Surveillance de réseau avec Icinga sur un Raspberry Pi

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 25 octobre 2012. Dernière mise à jour le 11 novembre 2012

https://www.bortzmeyer.org/icinga.html

Je vous ai déjà parlé ici de mes essais avec le Raspberry Pi <https://www.bortzmeyer.org/raspberry-pi.html> et de mon interrogation : « Bon, d'accord, ce petit ordinateur pas cher est très cool. Mais à quoi peut-il servir en vrai, à part à afficher les messages de "*boot*" Linux sur mon home cinema ? » Un des usages qui me semblait intéressant, vue notamment la faible consommation électrique du Pi, était de la surveillance de réseau. Cet article décrit la configuration avec le logiciel de surveillance Icinga.

Il y a des tas d'autres logiciels de surveillance de réseau. Le plus connu est sans doute Nagios. Ces logiciels sont tous composés :

- D'un ordonnanceur qui va lancer les tests, et déclencher les alarmes si les tests échouent (une fois qu'Icinga est installé, vous pouvez voir la file d'attente des tâches via le menu `System -> Scheduling Queue`),
- D'un ensemble de programmes de tests, allant d'un simple ping à des tests HTTP plus ou moins sophistiqués, en passant par tous les protocoles réseaux possibles, et naturellement à des mesures faites avec SNMP,
- D'un ensemble de programmes qui mettent en œuvre les alertes (envoi d'un message, d'un SMS, etc),
- Parfois d'une interface utilisateur, en général via le Web.

Comme Nagios est très populaire, de nombreux programmes de tests et d'alarmes ont été écrits pour lui (voir NagiosExchange <http://exchange.nagios.org/> et Monitoring Exchange <https://www.monitoringexchange.org/>), et son API d'écriture de "*plugins*" est donc devenue la référence. C'est une des raisons pour lesquelles j'ai choisi un programme issu de Nagios, ayant une API compatible, Icinga (pour savoir pourquoi Icinga et Nagios se sont séparés, voir cette histoire <http://www.freesoftwaremagazine.com/articles/nagios_and_icinga>). Je sais, il existe des tas d'autres programmes de surveillance, mais je n'avais pas la patience de les tester tous.

D'abord, pour le Pi <https://www.bortzmeyer.org/raspberry-pi.html> lui-même. C'est un vrai ordinateur, doté d'un processeur généraliste, de tout ce qu'il faut pour faire tourner un noyau moderne (en l'occurrence Linux), et d'assez de mémoire pour qu'un programme comme Icinga et ses "*plugins*" ne lui fasse pas peur. Vue, par top, voici la consommation de ressources d'Icinga après 24 h de fonctionnement (attention, certains des "*plugins*" peuvent être très gourmands) :

```
  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
  201 icinga    20   0 16508 2924  932 S   0.3  1.6   7:55.72 icinga
```

Et, sur le Pi, je gère actuellement 41 "*hosts*" et 126 "*services*".

Mon Pi tourne sous Arch Linux <https://www.bortzmeyer.org/archlinux.html>. Ce système m'a valu quelques mauvaise surprises dont une partie était due au matériel : le Pi n'a pas de protections contre une mauvaise alimentation électrique ou une mauvaise RAM et sa carte SD est... une carte SD, ce qui veut dire qu'elle est fragile et de durée de vie limitée. Pour une utilisation plus sérieuse d'Icinga sur un Pi, il faudrait brancher un disque dur USB et y mettre l'historique stocké par Icinga. Ici, c'est un usage léger, pour surveiller mes machines depuis la maison, je me suis donc contenté de la carte SD, de sauvegardes régulières, et de prières à Saint-Isidore.

Autre problème sur le Pi, son absence d'horloge stable. Bien sûr, j'ai un serveur NTP installé mais, parfois, cela ne fonctionne pas et il m'est arrivé de me retrouver en 1970, ce qui perturbe sérieusement Icinga, notamment pour l'affichage des tendances à long terme. (Et, oui, j'ai bien l'option `-s` du serveur NTP.) Il faut donc surveiller l'heure, après les redémarrages.

Une autre partie des surprises avec Arch Linux est venue du fait que ce système ne prétend pas être un système d'exploitation simple et pour n'importe qui. Il y a des pièges, il faut lire la documentation et, une fois que c'est fait, la relire, car Arch Linux change souvent (j'ai ainsi vécu le passage à systemd).

Ensuite, il faut installer Icinga sur Arch Linux. Une des solutions possibles est manuelle, l'autre utilise les paquetages. Dans les deux cas, il ne faut pas oublier d'installer également « nagios-plugins » qui contient les programmes de test. Il n'est pas obligatoire (si on utilise des tests qu'on a écrit ou bien récupérés ailleurs) mais, en pratique, on ne peut pas vivre sans. La compilation des deux programmes (icinga et nagios-plugins) n'est pas rapide sur le Pi, pensez à aller lire Hunger Games pendant ce temps (ou bien utilisez un compilateur croisé sur une machine plus rapide).

Pour l'installation sous forme de paquetage, icinga n'est pas dans le dépôt standard d'Arch Linux. Il faut donc utiliser AUR <http://aur.archlinux.org/> ("*Arch User Repository*", et merci à Samuel Tardieu pour la suggestion), un ensemble de paquetages non officiel. Pour installer des programmes AUR, je me suis servi de pacaur <https://github.com/Spyhawk/pacaur>. Une fois qu'on a installé cower <https://github.com/falconindy/cower> et pacaur, on peut taper :

```
% pacaur -S icinga nagios-plugins
```

On répond que, oui, on veut éditer le `PKGBUILD` et on le modifie pour mettre :

```
arch=('armv6h')
```

car la plupart des paquetages de l'AUR ne connaissent que i386. Ah, et puis on met un serveur HTTP, j'ai choisi Apache qui, lui, est dans les dépôts standards d'Arch Linux.

Donc, pour configurer Icinga, on commence évidemment par lire la documentation <http://docs.icinga.org/latest/en/>, puis on va dans `/etc/icinga` (tous les noms de fichiers et de répertoires sont ceux du paquetage Icinga de l'AUR). J'ai gardé les options par défaut dans `icinga.cfg` (à part le format de date), puis, dans `objects/`, copié les fichiers de configuration d'exemple avant d'en éditer certains : mon adresse de courrier dans `contacts.cfg`, les paramètres SSH dans `localhost.cfg`. Ces derniers sont l'occasion de voir comment on définit un service qu'on surveille :

```
define service{
        use                             local-service
        host_name                       localhost
        service_description             SSH
        check_command                   check_ssh
}
```

La commande `check_ssh` est définie dans `commands.cfg`. Elle se connecte au serveur SSH et vérifie qu'il répond. Comme tous les "*plugins*" Nagios, on peut aussi la tester à la main depuis le shell :

```
% /usr/share/nagios/libexec/check_ssh localhost
Connection refused
```

Ah, mais quel est ce problème ? C'est que, pour éviter d'être noyé sous les alarmes, j'ai mis le serveur SSH sur un autre port <https://www.bortzmeyer.org/sshd-port-alternatif.html>, 8422. Il faut donc indiquer ce port au programme de test :

```
% /usr/share/nagios/libexec/check_ssh -p 8422  localhost
SSH OK - OpenSSH_6.1 (protocol 2.0) | time=0.073474s;;;0.000000;10.000000
```

Cette fois, c'est bon, le programme s'est connecté, affiche la bannière du serveur SSH et des valeurs mesurées (pour, par exemple, déclencher une alarme si le serveur répond, mais trop lentement). Mettons maintenant cela dans le fichier de configuration (les options sont précédées d'un point d'exclamation) :

```
        check_command                   check_ssh!-p 8422
```

Cette fois, tout est prêt, on peut lancer Icinga :

```
# systemctl start icinga.service
```

et vérifier dans `/var/log/icinga/icinga.log` que tout se passe bien :

```
[1350940076] Icinga 1.7.2 starting... (PID=1807)
[1350940076] Local time is Mon Oct 22 23:07:56 CEST 2012
[1350940076] LOG VERSION: 2.0
[1350940077] Finished daemonizing... (New PID=1808)
[1350940077] Event loop started...
```

Vous avez noté qu'Icinga affiche les heures en secondes depuis l'epoch. Si cela vous agace, une simple solution est documentée <https://wiki.icinga.org/display/howtos/Readable+Timestamp+in+Logs>.

Maintenant, à ce stade, on peut configurer Apache, on va voir quelque chose. Icinga fournit un exemple en `/etc/icinga/apache.example.conf`, qui marche très bien. Une petite note sur la sécurité : si votre interface Web est accessible depuis l'extérieur de votre réseau, je vous suggère fortement de rendre HTTPS obligatoire : l'interface Web ne permet pas uniquement de regarder mais aussi de lancer des commandes. Il serait donc dommage de se faire sniffer le mot de passe dans un hotspot.

Dans la configuration Apache, il doit y avoir un truc du genre `AuthUserFile /etc/icinga/htpasswd.users` Ce fichier se gère (ajouter et retirer des utilisateurs, changer leur mot de passe) avec l'utilitaire Apache `htpasswd`. Une fois que tout cela est fait, rechargez Apache, et, si votre Pi est en `pi.example.net`, vous devriez pouvoir accéder à l'interface Web d'Icinga en `https://pi.example.net/icinga`. Si cela ne marche pas, il faut évidemment regarder les journaux, notamment `/var/log/httpd/error_log`. Si cela marche, je vous laisse explorer un peu l'interface et revenir plus tard.

Bon, vous en avez assez de regarder uniquement une seule machine, `localhost` et de voir que tout est vert ? (Si tout n'est pas vert, regardez le journal d'Icinga en `/var/log/icinga/icinga.log` et corrigez.) Il faut maintenant configurer d'autres machines à surveiller. Cela n'est pas trivial. Icinga est tout sauf simple. Relisez donc la documentation `<http://docs.icinga.org/latest/en/>`. On va éditer des fichiers en `/etc/icinga/conf.d`. Pour déclarer plusieurs machines similaires, Icinga dispose de la notion de gabarit ("*template*"). On écrit un gabarit avec tout ce qui est commun à toutes les machines d'un groupe, puis on déclinera pour chacune des machines. Par exemple, un gabarit est :

```
define host{
        name                            example-host
        use                             generic-host
        check_command                   check-host-alive
        check_period                    24x7
        check_interval                  2
        max_check_attempts              3
        contact_groups                  admins
        notification_period             24x7
        notification_options            u,d,r
        register 0
}
```

On a défini ici le gabarit `example-host`, lui-même dérivé d'un autre gabarit, fourni avec Icinga, `generic-host`. On a ajouté une commande de test de la machine (`check_command`), que `generic-host` ne définissait pas. (Si vos machines restent éternellement dans l'état "*Pending*" dans le menu `Status -> Host detail`, cela peut être parce que `check_command` manque.) Ensuite, on définit les moments où les tests sont faits (24x7 indique « en permanence », les périodes sont définie dans `/etc/icinga/objects/timeper` ainsi que les intervalles entre deux tests (ici deux minutes mais attention si vous lisez la configuration Icinga de quelqu'un d'autre, on peut changer les unités). Cet intervalle dépend du « coût » de la mesure (un problème typique des mesures actives). N'écroulez pas le réseau en voulant le mesurer ! `max_check_attempts` indique au bout de combien de tests ratés on déclenche une alarme. C'est un des gros avantages des programmes comme Icinga par rapport à la « surveillance de réseau du pauvre » où on lance depuis cron un programme qui teste et lance une alarme en cas d'échec. Un tel programme est en général bien trop bavard. Sauf si un service est ultra-critique, il est prudent de ne pas lever d'alarme trop tôt (les fausses alertes répétées tuent la vigilance du NOC.) Enfin, en cas d'alarme, on notifier `admins` (groupe défini dans `/etc/icinga/objects/contacts.cfg`), quelle que soit l'heure ou le jour (« 24x7 ») et pour les événements de type d (machine "*DOWN*"), u (machine "*UNREACHABLE*") et r ("*recovery*", pour être prévenu quand ça repart). Enfin, le `register 0` indique qu'il s'agit d'un gabarit, pas d'une machine.

Une fois qu'on a un gabarit, il faut déclarer des machines. Par exemple :

```
define host{
        use                     example-host
        host_name               gandalf
        address                 gandalf.example.net
}
```

Ici, on a simplement dit que `gandalf` était de type `example-host`, et donc **hérite** de toutes les définitions ci-dessus (commande, intervalle de test, notifications, etc). Son adresse peut être donnée sous forme d'une adresse IP ou, comme ici, d'un FQDN (mais voir plus loin pour le cas où les machines ont plusieurs adresses IP, notamment IPv4 et IPv6). Mettre le nom est plus pratique mais fait dépendre la surveillance du bon fonctionnement du DNS.

Mettez quelques machines comme cela, relancez Icinga et, dans l'interface Web, vous devriez voir plusieurs machines surveillées (et tout en vert, je vous le souhaite).

Avec ce réglage, on ne vérifiera que si la machine fonctionne et est joignable (le `check-host-alive`, défini dans `/etc/icinga/objects/commands.cfg` et qui, par défaut, est un simple ping). Mais si on veut surveiller des services comme HTTP ou SMTP ? Là, cela devient plus complexe (et j'ai déjà dit que la configuration d'Icinga n'était pas simple). Il faut voir qu'Icinga fait une distinction entre **machine** ("*host*") et **service** ("*service*"). `gandalf.example.net` est une machine, qui est surveillée, et il existe de 0 à N services qui tournent sur cette machine, chacun étant surveillé séparément. Je ne suis pas sûr que cette distinction entre machines et services soit pertinente aujourd'hui : si je veux vérifier que la résolution du nom `www.bortzmeyer.org` marche bien, comme trois machines servent la zone DNS, laquelle dois-je désigner comme hébergeant le service ?

Mais passons. Voici un exemple de configuration pour une machine qui a SMTP et HTTP, dans le fichier `/etc/icinga/conf.d/abgenomica.cfg` (Icinga charge tous les fichiers de ce répertoire, je mets donc un fichier par groupe de machines) :

```
define host{
        use                     abgenomica-host
        host_name               rosalind
        address                 rosalind.abgenomica.com
}

define service{
        use                             generic-service
        hostgroup_name                  ABgenomica
        service_description             HTTP
        check_command                   check_http
        }

define service{
        use                             generic-service
        host_name                       rosalind
        service_description             SMTP
        check_command                   check_smtp
        }
```

Le service SMTP est vérifié par le "*plugin*" `check_smtp` et teste `rosalind`. Le service HTTP, lui, ne s'applique pas à une machine mais à un groupe de machines, `ABgenomica`. On peut définir un groupe ainsi :

```
define hostgroup{
        hostgroup_name  ABgenomica
        members         rosalind,james,francis,maurice
}
```

Et, avec une telle configuration, les quatre machines nommées plus haut seront testées en HTTP.

Avec cela, en rechargeant Icinga, vous devriez avoir un nouveau groupe et de nouveaux "*hosts*" et "*services*" à admirer dans l'interface Web. Il y a un million d'autres options et je n'en ai donné ici qu'une petite partie, la documentation <http://docs.icinga.org/latest/en/objectdefinitions.html> vous en montre bien plus. Une option tout à fait inutile et donc indispensable est celle qui permet d'afficher une image à côté d'une machine. C'est l'option :

```
icon_image              rosalind.jpg
```

qui permet d'indiquer un fichier image (JPEG, GIF ou PNG). Il doit être de taille 40x40 pixels et carré (autrement, il sera déformé par Icinga à l'affichage). J'utilise ImageMagick pour des manipulations simples d'images, par exemple pour a réduire à la bonne taille :

```
% convert -verbose -geometry 40x40 original.jpg machine-name.jpg
```

Le programme de test check_http <http://nagiosplugins.org/man/check_http> a plein d'options utiles. Par exemple, `-r`, `-s` et `-e` permettent de tester la présence d'une chaîne de caractères dans la page Web. En effet, certains CMS peuvent avoir des problèmes sans pour autant renvoyer un code d'erreur HTTP. Ces options testent donc non seulement le code de retour HTTP (200 pour « tout va bien ») mais également le contenu de la réponse :

```
check_command                   check_http!-s "Bienvenue sur le site Web de la compagnie Machin"
```

Il existe des programmes de test pour des tas de protocoles réseaux existants. Par exemple, pour XMPP (RFC 6120[1]), il y a check_jabber <http://nagiosplugins.org/man/check_jabber> qui s'utilise ainsi (la commande n'est pas définie par défaut dans Icinga) :

```
define command{
    command_name        check_jabber
    command_line        $USER1$/check_jabber -H $HOSTADDRESS$ $ARG1$
}

define host{
        use                     generic-host
        host_name               myxmpp
        # Cela serait mieux si check_jabber savait suivre les
# enregistrements SRV dans la DNS... On le fait pour lui.
        address                 jabber.example.net
}
```

Et si vos services sont accessibles à la fois en IPv4 et en IPv6, comme c'est le cas de tous les services sérieux aujourd'hui ? Le Pi n'a évidemment pas de problème à faire de l'IPv6 (c'est un Linux, après tout) et Icinga lui-même n'a pas besoin de connaître le protocole de test, tout passe par les programmes de test extérieurs. Pour tester la simple connectivité, on peut déclarer un test `PING6` (regardez l'option `-6` à la fin, qui sera passée au programme de test) :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6120.txt`

```
define service{
        use                             generic-service
        hostgroup_name                  ABgenomica
        service_description             PING6
        check_command                   check_ping!200.0,20%!400.0,35%!-6
        }
```

Mais tester des machines « double-pile » (IPv6 **et** IPv4) soulève des problèmes. Par exemple, si on met juste un nom de domaine dans la directive `address` et qu'on fait des tests de services (DNS, HTTP, etc), le test n'essaiera qu'une seule des adresses (laquelle ? Cela dépend du programme de test et de réglages comme ceux mis dans `/etc/gai.conf`). Il est moins pratique de tester chaque adresse explicitement mais cela permet de s'assurer que le service fonctionne bien sur toutes les adresses. Voici un exemple de double test en HTTP, le programme de test `check_http` ayant, comme la plupart des programmes de tests, des options `-4` et `-6` :

```
define service{
        use                             generic-service
        host_name                       web-oarc
        service_description             HTTP4
        check_command                   check_http!-4 -H www.dns-oarc.net
        }
define service{
        use                             generic-service
        host_name                       web-oarc
        service_description             HTTP6
        check_command                   check_http!-6 -H www.dns-oarc.net
        }
```

Pour le DNS, c'est un peu plus délicat, le programme `check_dig` des "*Nagios plugins*" n'ayant pas d'option -4 ou -6 <https://sourceforge.net/tracker/?func=detail&aid=3586320&group_id=29880&atid=397600>. Une possibilité, tenant compte du fait qu'on désigne souvent un serveur DNS par son adresse, pas par son nom, est d'indiquer les adresses IP et d'utiliser deux macros différentes, `HOSTADDRESS` et `HOSTADDRESS6` dans la commande de test :

```
# https://www.dns-oarc.net/oarc/services/odvr
define host{
        use                     oarc-host
        # BIND
        host_name               odvr1
        address                 149.20.64.20
        address6                2001:4f8:3:2bc:1::64:20
}
define host{
        use                     oarc-host
        # Unbound
        host_name               odvr2
        address                 149.20.64.21
        address6                2001:4f8:3:2bc:1::64:21
}
...
define service{
        use                             generic-service
        host_name                       odvr1,odvr2
        service_description             DNS4
        check_command                   check_dig!-H $HOSTADDRESS$ -l bortzmeyer.fr -T SOA
        }

define service{
        use                             generic-service
        host_name                       odvr1,odvr2
        service_description             DNS6
        check_command                   check_dig!-H $HOSTADDRESS6$ -l bortzmeyer.fr -T SOA
        }
```

Dans ces deux exemples, IPv4 et IPv6 étaient testés complètement séparement. On peut aussi utiliser un « méta-programme de test » qui lance les programmes de test, une fois avec v4, une fois avec v6 et ne signale un succès que si les deux fonctionnent. Un exemple d'un tel méta-programme est check_v46 <http://gitorious.org/nagios-monitoring-tools/nagios-monitoring-tools/blobs/master/check_v46>.

De même, des directives comme `parents` (qui permet d'indiquer la dépendance d'une machine envers une autre) deviennent plus délicates en double pile, où un des protocoles (IPv4 ou IPv6) peut marcher alors que l'autre est en panne.

Autre utilisation avancée, le cas où plusieurs machines rendent un même service (architecture répartie) et où on ne voudrait une alarme que si un certain nombre d'entre elles sont en panne. Par exemple, on voudrait tester la connectivité Internet (il y a des bogues pénibles <http://bugs.freeplayer.org/task/10258> avec les accès Internet grand public). D'abord, on va chercher des amers publics <https://www.bortzmeyer.org/que-pinguer.html>. Ensuite, on les définit dans la configuration :

```
define hostgroup{
        hostgroup_name  Internet
        members         cymru,ovh,free,demarcq,rezopole
}

define host{
        name                            internet-host
        use                             generic-host
        parents                         freebox
        check_command                   check-host-alive
        check_period                    24x7
        check_interval                  5
        retry_interval                  2
        max_check_attempts              3
        contact_groups                  admins
        notification_period             24x7
        notification_options            u,d,r,f
        notifications_enabled 0
        register 0
}
```

Le `notifications_enabled 0` est là pour ne pas recevoir d'alertes si une seule de ces machines est en panne (inconvénient : cela affiche du rouge dans la partie "*Notifications*" de la page d'accueil, est-ce que quelqu'un sait comment le supprimer ?). On définit ensuite chaque machine :

```
define host{
        use                     internet-host
        host_name               cymru
        address                 fap.cymru.com
}
define host{
        use                     internet-host
        host_name               rezopole
        address                 ping.rezopole.net
}
define host{
        use                     internet-host
        host_name               demarcq
        address                 ping.demarcq.net
}
define host{
        use                     internet-host
```

```
        host_name               free
        address                 test-debit.free.fr
}
define host{
        use                     internet-host
        host_name               ovh
        address                 ping.ovh.net
}
```

Puis, pour le service à surveiller, on se sert du programme `check_cluster` qui permet de dire le nombre de machines qu'on veut voir en panne avant de sonner l'alarme. On définit d'abord la commande (j'ai créé un fichier `/etc/icinga/conf.d/mycommands.cfg` pour toutes ces commandes locales) :

```
define command{
       command_name     check_cluster_host
       command_line     $USER1$/check_cluster --host -w $ARG1$ -c $ARG2$ -d $ARG3$
       }
```

Puis on définit le service lui-même :

```
define service{
        use                             generic-service
        service_description Internet connectivity
        hostgroup_name Internet
        check_command         check_cluster_host!2!3!$HOSTSTATEID:cymru$,$HOSTSTATEID:ovh$,$HOSTSTATEID:free$,$H
        notifications_enabled 1
  }
```

Ici, j'ai dit qu'il fallait prendre le résultat des cinq machines et considérer que la panne de deux d'entre elles (première option) est un avertissement, la panne de trois (deuxième option) une erreur critique à signaler. Ainsi, je ne suis pas dérangé si une seule machine a un souci (ces amers publics fonctionnent sur une base de « on fait au mieux »). Avec le menu `Reporting -> Trends`, on peut maintenant voir de jolies images :

À noter que cette image montre que tout va bien. Si on veut une image avec un problème (ici, il a été artificiellement déclenché, ce n'était pas une vraie panne de ma SamKnows <`https://www.bortzmeyer.org/samknows.html`>) :

Une autre fonction indispensable d'un logiciel de surveillance réseau est la gestion de la « joignabilité » transitive <`http://docs.icinga.org/latest/en/networkreachability.html`>. On peut dire qu'une machine ou un service a besoin d'un autre, et, dans ce cas, en cas de panne, les alertes ne seront pas envoyées si la machine ou le service dont on a besoin est lui-même en panne. Ainsi, à la maison, toutes les connexions sont tributaires de ma Freebox. Si elle est en panne, plus rien ne marche sur l'Internet mais cela ne sert à rien d'envoyer des dizaines d'alertes. J'utilise donc la directive `parents` pour indiquer cette transitivité (regardez plus haut dans la définition du gabarit `internet-host`). Voir aussi la documention sur les « dépendences <`http://docs.icinga.org/latest/en/dependencies.html`> », qui permettent des choses plus subtiles comme de contrôler si les tests ont quand même lieu en cas de panne d'un truc dont on dépend par exemple :

```
define hostdependency{
        host_name                       freebox
        dependent_host_name             rosalind
        execution_failure_criteria   d,u
        notification_failure_criteria   d,u
        }
```

ne fera pas les tests sur `rosalind` si `Freebox` est "*DOWN*" ou "*UNREACHABLE*".

Et si je ne trouve pas dans la totalité des programmes existants, le programme de test que je veux ? Un des avantages de la famille Nagios est qu'il est simple d'écrire le sien. Prenons l'exemple de la surveillance d'un serveur whois (RFC 3912). Je ne l'ai pas écrit moi-même mais j'ai simplement récupéré un script shell de The Geekery <`http://jon.netdork.net/2009/03/09/nagios-and-monitoring-whois/`>. Il appelle la commande `whois` et vérifie qu'il y a une expression rationnelle précise dans la réponse. On définit la commande Nagios ainsi :

```
define command {
        command_name    check_whois
        command_line    /usr/local/share/nagios/libexec/check_whois $HOSTADDRESS$ $ARG1$ $ARG2$
        }
```

Le script lui-même étant disponible (en ligne sur `https://www.bortzmeyer.org/files/nagios_check_whois.sh`). Ensuite, cela s'utilise comme n'importe quelle commande (`status:.*ACTIVE` étant l'expression qui doit se trouver dans le résultat) :

```
define service {
        use                     generic-service
        host_name               whois.nic.example
        service_description     WHOIS
        check_command           check_whois!bortzmeyer.example!status:.*ACTIVE
        }
```

Un exemple bien plus perfectionné de test whois permet aussi de vérifier les dates d'expiration du domaine <`http://dns.measurement-factory.com/tools/nagios-plugins/check_whois.html`>.

Et si on ne veut pas écrire en shell mais en Python ? Il y a plein de solutions pour faire du Nagios en Python mais voici la bonne <`http://packages.python.org/nagiosplugin/`> (excellent tutoriel).

Pour les petits sites qui ont trois serveurs Debian à surveiller, c'est sans doute suffisant. Mais Icinga a aussi des possibilités pour des sites plus gros, répartis et ayant des exigences supplémentaires. Par exemple, on a souvent besoin de surveiller un service sur une machine distante, sans que ce service soit accessible de l'extérieur (l'espace disque libre, par exemple, ou bien un serveur qui, pour des raisons de sécurité, n'est pas accessible de l'extérieur, ce qui est souvent le cas des SGBD). Il existe plusieurs solutions à ce problème, par exemple avoir un programme de test qui exécute une commande distante avec SSH. Mais Nagios et Icinga ont une solution suggérée, plus simple (mais pas forcément aussi sûre), nommée NRPE <`http://docs.icinga.org/latest/en/nrpe.html`> pour "*Nagios Remote Plugin Executor*". Le principe est qu'un démon NRPE va tourner sur chaque machine sur laquelle on veut faire cette surveillance rapprochée (le serveur) et qu'un programme (le client) va tourner sur la machine de surveillance, se connectant au démon et lui demandant d'exécuter les commandes de test. Dan l'exemple qui suit, le serveur est une machine Debian et le client est le Raspberry Pi sur lequel tourne Icinga. NRPE étant en paquetage chez Debian, l'installation est simple :

```
# aptitude install nagios-nrpe-server
```

À configurer, il faut se préoccuper un peu de sécurité. Après tout, il va exécuter des commandes sur une requête d'une machine pas tellement authentifiée. J'utilise Shorewall <`https://www.bortzmeyer.org/filtrage-avec-shorewall.html`> pour configurer le pare-feu. J'ajoute donc au fichier `rules` de Shorewall (dans les exemples ci-dessous, le client est en `192.0.2.84` et le port du serveur est 5666) :

```
# Nagios Remote Execution (NRPE)
ACCEPT              net:192.0.2.84  fw  TCP     5666
```

Comme j'ai aussi les TCP wrappers, j'y autorise aussi NRPE dans `/etc/hosts.allow` :

```
# NRPE is protected by Shorewall and its own configuration file of allowed_hosts
nrpe: ALL
```

Pas besoin de plus de protection puisque, en effet, NRPE a aussi la sienne. Dans `/etc/nagios/nrpe.cfg`, je mets :

```
allowed_hosts=192.0.2.84
```

Un autre point très important est à configurer : les commandes à exécuter. Pour d'évidentes raisons de sécurité, la configuration par défaut (cela peut se modifier mais c'est très dangereux) de NRPE ne permet d'exécuter que des commandes dans une liste pré-définie, configurée sur le serveur NRPE. Par exemple, je mets dans le `nrpe.cfg` :

```
command[check_disks]=/usr/lib/nagios/plugins/check_disk -w 20% -c 10% -p /dev/xvda1 -p /dev/xvdb
```

et le client NRPE (Icinga) pourra alors demander l'exécution de la commande `check_disks` (sans avoir le choix des paramètres) qui vérifiera la part d'espace disque libre sur les deux disques indiqués.

Voilà pour le serveur NRPE. Maintenant, le client. Je n'ai pas trouvé de bon paquetage NRPE sur Arch Linux (celui dans l'AUR dépend de Nagios...), j'ai donc téléchargé NRPE depuis son site `<http://sourceforge.net/projects/nagios/files/nrpe-2.x/>` puis configuré (`./configure --enable-ssl --with-nrpe-user=icinga --with-nrpe-group=icinga -- with-nagios-user=icinga --with-nagios-`) et installé. On peut alors tester que tout va bien, qu'il peut parler au serveur :

```
% /usr/local/nagios/libexec/check_nrpe -H server.example.net
NRPE v2.12
```

Et pour une des commandes définies :

```
% /usr/local/nagios/libexec/check_nrpe -H server.example.net -c check_disks
DISK OK - free space: / 604 MB (26% inode=87%); /srv/d_aetius 2823 MB (29% inode=86%);| /=1682MB;1926;2167;0;240
```

Tout est bon, on peut configurer Icinga :

```
define command{
    command_name        check_nrpe
    command_line        /usr/local/nagios/libexec/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
   }
...
define service{
        use                             generic-service
        host_name                       server.example.net
        service_description             Disk
        check_command                   check_nrpe!check_disks
        }

define service{
        use                             generic-service
        host_name                       server.example.net
        service_description             Load
        check_command                   check_nrpe!check_load
        }
```

Et Icinga va alors se connecter périodiquement à `server.example.net` pour y lancer ces commandes de vérification.

On a vu les programmes de test, maintenant les alertes ("*notifications*"). Elles se configurent avec les directives `service_notification_commands` et `host_notification_commands` qui utilisent des programmes définis dans `objects/commands.cfg`. `notify-service-by-email` et `notify-host-by-emai` sont définis comme des envois de courrier. Attention, le courrier peut dépendre lui-même de ce qu'on veut surveiller. Par exemple, si la charge système est trop élevée, Icinga peut lever une alerte... qu'on ne pourra pas transmettre, les serveurs de courrier refusant typiquement d'accepter des messages dès que la charge dépasse une certaine valeur.

Ces commandes d'utilisation du courrier sont les seules livrées avec Icinga mais on peut facilement en définir d'autres.

Ah, au fait, si vous voulez écrire des programmes qui vont lire les données stockées par Icinga, où sont-elles ? Je n'ai pas encore creusé la question, mais la réponse est « cela dépend » car Icinga permet de stocker ces données dans un fichier ou dans un SGBD comme décrit ici <`https://wiki.icinga.org/pages/viewpage.action?pageId=3637460`>. Personnellement, je ne suis pas très chaud pour faire dépendre un système de surveillance, conçu pour signaler les pannes, d'un autre composant qui peut lui même être en panne.