# Objectifs de l'activité :

Théoriquement : Comprendre la supervision d'un hôte par Shinken.

Pratiquement :

* + Utiliser l'interface de visualisation de Shinken
  + Comprendre les fichiers de configuration principaux
  + Superviser un hôte
  + Installer et configurer SNMP sur un hôte
  + Tester des commandes
  + Installer un serveur de mail
  + Paramétrer et envoyer des notifications
  + Superviser un service

Matériel nécessaire**:**

* + une machine virtuelle avec Linux Debian 7.1 64 bits graphique installé pour avoir le navigateur, gedit et Shinken. Il est préférable de cloner une MV existante.

Résultat attendu**:**

* + La supervision de l'hôte local par Shinken avec envoi de mail en cas de problème.
  + Un compte rendu démontrant que la supervision se déroule avec succès et que les notifications sont envoyées devra être rédigé, les copies d'écran et les commentaires seront nécessaires.
  + La dernière partie comporte des questions dont les réponses devront figurer sur le compte rendu.

## Organisation

* + Travail individuel.

## Travail à faire

* + Chaque partie comporte des travaux à faire.

# Partie 1 : Les fichiers de configuration principaux

Travail à faire : ouvrir les fichiers mentionnés et les étudier.

Le fichier où se trouve les variables concernant les répertoires de travail de shinken  **dans /etc/default** et s'appelle **shinken.**

On y trouve notamment:

ETC=/etc/shinken

VAR=/var/lib/shinken/

BIN=/usr/bin

RUN=/var/run/shinken

LOG=/var/log/shinken

Tous les scripts de shinken font référence à ces variables sous la forme shell $ETC par exemple.

C'est la convention que nous utiliserons dans la suite de ce document.

Le fichier **$ETC/shinken.cfg** liste tous les fichiers de configuration qui seront chargés au démarrage.

Le mot clé **cfg\_dir** indique l'ensemble des fichiers de configuration principaux et des sous-répertoires (contenant aussi des fichiers de configuration) qui seront pris en compte au démarrage de **Shinken**.

# Configuration files with common objects like commands, timeperiods,

# or templates that are used by the host/service/contacts

cfg\_dir=commands

Adressage relatif à l'emplacement du fichier **shinken.cfg**

cfg\_dir=timeperiods

cfg\_dir=escalations

cfg\_dir=dependencies

# Now templates of hosts, services and contacts

Tous les fichiers et répertoires placés dans ces répertoires seront pris en compte

cfg\_dir=templates

# notification things

cfg\_dir=notificationways

# Now groups

cfg\_dir=servicegroups

cfg\_dir=hostgroups

cfg\_dir=contactgroups

# And now real hosts, services, packs and discovered hosts

# They are directory, and we will load all .cfg file into them, and

# their sub-directory

cfg\_dir=hosts

cfg\_dir=services

cfg\_dir=contacts

cfg\_dir=packs

cfg\_dir=modules

cfg\_dir=arbiters

cfg\_dir=schedulers

cfg\_dir=pollers

cfg\_dir=reactionners

cfg\_dir=brokers

cfg\_dir=receivers

cfg\_dir=realms

# You will find global MACROS into this file

#resource\_file=resource.cfg

cfg\_dir=resource.d

Pour être pris en compte, un fichier de configuration doit appartenir à un élément pointé par ce fichier.

Une erreur dans un des fichiers de configuration provoquera un arrêt du démarrage.

Il faut utiliser les options **check** pour vérifier l'ensemble des fichiers de configuration ou **debug** pour tracer le démarrage ou bien encore consulter en cas d’erreur le fichier /tmp/shinken\_checkconfig\_result ou bien les différents logs comme par exemple le log de l’arbiter **$LOG/arbiterd.log.**

root@debianWheezy:$ETC# service shinken check

Doing config check

. ok

root@debianWheezy:$ETC# service shinken -d start

…

. ok

Starting arbiter:

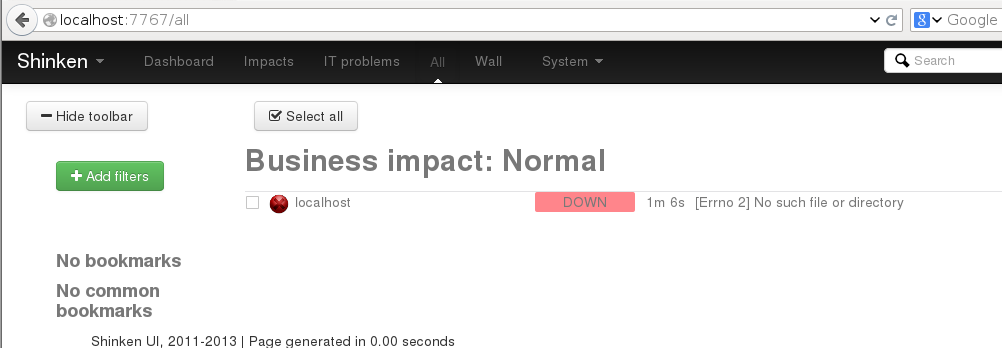
. ok

root@debianWheezy:$ETC# gedit/var/log/shinken/arbiterd.log

# Partie 2 : Pourquoi et comment une machine est-elle supervisée ?

Travail à faire **:** ouvrir les différents fichiers, les étudier, comprendre leurs relations et effectuer les actions permettant de tester « localhost ».

**Webui** (l'interface utilisateur Web proposée par défaut par Shinken) nous montre que la supervision ne fonctionne pas ☹. Mais que nous dit-elle exactement ?



**L**es différents états gérés par **Shinken** sont :

* Pour un hôte
  + Un hôte est une machine accessible par son adresse IP pouvant prendre 4 états (**UP, DOWN,UNREACHABLE, PENDING**)
    - **Up**🡺l'hôte répond,
    - **Down**🡺l'hôte ne répond pas,
    - **Unreachable**🡺l’hôte est injoignable car il se trouve derrière un autre hôte qui ne répond pas,
    - **Pending**🡺l’hôte n’est pas encore testé (au démarrage généralement).
* Pour un service
  + Un service est un élément supervisé sur un hôte (*qui doit donc être* ***UP***) pouvant prendre 5 états (**OK, WARNING, CRITICAL, UNKNOWN, PENDING**)
    - **Warning**🡺 des problèmes non bloquants,
    - **Critical**🡺 des problèmes bloquants,
    - **Unknown**🡺 état non testable car la commande (le plugin) a un problème,
    - **Pending**🡺 non encore testé (généralement au démarrage).

L'hôte ne répond pas dans notre cas, mais pourquoi devrait-il répondre et à quoi ne répond-il pas ?

## Les hôtes

**Pourquoi Shinken supervise la machine localhost ?**

Parce que l'installation a créé un fichier **localhost.cfg** dans le répertoire **$ETC/hosts.**

Tous les hôtes présents (soit manuellement, soit dynamiquement, comme on verra dans d'autres activités) dans ce répertoire seront donc supervisés.

Si on ouvre le fichier **localhost.cfg** on y lit :

define host{

use generic-host

contact\_groups admins

host\_name localhost

address localhost

}

Le mot clé **"define"** permet de définir un **objet** gérable par Shinken, ici un hôte (**host**) identifié par son nom (**localhost**). Cet identifiant est important car il peut être utilisé dans d'autres fichiers de configuration.

Son adresse est **localhost**, elle sera utilisée comme on le verra, par la variable **$HOSTADDRESS$** (on utilise un nom ou une adresse IP, dans notre cas **localhost** ou **127.0.0.1,** mais si on utilise un nom il faut une méthode de résolution de noms, dans ce cas le fichier **/etc/hosts**), et le groupe à contacter en cas de problème (**contact\_groups**), *souvenez vous de cela tout à l'heure.*

## Les modèles (templates)

Il y a peu de choses dans le fichier précédent car la définition utilise un **modèle** (**template**) par l'intermédiaire de la directive **use**.

Le modèle (**template**) utilisé est **generic-host**.

La plupart des modèles sont décrits (définis) dans le sous-répertoire **$ETC/templates/** ou comme on le verra plus loin dans **$ETC/packs/**

Dans ce répertoire on va trouver le fichier **generic-host.cfg.**

On va regarder plus précisément ce fichier :

# Generic host definition template - This is NOT a real host, just a template!

# Most hosts should inherit from this one

define host{

name generic-host

# Checking part

check\_command check\_host\_alive

max\_check\_attempts 2

check\_interval 5

# Check every time

active\_checks\_enabled 1

check\_period 24x7

# Notification part

# One notification each day (1440 = 60min\* 24h)

# every time, and for all 'errors'

# notify the admins contactgroups by default

contact\_groups admins,users

notification\_interval 1440

notification\_period 24x7

notification\_options d,u,r,f

notifications\_enabled 1

# Advanced option. Look at the wiki for more informations

event\_handler\_enabled 0

flap\_detection\_enabled 1

process\_perf\_data 1

# Maintenance period

#maintenance\_period workhours

# Dispatching

#poller\_tag DMZ

#realm All

# For the WebUI

#icon\_set server ; can be database, disk, network\_service, server

# This said that it's a template

register 0

}

Ce fichier définit un modèle identifié par le nom **generic-host**, la directive **register** nous indique qu'il s'agit d'un modèle grâce à la valeur 0. Un modèle n'est pas instancié, il doit être appelé par un objet.

Ce modèle appelle la commande **check\_host\_alive**.

## Les commandes

La commande **check\_host\_alive** est définie dans **$ETC/commands.cfg** /check\_host\_alive.cfg**.**

define command {

command\_namecheck\_host\_alive

command\_line $NAGIOSPLUGINSDIR$/check\_ping -H $HOSTADDRESS$ -w 1000,100% -c 3000,100% -p 1

}

Comme on peut voir la commande **check\_host\_alive** est une référence logique vers la commande réelle **check\_ping.**

Cette commande se situe dans le répertoire désignée par la variable $NAGIOSPLUGINSDIR$, cette variable est définie dans le fichier $ETC/resource.d/path.cfg .

# Nagios legacy macros

$USER1$=$NAGIOSPLUGINSDIR$

$NAGIOSPLUGINSDIR$=/usr/lib/nagios/plugins

#-- Location of the plugins for Shinken

$PLUGINSDIR$=/var/lib/shinken/libexec

Cette commande est une commande faisant partie des plugins nagios.

Mais si on regarde dans le répertoire **/usr/lib** on ne trouve pas de sous répertoire **nagios**. Il faut donc installer le plugin.

root@debianWheezy:~# apt-get install nagios-plugins

Lecture des listes de paquets... Fait

Construction de l'arbre des dépendances

…

Creating config file /etc/nagios-plugins/config/snmp.cfg with new version

Paramétrage de nagios-plugins (1.4.16-1) ...

Paramétrage de nagios-plugins-contrib (4.20120702) ...

Traitement des actions différées (« triggers ») pour « menu »

Las plugins installés sont nombreux :

root@debianWheezy:~# ls /usr/lib/nagios/plugins/

check\_apt check\_imap\_receive\_epncheck\_rbl

check\_backuppc check\_ipmi\_sensor check\_real

check\_breeze check\_ircd check\_rpc

check\_by\_ssh check\_jabber check\_rta\_multi

check\_cert\_expire check\_ldap check\_running\_kernel

check\_clamd check\_ldaps check\_sensors

check\_cluster check\_libs check\_simap

check\_dhcp check\_lm\_sensors check\_smtp

check\_dig check\_load check\_smtp\_send

check\_disk check\_log check\_smtp\_send\_epn

check\_disk\_smb check\_mailq check\_snmp

check\_dns check\_memcached check\_snmp\_environment

check\_dnssec\_delegation check\_mrtg check\_soas

check\_dummy check\_mrtgtraf check\_spop

check\_email\_delivery check\_multipath check\_ssh

check\_email\_delivery\_epn check\_mysql check\_ssl\_cert

check\_entropy check\_mysql\_health check\_ssmtp

check\_file\_age check\_mysql\_query check\_statusfile

check\_flexlm check\_nagios check\_swap

check\_fping check\_nntp check\_tcp

check\_ftp check\_nntps check\_time

check\_game check\_nt check\_udp

check\_haproxy check\_ntp check\_ups

check\_host check\_ntp\_peer check\_users

check\_hpasm check\_ntp\_time check\_wave

check\_hpjd check\_nwstat check\_webinject

check\_http check\_oracle check\_whois

check\_httpd\_status check\_overcr check\_zone\_auth

check\_icmp check\_packages check\_zone\_rrsig\_expiration

check\_ide\_smart check\_pgsql imap\_ssl\_cert

check\_ifoperstatus check\_ping imap\_ssl\_cert\_epn

check\_ifstatus check\_pop negate

check\_imap check\_printer urlize

check\_imap\_quota check\_procs utils.pm

check\_imap\_quota\_epn check\_radius utils.sh

check\_imap\_receive check\_raid

On y trouve notamment la commande **check\_ping**.

Cette commande est documentée ici : <https://www.monitoring-plugins.org/doc/man/check_ping.html>

L'option –w indique un "warning", -c un évènement "critical", rta signifie "round trip average travel" et –p le nombre de paquets "icmp echo" envoyés.

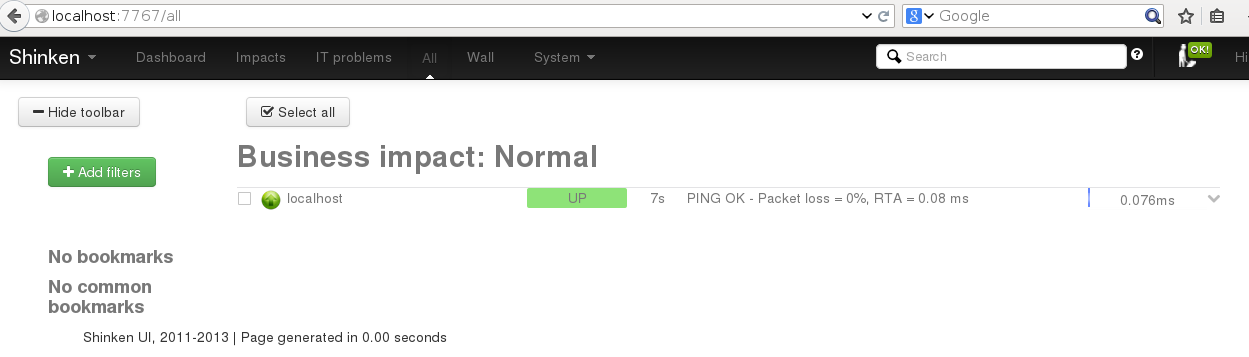
On peut d'ailleurs exécuter cette commande manuellement pour la tester.

root@debianWheezy:~# cd /usr/lib/nagios/plugins/

root@debianWheezy:/usr/lib/nagios/plugins# ./check\_ping -H localhost -w 1000,100% -c 3000,100% -p 1

PING OK - Paquets perdus = 0%, RTA = 0.29 ms|rta=0.291000ms;1000.000000;3000.000000;0.000000 pl=0%;100;100;0

Cela fonctionne. Que dit maintenant webui ?



La machine est up !

Remarque : **$HOSTADDRESS$** a été remplacé par l'adresse trouvée dans le fichier **$ETC/hosts/localhost.cfg,** directive **address.**

address localhost

# Partie 3 : Enrichir la supervision avec les packs

Travail à faire **:** ouvrir les différents fichiers, les étudier, comprendre leurs relations et effectuer les actions permettant de superviser localhost en tant que machine Linux.

Pour l'instant notre supervision se contente de tester la présence de notre hôte. On va lui demander un peu plus.

Regardons ce que propose la communauté pour superviser une machine Linux.

shinken@debianWheezy:~$ shinken search linux

glances (david-guenault) [pack,system,linux,glances] : Standard check through checkglances.py and glances server

linux-snmp (naparuba) [pack,linux,snmp] : Linux checks based on SNMP

linux-ssh (naparuba) [pack,linux,ssh] : Linux checks based on SSH without any script on distant server

pack-glances (david-guenault) [pack,system,linux,glances] : Standard check through checkglances.py and glances server

raspberrypi (frescha) [pack,linux,raspberrypi,server,os] : Standard checks

varnish-ssh (kokosny) [pack,linux,varnish,ssh] : varnish checks based on ssh

On a l'embarras du choix.

On va choisir dans un premier temps **linux-snmp**.

Mais avant d’installer quoi que ce soit, on va regarder le contenu du répertoire **$ETC/packs**

root@debianWheezy:/usr/lib/nagios/plugins# ls /etc/shinken/packs/

readme.cfg

Et lire le fichier **readme.cfg :**

#In this place you will find all your packs downloaded from shinken.iowebsite.

#

#you can freely adapt them to your own need

Ce répertoire est donc vide et est prévu pour recevoir les packs.

On installe le pack **linux-snmp** :

shinken@debianWheezy:~$ shinken install linux-snmp

Grabbing :linux-snmp

OK linux-snmp

Ce qui a pour conséquence de créer un sous répertoire dans le répertoire :

root@debianWheezy:~#ls /etc/shinken/packs/

linux-snmp readme.cfg

root@debianWheezy:~#ls /etc/shinken/packs/linux-snmp/

commands.cfg discovery.cfg linux-snmp.pack services templates.cfg

Qu'est ce qu'un pack ?

Un pack est un ensemble de fichiers de configuration et/ou de commandes développés par la communauté des utilisateurs et permettant la supervision des hôtes correspondant à des modèles génériques (Linux ici par exemple).

Si on regarde le fichier **commands.cfg** on voit un certain nombre de commandes qui seront appliquées sur les hôtes de type Linux.

# -----------------------------------------------------------------

#

# Linux standard check

#

# -----------------------------------------------------------------

#

define command {

command\_name check\_linux\_load

command\_line $PLUGINSDIR$/check\_snmp\_load.pl -H $HOSTADDRESS$ -C $\_HOSTSNMPCOMMUNITY$ -f -w $\_HOSTLOAD\_WARN$ -c $\_HOSTLOAD\_CRIT$ -T netsl -o $\_HOSTSNMP\_MSG\_MAX\_SIZE$

}

define command {

command\_name check\_linux\_disks

command\_line $PLUGINSDIR$/check\_snmp\_storage.pl -H $HOSTADDRESS$ -C $\_HOSTSNMPCOMMUNITY$ -m $\_HOSTSTORAGE\_PATH$ -f -w $\_HOSTSTORAGE\_WARN$ -c $\_HOSTSTORAGE\_CRIT$ -S0,1 -o $\_HOSTSNMP\_MSG\_MAX\_SIZE$

}

define command {

command\_name check\_linux\_cpu

command\_line $PLUGINSDIR$/check\_snmp\_load.pl -H $HOSTADDRESS$ -C $\_HOSTSNMPCOMMUNITY$ -f -w $\_HOSTCPU\_WARN$ -c $\_HOSTCPU\_CRIT$ -o $\_HOSTSNMP\_MSG\_MAX\_SIZE$

}

# Added -g flag since all linux system used are 64bits.

define command {

command\_name check\_linux\_network\_usage

command\_line $PLUGINSDIR$/check\_netint.pl -H $HOSTADDRESS$ -C $\_HOSTSNMPCOMMUNITY$ -n "$\_HOSTNET\_IFACES$" -g -2c -f -e -w $\_HOSTNET\_WARN$ -c $\_HOSTNET\_CRIT$ -q -k -y -M -B -m -P "$SERVICEPERFDATA$" -T "$LASTSERVICECHECK$" -o $\_HOSTSNMP\_MSG\_MAX\_SIZE$

}

define command {

command\_name check\_linux\_memory

command\_line $PLUGINSDIR$/check\_snmp\_mem.pl -w $\_HOSTMEMORY\_WARN$ -c $\_HOSTMEMORY\_CRIT$ -- -v 2c -c $\_HOSTSNMPCOMMUNITY$ $HOSTADDRESS$

}

define command {

command\_name check\_linux\_logfiles

command\_line $PLUGINSDIR$/check\_logfiles -f $\_HOSTCHKLOG\_CONF$

}

define command {

command\_name check\_linux\_time

command\_line $NAGIOSPLUGINSDIR$/check\_ntp\_time -H $HOSTADDRESS$ -w $\_HOSTNTP\_WARN$ -c $\_HOSTNTP\_CRIT$

}

Ces commandes sont situées dans le répertoire **$PLUGINSDIR$** ou **$NAGIOSPLUGINSDIR$,** soit comme défini dans le fichier **$ETC/resource.d/path.cfg**

# Nagios legacy macros

$USER1$=$NAGIOSPLUGINSDIR$

$NAGIOSPLUGINSDIR$=/usr/lib/nagios/plugins

#-- Location of the plugins for Shinken

$PLUGINSDIR$=/var/lib/shinken/libexec

Le modèle linux hérite lui-même du modèle **generic\_host**, comme on le constate en ouvrant le fichier templates.cfg

# The LINUX template.

define host {

name linux-snmp

use generic-host

check\_command check\_ping

register 0

# We will show the linux custom view

custom\_views +linux

\_SNMPCOMMUNITY $SNMPCOMMUNITYREAD$

\_SNMP\_MSG\_MAX\_SIZE 65535

\_LOAD\_WARN 2,2,2

\_LOAD\_CRIT 3,3,3

\_STORAGE\_WARN 90

\_STORAGE\_CRIT 95

\_CPU\_WARN 80

\_CPU\_CRIT 90

\_MEMORY\_WARN 80

\_MEMORY\_CRIT 95

\_NTP\_WARN 0.128

\_NTP\_CRIT 1

\_NET\_IFACES eth\d+|em\d+

\_NET\_WARN 90,90,0,0,0,0

\_NET\_CRIT 0,0,0,0,0,0

\_CHKLOG\_CONF $PLUGINSDIR$/logFiles\_linux.conf

\_STORAGE\_PATH /

}

define service {

name linux-service

use generic-service

register 0

aggregation system

}

Il utilise donc des commandes définies au niveau le plus haut mais il définit aussi ses propres commandes spécifiques dans son fichier **commands.cfg.**

Mais pourquoi toutes les commandes du fichier commands.cfg sont-elles appliquées au modèle Linux ?

## Les services

Une commande n'est pas appelée directement par l'hôte (à part le ping), les commandes sont appelées via les services car ce sont des services qui tournent sur les OS.

Le modèle Linux définit un service **linux-service** qui fait appel lui-même à un service générique **generic-service.**

Si on regarde maintenant la définition d'un service spécifique, **cpu.fcg** par exemple :

define service {

service\_descriptionCpu

use 20min\_long,linux-service

register 0

host\_name linux-snmp

check\_command check\_linux\_cpu

\_DETAILLEDESC Detect abnormal CPU usage

\_IMPACT Slow down applications hosted by the system

\_FIXACTIONS If recurrent situation then make performance audit

}

On constate 2 choses, l'appel au modèle **linux-service** et la relation établie entre le service et le modèle d'hôte **linux-snmp** par la directive **host\_name**.

Il reste maintenant à établir une relation entre notre hôte réel **localhost** et notre modèle **linux-snmp**.

C'est dans le fichier $ETC/hosts/localhost.cfg que cette relation est établie :

define host {

use linux-snmp,generic-host

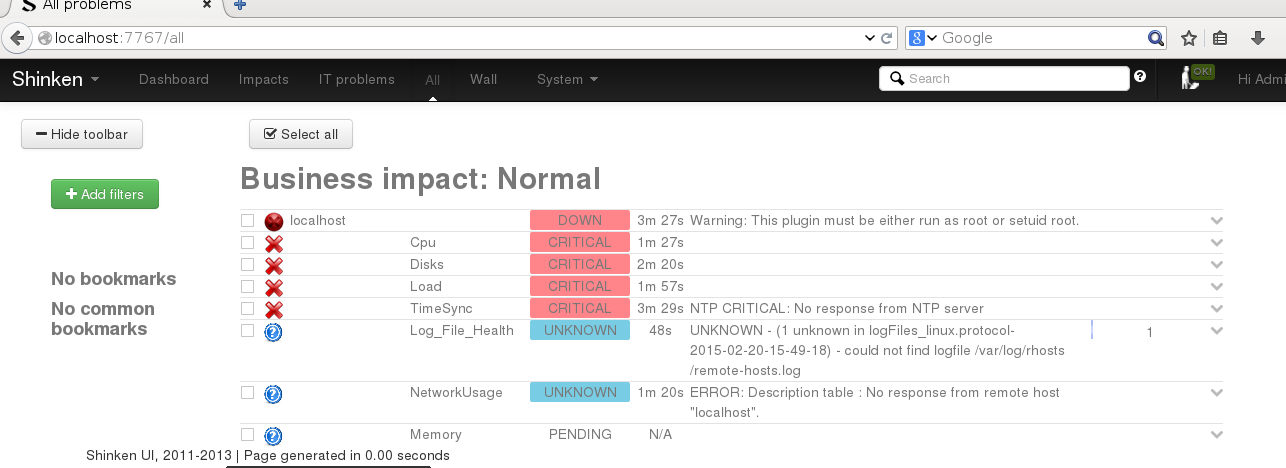
contact\_groups admins

host\_name localhost

address localhost

}

On redémarre shinken et on regarde le résultat sur webui.



On a bien la prise en compte de nouveaux indicateurs mais à priori ce n'est pas au point.

# Partie 4 : Mise au point de la supervision d'un hôte

Travail à faire : procéder aux différentes actions proposées pour effectuer correctement la supervision de **localhost** via **snmp**.

## Pourquoi notre ping ne marche plus ?

Première déconvenue, le ping ne fonctionne pas. Si on regarde de plus près, on s'aperçoit que la commande appelée par le modèle **linux-snmp** n'est pas **check\_ping** comme tout à l'heure mais **check\_icmp**.

Cette commande est définie dans $ETC/commands/check\_icmp.cfg.

## Check ping command

## Use ping to check connection statistics for a remote host.

# check\_ping -H <host\_address> -w <wrta>,<wpl>% -c <crta>,<cpl>% [-p packets]

# [-t timeout] [-4|-6]

define command {

command\_namecheck\_ping

command\_line $NAGIOSPLUGINSDIR$/check\_icmp -H $HOSTADDRESS$ -w 3000,100% -c 5000,100% -p 10

}

Contrairement à la commande **check\_host\_alive.cfg** utilisée précédemment, elle fait appel à la commande **check\_icmp** et non à la commande **check\_ping**, or si nous testons ces commandes on constate une erreur de droit avec l'utilisateur **shinken**.

shinken@debianWheezy:~$ /usr/lib/nagios/plugins/check\_ping -H localhost -w 1000,100% -c 3000,100% -p 1

PING OK - Paquetsperdus = 0%, RTA = 0.05 ms|rta=0.047000ms;1000.000000;3000.000000;0.000000 pl=0%;100;100;0

shinken@debianWheezy:~$ /usr/lib/nagios/plugins/check\_icmp -H localhost -w 1000,100% -c 3000,100% -p 1

Warning: This plugin must be either run as root or setuid root.

To run as root, you can use a tool like sudo.

To set the setuid permissions, use the command:

Chmod u+s yourpluginfile

check\_icmp: Failed to obtain ICMP socket: Opération non premise

On va modifier le **setuid** (voir http://fr.wikipedia.org/wiki/Setuid).

root@debianWheezy:/etc/shinken/commands# chmodu+s /usr/lib/nagios/plugins/check\_icmp

root@debianWheezy:/etc/shinken/commands# ls -l /usr/lib/nagios/plugins/check\_icmp

-rwsr-xr-x 1 root root 55312 juin 28 2012 /usr/lib/nagios/plugins/check\_icmp

Puis tester à nouveau la commande avec l'utilisateur shinken

shinken@debianWheezy:~$ /usr/lib/nagios/plugins/check\_icmp -H localhost -w 1000,100% -c 3000,100% -p 1

OK - localhost: rta 0,176ms, lost 0%|rta=0,176ms;1000,000;3000,000;0; pl=0%;100;100;; rtmax=0,176ms;;;; rtmin=0,176ms;;;;

Cette fois ça marche !

Voyons sur webui



## SNMP

On a choisit le modèle **linux-snmp**, il faut donc vérifier la configuration de **snmp** sur **localhost** et, le cas échéant, installer et configurer le démon **snmpd.**

Il faut installer le paquet **snmpd** si ce n'est pas déjà fait :

apt-get install snmpd

🡺 Attention il faut que dans le fichier **snmpd.conf** on écoute bien sur **localhost** car la variable **$HOSTADDRESS**$ prendra la valeur **localhost** qui sera traduite par **127.0.0.1** via **/etc/hosts.**

Il faut donc vérifier ou modifier le fichier de configuration pour que le serveur SNMP écoute sur le port 161 sur son adresse IP.

(pour vérifier si le port est à l’écoute, 2 solutions soit **netstat –ulnp** , soit **nmap –sU localhost** si on a installé **nmap**).

Pour cela, éditez le fichier **/etc/snmp/snmpd.conf** et modifiez la valeur du paramètre **agentAddress** avec la valeur **localhost** de votre serveur.

Mais il faut aussi modifier dans ce fichier les conditions d'accès à la MIB.

L'accès en lecture est défini par la directive **rocommunity.**

Par défaut, cet accès est restreint à une seule vue **systemonly** via l'option -V.

La vue **systemonly** est définie ainsi **:**

viewsystemonly included .1.3.6.1.2.1.1

viewsystemonly included .1.3.6.1.2.1.25.1

L'accès par défaut en lecture, faisant référence à la vue « systemonly », est défini ainsi :

rocommunity public default -V systemonly

Cela restreint l'accès aux deux branches .1.3.6.1.2.1.1 (« system ») et .1.3.6.1.2.1.25.1 (« hrSystem ») et uniquement celles-ci.

Il est possible de modifier ces restrictions en créant une vue « all », par exemple, qui sera utilisée ensuite à la place de « systemonly » :

view all included .1

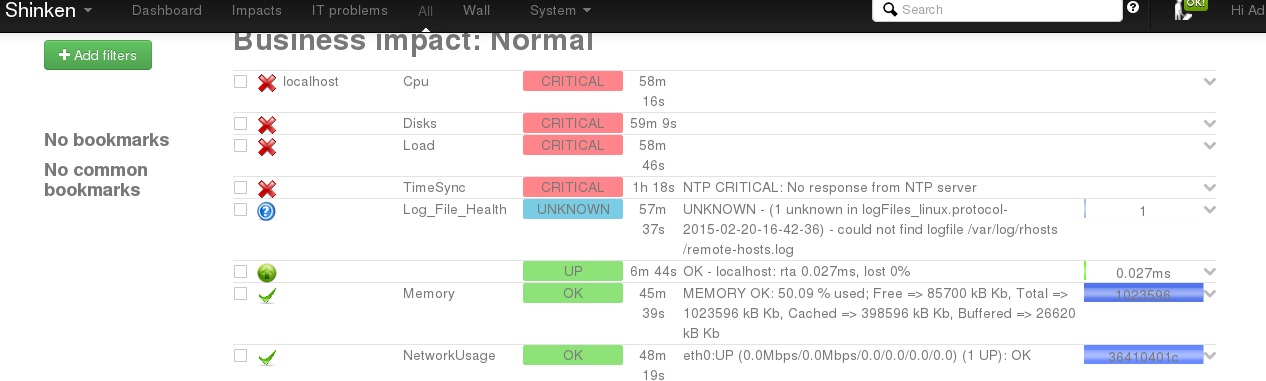
rocommunity public default -V all

Toute la branche « iso » sera ainsi accessible en lecture.

Il est nécessaire de redémarrer le service **snmpd** pour que le changement soit pris en compte.

service snmpd restart

Après cela on regarde le résultat dans webui :



Ça avance mais il y a encore un peu de travail de mise au point.

## Tester les commandes du modèle linux-snmp

On va continuer à tester manuellement les commandes présentes dans le modèle (**template**) **linux** qui ne fonctionne pas.

**Comment teste-t-on une commande déjà ?**

Il faut regarder quel est l'exécutable lancé par la commande, se positionner sur le répertoire et lancer la commande.

Observons la commande **check\_linux\_cpu :**

define command {

command\_namecheck\_linux\_cpu

command\_line $PLUGINSDIR$/check\_snmp\_load.pl -H $HOSTADDRESS$ -C $\_HOSTSNMPCOMMUNITY$ -f -w $\_HOSTCPU\_WARN$ -c $\_HOSTCPU\_CRIT$

}

Il faut remplacer toutes les variables par leurs valeurs.

**Où trouver les valeurs ?**

On va trouver les valeurs critiques **$\_HOSTCPU\_CRIT$** et warning **$\_HOSTCPU\_WARN$** par exemple du **cpu** dans le fichier **templates.cfg** du répertoire **linux** sous **$ETC/packs/linux-snmp/.**

# The LINUX template.

…

# We will show the linux custom view

custom\_views +linux

\_SNMPCOMMUNITY $SNMPCOMMUNITYREAD$

\_LOAD\_WARN 2,2,2

\_LOAD\_CRIT 3,3,3

Valeurs que l'on recherche

\_STORAGE\_WARN 90

\_STORAGE\_CRIT 95

\_CPU\_WARN 80

\_CPU\_CRIT 90

\_MEMORY\_WARN 90,20

\_MEMORY\_CRIT 95,50

\_NET\_WARN 90,90,0,0,0,0

\_NET\_CRIT 0,0,0,0,0,0

}

Ainsi on peut taper la commande :

shinken@debianWheezy:~$ /var/lib/shinken/libexec/check\_snmp\_load.pl -H localhost -C public -f -w 80 -c 90

Can't locate utils.pm in @INC (@INC contains: /var/lib/shinken/libexec /etc/perl /usr/local/lib/perl/5.14.2 /usr/local/share/perl/5.14.2 /usr/lib/perl5 /usr/share/perl5 /usr/lib/perl/5.14 /usr/share/perl/5.14 /usr/local/lib/site\_perl .) at /var/lib/shinken/libexec/check\_snmp\_load.pl line 22.

BEGIN failed--compilation aborted at /var/lib/shinken/libexec/check\_snmp\_load.pl line 22.

On s'aperçoit que la commande fait appel à un fichier **utils.pm** qu'elle ne trouve pas. Ce fichier appartient en fait aux plugins nagios et la commande le cherche dans le répertoire courant.

On va créer un lien symbolique du répertoire libexec vers le répertoire nagios/plugins (voir http://fr.wikipedia.org/wiki/Lien\_symbolique).

root@debianWheezy:/etc/shinken/commands# ln -s /usr/lib/nagios/plugins/utils.pm /var/lib/shinken/libexec/utils.pm

Testons à nouveau la commande après ça.

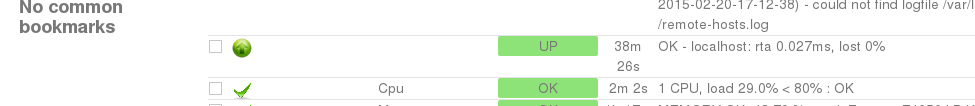
shinken@debianWheezy:~$ /var/lib/shinken/libexec/check\_snmp\_load.pl -H localhost -C public -f -w 80 -c 90

Argument "v6.0.1" isn't numeric in numeric lt (<) at /var/lib/shinken/libexec/check\_snmp\_load.pl line 685.

1 CPU, load 14.0% < 80% : OK | cpu\_prct\_used=14%;80;90

A priori ça marche (même si une erreur non bloquante est relevée dans le script).

Constat dans webui :



On peut procéder au même test pour les 3 autres commandes dans l'état UNKNOWN (ou bien aller vérifier directement sur **webUI l'effet de nos actions**)**.**

shinken@debianWheezy:~$ /var/lib/shinken/libexec/check\_snmp\_storage.pl -H localhost -C public -m / -f -w 90 -c 95 -S0,1 -o 65535

Argument "v6.0.1" isn't numeric in numeric lt (<) at /var/lib/shinken/libexec/check\_snmp\_storage.pl line 407.

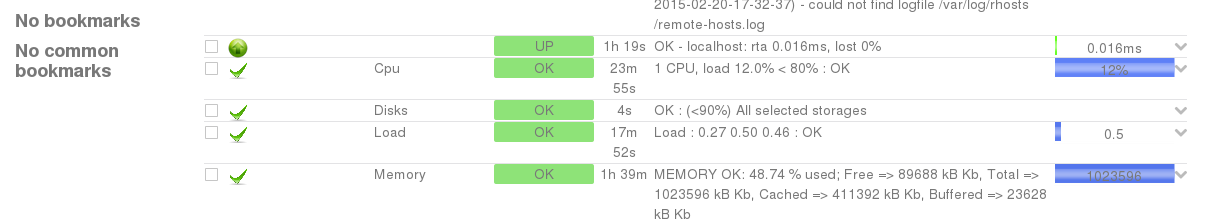
Argument "v6.0.1" isn't numeric in numeric lt (<) at /var/lib/shinken/libexec/check\_snmp\_storage.pl line 481.

CRITICAL : (>95%) /media/cdrom0: 100%used(3813MB/3813MB) | '/run/vmblock-fuse'=0MB;0;0;0;0 '/media/cdrom0'=3813MB;3432;3622;0;3813 '/sys/fs/fuse/connections'=0MB;0;0;0;0 '/'=6648MB;6915;7299;0;7683 '/dev'=0MB;9;10;0;10

On voit que le disque est en état critique à cause du CDROM qui est plein à 100%.

On va démonter le cdrom, juste pour avoir quelque chose de mieux.

Après éjection du CDROM, on a bien :



Pour les 2 autres erreurs il faudrait en fait définir un serveur **ntp** (voir <http://fr.wikipedia.org/wiki/Network_Time_Protocol>) pour notre machine et paramétrer correctement l'emplacement des logs (ce que nous ne ferons pas).

## Travail à faire :

Mettre en place NTP et vérifier sa prise en charge par Shinken.

Enlever la référence à la commande gérant les logs dans le modèle linux-snmp et vérifier le résultat sur webui

# Partie 5 : Paramétrage des notifications

Travail à faire : procéder aux différentes actions proposées pour effectuer le paramétrage des notifications sur votre système.

Notre objectif est de surveiller le fonctionnement et de réagir en cas de problème. Pour réagir, il faut être prévenu. La moindre des choses que l'on peut demander à notre superviseur c'est de nous alerter en cas de comportement anormal sur notre système. Cela s'appelle **notifier.**

On va donc envoyer des notifications à l'administrateur en cas de problème. Shinken permet de notifier de toutes les façons possibles et imaginables puisque tout est paramétrable ici. Mais nous ne disposons pour l'instant ( ☹ ) que d'une seule possibilité : l'envoi de mails. Pour cela il nous faut un serveur de mails. On va en installer un minimaliste juste pour comprendre comment fonctionne l'envoi des notifications.

## Installation et test de postfix sur le superviseur

On a besoin d'un serveur de mail pour envoyer des messages aux administrateurs en cas d'alertes.

On va installer **postfix** avec un paramétrage basique dans un premier temps (🡺 choisir **site internet** dans la configuration du paquet).

root@debianWheezy:# apt-get install postfix

Il faut modifier le fichier **/etc/postfix/main.cf** pour inhiber la résolution par **DNS** qui est active par défaut, et forcer la prise en compte du fichier **/etc/hosts :**

# See /usr/share/postfix/main.cf.dist for a commented, more complete version

# Debian specific: Specifying a file name will cause the first

# line of that file to be used as the name. The Debian default

# is /etc/mailname.

#myorigin = /etc/mailname

smtpd\_banner = $myhostname ESMTP $mail\_name (Debian/GNU)

biff = no

# appending .domain is the MUA's job.

append\_dot\_mydomain = no

# Uncomment the next line to generate "delayed mail" warnings

#delay\_warning\_time = 4h

readme\_directory = no

# TLS parameters

smtpd\_tls\_cert\_file=/etc/ssl/certs/ssl-cert-snakeoil.pem

smtpd\_tls\_key\_file=/etc/ssl/private/ssl-cert-snakeoil.key

smtpd\_use\_tls=yes

smtpd\_tls\_session\_cache\_database = btree:${data\_directory}/smtpd\_scache

smtp\_tls\_session\_cache\_database = btree:${data\_directory}/smtp\_scache

# See /usr/share/doc/postfix/TLS\_README.gz in the postfix-doc package for

# information on enabling SSL in the smtp client.

myhostname = debianWheezy.sisr5.org

alias\_maps = hash:/etc/aliases

alias\_database = hash:/etc/aliases

mydestination = debianWheezy, debianWheezy.sisr5.org, localhost.sisr5.org, localhost

relayhost =

mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128

mailbox\_command = procmail -a "$EXTENSION"

mailbox\_size\_limit = 0

Il faut ajouter ce paramètre

recipient\_delimiter = +

inet\_interfaces = all

smtp\_host\_lookup = native

Le dernier paramètre que l'on a rajouté (**smtp\_host\_lookup**) quand il est positionné à **native** évite de faire résoudre les noms systématiquement par le DNS car nous n'avons pas installé de DNS ici.

Ce paramètre force **postfix** à passer par le fichier **/etc/nsswitch.conf**, qui va lui donner comme méthode de résolution de noms **file** soit pour nous **/etc/hosts.**

En effet dans le fichier **/etc/nsswitch.conf**, on trouve ceci :

hosts: files mdns4\_minimal [NOTFOUND=return] dns mdns4

On vérifie aussi que le fichier **/etc/mailname** est correct :

root@debianWheezy:/home/roger# cat /etc/mailname

debianWheezy.sisr5.org

Il faut bien sûr redémarrer le service **postfix**:

root@debianWheezy:/home/roger# service postfix restart

## Création d'un utilisateur pour recevoir les notifications

On va créer un utilisateur pour tester notre notification (**roger** ici) :

root@debianWheezy:$ETC# adduser roger

Créez un utilisateur avec votre prénom et dans tout ce qui suit, remplacez **roger** par votre prénom.

## Test du serveur mail postfix et de l’utilisateur

Attention : Si le fichier **resolv.conf** ne définit pas un domaine de recherche par défaut, il faut mettre le nom complet, par exemple roger@debianWheezy.sisr5.org

On envoie un message avec la commande **mail** pour vérifier que ça fonctionne :

root@debianWheezy:/home/roger# mail -s "bonjour" roger@debianWheezy.sisr5.org

hello

j'essaie de t'envoyer un message

Cc:

Ctrl+d //pour sortir

Il suffit ensuite de se connecter avec **roger** (**su - roger**) et de vérifier qu'on a reçu les messages avec la commande **mail (**pour revenir à **root** tapez **exit).**

Avec le -, on récupère l'environnement de l'utilisateur (.profile)

roger@DebianWheezy:~$ mail

Mail version 8.1.2 01/15/2001. Type ? for help.

"/var/mail/roger": 6 messages 6 new

>N 1 root@debianWheezy Sat Feb 09 19:09 20/650 info: mpt raid status chang

N 2 root@debianWheezy Sat Feb 09 20:42 20/650 info: mpt raid status chang

N 3 root@debianWheezy Sat Feb 09 20:48 20/650 info: mpt raid status chang

N 4 root@debianWheezy Sat Feb 09 22:19 20/650 info: mpt raid status chang

N 5 root@debianWheezy Sun Feb 10 13:35 20/650 info: mpt raid status chang

N 6 root@debianWheezy Sun Feb 10 13:44 15/431 bonjour

& 6

Message 6:

From root@debianWheezy Sun Feb 10 13:44:15 2013

X-Original-To: roger@debianWheezy

To: roger@debianWheezy

Subject: bonjour

Date: Sun, 10 Feb 2013 13:44:15 +0100 (CET)

From: root@debianWheezy (root)

hello

j'essaie de t'envoyer une message

& q // pour sortir

Cela semble fonctionner.

Notre **postfix** est opérationnel pour ce que l'on veut faire.

On va passer à la suite, c’est-à-dire la configuration des notifications dans Shinken.

*Revenir à* ***root*** *par la commande* ***exit.***

## Commandes de notification par défaut

Pour envoyer des mails, Shinken doit disposer de commandes permettant de le faire.

Les commandes d’envoi de mail sont définies dans $ETC/notificationways :

root@debianWheezy:/etc/shinken/hosts# cat /etc/shinken/notificationways/email.cfg

# This is how emails are sent, 24x7 way.

define notificationway {

notificationway\_name email

service\_notification\_period 24x7

host\_notification\_period 24x7

service\_notification\_options c,w,r

host\_notification\_options d,u,r,f,s

service\_notification\_commands notify-service-by-email ; send service notifications via email

host\_notification\_commands notify-host-by-email ; send host notifications via email

}

**Rappel** : w (warning), c (critical) , r (recovery), d (down), u (unknown).

On peut observer qu’une période de temps (**24x7**) est respectée pour envoyer les notifications.

Cette période est définie dans **$ETC/timeperiods/24x7.cfg**

define timeperiod{

timeperiod\_name 24x7

alias Always

sunday 00:00-24:00

monday 00:00-24:00

tuesday 00:00-24:00

wednesday 00:00-24:00

thursday 00:00-24:00

friday 00:00-24:00

saturday 00:00-24:00

}

On comprend assez facilement que la période 24x7 définit une plage de 24h sur 24 et de 7 jours sur 7.

Les commandes utilisées par le service de notification sont définies dans le répertoire $ETC**/commands**.

## Notify Host by Email

define command {

command\_name notify-host-by-email

command\_line /usr/bin/printf "%b" "Shinken Notification\n\nType:$NOTIFICATIONTYPE$\nHost: $HOSTNAME$\nState: $HOSTSTATE$\nAddress: $HOSTADDRESS$\nInfo: $HOSTOUTPUT$\nDate/Time: $DATE$\n" | /usr/bin/mail -s "Host $HOSTSTATE$ alert for $HOSTNAME$!" $CONTACTEMAIL$

}

## Notify Service by Email

define command {

command\_name notify-service-by-email

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

}

L'envoi est réalisé en 2 étapes : il y a d'abord la génération du texte (par la commande **printf**) puis l’envoi par mail (par la commande **mail**).

Des macros (variables) sont utilisées, signalées par l'encadrement des $.

Par exemple, la macro **$HOSTNAME$** sera remplacée par **debianWheezy** dans le cas d'une alerte sur cette machine.

## Configuration du contact à alerter dans shinken

L’utilisateur créé précédemment existe pour Linux mais pas pour Shinken. Il faut créer un contact dans Shinken et l’associer à notre utilisateur.

Nous allons configurer ce contact dans le fichier **$ETC/contacts/roger.cfg** . Attention : on rajoute un contact aux 2 contacts existants (**admin** et **guest**).

define contact{

use generic-contact

contact\_name roger

email roger@debianWheezy.sisr5.org

}

Ce contact est identifié par le nom **roger,** il hérite de **generic-contact** qui définit des valeurs par défaut pour les paramètres de notification, notamment le droit de recevoir des notifications et la commande de notification utilisée.

Rappel : les modèles génériques sont définis dans le fichier **$ETC/templates**.

# Contact definition

# By default the contact will ask notification by mails

define contact{

name generic-contact

register 0

host\_notifications\_enabled 1

service\_notifications\_enabled 1

email shinken@localhost

can\_submit\_commands 1

notificationways email

}

## Configuration du déclenchement des alertes

Quand les notifications sont-elles envoyées par Shinken ?

La périodicité du déclenchement des notifications est définie de façon générale dans les modèles **generic\_host** et **generic-service**.*).*

# Generic service definition template - This is NOT a real service, just a template!

define service {

name generic-service ; The 'name' of this service template

active\_checks\_enabled 1 ; Active service checks are enabled

passive\_checks\_enabled 1 ; Passive service checks are enabled/accepted

notifications\_enabled 1 ; Service notifications are enabled

notification\_interval 10

notification\_period 24x7

event\_handler\_enabled 0 ; Service event handler is enabled

flap\_detection\_enabled 1 ; Flap detection is enabled

process\_perf\_data 1 ; Process performance data

is\_volatile 0 ; The service is not volatile

check\_period 24x7 ; The service can be checked at any time of the day

max\_check\_attempts 3 ; Re-check the service up to 3 times in order to determine its final (hard) state

check\_interval 5 ; Check the service every 5 minutes under normal conditions

retry\_interval 2 ; Re-check the service every two minutes until a hard state can be determined

notification\_options w,u,c,r ; Send notifications about warning, unknown, critical, and recovery events

contact\_groups admins,users

stalking\_options o,w,u,c

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL SERVICE, JUST A TEMPLATE

}

Ce fichier définit la façon dont Shinken vérifie l'état des objets supervisés.

Quand un objet est à l'état OK, il est vérifié toutes les **check\_interval** minutes.

S'il passe à l'état WARNING, CRITICAL ou UNKNOWN, il sera alors vérifié **max\_check\_attempts** fois à un intervalle de **retry\_interval** minutes avant de déclencher une notification. On considère alors que le service testé ou l’hôte est à l’état « hard ».

D’autres périodes sont définies dans le fichier **$ETC/templates/time\_templates.cfg**.

Il y a notamment la période **1hour\_long** qui nous intéresse car elle est utilisée par le service **Disks** qui va nous permettre de tester les notifications.

# Check every 1hour with hard state after 6hours

define service {

name 1hour\_long

use generic-service

max\_check\_attempts 6

normal\_check\_interval 60

retry\_interval 60

register 0

}

root@debianWheezy:~# cat /etc/shinken/packs/linux-snmp/services/disks.cfg

define service {

service\_description Disks

use 1hour\_long,linux-service

register 0

host\_name linux-snmp

check\_command check\_linux\_disks

\_DETAILLEDESC Overall disks usage

\_IMPACT Depends on disks, cause system instability

\_FIXACTIONS Clean the appropriate disks

}

Le service Disks redéfinit donc à son niveau quelques valeurs de **generic\_service**.

3 paramètres nous intéressent particulièrement :

* + max\_check\_attempts 6 🡺 Nombre de “check” renvoyant une erreur avant l’envoi d’une notification
  + normal\_check\_interval 60 🡺 temps entre 2 “check” renvoyant une erreur
  + retry\_interval 60 🡺 temps entre 2 “check” ne renvoyant pas d’erreur

Les temps sont exprimés en minutes.

Si on applique cela, la première notification nous parviendra après un temps calculé avec la formule suivante :

**normal\_check\_interval + (retry\_interval \* (max\_check\_attempts – 1)).** *moins 1 parce que le premier retour avec erreur compte.*

Donc en appliquant les valeurs du modèle **1hour\_long** on obtient :

**60 + (60 \* (6 – 1) ) = 360 minutes** soit **6 heures**

Ce qui est beaucoup trop pour nos tests.

On va donc utiliser plutôt le modèle de période suivant :

define service {

name 5min\_short

use generic-service

max\_check\_attempts 1

normal\_check\_interval 5

retry\_interval 2

register 0

}

En modifiant ainsi le service **Disks** de notre modèle linux\_snmp.

define service {

service\_description Disks

use 5min\_short,linux-service

register 0

host\_name linux-snmp

check\_command check\_linux\_disks

\_DETAILLEDESC Overall disks usage

\_IMPACT Depends on disks, cause system instability

\_FIXACTIONS Clean the appropriate disks

}

A ce stade nos alertes sont déclenchées dès la première erreur après un temps calculé avec la formule suivante :

**5+ (2 \* (1 – 1) ) = 5 minutes.**

Puis ensuite toutes les 2 minutes.

## Configuration du déclenchement des notifications

Une alerte déclenchée n’est pas pour autant notifiée.

L’autorisation de notification pour un service ou un hôte, la plage de notification et la fréquence de notification est définie de façon générale dans les modèles **generic\_host** et **generic-service**. *J'ai modifié ici la directive* ***notification\_interval*** *pour envoyer des messages toutes les 10 minutes dans les tests (par défaut 1440 soit 24 heures).*

Les contacts notifiés et le niveau d’erreur à partir duquel une notification est envoyée est aussi stipulé ici.

# Generic service definition template - This is NOT a real service, just a template!

define service{

name generic-service ; The 'name' of this service template

active\_checks\_enabled 1 ; Active service checks are enabled

passive\_checks\_enabled 1 ; Passive service checks are enabled/accepted

notifications\_enabled 1 ; Service notifications are enabled

notification\_interval 10

notification\_period 24x7

…

notification\_options w,u,c,r ; Send notifications about warning, unknown, critical, and recovery events

contact\_groups admins,users

stalking\_options o,w,u,c

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL SERVICE, JUST A TEMPLATE

}

Attention, il faut bien sûr préciser dans le fichier **$ETC/hosts/localhost.cfg** que l'on veut notifier notre contact en cas de problème.

define host {

use linux-snmp,generic-host

contact\_groups admins

host\_name localhost

address localhost

}

Dans notre cas, le groupe **admins** étant notifié, on va rajouter dans ce groupe l’utilisateur **roger** :

root@debianWheezy:~# cat /etc/shinken/contactgroups/admins.cfg

define contactgroup{

contactgroup\_name admins

alias admins

members admin,roger

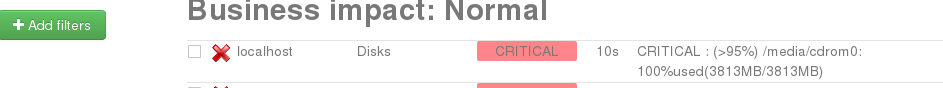
}

Après toutes ces modifications dans les fichiers de configuration, Il faut faire un contrôle de ces fichiers (*service shinken check)* et relancer Shinken.

# Partie 6 : Test des notifications

Travail à faire : procéder aux différentes actions proposées pour tester les notifications sur votre système.

Nous allons tester en provoquant une alerte 🡺 un moyen simple étant de connecter un DVD ISO à notre machine virtuelle, ce qui va générer une alerte *disque plein*.



Vérifions si on reçoit bien un mail en se connectant avec roger :

roger@debianWheezy.sisr5.org:~$ mail

"/var/mail/roger": 1 message 1 new

Message 112:

From shinken@debianWheezy.sisr5.org Mon Mar 2 17:50:20 2015

X-Original-To: roger@debianWheezy.sisr5.org

To: roger@debianWheezy.sisr5.org

Subject: \*\* PROBLEM alert - localhost/Disks is CRITICAL \*\*

Date: Mon, 2 Mar 2015 17:50:19 +0100 (CET)

From: shinken@debianWheezy.sisr5.org

Shinken Notification

Notification Type: PROBLEM

Service: Disks

Host: localhost

Address: localhost

State: CRITICAL

Et on peut suivre l'envoi régulier des mails dans **/var/log/mail.log :**

root@debianWheezy:~# tail -f /var/log/mail.log

Mar 2 16:04:49 debianWheezy postfix/qmgr[30063]: 748624014D: removed

Mar 2 16:20:20 debianWheezy postfix/pickup[14531]: 2FC0447304: uid=1001 from=<shinken>

Mar 2 16:20:20 debianWheezy postfix/cleanup[26492]: 2FC0447304: message-id=<20150302152020.2FC0447304@debianWheezy>

Mar 2 16:20:20 debianWheezy postfix/qmgr[30063]: 2FC0447304: from=<shinken@debianWheezy.sisr5.org>, size=564, nrcpt=1 (queue active)

Mar 2 16:20:20 debianWheezy postfix/pickup[14531]: 34C144014D: uid=1001 from=<shinken>

Mar 2 16:20:20 debianWheezy postfix/cleanup[26492]: 34C144014D: message-id=<20150302152020.34C144014D@debianWheezy>

Mar 2 16:20:20 debianWheezy postfix/qmgr[30063]: 34C144014D: from=<shinken@debianWheezy.sisr5.org>, size=575, nrcpt=1 (queue active)

Mar 2 16:20:20 debianWheezy postfix/local[26494]: 2FC0447304: to=<shinken@localhost>, relay=local, delay=0.09, delays=0.04/0.01/0/0.05, dsn=2.0.0, status=sent (delivered to command: procmail -a "$EXTENSION")

Mar 2 16:20:20 debianWheezy postfix/qmgr[30063]: 2FC0447304: removed

Mar 2 16:20:20 debianWheezy postfix/local[26495]: 34C144014D: to=<roger@debianWheezy.sisr5.org>, relay=local, delay=0.1, delays=0.05/0.02/0/0.04, dsn=2.0.0, status=sent (delivered to command: procmail -a "$EXTENSION")

Mar 2 16:20:20 debianWheezy postfix/qmgr[30063]: 34C144014D: removed

Et on peut observer aussi le contenu du fichier $LOG/schedulerd.log :

[1389521951] SERVICE NOTIFICATION: roger;localhost;Disks;CRITICAL;notify-service-by-email;CRITICAL : (>95%) /media/cdrom0: 100%used(3813MB/3813MB)

[1389522011] SERVICE NOTIFICATION: roger;localhost;Disks;CRITICAL;notify-service-by-email;CRITICAL : (>95%) /media/cdrom0: 100%used(3813MB/3813MB)

[1389522071] SERVICE NOTIFICATION: roger;localhost;Disks;CRITICAL;notify-service-by-email;CRITICAL : (>95%) /media/cdrom0: 100%used(3813MB/3813MB)

[1389522131] SERVICE NOTIFICATION: roger;localhost;Disks;CRITICAL;notify-service-by-email;CRITICAL : (>95%) /media/cdrom0: 100%used(3813MB/3813MB)

[1389522191] SERVICE NOTIFICATION: roger;localhost;Disks;CRITICAL;notify-service-by-email;CRITICAL : (>95%) /media/cdrom0: 100%used(3813MB/3813MB)

[1389522251] SERVICE NOTIFICATION: roger;localhost;Disks;CRITICAL;notify-service-by-email;CRITICAL : (>95%) /media/cdrom0: 100%used(3813MB/3813MB)

[1389522311] SERVICE NOTIFICATION: roger;localhost;Disks;CRITICAL;notify-service-by-email;CRITICAL : (>95%) /media/cdrom0: 100%used(3813MB/3813MB)

[1389522371] SERVICE NOTIFICATION: roger;localhost;Disks;CRITICAL;notify-service-by-email;CRITICAL : (>95%) /media/cdrom0: 100%used(3813MB/3813MB)

[1389522431] SERVICE NOTIFICATION: roger;localhost;Disks;CRITICAL;notify-service-by-email;CRITICAL : (>95%) /media/cdrom0: 100%used(3813MB/3813MB)

# Partie 7 : Supervision d'un service

Travail à faire : procéder aux différentes actions proposées pour mettre en place la supervision du serveur mail sur votre système.

Nous savons faire beaucoup de choses maintenant, mais les services que nous avons gérés jusqu'à présent à l'aide du modèle Linux sont assez proches de la machine. On aimerait superviser des services de niveau plus haut. Par exemple, un serveur mail, ça tombe bien on a justement le service **Postfix** actif sur notre machine !

**Comment faire ?**

On va créer un fichier dans **$ETC/services,** qu'on nomme comme on veut, ici **mailServeur.cfg**, et un service associé à l'hôte qu'on nomme comme on veut aussi (ici **SMTP**).

root@debianWheezy:$ETC/services# cat serveurMail.cfg

define service{

use generic-service ; Inherit default values from a template

host\_name localhost

service\_description SMTP

check\_command check\_tcp!25

}

Notre service est un service **TCP** qui écoute sur le port **25**. Une des façons de le superviser consiste tout simplement à tester si le service est bien à l'écoute sur ce port.

C'est le rôle de la commande **check\_tcp** que l'on trouve dans le fichier décrit ci-dessous : $ETC**/commands/check\_tcp.cfg**

root@debianWheezy:~# cat /etc/shinken/commands/check\_tcp.cfg

## Check a TCP port

# This plugin tests TCP connections with the specified host (or unix socket).

# check\_tcp -H host -p port [-w <warning time>] [-c <critical time>] [-s <send

# string>] [-e <expect string>] [-q <quit string>][-m <maximum bytes>] [-d

# <delay>] [-t <timeout seconds>] [-r <refuse state>] [-M <mismatch state>]

# [-v] [-4|-6] [-j] [-D <warn days cert expire>[,<crit days cert expire>]] [-S

# <use SSL>] [-E]

define command {

command\_name check\_tcp

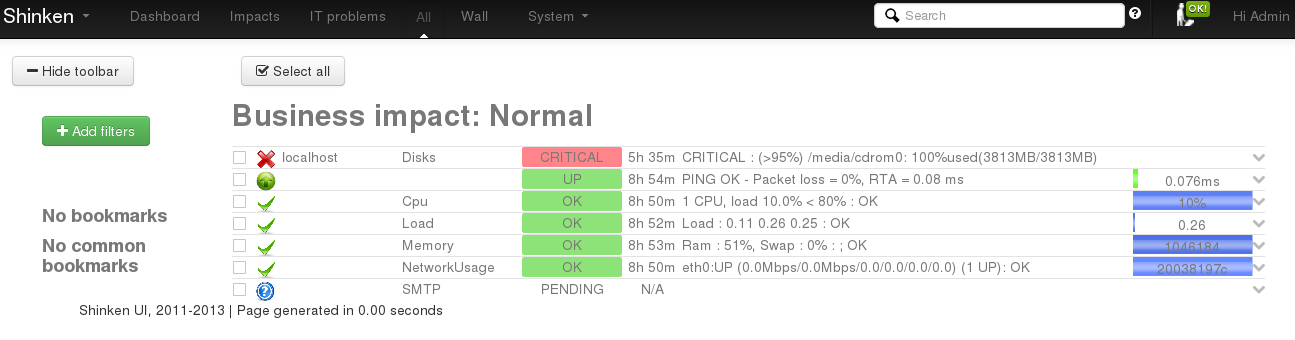
command\_line $NAGIOSPLUGINSDIR$/check\_tcp -H $HOSTADDRESS$ -p $ARG1$

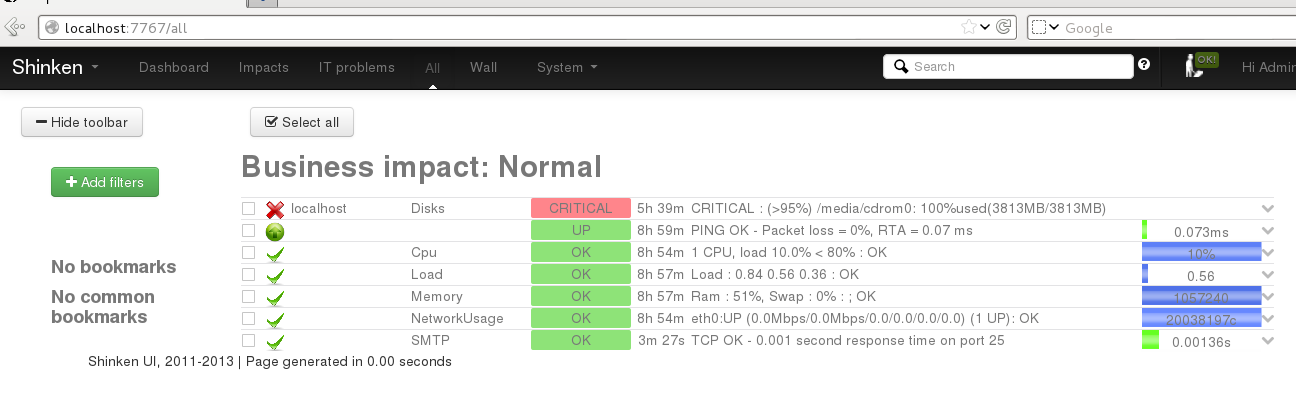
}

On constate que la commande admet un argument **$ARG1$**.

Pour passer une valeur (ici le numéro de port 25) à cet argument, on a utilisé le connecteur **!** comme ceci : **check\_tcp!25**.

On relance Shinken et on regarde si notre service est bien pris en compte dans l'interface **webUI :**





Ça fonctionne ! Et on est arrivé au bout de cette prise en main et des objectifs que l'on s'était fixés.

# Partie 8 : Faisons le point

On est arrivé au bout de notre objectif minimal de départ, c’est-à-dire superviser la machine hôte et le service qui tourne sur celle-ci, **postfix**, en notifiant les alertes à un contact.

On a fait beaucoup de choses pour cela. On va essayer de récapituler pour bien comprendre l'architecture de l'ensemble et revenir sur des éléments de vocabulaire.

## Les objets

Shinken définit plusieurs objets pour gérer la supervision ; ceux que nous avons utilisés ici sont :

* **Hosts** : cet objet représente tout équipement serveur ou réseau (**localhost** dans notre exemple).
* **Service** : cet objet est lié à un équipement et représente un service qu'il peut fournir (**SMTP** dans notre exemple).
* **Contact** : cet objet définit un contact avec des caractéristiques de notifications (**roger** dans notre exemple).
* **Contactgroup** : cet objet groupe les contacts (**admins** dans notre exemple).
* **Command** : cet objet définit les commandes utilisées pour la vérification des services.
* **Timeperiod** : cet objet spécifie les périodes auxquelles sont joignables les contacts (24x7 dans notre exemple).

Shinken définit d'autres objets que l'on n'a pas utilisés :

* **Hostgroup** : cet objet groupe les équipements pour les représenter de manière logique.
* **Hostdependency** et **servicedependency** : cet objet spécifie des dépendances entre services ou hosts pour limiter les alertes remontées en cas de défaillance d'un élément duquel dépendent les autres éléments.
* **Escalation** : cet objet définit l'organisation de la notification des alertes dans la hiérarchie des niveaux.

## La dépendance des objets :

Comme on a pu le constater, certains objets font appel à d'autres objets, il faut donc créer les objets dans un certain ordre.

* L'objet **command** ne dépend d'aucun autre objet.
* L'objet **timeperiod** ne dépend d'aucun autre objet.
* L'objet **contact** nécessite un objet **timeperiod.**
* L'objet **contactgroup** nécessite un objet **contact.**
* L'objet **hôte** nécessite un objet **command, timeperiod** et **contact.**
* L'objet **hostgroup** nécessite un objet **hôte.**
* L'objet **hôte** nécessite un objet **command, timeperiod** et **contact.**
* L'objet **service** nécessite un objet **command, timeperiod, contact** et **hôte.**
* L'objet **hostdependency** nécessite un objet **hôte.**
* L'objet **servicedependency** nécessite un objet **service.**
* L'objet **escalation** nécessite un objet **contactgroup** et **timeperiod.**

## La supervision d'un équipement et de ses services

Pour superviser un équipement, il faut créer un objet **host** et indiquer son nom (dans notre exemple **locahost**), son adresse IP (dans notre exemple 127.0.0.1) et un moyen de le tester (dans notre exemple **check\_icmp** ou **check\_host\_alive**).

On entend par service à la fois les services réseau (dans notre exemple SMTP) et les composants matériels d'un équipement (dans notre exemple le CPU). **Tout point de supervision sur un hôte est un service.**

## La supervision se fait par des sondes

Les commandes représentent les sondes qui sont lancées pour tester un équipement.

Par convention les sondes Nagios/Shinken acceptent ces paramètres :

* -H l'adresse IP à tester
* -W le seuil warning
* -C le seuil critical

On doit tester les commandes dans le **shell** avant de les faire exécuter par Shinken. Il faut les tester avec l'utilisateur Shinken.

Certaines commandes admettent des paramètres (dans notre exemple **check\_tcp**), on passe les paramètres en les faisant précéder d'un point d'exclamation "!" (dans notre exemple **check\_tcp!25**). Si on avait plusieurs paramètres, cela donnerait par exemple **check\_tcp!25!10**

Une commande est généralement appelée par un objet **hôte** ou un objet **service**. Elle ne peut s'exécuter de façon indépendante puisque par définition on doit lui spécifier l'objet supervisé.

## Le paramétrage est facilité par l'utilisation de modèles

Les modèles (*templates*) sont parfois nommés gabarits ou patrons.

Les modèles permettent de définir des typologies particulières d'équipement pour lequel on définit les éléments à superviser (les services) et la façon de les superviser (les commandes). En fait, ils permettent une factorisation des éléments communs de supervision et permettent ainsi de diminuer grandement le nombre de fichiers de configuration.

Cela simplifie énormément la mise au point de la supervision puisqu'il suffit de créer des hôtes ou des services de façon minimaliste (nom et adresse IP pour l'hôte par exemple) puis de les associer (avec la directive **use**) à des modèles existants.

Un hôte ou un service peut hériter d’un ou plusieurs modèles.

On peut créer ses propres modèles.

Un modèle est repéré par la directive **register** mise à la valeur 0 dans sa définition.

Dans notre exemple nous avons utilisé le modèle **linux** mais aussi le modèle **generic-host** et le modèle **generic-service**.

## Les modèles développés par la communauté d'utilisateurs de Shinken/Nagios sont fournis dans des packs

Les packs sont un ensemble de fichiers décrivant une bonne manière de superviser un sujet (dans notre exemple le pack **linux**).

C'est un ensemble de fichiers de configurations (dans notre exemple l'arborescence à partir $ETC**/packs/linux\_snmp).**

Un pack est généralement formé des fichiers suivants :

* templates.cfg : modèle pour un hôte ;
* commands.cfg : commandes de supervision spécifique au pack ;
* discovery.cfg : on verra ultérieurement ;
* services : c'est un répertoire qui contiendra pour chaque service spécifique au pack les fichiers de configuration.

## Récapitulons : que doit-on faire pour superviser un hôte Linux ?

* Définir un fichier hôte avec au minimum un nom et une adresse IP.
* Faire hériter ce fichier hôte du modèle linux.
* Tester les commandes présentes dans le modèle.
* Si les commandes ne fonctionnent pas, les mettre au point soit en les installant, soit en paramétrant de éléments complémentaires sur le superviseur (dans notre exemple on a mis au point quelques commandes) ou la machine supervisée (dans notre exemple **snmpd).**

Si on veut envoyer des notifications à un contact précis, il faut créer ce contact comme utilisateur dans la machine de supervision, puis comme contact dans Shinken et le faire hériter du modèle generic-contact.

Si on veut superviser un service particulier, on doit définir ce service en lui donnant un nom (SNMP dans notre exemple), l'associer à un hôte (localhost dans notre exemple) lui associer une commande (check\_tcp dans notre exemple).

# Partie 9 : Généralisons ce que nous avons appris

Travail à faire : Trois exercices d'application sont proposés, il faut superviser les types d'hôte suivants et donner tous les éléments de réponse aux questions.

## Une première application simple en autonomie 🡺superviser la machine Linux de son voisin

Cela devrait être assez facile. Il faut bien sûr au préalable que vos 2 machines communiquent.

Et puis après, que faut-il faire ?

La déclaration d'un nouvel **host** devrait suffire, il faut aussi que le service snmpd écoute sur l'adresse IP effective et non sur l'adresse « localhost ».

**A vous de jouer 🡺** n'oubliez de fournir une documentation comportant les fichiers de configuration commentés.

## Administration d'une machine Windows d'adresse x.xx.x (à adapter) avec notification à un utilisateur adminWindows.

* Donner le(s) fichier(s) que vous créez et son (leur) contenu.
* Donner le pack utilisé.
* Donner les commandes que vous testez avec les paramètres de tests.
* Donner si nécessaire les éléments nouveaux à installer/configurer sur le superviseur.
* Donner si nécessaire les éléments nouveaux à installer/configurer sur la machine supervisée.

Des éléments de réponse sont donnés dans l'activité 2.1 (**basés sur une version de Shinken 1.4 à adapter à la version 2.2**).

**Administration d'un commutateur Cisco 2960 d'adresse x.x.x.x(à adapter) avec notification à un utilisateur adminCisco.**

* Donner le(s) fichiers que vous créez et leur contenu.
* Donner le pack utilisé.
* Donner les commandes que vous testez avec les paramètres de tests.
* Donner si nécessaire les éléments nouveaux à installer/configurer sur le superviseur.
* Donner si nécessaire les éléments nouveaux à installer/configurer sur la machine supervisée.

Les éléments de réponse sont donnés dans l'activité 2.2 (**basés sur une version de Shinken 1.4 à adapter à la version 2.2**).

.

# Partie 10 : Automatisons la configuration de la supervision des hôtes et des services

Travail à faire : découvrir sur le réseau les éléments à superviser et produire automatiquement les fichiers de configuration correspondant.

On a pu constater dans les actions précédentes que la configuration de la supervision d'unobjet n'est pas une tâche anodine même en utilisant les "packs".

Dans un environnement constitué de nombreux serveurs ou actifs à superviser, cette tâche peut s'avérer très lourde. Shinken fournit une solution permettant la découverte des hôtes sur le réseau et l'écriture automatique des fichiers de configuration permettant leur prise en compte par le superviseur.

Les éléments de réponse sont donnés dans l'activité 2.3 (basés sur une version de Shinken 1.4 à adapter à la version 2.2)..