire modifier la page.	O	O	X	X
Modification d'un style CSS pour masquer le contenu actuel de la page et ajout d'un bout de code HTML.	O	O	O	O
Modification d'un style CSS pour masquer le contenu actuel de la page et modification d'un texte.	O	O	O	X

13.2.9. Améliorations possibles

Une comparaison des scripts JavaScript et des styles CSS pourrait être ajoutée pour diminuer le risque de faux négatif.

L'utilisation d'une moyenne pour rassembler les résultats des différentes méthodes n'est pas forcément la meilleure solution. Une réflexion pourrait être ouverte pour développer une meilleure façon de procéder.

La page web est téléchargée, stockée sur le disque dur puis traitée (extraction des textes, etc.). Lors du prochain test la page web est lue depuis le fichier puis traitée à nouveau. A chaque test il y a deux pages web qui sont traitées. Il serait possible de stocker les pages web après leur traitement. De ce fait, il n'y aurait que la page web qui vient d'être téléchargée qui serait traitée. Cette modification permettrait de diminuer le temps d'exécution du plugin mais générerait plus d'accès disque.

14. Difficultés rencontrées

14.1. Propriétés de configuration non documentées

Lorsque j'ai parcouru les fichiers de configuration, je me suis interrogé sur l'effet de certaines propriétés. En cherchant dans la documentation j'ai constaté que certaines propriétés n'étaient pas documentées.

14.2. Plugins non inclus par défaut

Après avoir terminé l'installation de Shinken, les plugins nécessaires à la configuration par défaut ne sont pas présents. Par conséquent, lorsque l'on se connecte sur l'interface web, il y a déjà des alertes indiquant que certains plugins ne sont pas présents. Un certain nombre peut être installé grâce au programme d'installation. Des méthodes pour installer les plugins sont fournies dans l'annexe A.1.

14.3. Compréhension de la logique et du fonctionnement des fichiers de configuration

Comme Shinken se veut compatible avec Nagios, son système de configuration est très similaire à celui de Nagios. La partie de la configuration qui est identique à celle de Nagios n'est pas forcément documentée. Ne connaissant pas Nagios, j'ai dû commencer par comprendre le système Nagios avant de pouvoir configurer Shinken.

14.4. Recherche d'informations dans les sources

Bien que le code Python soit agréable à lire, la recherche d'informations est difficile car il y a beaucoup de fichiers source et chaque fichier est utilisé par plusieurs daemons.

14.5. Problème d'ordonnancement des tests des hôtes

Shinken planifie les tests (checks) des hôtes selon plusieurs critères expliqués ici : http://www.shinken-monitoring.org/wiki/official/thebasics-hostchecks .

L'hôte que j'ai configuré avait son prochain test planifié à 30 jours dans le futur. Tous les paramètres de configuration étaient correctement configurés. Malgré plusieurs redémarrages de Shinken, la date du prochain test ne se mettait pas à jour.

J'ai donc du recréer un hôte dans les fichiers de configuration. La date du prochain test pour le nouvel hôte était correcte.

14.6. Utilisation de check_by_ssh

Le plugin `check_by_ssh` retournait l'erreur suivante : *host key verification failed*

L'erreur apparaissait uniquement lorsque la commande était exécutée par Shinken. En l'exécutant directement sur la console la commande fonctionnait.

Le problème se produit car la génération et la copie de la paire de clefs a été effectuée en tant que root alors que le plugin est exécuté en tant que shinken.

La solution consiste à se connecter sous l'utilisateur shinken grâce à la commande `su shinken` puis d'effectuer la génération et la copie de la paire de clefs. Pour tester le bon fonctionnement de la connexion SSH, exécuter la commande `ssh user_distant@machine_distante`. Cette commande va demander si on accepte la clef de la machine distante. Une fois la clef acceptée, cette dernière sera ajoutée au fichier contenant les clefs approuvées par l'utilisateur shinken.

14.7. Comparaison de deux listes en Python

Pour comparer deux listes en Python le programme itère une des deux liste puis vérifie si l'élément courant est présent dans l'autre liste. Si c'est le cas, l'élément est supprimé dans chaque liste. La somme des éléments restants dans les deux listes correspond aux différences.

Pourtant, il n'y a pas tous les éléments présents dans les deux listes qui ont été supprimés. Ceci est dû à l'implémentation Python des itérations de boucles. Lorsque l'on supprime un élément d'une liste, tout ce qui suit cet élément est décalé. Après avoir supprimé l'élément dans la liste, l'élément suivant se trouve à la position de l'élément qui a été supprimé. Par conséquent, quand on passe à l'itération suivante on aura sauté un élément. Pour régler le problème, le programme a été modifié pour ajouter les éléments à supprimer dans une liste d'éléments à supprimer. Une fois le parcours terminé, le programme parcourt la liste des éléments à supprimer tout en supprimant les éléments dans les deux listes comparées.

Le schéma ci-dessus représente graphiquement le problème. On remarque sur ce schéma que la valeur "Texte 2" n'est pas testée et n'est pas supprimée. On remarque aussi que la valeur "Texte 4" n'est pas testée.

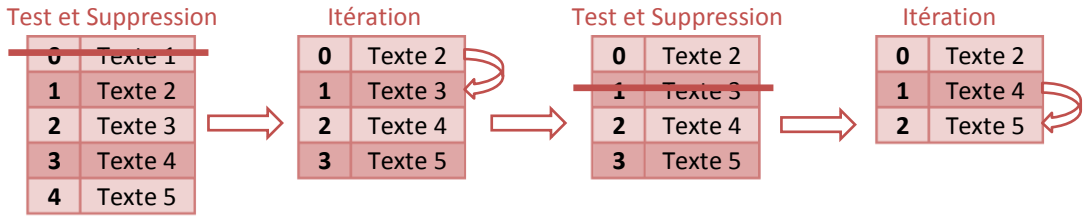
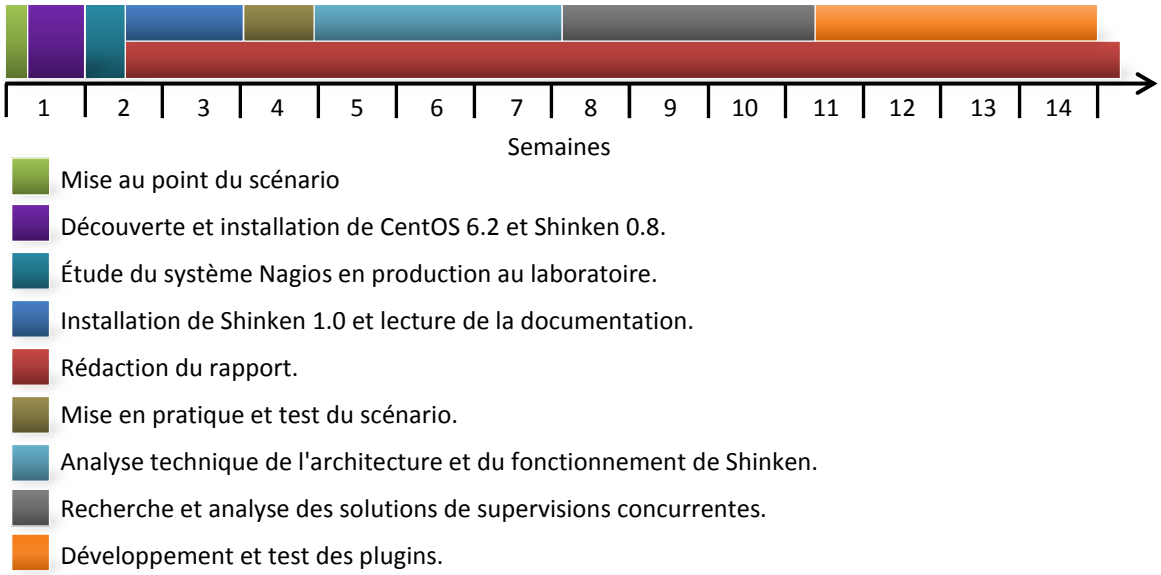


Figure 46 - Problème de comparaison de deux listes en Python

15. Répartition du temps

Ce projet d'approfondissement s'est déroulé sur quatorze semaines à raison de treize heures par semaines.

Le chronogramme ci-dessous détaille la répartition du travail sur les quatorze semaines.



16. Conclusion

Les objectifs de ce travail ont tous été atteints. Bien que le plugin développé puisse encore être amélioré et optimisé, la partie fonctionnelle a été obtenue et validée. Ce travail m'a permis d'approfondir le domaine de la supervision et de prendre conscience des besoins opérationnels en mesure de disponibilité.

La mise en place et l'utilisation d'une solution de supervision est une nécessité dans toutes les entreprises qui disposent de serveurs fournissant des services aux employés et/ou aux clients. L'importance de l'utilisation d'une solution de supervision ne doit pas être prise à la légère. En effet, tout système informatique d'une entreprise contient des éléments critiques qui doivent fonctionner à tout moment pour garantir la bonne marche de l'entreprise. Les sociétés ne disposent pas de versions papier de toutes leurs données, par conséquent, lorsque l'accès à ces données est interrompu, un certain nombre d'employés ne peuvent plus travailler ce qui résulte en une perte financière non négligeable.

Un outil de supervision tel que Shinken permet non seulement de détecter une panne d'un service ou d'une machine, mais aussi de d'en prévenir une en surveillant des indicateurs tel que la température ou encore le nombre d'erreurs d'E/S d'un disque dur, etc. Dans des outils tels que Shinken ou Nagios, cette fonctionnalité est implémentée par l'état "Warning".

Avant de mettre en place une solution de supervision, il est essentiel de connaitre précisément ses besoins opérationnels en mesure de disponibilité. Le chapitre Besoins opérationnels de mesure de disponibilité de ce document fournit la liste des besoins fondamentaux dans ce domaine. Cette dernière peut être utilisée pour aider à déterminer ses propres besoins. Une fois la liste des besoins établie, il est possible de concevoir (à l'aide d'outils de supervision) ou de choisir une solution de supervision adaptée. La solution peut, par la suite, être agrémentée par d'autres outils dans le but de répondre à des besoins annexes.

Un autre point important dans le choix d'une solution de supervision est le temps et l'argent que l'on souhaite investir dans le déploiement de la solution. La plupart des solutions propriétaires sont fournies prêtes à l'emploi ou comprennent le déploiement dans leur tarifs. Elles sont donc déployées rapidement au détriment d'un coût qui peut être élevé. Tandis que la plupart des solutions de type open source sont gratuites, elles nécessitent néanmoins un investissement temporel important. Les solutions open source proposent généralement d'acheter un service d'installation et/ou de maintenance du système.

Shinken est un produit moderne, modulaire et suffisamment stable pour être utilisé en production. Il est possible de migrer de Nagios à Shinken en moins d'une heure. Il doit cependant être utilisé en association avec d'autres outils pour former une solution de supervision complète. Son architecture distribuable devrait lui permettre d'être installé dans les plus grands datacenters dans lesquels Nagios montre ses limites. Des essais sur ces types de datacenter doivent encore être effectués pour valider le bon fonctionnement de Shinken et éventuellement détecter des goulots d'étranglement dans son architecture.

Le développement d'un plugin pour Shinken est une opération qui a très peu de contraintes. Tout fichier exécutable peut être un plugin à condition de retourner un code de retour correspondant à un des quatre états prédéfinis (Ok, Warning, Critical, Unknown). Comme Shinken utilise les mêmes codes de retour que Nagios, les plugins développés pour l'un fonctionnent également sur l'autre.

Le développement du plugin *check_web_defacing* m'a permis de me confronter à la problématique non triviale de la comparaison du contenu de deux fichiers. Les algorithmes standards de comparaison de fichiers étant de complexité exponentielle, j'ai dû élaborer des méthodes alternatives moins coûteuses en temps de calcul.

17. Liens utiles & références

- Site web officiel de Shinken : http://www.shinken-monitoring.org/
- Site web officiel de Nagios : http://www.nagios.org/
- Documentation (Wiki) de Shinken : http://www.shinken-monitoring.org/wiki/
- Documentation de Nagios : http://support.nagios.com/
- Plugins pour Nagios (fonctionnant sur Shinken) : http://exchange.nagios.org/directory/Plugins
- Un article rédigé par le créateur de Shinken (Jean Gabes) paru dans le *Linux Magazine N° 49* : http://diamond.izibookstore.com/produit?internal_reference=lmhs49
- Documentation de Beautiful Soup : http://www.crummy.com/software/BeautifulSoup/bs4/doc/
- Documentation de Python : http://docs.python.org/
- Algorithme *Longest common subsequence problem*: http://en.wikipedia.org/wiki/Longest_common_subsequence_problem
- Interview de Jean Gabes : http://blog.nicolargo.com/2011/08/interview-de-jean-gabes-le-createur-de-shinken.html
- Site web de Jean Gabes: http://www.gabes.fr/jean/

A. Annexes

A.1. Installation de CentOS 6.2

Après avoir booté sur le DVD1 sélectionner **Install or ugrade an existing system**.

Ensuite sélectionner **Skip** puis **OK**.

Sélectionner **French** puis **[ENTER]**

Sélectionner **fr_CH-latin1** puis **[ENTER]**.

Si un avertissement apparait sélectionner **Réinitialiser**.

Sélectionner le fuseau horaire **Europe/Zurich** puis cliquer sur **Valider**.

Entrer un mot de passe root (par exemple **rootroot**). Puis **Valider.**

Attention: le mot de passe rootroot ne doit pas être utilisé dans un environnement en production.

Si un avertissement apparait sélectionner **Utiliser malgré tout**.

Sélectionner **Utiliser la totalité du disque** puis **[ENTER]** puis **Valider**.

Cliquer sur **Write changes to disk**.

Une fois l'installation terminée cliquer sur **Redémarrer**.

Pour se connecter à la machine il faut utiliser le login **root** et le password défini précédemment (**rootroot**).

A.2. Configuration réseau (sur CentOS)

Activation de la carte réseau: `ifconfig eth0 up`

Configuration d'un serveur DNS: `echo "nameserver 8.8.8.8" > /etc/resolv.conf`

Configurer les paramètres de la carte réseau `vi /etc/sysconfig/network-scripts/ifcfg-eth0`

Modifier **ONBOOT="no"** en **ONBOOT="yes"**.

Ajouter les lignes suivantes:

BOOTPROTO=none
NETWORK=[adresse du réseau]
NETMASK=[masque de sous réseau]
IPADDR=[adresse IP de Shinken]

Configurer la route par défaut: `vi /etc/sysconfig/network`

Modifier **HOSTNAME=localhost.localdomain** par **HOSTNAME=shinken**

Ajouter **GATEWAY=**[IP de la passerelle par défaut]

Redémarrer le service network: `service network restart`

Les paramètres peuvent être vérifiés grâce aux commandes `ifconfig`, `route`, `ping`, …

A.3. Installation de Shinken 1.0

La manière la plus simple d'installer Shinken est de le télécharger et d'utiliser le script d'installation.

Si `wget` n'est pas disponible exécuter: `yum install wget`

Télécharger Shinken: `wget http://shinken-monitoring.org/pub/shinken-1.0.tar.gz`

Décompresser le contenu de l'archive: `tar -yxf shinken-1.0.tar.gz`

Installation de *redhat-lsb* (dépendance du script d'installation): `yum install redhat-lsb`

Installer Shinken: `./shinken-1.0/install -i` (~10mn)

Installer les plugins:

```
./shinken-1.0/install -p nagios-plugins
./shinken-1.0/install -p manubulon
```

Démarrer Shinken: `/etc/init.d/shinken start`

Ensuite il faut soit ajouter une règle dans le pare-feu, soit désactiver le pare-feu.

Désactiver le pare-feu: `service iptables stop`

Pour que le pare-feu ne se lance pas au démarrage il faut exécuter:

```
chkconfig --level 3 iptables off
```

Accéder à l'interface web de Shinken: ***http://[ip configurée précédemment]:7767/***

Le nom d'utilisateur est **admin** et le mot de passe est **admin**.

A.4. Installer des plugins

Il existe plusieurs méthodes pour installer des plugins. Celle qui fonctionne toujours consiste à copier les fichiers du plugin dans le répertoire ***libexec*** de Shinken (*/usr/local/shinken/libexec*).

Certains packs de plugins peuvent être installés avec le script d'installation de Shinken. La commande est la suivante: `./install -p [nom du plugin]`
Pour installer les plugins Nagios il faut exécuter: `./install -p nagios-plugins`
Pour installer les plugins SNMP il faut exécuter: `./install -p manubulon`
La liste des packs disponibles peut être obtenue en exécutant: `./install --help`

Les plugins Nagios peuvent être installés avec le système de paquets de CentOS:

```
yum install nagios-plugins-all
```

Les plugins sont installés dans le répertoire */usr/lib64/nagios* (ou */usr/lib/nagios* pour les OS 32 bits)

Il faut copier ces plugins dans le répertoire Shinken grâce à la commande

```
cp  /usr/lib64/nagios/plugins/*  /usr/local/shinken/libexec/
```

Si le plugin n'est pas déjà présent dans la configuration, il faut l'ajouter dans *etc/commands.cfg*.

A.5. Installation et configuration du système d'envoi d'emails

Il faut tout d'abord installer les paquets mailx et postfix: `yum install mailx postfix`

Editer le fichier de configuration de postfix `vi /etc/postfix/main.cf` et ajouter les lignes suivantes:

```
myorigin = tddeig.ch
myhostname = Shinken
mydestination = Shinken, localhost.localdomain, , localhost
relayhost = [smtpauth.bluewin.ch]:587
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = loopback-only
inet_protocols = all
smtpd_banner = $myhostname ESMTP $mail_name
append_dot_mydomain = no
readme_directory = no
biff = no
smtpd_use_tls = yes
smtp_enforce_tls = no
smtp_sasl_auth_enable = yes
smtp_sasl_password_maps = hash:/etc/postfix/sasl/sasl_passwd
smtp_sasl_security_options = noanonymous
smtp_always_send_ehlo = yes
```

Le compte du serveur SMTP doit être déclaré dans le fichier */etc/postfix/sasl/sasl_passwd* (créer le répertoire *sasl* s'il n'existe pas).
La syntaxe est la suivante: `adresse_du_serveur utilisateur:mot_de_passe`

Pour prendre en compte la configuration du compte pour le serveur SMTP exécuter:
`postmap /etc/postfix/sasl/sasl_passwd`

Redémarrer le serveur mail: `/etc/init.d/postfix restart`

Envoyer un email de test:
`echo "Shinken" | mail adresse@email.com -s "Message de test"`

A.6. Installer un serveur SSH sur Ubuntu

Télécharger et installer le serveur SSH `apt-get install openssh-server`

Les paramètres par défaut permettent de se connecter depuis une machine distante pour autant que l'on ait un compte sur la machine cible. Il est impossible de se connecter avec le compte root sur une machine Ubuntu.

Redémarrer le serveur SSH : `service ssh restart`

Si la connexion depuis une machine distante ne fonctionne pas, essayer de désactiver le pare-feu :
`service ufw stop`

Pour plus d'informations se référer à la documentation d'Ubuntu : http://doc.ubuntu-fr.org/ssh

A.7. Générer une paire de clefs pour SSH

Pour se connecter à un serveur SSH via le mécanisme de clefs publique & partagée il faut tout d'abord disposer d'un compte sur le serveur SSH.

Sur la machine source (celle qui veut se connecter à un serveur SSH) générer les clefs à l'aide de la commande `ssh-keygen` (accepter les paramètres par défaut).

Une fois la paire de clefs créée il faut placer la clef publique sur le serveur SSH. Sur la machine source lancer la commande `ssh-copy-id utilisateur@adresse-ip`

Il faut maintenant approuver le serveur SSH sur la machine source. La méthode la plus simple est d'essayer de se connecter au serveur SSH grâce à la commande `ssh utilisateur@adresse-ip`

L'empreinte de la clef du serveur SSH va s'afficher et un message demandera de la confirmer. Il faut vérifier que l'empreinte correspond à celle du serveur SSH et, dans le cas échéant, confirmer la clef.

A.8. Plugin check_ordonnanceur

Un plugin a été créé pour déterminer si l'ordonnanceur de Shinken planifie le prochain test au moment où le test est lancé ou après que le test se soit terminé.

Ce plugin très simple attend un certain délai avant de retourner le résultat.

Il prend en entrée deux arguments:

- Le temps d'attente en secondes.
- Le code de retour que le plugin va retourner à la fin de l'attente.

Le code ci-dessous a été déposé dans le répertoire *libexec*.

```
#!/bin/sh
echo "Test scheduler"
sleep $1
exit $2
```

Code 2 - cheeck_ordonnanceur.sh

La commande ci-dessous a été définie dans le fichier *etc/commands.cfg*. Elle indique au plugin d'attendre 60 secondes avant de retourner 1 (Warning).

```
define command{
        command_name    check_ordonnanceur
        command_line    $USER1$/check_ordonnanceur.sh 60 1
}
```

Figure 47 - Commande check_ordonnanceur

Pour pouvoir tester cette commande, un service faisant la liaison entre la commande et une machine (host) a été ajouté.

```
define service{
        use                     generic-service
        host_name               MachineA
        service_description     Test ordonanceur
        check_command           check_ordonnanceur
        normal_check_interval   1
        retry_check_interval    1
        initial_state           u
}
```

Figure 48 - Service check_ordonnanceur

Après avoir redémarré Shinken il est possible de tester l'ordonnanceur de Shinken. Avec la configuration ci-dessus l'intervalle entre deux tests devrait être de 60 secondes si Shinken replanifie le test suivant après avoir lancé le test ou de 120 (60+60) secondes si le test suivant est planifié après que le test soit terminé.

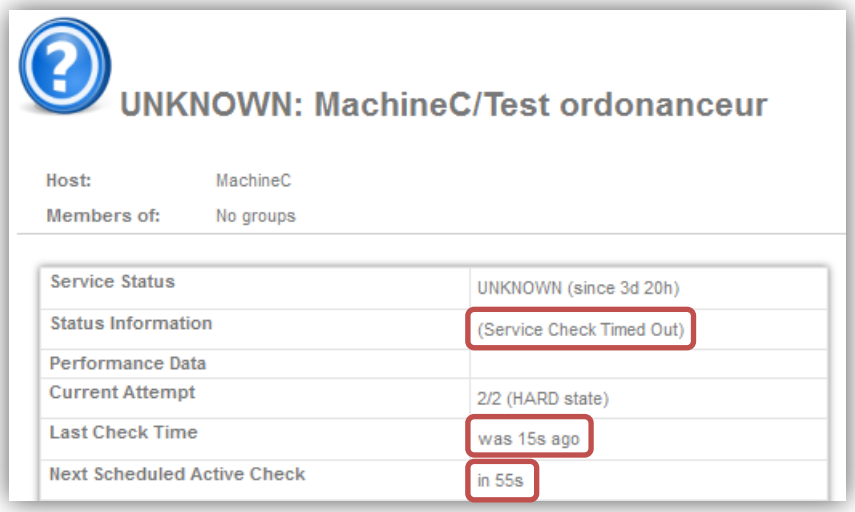


Figure 49 - Capture d'écran du test de l'ordonnanceur

Comme on peut voir sur la capture d'écran ci-dessus la réalité est légèrement différente. L'intervalle entre deux tests est d'environ 70 secondes. Ceci est dû au paramètre *service_timeout* (présent dans le fichier *etc/nagios.cfg*). Ce paramètre définit le temps d'exécution maximal d'un test et est configuré par défaut à 10 secondes. Si ce temps est dépassé, Shinken tue le processus du test et renvoie le message *Service Check Timed Out*.

Bien que d'autres méthodes d'ordonnancement soient possibles, celle utilisée dans Shinken peut être considérée comme logique car le temps avant le prochain test peut changer en fonction du résultat du test (par exemple avec la propriété *retry_check_interval*).

A.9. Installer Beautiful Soup

La méthode la plus simple et portable pour installer Beautiful Soup est de télécharger les sources de la dernière release puis de l'installer avec le script *setup.py* prévu à cet effet.

Télécharger les sources :
`wget http://www.crummy.com/software/BeautifulSoup/bs4/download/4.0/beautifulsoup4-4.1.0.tar.gz`

Extraire les sources: `tar xzvf beautifulsoup4-4.1.0.tar.gz`

Entrer dans le répertoire qui vient d'être créé : `cd beautifulsoup4-4.1.0`

Lancer l'installation : `python setup.py install`

B. Tables des illustrations

B.1. Figures

B.2. Tableaux