N°45 DÉCEMBRE 2009 JANVIER 2010

France Métro : 6,50 € / DOM : 7 €
TOM Surface : 950 XPF / POL. A :1400 XPF
CH : 13,80 CHF / BEL.PORT.CONT : 7,50 €
CAN : 13 $CAD / TUNISIE : 8,80 TND / MAR : 75 MAD

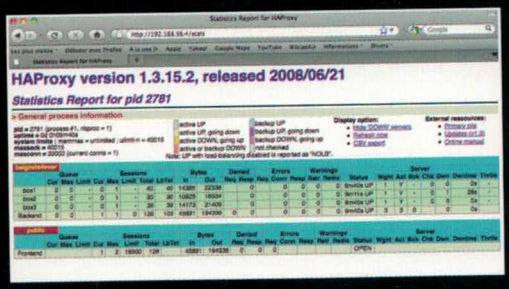
GNU LINUX MAGAZINE / FRANCE

HORS-SÉRIE

Administration et développement sur systèmes UNIX

SERVEUR MANDATAIRE

Comprenez comment utiliser un proxy TCP générique avec HAProxy et HTTP



HTTP OVER SSL/TLS

Découvrez les 12 façons d'installer un proxy pour votre serveur HTTPS

SMTP / EXIM4 / IMAP

Migrez et complétez une installation SMTP4 vers IMAP, SSL/TLS, Webmail, ...

DRBD & POSTGRESQL

Assurez une réplication de votre SGBD au niveau des disques avec DRBD

SPÉCIAL

RETOURS D'EXPÉRIENCES POUR SYSADMINS

10 SOLUTIONS CONCRÈTES

LDAP & DNS

Utilisez un backend LDAP avec votre DNS Bind 9

BZR & TRAC

Déployez un système de développement communautaire avec Bazaar, SSH et Trac

DEBIAN

Créez un système d'installation Debian automatique personnalisé

SAUVEGARDES

Gérez vos sauvegardes incrémentales avec Rdiff-backup

L 15066 - 45 H - F: 6,50 € - RD

SOMMAIRE

USER

SUPERVISION

SYSADMIN

NETADMIN

CODE(S)

ABONNEMENT

ÉDITO

Partagez et échangez !

Voilà bien quelque chose que les développeurs, les administrateurs système, les administrateurs réseau et tout autre spécialiste d'un point ou d'un autre du monde de l'informatique et des sciences techniques font depuis la nuit des temps.

L'échange et le partage sont à la source du logiciel libre, car l'information n'est pas la matière. Ainsi, entre gens de bonne compagnie, on partage ses expériences, ses découvertes et ses infortunes sans pour autant en être dépouillé.

C'était déjà vrai du temps reculé des scientifiques fous de tous bords, des alchimistes ou même des personnages impliqués dans ce qu'on appelait à l'époque les magies noires. Cela est encore vrai de nos jours. Après tout, comme le disait Arthur C. Clarke, « Toute technologie suffisamment avancée est indiscernable de la magie ». Ne vous êtes-vous jamais dit, à un moment calme de votre vie (oui, celui-là, il y a deux ans, là), que les personnes non versées dans vos domaines de prédilection pouvaient peut-être estimer que ce que vous faites relève de la magie ?

Moi même, par moment, en sachant pertinemment « qu'est-ce qui fait quoi ? » sur le net ou dans mes machines, je ne peux m'empêcher de trouver cela fantastique, surréaliste et magique. Pourtant, il ne s'agit que (sic) du fruit de notre travail à tous. Le fruit d'une connaissance partagée et poussée toujours plus loin en avant.

Avec ce numéro, nous avons souhaité partager un peu plus formellement la magie. Oh, pas des tours de passe-passe ou des apparitions miraculeuses de solutions, non. Laissons cela au, je cite, « responsable SI local en carton à peine foutu de brancher une imprimante » (l'auteur de cette si judicieuse description se reconnaîtra). Non, je vous parle de magie qu'on fabrique avec les mains et la nuit, celle qui bouge les choses et transforme votre réseau en une œuvre magnifique et harmonieuse...

Mesdames et messieurs les lecteurs, place aux artistes et... désolé pour vos nuits.

Denis Bodor

GNU/Linux Magazine est membre de l'APRIL

Membre April www.april.org

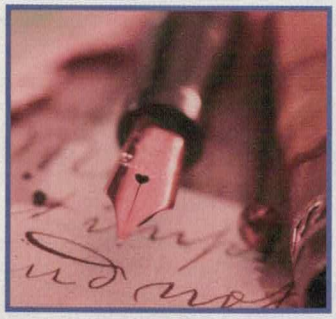
Gnu /Linux Magazine France Hors-série est édité par Diamond Editions

LES ÉDITIONS DIAMOND

B.P. 20142 - 67603 Sélestat Cedex
Tél. : 03 67 10 00 20
Fax : 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com
www.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Secrétaire de rédaction : Véronique Wilhelm
Relecture : Dominique Grosse
Conception graphique : Fabrice Krachenfels

Responsable publicité : Tél. : 03 67 10 00 26
Service abonnement : Tél. : 03 67 10 00 20
Impression : VPM Druck Allemagne
Distribution France :
(uniquement pour les dépositaires de presse)
MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
Service des ventes :
Distri-médias :
Tél. : 05 61 72 76 24
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN :** 1291-78 34
Commission paritaire : K78 976
Périodicité : Bimestrielle
Prix de vente : 6,50 €

Signer ses mails avec S/MIME



Auteur

■ Denis Bodor

L'utilisation des certificats SSL/TLS est très répandue dans le monde du web. Lorsqu'on parle, en effet, de ce type de choses, on pense naturellement à HTTPS. Mais, l'utilisation de ce standard va bien plus loin avec la configuration de VPN, l'authentification de clients (web ou autres), mais également la signature des messages électroniques. Chose qui nous intéresse ici.

Comme vous le savez peut-être, dans le monde de la signature électronique pour la messagerie, deux philosophies se partagent la vedette. Nous avons, d'une part, GnuPG/PGP et le concept de réseau de confiance (Web of trust) et, de l'autre, S/MIME et le principe de l'autorité de certification.

OBJECTIF(S)

Le but : renforcer la sécurité et la protection de sa vie privée par l'ajout d'un mécanisme de chiffrement et de signature électronique de sa correspondance par mail. Parmi les deux solutions possibles, nous choisissons de prendre la voie de l'utilisation des certificats SSL/TLS obtenus via une autorité de certification. Nous voulons comprendre les mécanismes, manipuler les certificats et clefs et, enfin, configurer notre solution de messagerie côté client, le tout le plus simplement possible.

OUTIL(S) UTILISÉ(S)

Mutt est un client de messagerie (MUA pour Mail User Agent) en mode console pour les systèmes UNIX. Il a été créé par Michael Elkins en 1995 et existe sous presque tous les UNIX en logiciel libre ou non. « All mail clients suck. This one just sucks less » est le slogan du projet, partant du principe que tous les clients mail sont « pourris », mais que Mutt l'est un peu moins. Mutt est en mode console, ce qui lui offre l'avantage d'être portable et accessible via SSH ou même une liaison série.

1 GNUPG ET LE RÉSEAU DE CONFIANCE

GnuPG et OpenPGP ont déjà largement et fréquemment rempli les colonnes de *GLMF*. Je vais cependant rappeler le principe de fonctionnement. Le point-clef de la signature électronique n'est pas simplement de s'assurer de l'intégrité des messages. Pour cela, un simple hash serait suffisant. Il faut, en effet, arriver à relier une personne (une identité) avec une signature. Vous, tout seul, générant une paire de clefs PGP et l'utilisant pour signer vos messages, ne prouvez en rien votre identité. La simple existence de la signature ne démontre qu'une seule chose : un lien entre les différents mails signés. Tous sont originaires du même utilisateur, mais rien ne prouve qui il est vraiment et qu'il existe un lien entre l'identité affichée et la personne physique ou morale en question.

Pour établir ce lien identité/clefs avec GnuPG/OpenPGP, on utilise le mécanisme de signature de clef. Ainsi sont organisés des *key signing parties* où chaque utilisateur vient avec son empreinte de clef et ses papiers d'identité. Il s'authentifie auprès des autres personnes sur place en prouvant une relation entre l'empreinte et son identité. Ces personnes signent sa clef publique. Ceci revient à apposer un sceau « moi, X, affirme que la clef Y est bien celle de monsieur Z ». Plus on a de signatures de ce type sur sa clef publique, plus on renforce le lien clef/identité. Tout un mécanisme de gestion de niveaux de confiance permet de pondérer la valeur des signatures. Ainsi, un utilisateur connu pour signer facilement et sans vraiment vérifier l'identité de la personne verra ses signatures perdre de la valeur par rapport à celles d'une utilisation plus stricte.

et Mutt

Plus le nombre de signataires et le nombre d'interconnexions sont importants, plus il devient facile d'établir un chemin entre la clef ayant servi à signer un message et une signature qu'on aura nous-même faite sur la clef d'un utilisateur. Ainsi, si quelqu'un dont j'ai vérifié l'identité a signé la clef d'une personne qui a signé la clef d'une autre qui elle-même aura servi à signer la clef utilisée pour le mail que j'ai sous les yeux, je suis *relativement* sûr de la valeur de cette signature et de l'identité de mon correspondant.

Vous l'aurez compris, tout ceci repose sur un réseau de personnes qui se connaissent (au moins de vue). Ceci pose deux problèmes. Il faut d'une part établir socialement des liens avec des personnes soucieuses de leur correspondance électronique, ce qui peut être difficile selon sa situation géographique. En effet, la majorité des gens ne comprennent pas le concept de signature électronique, ni l'intérêt de signer ou chiffrer leurs messages. « Je n'ai rien à cacher » est la réponse naïve qu'on entend le plus souvent. Il est donc nécessaire de s'entourer de personnes ayant la même sensibilité que soi dans ce domaine ou planifier de longues et pénibles heures d'explication.

Autre problème, les signatures sont datées et donc traçables. En d'autres termes, avec un fichier de clefs publiques conséquent et à jour, il devient possible d'établir des liens entre les personnes, les dates et les lieux. Bref, le réseau de confiance peut devenir un réseau de surveillance plus ou moins performant. Un peu comme il est possible de grapher les dépendances entre des paquets Debian avec **debtree**, il est possible de dessiner le réseau social d'une personne. Il existe même un outil pour cela : il s'appelle **sig2dot** (paquet **signing-party** dans Debian).

Note

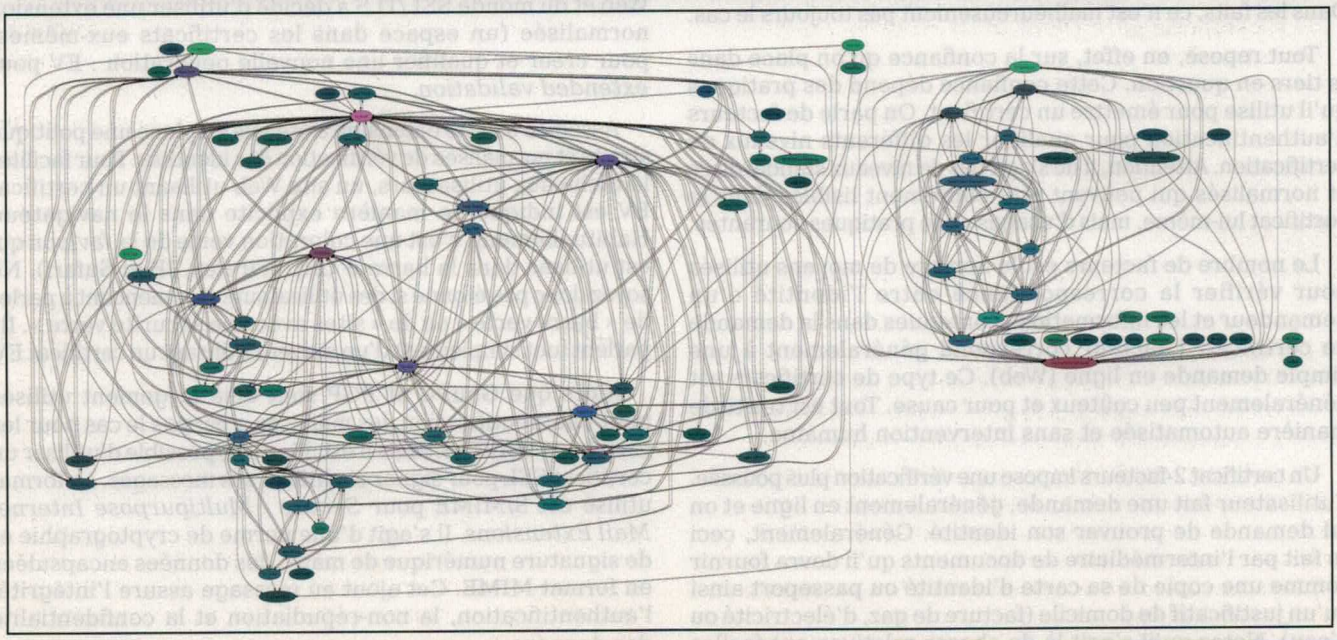
Grapher les relations entre les clefs GPG

Pour dessiner les relations entre les clefs d'un trousseau, il suffit d'installer les paquets **signing-party** et **graphviz** nous fournissant les commandes **sig2dot** et **dot**. On peut ensuite procéder comme ceci :

```
% gpg --no-options --list-sigs | sig2dot > /tmp/keyring.dot
% dot -Gcharset=latin1 -Tps /tmp/keyring.dot > /tmp/keyring.ps
```

Pour ensuite visualiser directement le fichier PostScript ou le convertir en un autre format :

```
% gv /tmp/keyring.ps
% convert /tmp/keyring.ps /tmp/keyring.jpg
% display /tmp/keyring.jpg
```



2 S/MIME ET L'AUTORITÉ DE CERTIFICATION

S/MIME, tout comme l'utilisation bien connue de SSL/TLS et des certificats serveur, repose sur un mécanisme centralisé. Une autorité supérieure, digne de confiance, appelée « autorité de certification », qui délivre les certificats (la clef publique signée par ses soins), moyennant finance. Elle est garante du lien clef/identité. Il existe plusieurs autorités de certification (des centaines), certaines plus strictes que d'autres dans leurs procédures de production de certificats. Nous en reparlerons plus loin.

Le principe de fonctionnement est très différent de celui d'un réseau de confiance. Une société commerçante vend un service. Elle est réputée de confiance, car c'est son activité principale (ou l'une de ces activités) et que celle-ci est validée par une autre haute autorité (n'est pas autorité de certification qui veut). On est donc en droit d'attendre un service de qualité.

Les tiers de confiance forment également un réseau ou plus exactement une chaîne. Nous avons des autorités de certification racines, habilitées à délivrer des certificats destinés à une utilisation directe, mais également des certificats intermédiaires. Ainsi, une autorité peut déléguer une partie de son travail à une autre entreprise. Un certificat vendu par cette dernière pour un serveur HTTP, par exemple, sera signé par la clef de l'entreprise en question. Le certificat associé à cette clef sera, quant à lui, signé par l'autorité de certification racine. On peut ainsi retracer un chemin vers l'autorité de certification principale, le tiers de confiance. Attention, ce n'est pas parce que ce chemin peut être parcouru que les applications utilisant de tels certificats le font. C'est un problème souvent mis en avant lorsqu'il s'agit de la gestion des identités et des certificats.

Si un utilisateur ou un serveur dispose d'un certificat émis par une véritable autorité (racine ou intermédiaire), on peut donc, en principe, être sûr de l'identité associée. Dans les faits, ce n'est malheureusement pas toujours le cas.

Tout repose, en effet, sur la confiance qu'on place dans le tiers en question. Cette confiance dépend des pratiques qu'il utilise pour émettre un certificat. On parle de facteurs d'authentification pour qualifier les différents niveaux de certification. Attention, il ne s'agit pas de niveaux standardisés et normalisés qui peuvent être clairement lisibles dans le certificat lui-même, mais d'usages et de pratiques courantes.

Le nombre de facteurs est le nombre de moyens utilisés pour vérifier la correspondance entre l'identité d'un demandeur et les informations contenues dans la demande de certificat. 1-facteur correspond généralement à une simple demande en ligne (Web). Ce type de certificats est généralement peu coûteux et pour cause. Tout est traité de manière automatisée et sans intervention humaine.

Un certificat 2-facteurs impose une vérification plus poussée. L'utilisateur fait une demande, généralement en ligne et on lui demande de prouver son identité. Généralement, ceci se fait par l'intermédiaire de documents qu'il devra fournir comme une copie de sa carte d'identité ou passeport ainsi qu'un justificatif de domicile (facture de gaz, d'électricité ou d'eau). Notez qu'il s'agit là de choses relativement faciles à falsifier ou à produire de toutes pièces. Un fax ou une photocopie n'est rien d'autre qu'une image en noir et blanc.

Enfin, nous avons les certificats 3-facteurs pour lesquels s'ajoute une vérification en plus des précédentes, généralement sous la forme d'un appel téléphonique. Là encore, dans l'absolu, il n'est pas difficile pour une personne mal intentionnée de tricher sur ce point.

Bien entendu, les certificats 2 ou 3-facteurs font intervenir des humains dans les procédures de vérification. Ainsi, vous pouvez acquérir pour moins de 20 euros un certificat serveur 1-facteur chez RapidSSL et un 3-facteur chez un fournisseur français comme TBS par exemple pour lequel vous devrez débourser quelques 80 euros pour un certificat ChamberSign Fiducio de classe III référencé par plusieurs ministères pour les téléprocédures.

Il faut bien comprendre que le lien entre un individu et un certificat SSL ne peut être établi que via un nombre plus important de facteurs. Un certificat 1-facteur pour un serveur HTTP ne prouve qu'une seule chose : il correspond au domaine pour lequel il a été émis. La belle affaire !

Peut-on dire pour autant que l'utilisation d'une autorité de certification est moins bonne que celle découlant d'un réseau de confiance ? Clairement non. Un certificat délivré de manière automatisée n'aura pas plus de valeur, ni moins, qu'une clef PGP possédant un chemin trop complexe vers un utilisateur de confiance. Je peux également créer artificiellement un réseau de confiance auto-signé. Cela peut prendre du temps, mais je peux finir par forger un nombre d'identités et un réseau social tout aussi crédible que factice (attention à la schizophrénie implicite).

Le lien entre une clef publique et une identité est primordial. Voilà pourquoi le CA/Browser Forum, un groupement (consortium) des plus importants acteurs du Web et du monde SSL/TLS a décidé d'utiliser une extension normalisée (un espace dans les certificats eux-mêmes) pour créer et qualifier une nouvelle génération : EV pour *extended validation*.

Associée à cette extension a été mis en place une politique stricte et normalisée de vérification des identités. Pour faciliter la tâche aux utilisateurs, un site Web utilisant un certificat EV est indiqué de manière explicite dans le navigateur. Habituellement, c'est une coloration verte de la *favicon* qui est utilisée dans la barre d'URL (Firefox, IE et Safari). Ne soyez donc pas étonné si des utilisateurs commencent à parler de « barre verte » ou de « sites avec une sécurité verte ». Ils parlent tout simplement d'un serveur utilisant un certificat EV.

Alors que GnuPG et PGP sont déjà largement utilisés pour l'authentification des mails, ce n'est pas le cas pour les certificats SSL/TLS. Mais il est, bien sûr, possible d'utiliser un certificat SSL pour signer et chiffrer ses messages. Le format utilisé est S/MIME pour *Secure / Multipurpose Internet Mail Extensions*. Il s'agit d'une norme de cryptographie et de signature numérique de mails, des données encapsulées en format MIME. Cet ajout au message assure l'intégrité, l'authentification, la non-répudiation et la confidentialité des données.

3 EN PRATIQUE, OBTENTION ET INSTALLATION DU CERTIFICAT

3.1 Obtention du certificat

La manière la plus simple dans un premier temps est d'utiliser un certificat personnel gratuit. Bon nombre d'autorités de certification permettent d'obtenir ce type de choses. Il s'agit, avant tout, d'un produit d'appel permettant ensuite de proposer des offres plus complètes ou plus orientées serveur.

Ces certificats sont générés sans autres vérifications que celle de l'adresse email utilisée. Il n'y a donc pas de lien direct entre une identité et un certificat. Certaines autorités proposent des certificats 2 ou 3-facteurs payants mais, tout comme avec les certificats serveur, encore faut-il que votre correspondant sache faire la différence entre ce type de produits et les certificats gratuits de bien moindre intérêt. A quoi bon disposer d'un certificat 3-facteurs si quelqu'un d'autre a trouvé le moyen d'en obtenir un, 1-facteur, avec une adresse mail qui ressemble à la vôtre. Après tout **pierre.dupond@yahoo.fr** n'est pas une adresse plus « personnelle » que **pierre.dupont@gmail.fr**. Si votre correspondant ne voit pas l'intérêt d'un certificat 3-facteurs, pour lui, l'une et l'autre adresses pourraient bien être votre adresse.

Bien entendu, l'utilisation d'un certificat personnel avec des canaux de communication publics (liste de diffusion par exemple) tendra à renforcer la relation entre une personne et un certificat. Plus exactement, il permettra de lier entre eux les messages émis et signés par la même clef et donc d'associer un profil au certificat, mais pas nécessairement à un patronyme (tout comme pour une clef PGP/GPG en somme).

La plupart des pages Web permettant d'obtenir un certificat personnel sont entièrement automatisées. C'est votre navigateur qui génèrera la demande de certificat et la proposera au site (voir article sur EJBCA, la PKI SSL dans *GLMF 121*). A son tour, le serveur va manipuler les données et signer la demande avant de renvoyer le certificat au navigateur. Aucune clef secrète ne circule entre navigateur et serveur. Attention toutefois, certains services Web proposent de générer la demande et le certificat. Comprenez bien que, alors, l'autorité de certification aura connaissance de votre clef privée.

A titre d'exemple, voici comment cela peut se passer avec un certification 1-facteur.

- Vous pointez votre navigateur sur la page Web du service proposé et passez commande. Votre navigateur génère une requête de certificat et l'envoie au serveur.
- Vous êtes invité à confirmer votre adresse mail par la méthode classique (envoi d'un mail et clic sur un lien).
- En suivant le lien, vous récupérez les informations sur l'état de votre requête. Votre adresse mail est confirmée et généralement le certificat est immédiatement disponible.
- Vous pouvez alors récupérer le certificat en le « téléchargeant ». En réalité, celui-ci va s'installer automatiquement dans votre navigateur.

3.2 Exportation du certificat

A ce stade, vous disposez d'un certificat signé par l'autorité de certification ainsi que la clef privée associée. Cependant, pour en faire usage avec une vaste gamme d'applications, il est nécessaire d'enregistrer le duo (certificat et clef) dans un format plus polyvalent. C'est généralement le format PEM (*Privacy Enhanced Mail*) qui n'est autre qu'un format DER (*Distinguished Encoding Rules*) encodé en Base64 et placé entre des lignes/bornes `-----BEGIN CERTIFICATE-----` et `-----END CERTIFICATE-----`. Un autre format qu'on retrouve parfois est PKCS#12.

Les PKCS pour *Public Key Cryptographic Standards* ou, en français, « standards de cryptographie à clé publique » sont un ensemble de spécifications conçues par RSA Security, non un standard. Cette société n'étant pas un organisme de normalisation, PKCS ne peut être considéré comme une norme. Cependant, les PKCS étant largement adoptés, une partie des spécifications ont été reformulées sous forme de RFC (*Requests For Comments*, tous les standards d'Internet sont des RFC) par l'IETF (*Internet Engineering Task Force*, groupe informel, international et ouvert, participant à l'élaboration de standards).

PKCS#12, ou PKCS 12, définit un format de fichiers permettant de stocker la clef privée et le certificat de clef publique correspondant, tout en les protégeant par un mot de passe. Le navigateur Firefox ne sait exporter les clefs qu'au format PKCS#12. Vous devrez donc passer par le menu ***Édition***, ***Préférences***, puis cliquez sur l'icône ***Avancé*** et l'onglet ***Chiffrement***. Là, en cliquant sur ***Afficher les certificats***, vous aurez accès à l'ensemble des données de chiffrement disponibles pour Firefox : certificats des autorités de certifications connues, certificats des serveurs Web HTTPS visités, certificats personnels, etc.

Comme le montre la capture sur cette page, la direction générale des impôts utilise également cette technique pour les télédéclarations. En effet, on peut considérer le fait de renseigner un nom, un numéro de télédéclarant et un total de référence de l'année précédente comme une procédure de délivrement de certificat 3-facteurs.

Pour exporter un ou plusieurs certificats et leur clef privée sous forme de fichier PKCS#12 (extension `.p12`), il vous suffit de sélectionner la ligne en question et de cliquer sur ***Sauvegarder***. Une boîte de dialogue apparaît, exigeant un mot de passe pour protéger les données après vous avoir demandé de spécifier un nom de fichier. Le niveau de sécurité, découlant de la complexité du mot de passe, est indiqué sous la forme d'une barre de progression.

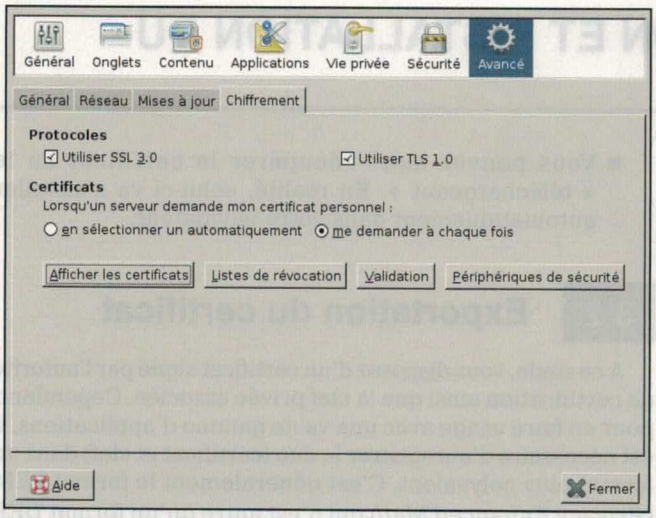


Fig. 1 : Les préférences de Firefox donnent accès à toutes les options de chiffrement et de gestion de certificats, mais également à la gestion d'éventuels périphériques d'authentification comme des cartes à puce ou des dongles USB.

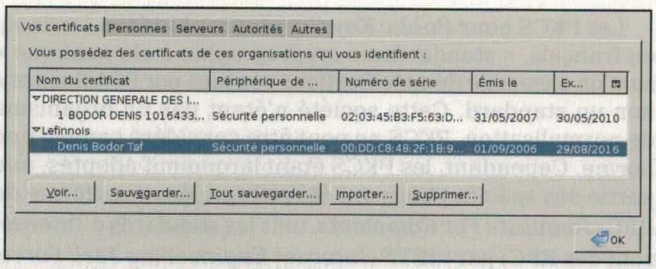


Fig. 2 : La gestion des certificats dans le navigateur vous permet de gérer les certificats des autorités de certifications connues, certificats des serveurs Web HTTPS visité, certificat personnels, etc.

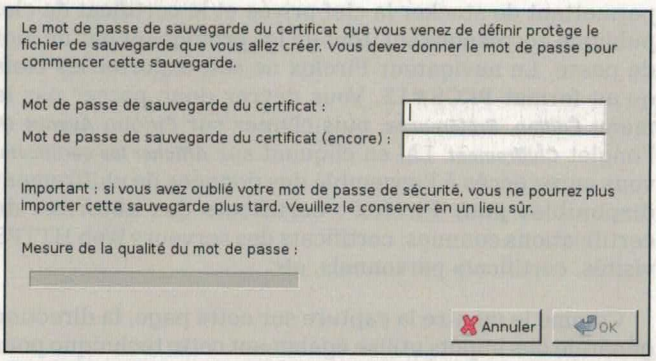


Fig. 3 : L'exportation d'un couple certificats plus clef privée nécessitera la saisie d'un mot de passe comme l'exige les spécifications du format utilisé, le PCKS#12.

Avant de convertir le fichier, il est possible d'afficher un certain nombre d'informations sur les données qu'il contient avec :

```
% openssl pkcs12 -in export.p12 -info -noout
Enter Import Password:
MAC Iteration 1
MAC verified OK
PKCS7 Data
Shrouded Keybag: pbeWithSHA1And3-KeyTripleDES-CBC, Iteration 1
PKCS7 Encrypted data: pbeWithSHA1And40BitRC2-CBC, Iteration 1
Certificate bag
```

Nous procédons ensuite à la création du fichier au format PEM :

```
% openssl pkcs12 -in export.p12 -out certetclef.pem -des
Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Nous obtenons ainsi **certetclef.pem** qui contient à la fois le certificat et la clef privée. En réalité, il s'agit de deux contenus réunis en un seul fichier. Ceci est, en effet, possible en raison du format PEM lui-même utilisant des délimiteurs comme précisé précédemment (**BEGIN RSA PRIVATE KEY**, et **BEGIN CERTIFICATE** par exemple). Un simple affichage du contenu de **certetclef.pem** avec **cat** ou **less** vous le démontrera.

Notez qu'un mot de passe vous est demandé trois fois de suite. La première permet la lecture des données du fichier PKCS#12 et les deux suivantes, la protection du fichier PEM (définition et confirmation du mot de passe).

Enfin, pour conclure cette partie, sachez qu'il est possible d'afficher les informations de certification en partant du contenu du fichier PEM avec la commande **x509** de l'outil **openssl** : **openssl x509 -in certetclef.pem -text**. Vous retrouverez alors toutes les informations que vous avez pu voir s'afficher dans votre navigateur.

Note

Sauvegarde de vos clefs et certificats

Si, comme moi, vous avez opté pour la télédéclaration et le paiement fiscal en ligne (impôts sur le revenu, taxe d'habitation, etc.), vous avez obtenu un certificat de la part de la direction des impôts. Ce certificat a une validité à durée limitée et il vous faudra le renouveler à intervalles réguliers. Il serait toutefois dommage de devoir refaire une demande suite à un crash disque ou à l'effacement de votre **~/.mozilla**.

Bien entendu, il est toujours possible de sauvegarder vos données Firefox avec le reste de vos disques. L'idée, ici, serait plutôt de détacher les données de l'application, de les stocker en PEM et ensuite dans un format plus polyvalent et adaptable. Nous utilisons donc l'outil **openssl** comme ceci :

```
% openssl rsa  -in certetclef.pem | a2ps -B -1 -o clef.ps
% ps2pdf clef.ps
% openssl x509 -in certetclef.pem | a2ps -B -1 -o cert.ps
% ps2pdf cert.ps
```

Vous obtenez alors deux fichiers PDF qu'il vous est possible d'imprimer et mettre en sécurité. Comme vous le savez, le bon vieux papier a une capacité de rétention de l'information largement supérieure à celle d'un support informatique. Maintenant, vous pouvez toujours graver ces données encodées Base64 sur du marbre, si vous avez du temps devant vous...

4 INTÉGRATION DANS LE CLIENT MAIL MUTT

Nous pouvons à présent intégrer notre certificat et sa clef privée dans notre configuration du MUA Mutt. Nous considérons ici que vous disposez déjà d'une configuration viable et fonctionnelle du client. Sa configuration POP3, IMAP ou mbox/Maildir local sort totalement du cadre de cet article.

Mutt, selon la version binaire dont vous disposez, saura ou non manipuler directement des données chiffrées et/ou signées en S/MIME. Pour le savoir, il vous suffit d'utiliser l'option **-v**. Si dans la sortie (dense) qui s'affiche il est fait mention de **+CRYPT_BACKEND_CLASSIC_SMIME**, c'est signe que votre Mutt a été compilé avec les options adéquates.

Pour l'intégration, nous commençons par créer une arborescence de gestion et de stockage des certificats. Ceci peut se faire automatiquement via la commande **smime_keys init** depuis votre répertoire personnel (cette commande est disponible avec le paquet **mutt** sous Debian GNU/Linux).

Il ne vous reste plus alors qu'à ajouter votre couple certificat/clef dans le répertoire qui fera office de base de données avec :

```
% smime_keys add_pem certetclef.pem
You may assign a label to this key, so you don't have to remember
the key ID. This has to be _one_ word (no whitespaces).
Enter label: me
added certificate: /home/denis/.smime/certificates/9ec3a561.0.
certificate 4b8656c1.0 (me) for lefinnois@lefinnois.net added.

==> about to verify certificate of lefinnois@lefinnois.net
/home/denis/.smime/certificates/4b8656c1.0: OK
added private key: /home/denis/.smime/keys/4b8656c1.0 for lefinnois@
lefinnois.net
```

Comme vous pouvez le constater, on vous demandera de spécifier un nom (label) pour cet ajout afin de faciliter la sélection si vous installez plusieurs certificats. Une fois ce nom unique validé, l'ensemble est mis à jour et **smime_keys** vous donne toutes les informations nécessaires. Notre certificat et notre clef privée sont référencés sous **4b8656c1.0**. On retrouve d'ailleurs la correspondance label/référence dans les fichiers **.index** de **.smime/** :

```
% cat .smime/keys/.index
lefinnois@lefinnois.net 4b8656c1.0 me
% cat .smime/certificates/.index
lefinnois@lefinnois.net 4b8656c1.0 me 9ec3a561.0 t
```

On remarquera que cette dernière ligne référence également un autre certificat. Il s'agit de celui d'AC ayant signé notre requête (ici TBS). On pourra facilement s'en assurer en utilisant la commande **openssl x509 -text -noout -in .smime/certificates/9ec3a561.0**

Dernier point de configuration, nous devons nous pencher sur notre fichier **.muttrc** afin de préciser au MUA la manière dont il doit utiliser ces nouvelles informations. Nous ajoutons à cet effet les lignes suivantes :

```
set smime_is_default
set crypt_autosign = yes
set crypt_replyencrypt = yes
set crypt_replysign = yes
set crypt_replysignencrypted = yes
set crypt_verify_sig = yes
set smime_default_key="4b8656c1.0"
```

Nous définissons ainsi une signature automatique, une réponse chiffrée à un message chiffré, idem pour la signature, idem pour les deux et, enfin, une vérification automatique des signatures sur les messages en réception. **smime_default_key** nous permet de préciser la clef par défaut à utiliser pour signer nos messages.

Dans l'interface de Mutt, à l'étape de confirmation d'envoi, nous trouverons donc :

```
    From: Denis Bodor <moi@mail.com>
      To: lefinnois@lefinnois.net
      Cc:
     Bcc:
Subject: un sujet idiot
Reply-To:
     Fcc: ~/sent
     Mix: <no chain defined>
  S/MIME: Signer
signer en tant que : 4b8656c1.0
```

Il nous sera alors possible d'utiliser la touche S (majuscule) nous permettant de choisir un autre comportement que la simple signature :

```
(c)hiffr S/MIME, (s)ign, ch. (a)vec, s. (e)n tant que, les (d)eux, clai(r)
```

Le message n'est pas très clair, mais cela signifie :

- **c** : chiffrer ;
- **s** : signer ;
- **a** : chiffrer avec la clef... ;
- **e** : signer en tant que... ;
- **d** : signer et chiffrer ;
- **r** : message en clair.

CONCLUSION

Nous voici arrivés au terme de cet article. Les explications présentées ici découlent d'une mise en œuvre de Mutt avec support S/MIME, mais sont parfaitement applicables à d'autres MUA. Nous n'avons pas exploré outre mesure le cas de la réutilisation des certificats obtenus par des moyens annexes (direction des Impôts ou certificats autosignés). En effet, ceux-ci représentent clairement des « détournements », voire des *hacks*, par rapport à l'obtention ou l'achat d'un véritable certificat auprès d'une AC reconnu. Ceci pourrait être laissé en guise d'exercice au lecteur curieux ou faire l'objet d'une annexe à cet article dans un prochain numéro. ■

Centreon/Nagios : le couple

Le duo Centreon/Nagios est devenu au fil du temps une solution pleinement viable pour la supervision qui rivalise, en termes d'ergonomie, avec des solutions comme Zabbix.

Auteur

- Jean Gabès

1 NAGIOS RESTE LA BASE DE LA SUPERVISION

Nagios est la référence en matière de logiciel de supervision *open source*. Développé depuis plus de 10 ans, il possède une communauté sans égale et des possibilités très étoffées. Il a cependant une réputation, méritée, d'être relativement complexe à prendre en main, notamment la partie configuration. Nous y reviendrons un peu plus tard. Pour l'instant, regardons un peu les principales caractéristiques de notre champion afin de mieux comprendre l'intérêt de Centreon.

OBJECTIF(S)

Nous voulons ici mettre en place une solution de supervision. Comme le dit Wikipédia, « La supervision est la surveillance du bon fonctionnement d'un système ou d'une activité ». Voilà qui est bien vague. En d'autres termes, il s'agit de confier la surveillance de l'ensemble d'un système ou d'un réseau à un service. En cas de défaillance, celui-ci pourra réagir, simplement prévenir un responsable ou déclencher une alerte à grande échelle. Une solution de supervision permet également d'obtenir une vue globale d'une infrastructure complexe.

OUTIL(S) UTILISÉ(S)

Nagios, anciennement appelé Netsaint, est une application permettant la surveillance système et réseau. Elle surveille les hôtes et services spécifiés, alertant lorsque les systèmes vont mal et quand ils vont mieux. C'est un logiciel libre sous licence GPL, entièrement modulaire. Cette modularité est souvent vue comme une complexité. Pour corriger cette perception, nous avons Centreon qui fournit une interface simplifiée très accessible. Les sysadmins préférant (par moment) l'aspect technique ont toujours accès aux informations de Nagios.

1.1 Les grands principes de Nagios

Pour ceux qui n'ont pas lu les précédents articles sur ce sujet, Nagios est un ordonnanceur de vérifications. Il ne sait faire que l'ordonnancement, les tests en eux-mêmes sont laissés à la charge de *plugins*. L'auteur est parti du principe qu'il lui était impossible de prévoir toutes les vérifications possibles dans un seul outil. Il a donc choisi de n'en faire aucun et de se concentrer sur l'ordonnancement. Il respecte ainsi le principe *KISS* d'Unix qui consiste à ne faire qu'une chose, mais à la faire bien.

L'administrateur doit fournir diverses informations à Nagios pour que celui-ci puisse effectuer son travail. Outre les options propres au *daemon*, il doit lui spécifier les éléments à superviser. Ce sont les hôtes (par exemple les serveurs). Il doit également définir les services à surveiller sur ces hôtes. Il faut les voir comme un point de supervision et non pas comme un simple service Unix. Par exemple, la supervision de la charge CPU est un service, tout comme l'est la connexion au serveur web hébergé par ce même serveur.

L'administrateur a également besoin de définir qui Nagios doit prévenir en cas de souci. Ce sont les contacts. Ils sont avertis par le biais de plugins, qui, pour la plupart, se contentent d'envoyer un email, mais peuvent faire bien d'autres choses comme mettre à jour un flux RSS ou bien afficher un message sur le clavier de l'administrateur si ce dernier possède un petit écran LCD.

gagnant de la supervision

1.2 Les plugins de vérification

Nous avons évoqué le fait que Nagios sous-traite la vérification à des plugins. Ce sont des programmes que va appeler Nagios. Ce dernier en attend deux choses :

- une ligne sur la sortie standard ;
- un code retour.

La ligne est présente essentiellement pour retourner un texte à l'administrateur sur l'alerte. Nous verrons un peu plus loin qu'elle peut faire un peu plus que cela. Le plus important pour Nagios est le code retour du programme. Suivant la valeur retournée, Nagios va connaître le résultat de la vérification. Les différents éléments au sein de Nagios peuvent avoir différents statuts qui seront directement liés au retour des vérifications. Ces états sont présentés dans les deux tableaux suivants :

Pour les hôtes :

État	Signification	Code retour
UP	Hôte disponible	0
DOWN	Hôte indisponible	1 ou 2

Pour les services :

État	Signification	Code retour
OK	Tout va bien	0
WARNING	Quelque chose n'est pas normal	1
CRITICAL	Erreur critique	2
UNKNOWN	État inconnu	3

Voici l'exemple d'un plugin effectuant la vérification de l'existence du fichier **/tmp/monfichier.txt**.

```
#!/bin/bash
if [ -f /tmp/monfichier.txt ]; then
  echo "Le fichier existe bien."
  exit 0
else
  echo "Le fichier n'existe pas."
  exit 2
fi
```

Il pourra être utilisé pour un service alertant l'administrateur si le fichier n'existe pas.

1.3 La configuration de Nagios

1.3.1 Un long apprentissage

Point souvent décrié par les administrateurs lorsqu'ils commencent à mettre en place Nagios, la configuration est pourtant l'une de ses plus grandes forces. Le fichier de configuration principal est **nagios.cfg**. Il est situé par défaut dans **/usr/local/nagios/etc**. Il possède un nombre important de propriétés permettant de régler finement l'ordonnancement général des vérifications et des alertes que nous n'étudierons pas ici. Il permet également d'appeler d'autres fichiers de configuration utilisés pour la définition des éléments à surveiller grâce à l'option **cfg_file**.

La structure standard de Nagios est d'avoir un fichier de configuration par type d'élément (hôte, service, etc.). Il est aussi possible de tout mettre dans le même fichier, mais il est dur de s'y retrouver par la suite. Au final, les éléments à configurer sont les suivants :

- les contacts ;
- les hôtes ;
- les services ;
- les commandes de supervision et de notification ;
- les périodes de temps qui permettent d'avoir une supervision très fine sur les plages de temps où sont surveillées les éléments ou bien où les administrateurs sont alertés.

1.3.2 La structure de la configuration des éléments

La configuration de tous ces éléments suit ce modèle :

```
define type{
  parametre1=valeur1
  parametre2=valeur2
}
```

Dans le cas d'un hôte par exemple, la configuration minimale est la suivante :

```
define host{
  host_name   serveur-web
  alias       Serveur Web
  address     192.168.0.1
  check_command   check-host-alive
  check_interval  5
  retry_interval  1
  max_check_attempts  3
  check_period    24x7
  contact_groups   admins-web
  notification_interval   120
  notification_period    24x7
  notification_options   d,u,r
}
```

Sans entrer dans les détails, elle définit un hôte nommé **serveur-web** qui a pour adresse **192.168.0.1** et est surveillé avec la commande **check-host-alive** (en général un simple

ping) toutes les 5 minutes. En cas de soucis répétés (3 retours en erreur), il lève une notification toutes les 120 minutes vers les administrateurs du groupe **admins-web**, et ce, 24 heures sur 24.

1.3.3 Une configuration complète

Autant le dire de suite : cette configuration minimale n'est pas triviale et peut effrayer bon nombre d'administrateurs qui souhaitent juste être informés lorsque leur serveur n'est plus disponible. Ici, nous n'avons configuré qu'un seul serveur, mais il faut faire de même pour tous les autres. Sans utiliser de techniques de configuration avancées, pour une centaine de serveurs, ceci fait plus de 1400 lignes. Et nous n'avons pas encore configuré les services qui sont surveillés sur ces serveurs ! De plus, nous nous sommes limités aux options obligatoires, mais de nombreuses autres existent...

1.4 La gestion de la métrologie

1.4.1 Nagios et la métrologie

Activité souvent liée à la supervision, la métrologie consiste à récupérer et « grapher » des données numériques comme l'espace disque ou la charge CPU d'une machine. Il est intéressant de remarquer que les indicateurs lus sont les mêmes que ceux utilisés pour bon nombre de plugins de supervision. Prenons l'exemple de la charge CPU : un plugin récupère cette donnée et la compare avec un seuil prédéfini et lève, si besoin est, une alerte. Dans le cadre de la métrologie, cette valeur est relevée et mise en base dans le but d'être intégrée dans un graphique. L'opération de récupération est identique dans les deux cas. Partant de ce constat, l'auteur de Nagios a permis aux sondes de supervision de renvoyer des données de métrologie.

Rappelons que Nagios est un ordonnanceur dédié à la supervision, respectant le principe KISS d'Unix. Son rôle premier n'est pas de gérer la masse des données de métrologie et encore moins de générer de jolis graphiques pour les administrateurs. Il n'est qu'un vecteur pour la récupération des données. Il les place simplement dans un fichier plat qui sera lu par une application tierce qui s'occupera du reste.

1.4.2 La remontée des données de métrologie

Pour pouvoir renvoyer des données de performances, les sondes doivent respecter un format particulier. Nous avons vu que les plugins de supervision utilisent le code retour pour indiquer l'état de l'élément supervisé et la sortie standard pour indiquer sommairement à l'administrateur quel est le problème. C'est cette dernière que nous allons utiliser ici. Cette ligne va pouvoir être coupée en deux morceaux indépendants par le caractère « | ».

Ce qu'il y a avant ce caractère est le texte retourné à l'administrateur, ce qu'il y a après concerne les données de métrologie. S'il est absent, Nagios considère qu'il n'y a tout simplement pas de données de performance retournées. Le format de ces données est fort simple :

```
'label'=valeur[unité]
```

Par exemple, dans le cas de notre sonde pour l'utilisation du CPU :

```
CPU=90%
```

Un autre format plus complet est possible :

```
'label'=valeur[unité];[warn];[crit];[min];[max]
```

Ceci donne par exemple, avec notre sonde, et un seuil d'avertissement à 80% et un d'alerte à 95% :

```
CPU=90%;80;95;0;100
```

Bien sûr, une sonde peut renvoyer plusieurs données sur la même ligne. Il suffit de les séparer par une espace. Par exemple, dans le cas d'une supervision d'un processeur *dual core*, nous pouvons obtenir :

```
Critical : CPU is too high | CPU1=99%;80;95;0;100 CPU2=20%;80;95;0;100
```

Une fois que la sonde a retourné les données à Nagios, ce dernier extrait les données de performances et les place dans un fichier choisi par l'administrateur, généralement **/usr/local/nagios/var/service-perfdata**. On peut voir ci-dessous la ligne ajoutée par Nagios suite au lancement de la sonde pour le service Cpu sur la machine serveur-web :

```
1250686984      serveur-web      Cpu     CPU is too high CRITICAL
CPU1=99%;80;95;0;100 CPU2=20%;80;95;0;100
```

Le premier nombre est le temps au format Unix où la donnée a été exportée.

1.4.3 Les outils de gestion des données de performance

Au fil du temps, plusieurs outils permettant de récupérer ces données ont vu le jour. Certains étaient très performants, mais difficiles à configurer comme PNP4Nagios ou bien très faciles à utiliser, mais nécessitant des ressources indécentes comme Perfparse. Il était relativement difficile pour les administrateurs de choisir une solution convenable.

1.5 L'export des données

Nagios alerte les administrateurs par le biais des notifications, mais il est également utile d'exporter les données de disponibilité pour que d'autres programmes puissent les mettre en forme et proposer des consoles de supervision aux administrateurs. Deux méthodes se sont succédées au fil du temps.

1.5.1 L'export en fichier plat

La première version de l'export s'est faite par un simple fichier plat. Habituellement situé dans le fichier **/usr/**

`local/nagios/var/status.dat`, il est généré régulièrement, généralement toutes les 10 secondes. Il contient toutes les données nécessaires pour les applications tierces.

Ces informations sont pratiques, mais également nombreuses. La taille de ce fichier n'est pas anodin pour les grandes configurations. Pour un Nagios ayant 600 hôtes et 7000 services, il pèse près de 10 Mo. Deux problèmes se posent alors :

- le temps des écritures/lectures sur disque ;
- le temps de traitement CPU.

Le premier problème n'en est pas vraiment un au final. Ecrire 10 Mo toutes les 10 secondes et le lire encore plus souvent n'est pas pénalisant en plaçant ce fichier sur un *filesystem* en mémoire comme `/dev/shm`. Ce fichier étant temporaire et non utilisé pour les relances de Nagios, même une coupure du système, et donc une perte du fichier, n'est pas importante.

Le second problème est lui bien plus pénalisant. Pour obtenir une information, les outils sont obligés de lire et traiter le fichier en entier. Par exemple, dans le cas d'une interface web qui affiche les erreurs, chaque rechargement de page implique une analyse complète du fichier. Celle-ci peut prendre facilement 2 à 3 secondes pour des fichiers importants. En plus de générer des délais d'affichage très longs pour les sites en questions, il ne faut pas oublier que chaque administrateur va avoir la console d'ouverte et rafraîchie très régulièrement. On arrive donc à des situations où 1 ou 2 processeurs sont utilisés en continu juste pour l'affichage des consoles ! Il ne reste plus beaucoup de ressources à Nagios pour faire son travail.

1.5.2 L'export en base de données

C'est pour cette raison qu'une fonctionnalité est apparue : l'export des informations en base de données. Elle repose sur l'utilisation de l'Event Broker de Nagios. Ce dernier permet à des modules d'être chargés en mémoire par Nagios et d'obtenir des informations de ce dernier. Le module qui nous intéresse ici se nomme `ndomod`. Il fait partie de NDOUtils. Ce module exporte les informations de Nagios vers une *socket* (de type *file socket* ou bien réseau). Celle-ci est ouverte par un daemon nommé Ndo2db qui est chargé de récupérer les données et de les placer en base. Actuellement, seules les bases MySQL et PostgreSQL sont supportées.

L'intérêt d'avoir un daemon déporté réside dans le fait que plusieurs Nagios peuvent lui communiquer des informations à stocker ensuite dans une seule et même base de données. Nous verrons cela plus en détail par la suite.

L'autre grand intérêt de la base de données se situe dans les recherches d'informations que vont effectuer les outils. Il est bien plus simple et efficace de passer par des requêtes SQL pour rechercher, par exemple, l'état d'un hôte particulier que de traiter le fichier `status.dat` en entier. Les outils actuels utilisent tous l'export en base pour obtenir les informations.

1.6 Nagios en résumé

Faisons un petit point rapide sur ce qu'est Nagios en résumant son fonctionnement global sur le diagramme suivant :

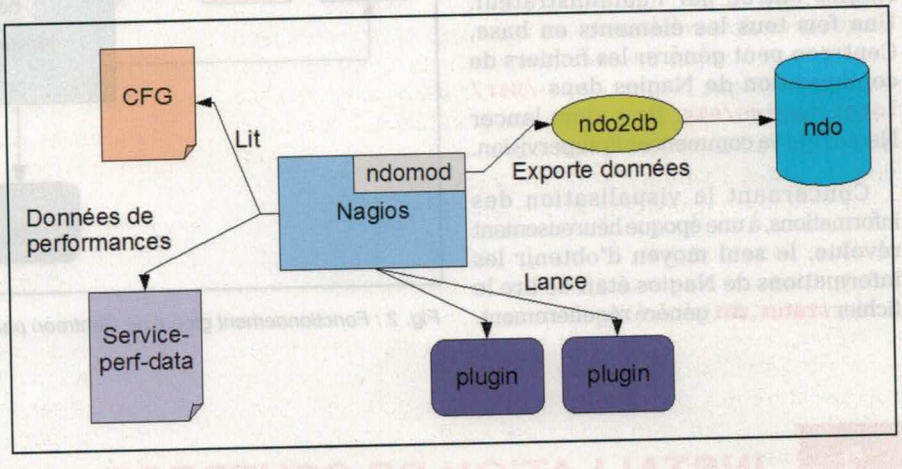


Fig. 1 : Fonctionnement général de Nagios

2 INTÉRÊTS ET HISTOIRE DE CENTREON

2.1 Un outil répondant à un besoin fort

Centreon a vu le jour sous le nom d'Oreon en 2003 sous l'impulsion de deux développeurs français. Ils sont partis d'un constat simple : Nagios est complexe à configurer. Partie comme une interface Web ayant ses données en base de données, Oreon a fortement évolué au fil du temps pour s'occuper de plus en plus d'aspects de la supervision et forme aujourd'hui avec Nagios l'une des solutions les plus complètes en matière de supervision et de métrologie.

Une fois la partie configuration implémentée, ce fut le tour de la partie visualisation. Celle-ci permet d'avoir une console de supervision avec les alertes en temps-réel. Nagios fournit déjà une telle interface, mais elle est codée en C et n'est pas très ergonomique. L'interface de Centreon permet de filtrer très efficacement les informations comme nous le verrons par la suite.

Comme dit précédemment, la gestion de la métrologie était un gros problème pour les administrateurs. Centreon gère cela avec une solution nommée **CentStorage** qui récupère les données du fichier `service-perf-data`. D'autres problématiques auxquelles les administrateurs étaient confrontées ont été gérées par la suite comme la gestion des *traps* SNMP ou bien la mise en place des environnements distribués, comme nous le verrons par la suite.

2.2 Le fonctionnement global de Centreon

Centreon est composé d'une interface Web en PHP pour la configuration et la visualisation et de deux daemons en Perl. L'interface fait appel à une base de données MySQL. C'est dans cette dernière que va se situer la configuration Nagios entrée par l'administrateur. Une fois tous les éléments en base, Centreon peut générer les fichiers de configuration de Nagios dans `/usr/local/nagios/etc`. Il va alors lancer Nagios qui va commencer la supervision.

Concernant la visualisation des informations, à une époque heureusement révolue, le seul moyen d'obtenir les informations de Nagios était de lire le fichier `status.dat` généré régulièrement. Comme dit précédemment, le coût de ce traitement était très important sur les configurations importantes. Lors de l'arrivée de l'export en base de données, Centreon l'a utilisé pour alléger son fonctionnement. Une fois Nagios lancé, il exporte les données dans sa propre base de données. Celle-ci est lue par Centreon qui affiche les informations pour ses utilisateurs.

Le fonctionnement global de Centreon est illustré sur le diagramme suivant. Il est très général, et certains points seront analysés plus finement par la suite.

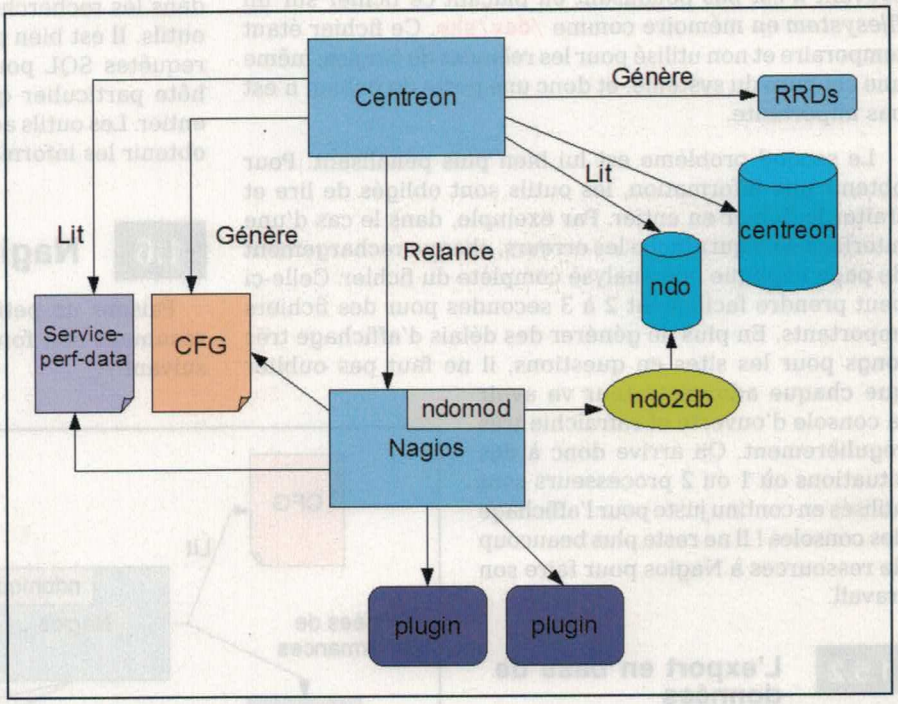


Fig. 2 : Fonctionnement global de Centreon par rapport à Nagios

3 INSTALLATION DE CENTREON

3.1 Une installation tout en un

La mise en place de Centreon n'est pas nécessairement une partie de plaisir. C'est pour cela que quelques courageux ont décidé de créer la distribution *FAN* (*Fully Automated Nagios*). Basée sur Centos, elle est aussi simple à installer que sa grande sœur. Elle propose en plus Nagios, Centreon, NagVis (un outil de visualisation d'alertes avancées) et enfin NaReTo (un outil de *reporting*). Tous ces outils sont déjà préconfigurés et l'administrateur n'a plus qu'à ajouter ses éléments à surveiller pour avoir une solution complète de supervision.

3.2 L'installation de Centreon

Pour ceux qui souhaitent garder leur distribution habituelle, nous allons traiter les principales étapes dans l'installation de Centreon.

3.2.1 Les pré-requis

Celle-ci est un peu comme la première configuration de Nagios : relativement effrayante. En fait, il n'en est rien, et c'est surtout la liste des pré-requis qui fait plus peur

qu'autre chose. Mais, après tout, ceci signifie que l'outil ne réinvente pas la roue, ce qui est plutôt bon signe au final.

Les pré-requis principaux sont Nagios, MySQL, PHP5, RRDTool. Le reste concerne des bibliothèques Perl et PHP. Sur une Centos, il suffit de lancer :

```
yum install php gd gd-devel rrdtool net-snmp sudo httpd php-mysql php-pear php-snmp php-posix gd-devel libpng libpng-devel perl-Config-IniFiles perl-Crypt-DES perl-Digest-Hmac perl-Digest-SHA1 perl-GD perl-IO-Socket-INET6 perl-Net-Snmp perl-rrdtool perl-Socket6 php-gd php-ldap sudo
```

Nous supposons ici que Nagios et NDOUtils sont déjà installés. Dans le cas contraire, il suffit d'ajouter l'installation des paquets **nagios** et **ndoutils**.

3.2.2 L'installation

La récupération de Centron s'effectue sur http://www.centreon.com. L'installation se fait avec le script **install.sh** fourni :

```
tar xvfz centreon-2.0.tar.gz
cd centreon-2.0
./install.sh -i
```

Un système de *template* automatisant l'installation sur les distributions les plus courantes est disponible. Nous ne l'utiliserons pas ici pour le besoin de la démonstration.

Ne seront listées ici que les options où le choix est un peu plus complexe que le fait de bien vouloir installer Centreon ou bien de créer des répertoires inexistants :

```
Where is the RRD perl module installed [RRDs.pm] default to [/usr/lib/perl5/RRDs.pm]
/usr/lib/perl5/vendor_perl/5.8.8/i386-linux-thread-multi/RRDs.pm
```

Puis, un peu plus loin :

```
Where is PEAR [PEAR.php] default to [/usr/share/php/PEAR.php]
/usr/share/pear/PEAR.php
```

L'emplacement de ces fichiers est obtenu avec la commande **locate**.

Nous arrivons enfin à un message peu clair :

```
I think you'll have a problem with
'Default requiretty' in sudo file
Press enter to continue.
```

Il faut ouvrir une nouvelle session *shell* pour éditer le fichier **/etc/sudoers** et commenter la ligne suivante :

```
Defaults requiretty
```

Sans cela, le daemon Apache qui fonctionne sans tty ne pourrait pas lancer la commande **sudo** qui permet de lancer Nagios. L'auteur du script a promis un message plus clair dans la prochaine version :)

3.3 La configuration de Centreon

Une fois installé, il est nécessaire de finir l'installation de Centreon avec la configuration de sa connexion vers la base de données. Cette partie s'effectue depuis l'interface web. Pour y accéder, il suffit de se connecter sur http://monserveur/centreon.

Les premières pages permettent d'accepter la licence GPLv2 de l'outil, les suivantes permettent de spécifier à Centreon où se trouve Nagios (en général **/usr/local/nagios/**). Les autres pages sont dédiées à la mise en place de la base de données et de la configuration d'un compte administrateur. C'est en fait très classique dans le monde des applications web et l'administrateur ne sera vraiment pas perdu.

4 LA GESTION DE LA CONFIGURATION DE NAGIOS

Une fois installé, nous pouvons commencer à utiliser la première fonctionnalité de Centreon : la configuration de Nagios.

4.1 La configuration de Nagios

Cette étape est relativement aisée dans le sens où Centreon apporte déjà une configuration standard de Nagios qui conviendra à la grande majorité des utilisateurs. Ceci se passe dans l'onglet ***Configuration > Nagios > nagios.cfg***. Tout d'abord, nous remarquons que nous avons le choix entre plusieurs configurations de Nagios. En effet, Centreon est capable de gérer plusieurs Nagios, et chacun peut avoir sa propre configuration. Celle qui nous intéresse pour l'instant est celle du Nagios local à la machine Centreon qui se nomme **Nagios CFG 1**.

Le premier onglet permet de définir les fichiers utilisés par Nagios, le second concerne les options propres à l'ordonnancement des tests. Les deux onglets suivants concernent les options de logs et l'export des données de Nagios. Enfin, les trois derniers onglets portent sur le tuning et les options de débogage.

Pour appliquer cette configuration et lancer Nagios, il suffit d'aller dans ***Configuration > Nagios***, de cocher les cases

Move Export Files et ***Restart Nagios*** et enfin de cliquer sur le bouton ***Export***. Centreon va redémarrer le service Nagios par l'intermédiaire d'une commande **sudo**.

4.2 La configuration des éléments

4.2.1 Une configuration préalable

Pour l'instant, il n'y a qu'une configuration sommaire concernant les éléments à superviser. Seule la machine locale est monitorée. Nous allons ajouter nos machines et nos services, mais, avant cela, il est nécessaire d'avoir des contacts et des commandes de supervision définies. Ceci se passe dans les onglets ***Configuration > Users*** et ***Configuration > Commands***. Concernant les commandes, Centreon fournit déjà un grand nombre de commandes de vérifications ou de notifications.

Concernant les utilisateurs, il suffit d'ajouter les administrateurs en charge des éléments à surveiller en s'inspirant de l'utilisateur déjà défini par Centreon. Il est également possible d'importer des utilisateurs depuis une base Ldap et d'avoir ainsi une synchronisation des mots de passe.

4.2.2 Les choses sérieuses commencent

Une fois les contacts et commandes définies, il est possible de procéder à la configuration des hôtes et des services. Nous allons commencer par les hôtes, car les services s'appliquent dessus. Cette configuration peut être longue, voire très longue suivant le nombre d'éléments que vous avez à configurer.

Elle peut être d'autant plus longue si vous n'utilisez pas les possibilités avancées de Nagios en termes de configuration comme les techniques d'héritage. Elles ont été vues dans un précédent article, mais nous allons ici rappeler la principale d'entre elles : la configuration à partir d'un modèle. Supportée par Centreon, elle permet de définir un élément qui ne sera pas réellement surveillé, mais qui servira de base de configuration pour de nombreux autres. Par exemple, vous pouvez définir un modèle pour vos serveurs Linux qui ont en commun d'être surveillé 24h sur 24 et d'alerter les administrateurs durant cette même période. Ces définitions se font dans ***Configuration > Hosts > Templates***. Contrairement aux hôtes normaux, les modèles ne sont pas obligatoirement complets du point de vue de la configuration. Ce qui importe, c'est que les hôtes qui vont l'utiliser le soient.

Pour que ces derniers l'utilisent, il faut créer un hôte dans ***Configuration > Hosts > Hosts*** et changer sa propriété `Host Multiple Templates`. Il est possible de procéder exactement de la même façon avec les services. Plusieurs modèles peuvent être utilisés conjointement.

Centreon apporte un réel gain dans la configuration des hôtes et des services. Il est possible de dupliquer un élément ou bien d'en modifier plusieurs en une seule opération. De plus, les écrans de configuration sont très bien pensés et organisés. Les propriétés principales sont proposées sur les deux premiers onglets et suffisent dans 99% des cas. L'administrateur n'est ainsi pas perdu dans une montagne d'options dont il ne comprend pas l'intérêt. Un simple clic mène à la documentation de Nagios afin d'aider à comprendre les différentes options qui s'offrent à l'utilisateur.

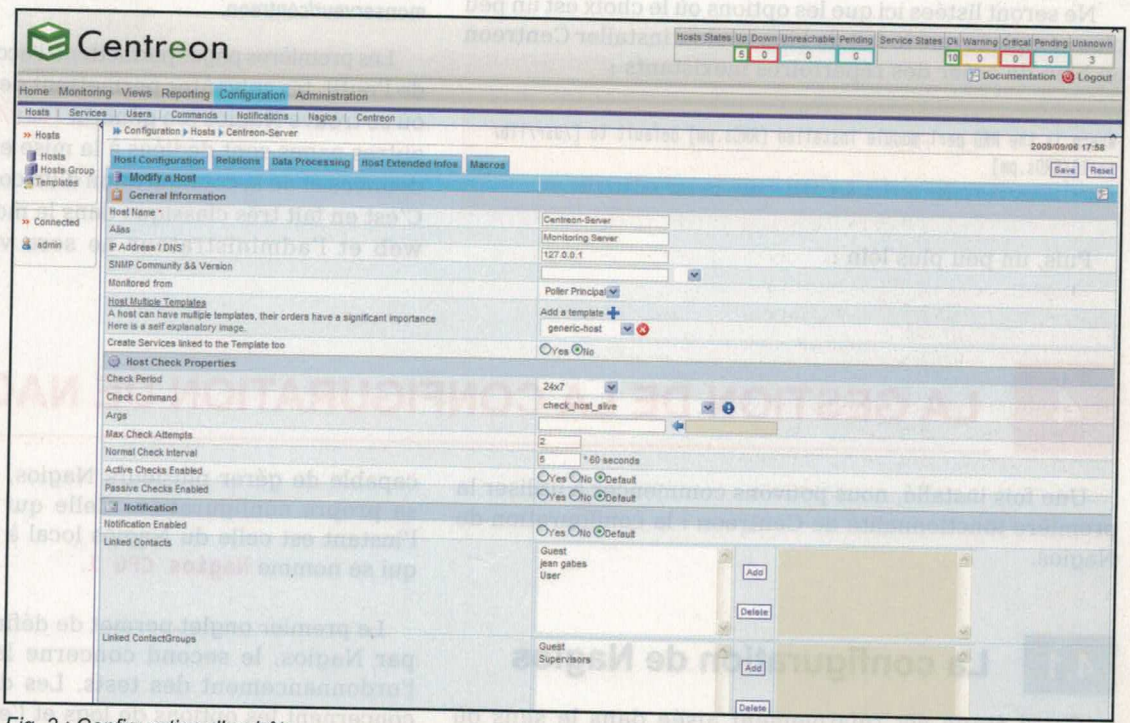


Fig. 3 : Configuration d'un hôte

4.2.3 Quelques manques

Tout n'est pas encore rose dans le monde de la configuration dans Centreon. Certaines options avancées de Nagios comme

les techniques d'héritages additifs ne sont pas encore disponibles, mais ne devraient pas tarder. Celles-ci n'étant pas encore largement répandues, peu d'administrateurs remarqueront ce manque pour l'instant.

5 LA VISUALISATION DES ALERTES

5.1 Les alertes en temps réel

Autre partie importante de Centreon : la visualisation des alertes en temps réel. Comme dit précédemment, Centreon va chercher les informations dans la base NDO continuellement par Nagios. L'interface de Centreon est très agréable à utiliser et fait une grande part à l'Ajax. En très peu de clics, l'utilisateur cherche, tri et filtre les informations qui l'intéressent.

Plusieurs types d'écrans sont proposés afin de convenir à tous les types de demandes. Nous pouvons voir les informations par hôtes, par services, par groupes d'hôtes ou de services, et, pour chacun, des filtres sont disponibles afin de ne voir que les problèmes et/ou les problèmes non encore pris en compte par un administrateur.

Il suffit de survoler, avec la souris, un hôte ou un service afin d'avoir les informations les plus importantes sur son état et un clic vous donne accès à l'ensemble de ses informations.

5.2 Remonter dans l'historique des alertes

En plus de l'aspect temps réel, Centreon propose des fonctions pour remonter l'historique des alertes. Ceci peut être pratique, par exemple, en cas de souci s'étant produit la nuit. Arrivé le matin, l'administrateur va chercher à comprendre ce qui s'est passé en analysant les messages d'alertes reçus, mais également tous les problèmes qui se sont produits peu de temps auparavant.

La masse d'information peut être très importante. C'est pourquoi Centreon propose là encore de nombreuses options de tri et de filtre. De cette manière, l'administrateur peut rechercher le plus efficacement possible les problèmes qui se sont posés et surtout qu'elle en a été l'origine.

Par exemple, il est possible de ne voir que les avertissements ou les alertes critiques et vérifiées d'un groupe de machines particulier entre 2 et 3 heures du matin. Avec de tels critères, là où la recherche prenait un quart d'heure à un administrateur chevronné en **grep**, n'importe qui peut faire de même en moins de 30 secondes. L'administrateur peut alors passer plus de temps à comprendre l'erreur et chercher à mettre en place des solutions palliatives pour qu'elle ne se reproduise plus.

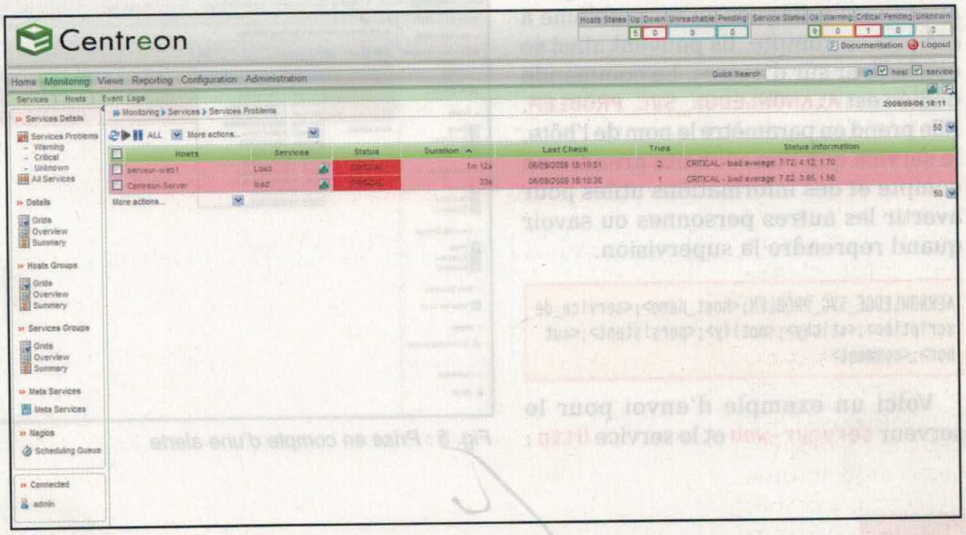


Fig. 4 : Vues des alertes dans Centreon

6 DONNER DES ORDRES À NAGIOS

6.1 Pourquoi donner des ordres à Nagios ?

Nagios fait ce qu'on lui demande : il a une configuration avec des hôtes et des services qu'il doit surveiller régulièrement. Mais, il n'en reste pas moins à l'écoute des nouvelles demandes des administrateurs une fois lancé.

Ils peuvent lui demander de changer un paramètre particulier comme une période de surveillance ou bien le prévenir qu'ils sont en train d'effectuer des manipulations

sur un serveur et qu'il est donc inutile qu'il les prévienne d'erreurs qui pourraient se produire sur ces derniers.

Ils peuvent également demander à Nagios d'avancer une vérification pour voir disparaître de la console de supervision un problème qu'ils viennent juste de régler.

Toutes ces demandes sont pour Nagios des commandes externes. Elles sont très nombreuses, même si au final la plupart des administrateurs n'utiliseront qu'une poignée d'entre elles.

6.2 Comment ordonner à l'ordonnanceur ?

Donner un ordre à Nagios est très facile lorsqu'on a un accès shell sur la machine où se trouve Nagios. En effet, une simple commande **echo** ou **printf** suffit. Nagios lit en continu les ordres qui peuvent lui être donnés sur un fichier particulier : un *pipe* nommé situé dans **/usr/local/nagios/var/rw/nagios.cmd**. Les commandes sont normées et ressemblent toutes à :

```
[DATE] COMMAND_NAME;arg1;arg2;arg3
```

Par exemple, un utilisateur peut avertir ses collègues qu'un problème a été pris en compte. Ils peuvent ainsi se concentrer sur d'autres. La commande dédiée est **ACKNOWLEDGE_SVC_PROBLEM**. Elle prend en paramètre le nom de l'hôte, le service que l'on souhaite prendre en compte et des informations utiles pour avertir les autres personnes ou savoir quand reprendre la supervision.

```
ACKNOWLEDGE_SVC_PROBLEM;<host_name>;<service_de
scription>;<sticky>;<notify>;<persistent>;<aut
hor>;<comment>
```

Voici un exemple d'envoi pour le serveur **serveur-web** et le service **Http** :

```
/bin/printf "[%lu] ACKNOWLEDGE_SVC_PROBLEM;serveur-web;Http;1;1;1;Admin
Web;Ok on s'en occupe\n" `date +%s` > /usr/local/nagios/var/rw/nagios.cmd
```

6.3 L'utilité de Centreon

A vrai dire, l'utilisation du **printf** est bien pratique lorsque l'on a un shell sous la main, mais il y a plus ergonomique tout de même. Il faut connaître toutes les commandes, ce qui n'est pas rien. De plus, donner un accès shell à tous les administrateurs ou les superviseurs à la machine Nagios n'est pas nécessairement du goût de tout le monde.

Et là encore, Centreon est utile. Il permet de lancer ces commandes simplement depuis l'interface Web sans avoir à connaître les noms des commandes. Ces actions sont présentes sur la liste des hôtes ou des services si l'on souhaite l'appliquer à un grand nombre d'éléments, mais également sur chaque page d'un élément spécifique. Toutes les actions définies par Nagios ne sont pas proposées. Seules les plus utiles le sont afin de ne pas surcharger inutilement l'interface.

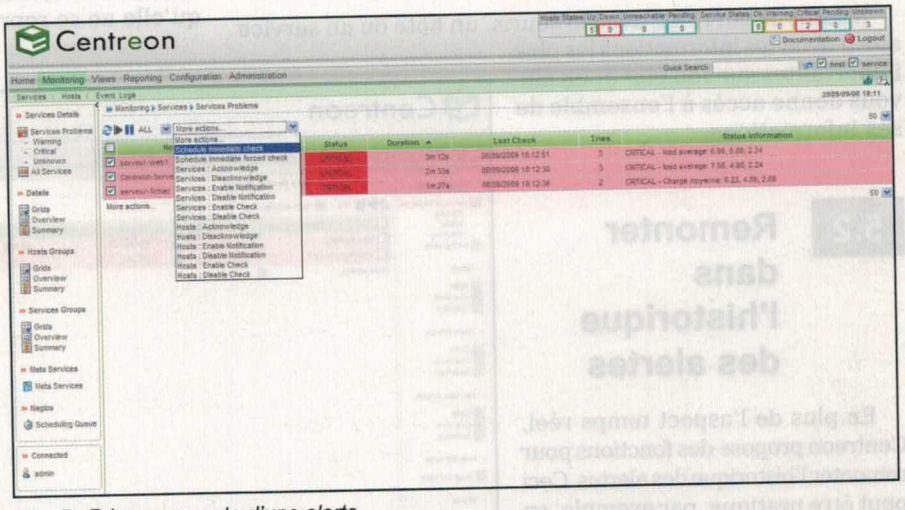


Fig. 5 : Prise en compte d'une alerte

7 LA GESTION DE LA MÉTROLOGIE

7.1 La métrologie est très utile

La métrologie consiste à récupérer des données numériques pour les transformer en graphiques lisibles par les administrateurs. Si cette fonctionnalité peut paraître bien secondaire à bon nombre d'entre nous, il n'en est rien. Elle peut, à elle seule, justifier le coût de la mise en place d'une solution de supervision/métrologie. Si l'aspect graphique plait indéniablement aux responsables et aide à faire passer la facture, c'est dû au fait qu'elle est l'une des bases dans le dimensionnement des serveurs et équipements réseau lors des renouvellements de parcs.

Les deux ressources qui gagnent le plus à ce jeu-là sont indiscutablement la mémoire et les CPU. Lorsqu'un administrateur sort une courbe où la mémoire de la machine n'a jamais atteint les 20% et que le CPU a plafonné à 10%, le renouvellement par un serveur de même gabarit est vite remis en question. Les gains sont importants et très rapides à estimer. Ils sont parfois les seuls que possèdent les administrateurs pour justifier la mise en place de la solution de supervision/métrologie.

7.2 CentStorage

Comme nous l'avons vu, Nagios ne traite pas directement les données de métrologie, mais est un vecteur privilégié pour les récupérer. Il les stocke tout simplement dans un fichier plat. Centreon possède un daemon nommé **CentStorage** qui est chargé de la partie métrologie. Il est codé en Perl. Il récupère les données de ce fichier et envoie les informations dans deux types de bases de données suivant le souhait de l'administrateur.

La première est constituée de multiples bases RRDTools. Elles permettent de conserver les données sur de grandes périodes en agrégeant les valeurs les plus anciennes : si, dans les derniers jours, toutes les informations sont conservées, plus l'on remonte dans le temps, plus des moyennes sur des périodes de plus en plus grandes sont conservées. Ceci permet d'avoir des informations précises sur le court terme, et les tendances sur de longs termes (par exemple sur la dernière année), le tout en ayant des fichiers de taille raisonnable (moins de 5Mo en moyenne).

La seconde destination des informations est une base MySQL tout ce qu'il y a de plus standard. La durée de rétention est bien plus faible que celle des bases RRDTools (mais paramétrable comme pour ces dernières) et ne sert qu'à les régénérer en cas de souci.

7.3 La visualisation des courbes

Si l'intégration des données est importante, les possibilités d'affichage des courbes le sont tout autant. De ce côté, Centreon est très ergonomique. La visualisation ne s'arrête pas à un service à la fois. Il est possible de sélectionner toutes les données d'un groupe de machines. Les utilisateurs peuvent également choisir la période de temps considérée.

Il est ainsi possible d'avoir sur une seule page toutes les courbes de charge des serveurs. Ceci peut être pratique afin de réorganiser les ressources entre vos machines ou d'identifier la source d'un ralentissement.

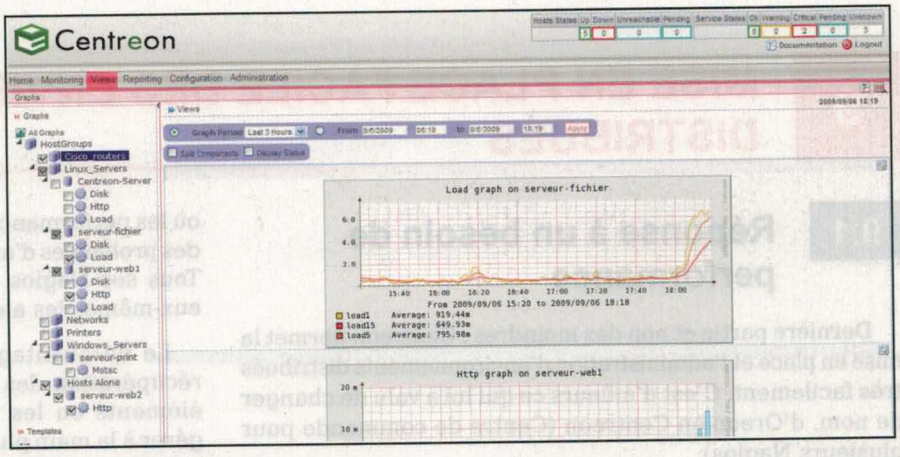


Fig. 6 : Exemple de courbes obtenues

8 LA GESTION DES TRAPS SNMP

8.1 Un problème complexe à gérer sans outil dédié

Les traps SNMP sont un moyen très répandu pour remonter des problèmes. Un élément (*switch*, routeur ou serveur) envoie à un autre un paquet contenant une alerte identifiée par une OID définie dans des fichiers MIB où se trouve la signification de l'erreur et son format précis. Il faut bien entendu que le receveur de l'alerte soit en écoute. C'est le rôle du daemon **snmptrapd**. Ce dernier réceptionne le paquet Trap et le transmet à un autre programme, **snmptt**, qui va traduire l'erreur et réagir en fonction. L'une de ses possibilités, qui nous intéresse ici, consiste à relayer cette alerte à Nagios. Ce dernier sera en charge de lever les notifications aux administrateurs si besoin est. L'envoi de l'alerte à Nagios passe par une commande externe. Un script nommé `submit_check_result` est fourni avec Nagios pour effectuer simplement cette action.

Si la mise en place de snmptrapd est simple, la configuration de snmptt est moins aisée : il a besoin de savoir quoi faire, et ce, pour chaque OID à traiter. Il est nécessaire de passer par la phase de conversion des fichiers MIB en actions à effectuer. Nous allons simplement lui demander de lancer une alerte à Nagios sur l'hôte qui a levé la trap (`$r` dans snmptt) sur le service TRAP.

```
snmpttconvertmib -in=MAMIB.MIB --out=/etc/snmp/snmptt.conf.monequipement
--exec='/usr/local/nagios/libexec/eventhandlers/submit_check_result $r TRAP 1'
```

Le fichier `/etc/snmp/snmptt.conf.monequipement` doit ensuite être ajouté dans la configuration de snmptt afin d'être pris en compte.

8.2 La solution de Centreon

Cette étape est pénible et peu pratique. Toutes les alertes d'un fichier vont arriver sur le même service et vont donc alerter les mêmes personnes. Une solution plus souple est préférable. Centreon est capable de gérer cette situation.

Ce dernier propose à l'administrateur de définir des équipementiers. Ces derniers sont simplement les fabricants des équipements. Ils vont pouvoir se voir chargés des MIB. Une fois les alertes chargées dans la base Centreon, l'administrateur va pouvoir faire le lien entre des hôtes, des services et des traps. Ces relations sont conservées en base. Il va ensuite générer les fichiers de configuration de snmptt et lui faire lancer pour chaque alerte la commande trapHandler. Cette dernière va être appelée de la manière suivante

```
/usr/local/nagios/libexec/traps/trapHandler $aA $A $o "Texte de la trap $*"
```

où **$aA** est l'adresse IP de l'émetteur, **$A** son nom et **$o** l'oid de la trap.

Ayant les informations sur l'OID et l'hôte qui a envoyé la trap, elle va aller regarder en base si une relation existe avec un service. Si c'est le cas, elle va simplement lancer la commande **submit_check_result** pour lever une alerte au sens Nagios.

L'administrateur n'a ainsi plus à se préoccuper de la configuration de snmptt. Il peut découper plusieurs types d'alertes d'une même Mib sur plusieurs services, et peut ainsi alerter différentes personnes au besoin.

9 MISE EN PLACE FACILE DES ENVIRONNEMENTS DISTRIBUÉS

9.1 Réponse à un besoin de performance

Dernière partie et non des moindres : Centreon permet la mise en place et l'administration d'environnements distribués très facilement. C'est d'ailleurs ce qui lui a valu de changer de nom, d'Oreon en Centreon (Centre de commande pour plusieurs Nagios).

Les environnements distribués consistent à mettre en œuvre plusieurs Nagios. Ils sont utiles dans des situations où les performances d'un seul ne suffisent plus ou bien pour des problèmes d'architectures réseau ou organisationnelles. Tous ses Nagios ont leur propre configuration et lèvent eux-mêmes les alertes.

Le désavantage de ces architectures réside dans la récupération des informations, notamment les états des éléments ou les informations de performance. Avoir à gérer à la main plusieurs configurations n'est pas un mince problème non plus.

9.2 La gestion par Centron

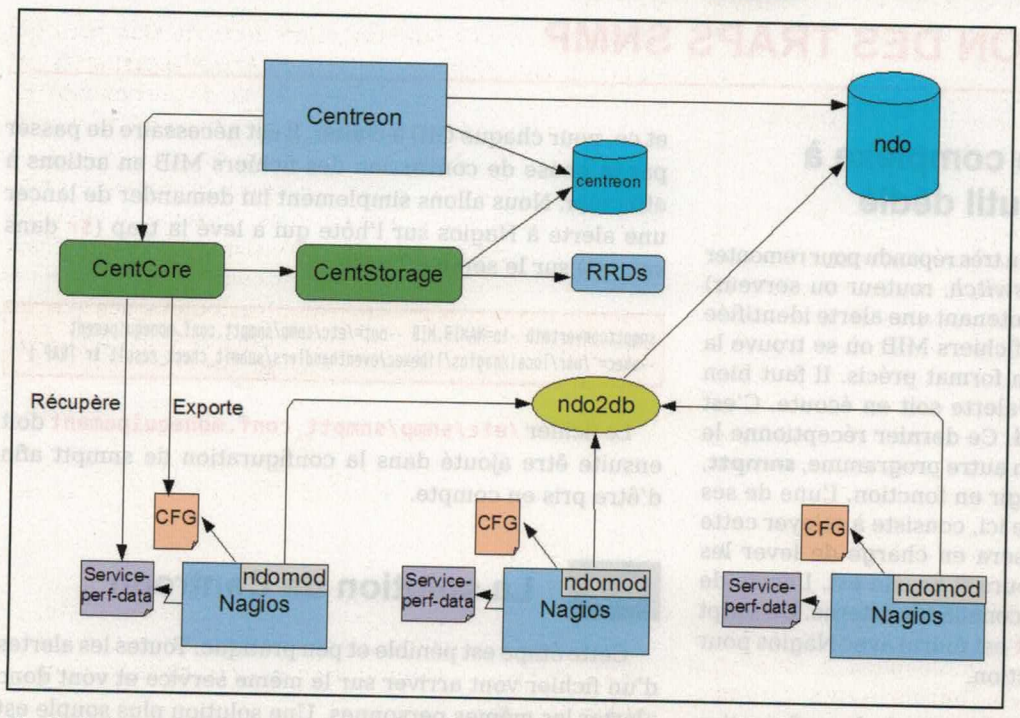


Fig. 7 : Architecture Nagios distribuée avec Centreon

Centreon gère tout cela. Les administrateurs peuvent définir des *pollers* qui sont tout simplement les Nagios distants. Comme nous l'avons vu, Centreon récupère les données d'états des éléments dans la base du module NDO. Elle va être mise encore à contribution ici : chaque Nagios distant va voir son module **ndomod** envoyer ses informations vers un unique daemon **ndo2db** qui va les placer en base. Centreon permet de gérer la configuration de chaque poller que ce soit au niveau de Nagios et de son fichier **nagios.cfg** ou bien de son module **ndomod** et son fichier de configuration **ndomod.cfg**.

Chaque hôte que l'administrateur a défini va se voir attribué un poller. Par

défaut, c'est celui en local sur le serveur Centreon qui s'en occupe. Les services rattachés à cet hôte se verront également supervisés par le même poller.

L'administrateur peut, depuis Centreon, tester les configurations de tous ses Nagios depuis l'interface Web. Une fois que toutes sont validées, il peut en demander l'envoi vers les serveurs distants. C'est le daemon **CentCore** qui va s'en occuper.

Ce dernier va simplement envoyer par **SCP** les fichiers de configuration et demander par **SSH** de redémarrer le service Nagios distant. La mise en place d'un Nagios distant est très simple : en plus du binaire Nagios et des plugins de vérification, seul un compte `nagios` et quelques ajouts dans le fichier `/etc/sudoers` pour ce dernier sont nécessaires.

CentCore est également utile pour récupérer les données des Nagios distants : il est en charge de rapatrier les logs des Nagios distants pour analyse locale et également les fichiers `service-perf-data` qu'il transmet ensuite à CentStorage pour que ce dernier les traite comme s'ils provenaient du Nagios local.

De cette manière, l'administrateur obtient une console d'administration unique pour l'ensemble de sa supervision. Il peut la configurer et l'administrer en ayant uniquement l'accès à la console de Centreon.

Nous pouvons résumer cette architecture par le diagramme figure 7.

10 IL RESTE ENCORE BEAUCOUP À VOIR

10.1 Dans Centreon

Centreon permet encore beaucoup de choses très pratiques. Une page nommée `Nagios Statistics` permet d'observer les performances de Nagios. Elle est très pratique afin de voir le nombre de vérifications effectuées, mais également le temps de latence de Nagios. Cet indicateur est primordial dans l'analyse de ses performances. Il représente la différence de temps entre le lancement effectif d'une vérification et ce qui était prévu. Un temps de l'ordre d'une seconde indique que Nagios a suffisamment de ressources pour effectuer son travail. Dans le cas contraire, il faut lui permettre de reprendre son souffle en diminuant le nombre d'éléments supervisés, utiliser les options de tuning de Nagios ou mettre en place une supervision distribuée.

Une autre possibilité offerte par Centreon réside dans les ACL (liste de contrôle d'accès). Chaque utilisateur peut se voir accroché des autorisations et interdictions sur les pages de Centreon. Il est ainsi possible de n'autoriser un administrateur réseau qu'à la visualisation d'alertes des machines qui le concernent et de lui interdire tout le reste. Un autre aurait accès aux alertes sur les serveurs, mais également à la partie configuration. Les possibilités sont très étendues : l'accès à toutes les pages sont indépendantes les unes des autres.

Une partie `reporting` est également disponible. Elle permet de voir les taux de disponibilités des éléments dans Nagios. On peut ainsi savoir combien de temps un groupe d'hôtes ou un groupe de services a été indisponible durant la dernière année. Les problèmes à la source de l'indisponibilité sont également fournis. Autant dire que ces pages seront celles qui vont intéresser le plus les responsables :)

La mise en place d'un Centreon peut effrayer les administrateurs qui ont déjà un Nagios en place avec une configuration faite aux petits oignons. Heureusement pour eux, Centreon propose une option pour importer cette configuration et la charger en base comme si elle avait été créée par l'interface standard.

10.2 L'aventure continue

Nous venons de voir les points forts de Centreon qui en font le meilleur ami de Nagios. Arrivé à maturité, cet outil fait bien plus que son premier rôle de gestionnaire de configuration. Il a pris en charge de plus en plus d'aspects de la supervision basée sur Nagios. Comme nous l'avons vu, la situation n'est pas toute rose. Certaines fonctions récentes de Nagios ne sont pas encore gérées. Certaines techniques d'héritage de configuration sont ainsi impossibles à utiliser avec Centreon. Son évolution au cours des 6 dernières années a été rapide et constante. Ne doutons pas que ses développeurs et sa communauté vont continuer à en faire l'outil qui sied le mieux à Nagios. ■

Auteur : Jean Gabès

Auteur du livre « Nagios 3 pour la Supervision et la Métrologie. Déploiement, Configuration et Optimisation » aux éditions Eyrolles

Annexe : PNP4Nagios

PNP4Nagios est une extension pour Nagios dont l'objectif est d'extraire les données de performance remontées à Nagios et de les présenter sous forme de graphes générés par rrdtool.

Auteur

- Yves Mettier

1 INTRODUCTION

Autrefois, au début du XXIe siècle, les outils propriétaires se faisaient la guerre pour présenter des rapports de performance des serveurs. Ou plutôt, il s'agissait des développeurs et des commerciaux qui vendaient ces outils. Ils n'étaient pas trop gênés par la concurrence du libre, car il n'existait rien de sérieux. Il y avait bien un projet APAN (*Advanced Performance Addon for Nagios*) à greffer à l'outil de supervision Nagios, mais il était à l'abandon. Un autre projet, Perfparse, également une extension pour Nagios, était en plein développement, mais reposait principalement sur les épaules d'une seule personne. Il avait la particularité de ne pas reposer sur des bases RRD (au contraire de la plupart des projets actuels) qui devenaient à la mode, et de proposer une interface graphique sous forme de CGI programmée en C (quelle horreur !). Alors que le dernier développeur de Perfparse baissait en cadence et n'arrivait plus à implémenter les fonctionnalités souhaitées par les utilisateurs, d'autres projets naissaient. L'un d'eux, PNP4Nagios, est celui que nous allons vous présenter.

Pourquoi avoir insisté sur Perfparse ? D'une part, parce que l'auteur de l'article a activement participé à Perfparse. D'autre part, à cause de l'acronyme PNP4Nagios qui veut dire *PNP is Not Perfparse - for Nagios*.

PNP4Nagios n'est pas Perfparse sur de nombreux points :

- L'analyseur de données de performance est programmé en Perl.
- L'interface utilisateur est en PHP.
- La base de données est une base en tourniquet (ceux qui connaissaient la traduction de RRD en français lèvent le doigt) générée et utilisée par RRDtool.

Seul point commun : l'analyseur peut être lancé en tant que démon, à l'aide d'un *pipe* ou directement comme commande de service, et ce, pour supporter une forte charge (avec le démon) ou pour avoir une installation simple (commande de service).

Partant d'une configuration de base Nagios 3 fonctionnelle, nous allons compiler et installer PNP4Nagios, l'intégrer à Nagios, puis nous amuser un peu avec les modèles (*templates*) pour présenter les graphiques.

2 PRÉ-REQUIS : NAGIOS

Nous partons du principe que vous avez un Nagios 3 installé, fonctionnel, et que vous savez vous en servir. Si tel n'est pas le cas, visitez **http://nagios.org** pour en savoir plus. Si vous disposez d'une distribution Debian Lenny (5.0), voici un rappel de la façon d'installer Nagios :

```
$ sudo aptitude install nagios3
```

Pour pouvoir utiliser correctement l'interface utilisateur sur **http://localhost/nagios3/**, vous devez revoir la configuration de Nagios pour l'interfacer avec celle du serveur web Apache (que nous supposons installé). Éditez le fichier **/etc/nagios3/nagios.cfg** et activez les commandes externes :

```
check_external_commands=1
```

Puis, suivez la documentation du fichier **/usr/share/doc/nagios3/README.Debian** en lançant les commandes suivantes :

```
$ sudo /etc/init.d/nagios3 stop
$ sudo dpkg-statoverride --update --add nagios www-data 2710 /var/lib/
nagios3/rw
$ sudo dpkg-statoverride --update --add nagios nagios 751 /var/lib/nagios3
$ sudo /etc/init.d/nagios3 start
```

Ces commandes changent les permissions permettant d'atteindre le fichier de commandes **/var/lib/nagios3/rw**. Il faut évidemment arrêter Nagios au préalable pour qu'il n'interagisse pas avec ces commandes. Et tout aussi évidemment le relancer ensuite.

Enfin, activez le compte administrateur de Nagios en créant le fichier **/etc/nagios3/htpasswd.users** si nécessaire et en lui ajoutant ou changeant le mot de passe :

```
$ sudo htpasswd -c /etc/nagios3/htpasswd.users nagiosadmin
```

À ce stade, vous avez un Nagios fonctionnel et basique sur votre serveur avec au moins les greffons suivants :

- *gateway / PING* ;
- *localhost / Current Load* ;
- *localhost / Current Users* ;
- *localhost / HTTP* ;
- *localhost / SSH* ;
- *localhost / Total Processes*.

Si le service SSH est en erreur, vérifiez que le serveur OpenSSH est bien installé et en état de fonctionnement !

3 COMPILATION DE PNP4NAGIOS

Nous allons maintenant installer PNP4Nagios. Dans un premier temps, récupérez les sources sur http://www.pnp4nagios.org/. Nous travaillons avec la dernière version stable, 0.4.14 lors de la rédaction de cet article.

Vérifiez que vous avez le nécessaire pour compiler, que rrdtool est installé, et que vous disposez bien de l'extension RRDTool pour Perl. Sur Debian, exécutez la commande suivante pour cela :

```
$ sudo aptitude install build-essential rrdtool librrds-perl
```

Dans le répertoire **pnp4nagios-0.4.14**, la commande **./configure** n'est pas anodine. En effet, nous allons modifier quelques chemins :

- **/opt/pnp4nagios/0.4.14/var/perfdata.log** : le journal de Nagios contenant les données de performance ;
- **/opt/pnp4nagios/data/rrd** : le répertoire contenant les bases tourniquet de PNP4Nagios ;
- **/opt/pnp4nagios/data/spool** : le répertoire contenant des fichiers temporaires de journaux à analyser ;

Vous pouvez (vous devez ?) placer le répertoire **/opt/pnp4nagios/data** sur un système de fichiers dédié aux données applicatives. Dans notre cas, le choix de **/opt/pnp4nagios/data** suppose que ce répertoire est un lien vers un tel système de fichiers.

```
$ ./configure --prefix=/opt/pnp4nagios/0.4.14 --with-perfdata-logfile=/
opt/pnp4nagios/0.4.14/var/perfdata.log --with-perfdata-dir=/opt/
pnp4nagios/data/rrd --with-perfdata-spool-dir=/opt/pnp4nagios/data/spool
checking for a BSD-compatible install... /usr/bin/install -c
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking for gcc... gcc
[...]
  General Options:
  -------------------------       -------------------
  Nagios user/group:              nagios nagios
  Install directory:              /opt/pnp4nagios/0.4.14
  HTML Dir:                       /opt/pnp4nagios/0.4.14/share
  Config Dir:                     /opt/pnp4nagios/0.4.14/etc
  Location of rrdtool binary:     /usr/bin/rrdtool Version 1.3.1
  RRDs Perl Modules:              FOUND (Version 1.3001)
  RRD Files stored in:            /opt/pnp4nagios/data/rrd
  process_perfdata.pl Logfile:    /opt/pnp4nagios/0.4.14/var/perfdata.log
  Perfdata files (NPCD) stored in: /opt/pnp4nagios/data/spool
  Review the options above for accuracy.  If they look okay,
  type 'make all' to compile.
```

Les développeurs ont pensé à nous en nous disant quoi faire :

```
$ make all
[...]
*** Compile finished ***
  make install
    - This installs the main program and HTML files
```

Ici, merci les développeurs, mais nous le savions. C'est l'habitude.

```
$ sudo make install
[...]
*** Main program, Scripts and HTML files installed ***
Please run 'make install-config' to install sample
configuration files
Please run 'make install-init' if you want to use
BULK Mode with NPCD
```

Voilà qui est nouveau. Heureusement que les développeurs nous indiquent la marche à suivre. Mais nous allons n'installer que les fichiers de configuration d'exemples. En effet, nous faisons le choix d'une petite plateforme de surveillance, avec quelques serveurs et quelques services. Aussi, point n'est besoin de s'intéresser aux performances apportées par le démon NPCD (qui plante parfois, alors un greffon Nagios de type **check_npcd** est de très bon goût).

```
$ sudo make install-config
```

L'installation est finalement assez simple, bien que non habituelle. Nous allons maintenant intégrer PNP4Nagios à Nagios.

4 INTÉGRATION DE PNP4NAGIOS DANS NAGIOS 3

Pour utiliser PNP4Nagios, la première chose à modifier dans la configuration de Nagios est d'activer les données de performance. Pour ceux qui n'en ont pas besoin, mieux vaut qu'elles soient désactivées par défaut. Quant à nous, modifions **/etc/nagios3/nagios.cfg** :

```
process_performance_data=1
```

Activer les données de performance ne suffit pas. Il faut aussi les communiquer à PNP4Nagios. Comme nous l'avons évoqué plus haut, il existe au moins trois méthodes pour cela. L'une consiste à envoyer les données à PNP4Nagios en utilisant un pipe. Nous déconseillons fortement cette façon de faire, car, en cas de dysfonctionnement (de PNP4Nagios, de Nagios ou du serveur), les données en cours de traitement seraient irrémédiablement perdues. Les deux autres

consistent à demander à Nagios d'écrire ces données dans un fichier et de faire analyser ce fichier à PNP4Nagios de façon indépendante. Ainsi, vous pouvez configurer le démon **npcd** qui lancera l'analyseur PNP4Nagios pour traiter les données. C'est a priori la meilleure façon de faire pour les grosses configurations avec des milliers de services, car l'écriture et la lecture des données sont totalement dissociées. Pourtant, nous avons choisi de demander à Nagios de lancer PNP4Nagios, car cette méthode est plus simple à configurer et convient très bien pour de petites configurations (quelques dizaines de services).

Remarque

Nous ne configurons les données de performance que pour les services (directives commençant par **service_**). Si vous souhaitez aussi surveiller les serveurs, adaptez ce qui suit avec les directives en **host_**.

Modifiez encore le fichier **/etc/nagios3/nagios.cfg** pour indiquer à Nagios où écrire les données de performance et sur quel modèle :

```
service_perfdata_file=/var/log/nagios-service-perfdata

service_perfdata_file_template=DATATYPE::SERVICEPERFDATA\
tTIMET::$TIMET$\tHOSTNAME::$HOSTNAME$\tSERVICEDESC::$SERVICEDESC$\tS
ERVICEPERFDATA::$SERVICEPERFDATA$\tSERVICECHECKCOMMAND::$SERVICECHE
CKCOMMAND$\tHOSTSTATE::$HOSTSTATE$\tHOSTSTATETYPE::$HOSTSTATETYPE$\
tSERVICESTATE::$SERVICESTATE$\tSERVICESTATETYPE::$SERVICESTATETYPE$\
tSERVICEOUTPUT::$SERVICEOUTPUT$

service_perfdata_file_mode=a
```

Attention : gardez la directive **service_perfdata_command** commentée. Nous demandons à Nagios d'écrire les données lui-même dans un fichier et ne souhaitons pas exécuter en parallèle une commande qui ferait la même chose.

Enfin, nous allons profiter d'une directive qui permet de traiter les données de performance à intervalle régulier. Cette fonctionnalité a probablement été implémentée pour effectuer de la rotation de journaux. Pour nous, c'est l'endroit où nous interfaçons Nagios et PNP4Nagios. Éditez encore une dernière fois **/etc/nagios3/nagios.cfg** :

```
service_perfdata_file_processing_interval=30

service_perfdata_file_processing_command=process-service-perfdata-file
```

Nous allons ainsi lancer la commande **process-service-perfdata-file** toutes les 30 secondes. Cette commande est à définir dans le fichier **/etc/nagios3/command.cfg** :

```
define command{
  command_name process-service-perfdata-file
  command_line /usr/bin/perl /opt/pnp4nagios/0.4.14/libexec/process_
perfdata.pl --bulk=/var/log/nagios-service-perfdata
}
```

Nous y sommes presque : Nagios et PNP4Nagios sont interfacés. Mais, pour le vérifier, il faudrait avoir accès à ce dernier avec notre navigateur. Configurons le serveur web Apache pour prendre en compte le chemin **/pnp4nagios**. Si sa configuration est modulaire (comme sur Debian), ajoutez un fichier (nommé par exemple **pnp4nagios.conf** – que c'est original !) avec le contenu suivant dans **/etc/apache2/conf.d** :

```
Alias /pnp4nagios /opt/pnp4nagios/0.4.14/share

<Directory /opt/pnp4nagios/0.4.14/share>
  Options FollowSymLinks

  DirectoryIndex index.php

  AllowOverride AuthConfig
  Order Allow,Deny
  Allow From All

  AuthName "Nagios Access"
  AuthType Basic
  AuthUserFile /etc/nagios3/htpasswd.users
  require valid-user
</Directory>
```

Remarque

Si la configuration de votre serveur web n'est pas modulaire, nous vous conseillons de la rendre modulaire avec la directive **Include /etc/apache2/conf.d/**. Sinon, concaténez les lignes ci-dessous à votre unique fichier de configuration.

Relancez le serveur web :

```
$ sudo /etc/init.d/apache2 restart
```

Vérifiez maintenant l'URL **http://localhost/pnp4nagios/index.php?host=localhost**. Si vous n'avez rien, attendez un peu. En effet, si vous avez été trop vite, les bases tourniquet ne sont pas encore créées. Attendez 5 à 10 minutes (combien de cafés ?) et testez à nouveau. Il se peut aussi que vous ayez des messages d'erreur de PNP4Nagios, comme le module **GD** pour PHP qui manque. Sur Debian, installez ce module ainsi :

```
$ sudo aptitude install php5-gd
```

Une fois que l'URL précédente fonctionne comme souhaité, vous pouvez profiter d'une fonctionnalité de l'interface utilisateur de Nagios : les *extra actions*. Cela consiste à ajouter un petit logo (une étoile, ou plutôt une explosion) à côté des services concernés, avec la directive **action_url**. Pour chacun des services, vous pourriez donc ajouter la ligne suivante :

Remarque

Pour gagner en performance, vous pouvez placer le journal de performance de Nagios dans un système de fichiers monté en mémoire vive comme tmpfs. Avec des droits restrictifs, un fichier dans un sous-répertoire de **/tmp** pourrait convenir. Mais, en contrepartie des performances gagnées, vous pouvez perdre quelques dizaines de secondes de données de performance en cas de crash du serveur, puisque les données ne sont plus écrites physiquement sur un disque. D'un autre côté, si vous préférez écrire le journal physiquement sur le disque, rappelez-vous que les données transitent par un cache et que celui-ci n'est pas forcément écrit sur le disque au moment du crash du serveur. À vous de voir entre **/tmp/nagios3/service-perfdata** et **/var/log/nagios-service-perfdata**...

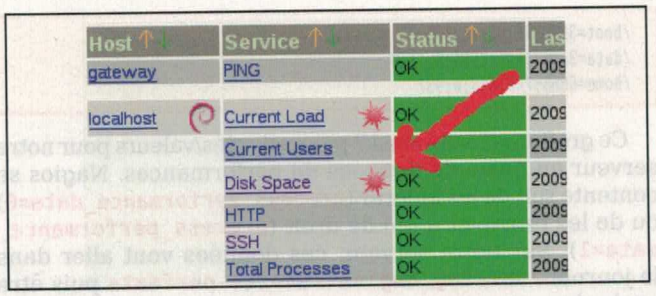


Fig. 1 : Logo pour action_url

```
action_url /pnp4nagios/index.php?host=$HOSTNAME$&srv=$SERVICEDESC$
```

Cependant, nous allons préférer l'utilisation de l'héritage multiple, fonctionnalité apportée dans la version 3 de Nagios. Nous allons définir un service modèle ci-dessous :

```
define service {
  name       srvpnp
 register   0
  action_url /pnp4nagios/index.php?host=$HOSTNAME$&srv=$SERVICEDESC$
}
```

Pour chaque service dont vous voulez qu'il affiche l'icône d'action complémentaire, vous allez préciser qu'il hérite également de **srvpnp**. Ainsi, au lieu de lire

```
use generic-service
```

ce sera :

```
use generic-service,srvpnp
```

Relancez Nagios pour prendre cette modification en compte :

```
$ sudo /etc/init.d/nagios3 restart
```

L'intérêt d'utiliser l'héritage est que vous centralisez l'URL de PNP4Nagios. Si vous devez changer cette URL, par exemple parce que, votre serveur croulant sous la charge, vous avez décidé de déplacer Apache et PNP4Nagios sur un autre serveur, vous pourrez la changer dans le service **srvpnp** uniquement. Vous n'aurez pas à modifier chaque **action_url** dans chaque service un à un en risquant un oubli ou n'importe quelle autre erreur humaine.

Si vous avez modifié le service **check_all_disks** pour qu'il hérite de **srvpnp**, vous verrez le petit logo d'explosion en face de *Disk Space* sur l'interface web. Si vous ne l'avez pas fait, faites-le, car nous allons maintenant nous amuser avec les données de performance des disques.

5 PNP4NAGIOS EN FRENCH

Avez-vous remarqué que la page affichant les graphes était en anglais ? Les images attirent tellement l'œil...

Pour configurer PNP4Nagios en français, éditez le fichier **/opt/pnp4nagios/0.4.14/etc/config.php**. Changez la langue par défaut :

```
$conf['lang'] = "fr";
```

Mais, cela ne suffit pas. Modifiez également la fin du fichier de configuration :

```
$views[0]["title"] = "4 Heures";
$views[0]["start"] = ( 60*60*4 );
```

```
$views[1]["title"] = "24 Heures";
$views[1]["start"] = ( 60*60*24 );

$views[2]["title"] = "Une semaine";
$views[2]["start"] = ( 60*60*24*7 );

$views[3]["title"] = "Un mois";
$views[3]["start"] = ( 60*60*24*30 );

$views[4]["title"] = "Un an";
$views[4]["start"] = ( 60*60*24*365 );
```

Remarquez qu'il est facile de changer la durée représentée dans un graphe, voire d'ajouter une nouvelle durée (n'oubliez pas d'incrémenter le compteur !).

6 PRINCIPE DES DONNÉES DE PERFORMANCE ISSUES DES GREFFONS NAGIOS

Nagios donne accès à deux types de données de performance. Ce sont celles issues de Nagios, qui permettent de connaître un peu mieux son état en temps réel. Elles ne nous intéressent pas dans cet article. Ce sont aussi celles générées par certains greffons, comme **check_disk**.

Lorsqu'un greffon retourne un résultat à Nagios, il peut être décomposé en trois éléments :

- le code retour, qui indique si tout va bien, si c'est catastrophique ou si c'est pire ;
- le message (sur une ligne) et la partie qui précède le pipe (lorsqu'il y en a un) pour expliciter l'état dans un format lisible par un humain (quoique, des fois...) ;
- la partie du message qui suit le pipe, qui contient les données de performance.

Ces données de performance obéissent à certaines règles que nous rappelons ici. Leur format est une suite de paires *'clé'=valeur* séparées par une espace. Une paire a le format suivant :

```
'clé'=valeur[unité];[warn];[crit];[min];[max]
```

- Les clés peuvent contenir n'importe quel caractère, mais doivent être placées entre apostrophes si elles contiennent une espace ou un caractère « = ». Les apostrophes sont facultatives dans les autres cas et peuvent être remplacées par des guillemets (double *quotes*) si vous voulez qu'une clé contienne une apostrophe.
- La longueur des clés a deux limitations. Il faut, d'une part, se rappeler que l'intégralité du message ne

doit pas dépasser 1024 caractères (sinon, Nagios ne le digère pas). D'autre part, si, comme nous avec PNP4Nagios, vous voulez vous en servir avec RRDTool, vous devez vous limiter à 19 caractères.

- Les valeurs pour **warn**, **crit**, **min** et **max** peuvent être nulles et les point-virgules inutiles en fin de valeur sont facultatifs.
- L'unité peut être **s**, **us** ou **ms** pour des valeurs de temps (basées sur les secondes), **%**, **B**, **KB**, **MB**, **TB** pour des octets (ou multiples d'octets) ou **c** pour un compteur (comme le nombre de spams reçus). Dans le cas de nombres (comme le nombre d'utilisateurs), l'unité ne doit pas être indiquée (d'ailleurs, qu'auriez-vous mis ?).

Ainsi, le greffon **check_disk** renvoie-t-il ce résultat sur notre serveur (il est sur une ligne, mais nous le reformatons pour une meilleure lisibilité) :

```
DISK OK - free space: / 979 MB (21% inode=63%); /lib/init/rw 251 MB
(100% inode=99%); /dev 9 MB (99% inode=98%); /dev/shm 251 MB (100%
inode=99%); /boot 55 MB (64% inode=99%); /data 1744 MB (98% inode=99%);
/home 820 MB (92% inode=97%);|
  /=3475MB;3754;4223;0;4693
 /lib/init/rw=0MB;200;225;0;251
  /dev=0MB;8;9;0;10
  /dev/shm=0MB;200;225;0;251
```

```
/boot=30MB;72;81;0;91
/data=34MB;1499;1686;0;1874
/home=68MB;749;843;0;937
```

Ce greffon renvoie ainsi 7 paires de clés/valeurs pour notre serveur en guise de données de performances. Nagios se contente soit de les ignorer (**process_performance_data=0**) ou de les renvoyer à qui de droit (**process_performance_data=1**). Sur notre serveur, ces données vont aller dans le journal **/var/log/nagios-service-perfdata** puis être analysées par **process_perfdata.pl** toutes les 30 secondes (mode **bulk**).

PNP4Nagios (et plus particulièrement **process_perfdata.pl**) va, la première fois, créer une base tourniquet, dans le répertoire que nous avions indiqué **/opt/pnp4nagios/0.4.14/data/rrd/**, dans un fichier **localhost/Disk_Space.rrd** (du nom du serveur suivi du nom du service) avec RRDtool. Cette base sera alimentée à chaque fois que **process_perfdata.pl** aura des données de performance à analyser. Il en est de même pour les autres services produisant des données de performance (**Current_Load**, **Current_Users** et **HTTP** pour une installation de base sur Debian Lenny 5.0).

Nous rappelons ici que les bases tourniquet ainsi créées n'ont pas besoin de maintenance. Du fait de leur structure en tourniquet, le fichier qui les contient ne change jamais de taille.

7 PROGRAMMER UN MODÈLE (TEMPLATE) POUR RRDTOOL DANS PNP4NAGIOS

Nous voici maintenant avec des bases tourniquet et, en particulier, celle qui va nous intéresser, **Disk_Space.rrd**. Si vous cliquez sur le logo d'explosion correspondant ou que vous visitez l'URL **http://localhost/pnp4nagios/index.php?host=localhost**, vous allez obtenir un ensemble de graphes produits par RRDtool, d'une couleur jaune assez moche ou plutôt, pour rester positif, d'une couleur à vous rappeler le soleil et les fleurs de pissenlit qui ondulent dans le vent dans la fraîcheur d'une soirée estivale (voir Fig. 2).

Si vous regardez bien, ce jaune n'est pas le problème. La page web que vous visualisez propose des graphes pour des durées de 4h, 24h, une semaine, un mois et un an. Ça, c'est encore un bon point pour PNP4Nagios. Là où rien ne va plus, c'est que pour chacune de ces durées, vous avez un graphe pour CHAQUE système de fichiers monté, incluant de façon inutile **/lib/init/rw**, **/dev**, **/dev/shm**, voire **/boot** (quoique les collectionneurs d'images du noyau devraient surveiller ce dernier quand même).

Pour n'afficher que les systèmes de fichiers qui nous intéressent, nous avons trois stratégies possibles :

- modifier le greffon Nagios pour qu'il ne renvoie que les données de performance pour les systèmes de fichiers qui nous intéressent ;
- programmer un modèle dans PNP4Nagios qui liste spécifiquement les systèmes de fichiers qui nous intéressent ;
- programmer un modèle dans PNP4Nagios qui ignore les systèmes de fichiers sus-cités.

Afin d'anticiper l'apparition éventuelle de nouveaux systèmes de fichiers, nous préférons cette dernière manière

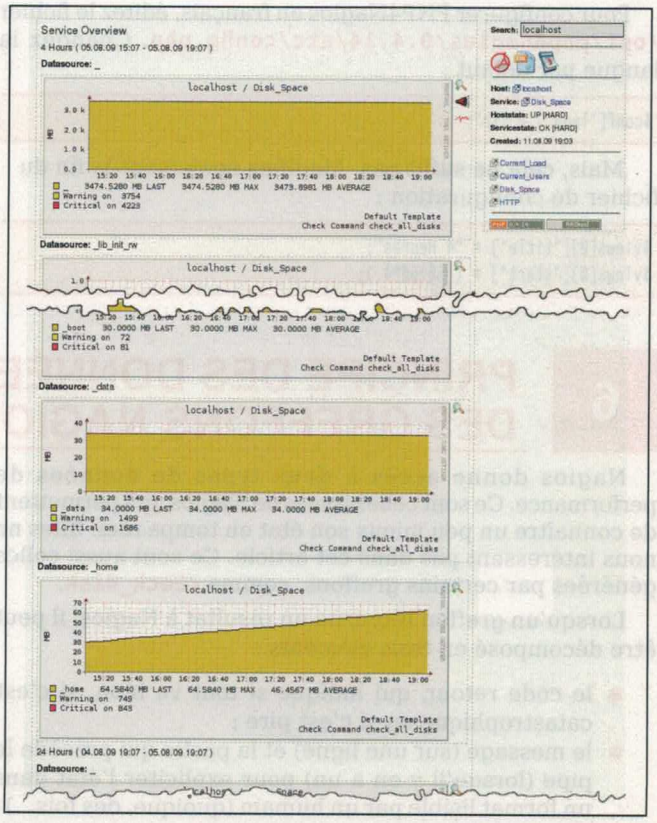


Fig. 2 : Extrait de la page des graphes du service Disk Space

de faire. Nous allons par ailleurs utiliser une ligne au lieu de l'aire jaune pour montrer l'évolution des systèmes de fichiers. Ces lignes pourront être tracées sur un même graphe. Mais, l'expérience montre que pour mieux visualiser les résultats, il vaut mieux disposer d'un graphe pour les valeurs en dessous de 10 Go et un autre pour les valeurs au-dessus. Nous nous limiterons à cette distinction bien qu'une suite de Fibonacci puisse faire l'affaire.

8 UN PEU DE PHP

PNP4Nagios propose des modèles par défaut qui se trouvent dans **/opt/pnp4nagios/0.4.14/templates.dist/**. Vous devez les laisser à cet endroit, car ils sont effectivement utilisés. Par contre, nous allons programmer le nôtre dans **/opt/pnp4nagios/0.4.14/templates/**. Les modèles de **templates/** ont priorité sur **templates.dist/**. Le nom d'un modèle est le nom de la commande (telle que définie dans **/etc/nagios-plugins/conf.d/** et repris dans la configuration de Nagios avec la directive **check_command**) avec l'extension **.php**. Ainsi, avec notre configuration de base, le fichier de modèle s'appellera **check_all_disks.php**.

Un fichier de modèle définit trois variables ou, plutôt, trois tableaux dont chaque élément correspond à un graphe (pour une durée donnée). Rappelons que les diverses durées (4h, 24h, une semaine...) peuvent être paramétrées dans le fichier **/opt/pnp4nagios/0.4.14/etc/config.php** comme nous l'avons vu quand nous parlions de traduction plus haut. Pour une durée donnée, PNP4Nagios présente autant de graphes qu'il y a de DS (**Datasource**) dans la base tourniquet RRA. Pour en revenir à nos tableaux, à chaque élément correspond un DS.

Les trois tableaux sont les suivants :

- **opt[]** : les options pour **rrdtool graph** ;
- **def[]** : les définitions et tous les autres éléments de calcul, dessin... pour **rrdtool graph** ;
- **ds_name[]** : le nom à afficher pour le **Datasource** (par défaut, il correspond au DS défini dans la base tourniquet).

Comme il s'agit de code PHP, nous pouvons nous servir de variables prédéfinies. Ce sont des tableaux (à chaque DS un élément) dont le nom correspond à ce que vous pouvez trouver dans un fichier XML (dont nous avons soigneusement ignoré l'existence jusqu'ici dans l'article). Lors de la création d'un fichier RRD, PNP4Nagios crée, dans le répertoire **/opt/pnp4nagios/0.4.14/data/rrd/**, le fameux fichier **greffon.rrd**, mais aussi un fichier **greffon.xml**. Voici un extrait d'un tel fichier, **Disk_Space.rrd** :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<NAGIOS>
  <DATASOURCE>
    <TEMPLATE>check_all_disks</TEMPLATE>
    <IS_MULTI>0</IS_MULTI>
    <DS>1</DS>
    <NAME>_</NAME>
    <UNIT>MB</UNIT>
   <ACT>3474</ACT>
   <WARN>3754</WARN>
    <WARN_MIN></WARN_MIN>
    <WARN_MAX></WARN_MAX>
    <WARN_RANGE_TYPE></WARN_RANGE_TYPE>
    <CRIT>4223</CRIT>
    <CRIT_MIN></CRIT_MIN>
    <CRIT_MAX></CRIT_MAX>
    <CRIT_RANGE_TYPE></CRIT_RANGE_TYPE>
    <MIN>0</MIN>
    <MAX>4693</MAX>
  </DATASOURCE>
  <DATASOURCE>
    <TEMPLATE>check_all_disks</TEMPLATE>
    [...]
    <DS>2</DS>
   <NAME>_lib_init_rw</NAME>
    [...]
```

Les tableaux qui sont susceptibles de nous intéresser sont donc **DS[]** qui indique le DS, **NAME[]** qui indique le nom du **Datasource** et, dans notre cas, le nom du système de fichiers, éventuellement **WARN[]** et **CRIT[]** pour afficher des lignes horizontales indiquant les valeurs de **warning** et **critical**.

Maintenant que vous voilà familiarisé avec les modèles, nous allons passer à la pratique. Il existe deux méthodes pour réaliser vos modèles, soit en partant de **rrdtool graph**, soit en s'inspirant des modèles existants (dans le répertoire **templates.dist/**. Le mieux est de mixer les deux méthodes. Quant à nous, un article se prêtant mal à la présentation des modèles existants (nous n'allons pas recopier le code de PNP4Nagios), nous allons approfondir la méthode **rrdtool graph**.

9 UN PEU DE RRDTOOL

Nous disposons des fichiers **rrd** et des informations (en particulier les DS). Voici une ligne de commande pour générer un graphe des 4 dernières heures pour le système de fichiers « / » (le DS correspondant est donc 1 – voyez l'extrait de fichier XML ci-dessus) :

```
NOW=`date "+%s"`
YESTERDAY=`expr $NOW - 14400`
RRDFILE=/opt/pnp4nagios/data/rrd/localhost/Disk_Space.rrd
```

```
rrdtool graph /tmp/Disk_Space.png \
        --start $YESTERDAY --end $NOW \
        --imgformat PNG \
DEF:var=$RRDFILE:1:AVERAGE \
LINE1:var#880000:"/ " \
GPRINT:var:AVERAGE:"%6.0lf Mb\n"
```

Ce script génère une image au format PNG avec la courbe d'utilisation du système de fichiers « / ». Nous affichons

non pas du jaune comme le fait PNP4Nagios, mais une ligne (**LINE1** au lieu de **AREA**).

Transcrivons cette ligne de commande en modèle pour PNP4Nagios. Nous n'allons, par contre, pas nous limiter à un seul système de fichiers, mais nous allons au contraire tous les représenter.

```
01 <?php
02 $n = 1;
03 foreach ($DS as $i) {
04     if(
05         ($NAME[$i] != "_lib_init_rw") &&
06         ($NAME[$i] != "_dev") &&
07         ($NAME[$i] != "_dev_shm")
08        ) {
09         $opt[$n] = "";
10
11         $ds_name[$n] = $NAME[$i];
12
13         $def[$n] = "DEF:var=$rrdfile:$DS[$i]:AVERAGE " ;
14         $def[$n] .= "LINE1:var#880000:\"$NAME[$i]\"\g " ;
15         $def[$n] .= "GPRINT:var:AVERAGE:\"\t/ %6.0lf / $MAX[$i]
$UNIT[$i]\\n\" " ;
16         $n++;
17     }
18 }
19 ?>
```

Tous les représenter ? Non, nous avons profité des lignes 3 à 7 pour effectuer un test et éliminer les systèmes de fichiers qui ne nous importent guère. Nous avons également utilisé la variable **$MAX[]** pour afficher la taille maximale de chaque système de fichiers dans la légende (**GPRINT**) et **$UNIT[]** pour l'unité. Notez également que nous devons échapper les guillemets de RRDtool, car ils s'inscrivent dans une chaîne de caractères PHP.

Sauvegardez ce fichier modèle dans **check_all_disks.php** et rafraîchissez l'affichage des graphes de vos systèmes de fichiers. Le jaune devrait disparaître au profit de la ligne et les trois systèmes de fichiers que nous avons éliminés ne devraient plus apparaître.

10 AMÉLIORATION D'UN MODÈLE

Nous programmons en PHP. Par conséquent, il est facile de profiter du langage pour améliorer le modèle. Nous allons tout d'abord remettre les *slashs* que RRDtool nous remplace par des tirets bas (*underscore*). Voici le nouveau code :

```
11         $name = preg_replace('/_/', '/', $NAME[$i]);
12         $ds_name[$n] = $name;
13
14         $def[$n] = "DEF:var=$rrdfile:$DS[$i]:AVERAGE " ;
15         $def[$n] .= "LINE1:var#880000:\"$name\"\g " ;
16         $def[$n] .= "GPRINT:var:AVERAGE:\"\t/ %6.0lf / $MAX[$i]
$UNIT[$i]\\n\" " ;
```

Nous allons également remettre la légende verticale (l'unité), le titre du graphe et préciser que nous voulons un graphe dont l'origine est 0. Pour cela, ligne 9, mettez ceci :

```
09         $opt[$n] = "--vertical-label '$UNIT[$i]' -l0 --title
\"Filesystems for $hostname\" ";
```

Les capacités des disques durs étant plutôt de l'ordre du gigaoctet, et, partant du principe que le greffon Nagios retourne systématiquement des mégaoctets, nous allons modifier l'unité. RRDTool nous permet cela avec la directive **CDEF** qui consistera, dans notre cas, à diviser les valeurs par 1024 à la volée. Rappelons que RRDTool utilise la notation polonaise inverse (RPN en anglais) et que la ligne **CDEF:cvar=var,1024,/** signifie ceci : prends la variable **var**, prends **1024** et effectue une division avec ce que tu viens de prendre. Le code devient alors :

```
09         $opt[$n] = "--vertical-label 'Gb' -l0 --title \"Filesystems
for $hostname\" ";
[...]
14         $def[$n] = "DEF:var=$rrdfile:$DS[$i]:AVERAGE " ;
15         $def[$n] .= "CDEF:cvar=var,1024,/ " ;
16         $def[$n] .= "LINE1:cvar#880000:\"$name\"\g " ;
17         $def[$n] .= "GPRINT:var:AVERAGE:\"\t/ %6.0lf / $MAX[$i]
$UNIT[$i]\\n\" " ;
```

Notez que le graphe (directive **LINE1** et option **--vertical-label**) et la légende (**GPRINT**) sont bien distincts. Le graphe est passé en gigaoctet alors que la légende reste en mégaoctets. Vous pouvez évidemment modifier cela pour afficher des Go partout.

Rassemblons maintenant les courbes sur deux graphes, l'un pour les capacités supérieures à 10 Go et l'autre pour les autres. Nos tableaux vont se limiter à deux éléments en fonction du seuil. Nous allons de plus utiliser un tableau temporaire **$line[]**. Voici le code de notre nouveau modèle :

```
01 <?php
02 $seuil = 10000;
03
04 $ds_name[1] = ""; $def[1] = ""; $line[1] = "";
05 $ds_name[2] = ""; $def[2] = ""; $line[2] = "";
06 $opt[1] = "--vertical-label 'Gb' -l0 --title \"Filesystems for $hostname\" ";
07 $opt[2] = "--vertical-label 'Gb' -l0 --title \"Filesystems for $hostname\" ";
08 $color[1] = "#880000";
09 $color[2] = "#FF8000";
10 $color[3] = "#00FF30";
11 $color[4] = "#22A0FF";
12 $color[5] = "#A00088";
13 $color[6] = "#A0B088";
14
15 $nc[1] = 1; $nc[2] = 1;
16
```

```
17 foreach ($DS as $i) {
18     if(
19         ($NAME[$i] != "_lib_init_rw") &&
20         ($NAME[$i] != "_dev") &&
21         ($NAME[$i] != "_dev_shm")
22        ) {
23         if($MAX[$i] <= $seuil) {
24           $n = 1;
25         } else {
26           $n = 2;
27         }
28
29         $name = preg_replace('/_/', '/', $NAME[$i]);
30         if($name == "/") $name = "/\t";
31         $ds_name[$n] .= " $name";
32
33         $def[$n] .= "DEF:var$nc[$n]=$rrdfile:$DS[$i]:AVERAGE " ;
34         $line[$n] .= "CDEF:cvar$nc[$n]=var$nc[$n],1024,/ " ;
35         $line[$n] .= "LINE1:cvar$nc[$n]".$color[$nc[$n]].":\"$name\"\g " ;
36         $line[$n] .= "GPRINT:var$nc[$n]:AVERAGE:\"\t%6.0lf /
$MAX[$i] $UNIT[$i]\\n\" " ;
37         $nc[$n]++;
38     }
39 }
40 $def[1] .= $line[1];
41 $def[2] .= $line[2];
42 ?>
```

Vous constatez que nous avons très simplement adapté le modèle précédent en modifiant la gestion de l'index `$n`. Celui-ci n'est plus un compteur, mais une valeur dans un ensemble fini, 1 ou 2 dans notre cas. Par ailleurs, avec plusieurs courbes sur un même graphe, nous sommes obligés à deux adaptations. L'une est le nom de chaque courbe, qui ne peut plus être `var`. Nous gérons un compteur de courbes, dans le tableau `$nc[]`. La première courbe s'appelle donc `var1`, la deuxième `var2` et ainsi de suite, et de même pour le second graphe. L'autre adaptation est de disposer d'une couleur pour chaque courbe. Nous définissons un tableau de couleur et chaque courbe aura une couleur prise dans ce tableau (en fonction du compteur de courbes que nous venons de décrire). La ligne 34 revient donc à exécuter ceci pour la première courbe du second graphe : `$line[2] .= "LINE1:cvar1#880000:\"/data\"\g " ;` avec `$nc[2]` (le second graphe) qui vaut 1 (la première courbe).

Notez au passage la ligne 30 que nous avons ajoutée : le système de fichiers « / » porte un nom trop court, qui empêche les colonnes (affichées avec `GPRINT`) d'être alignées. Nous forçons une tabulation pour ce cas particulier.

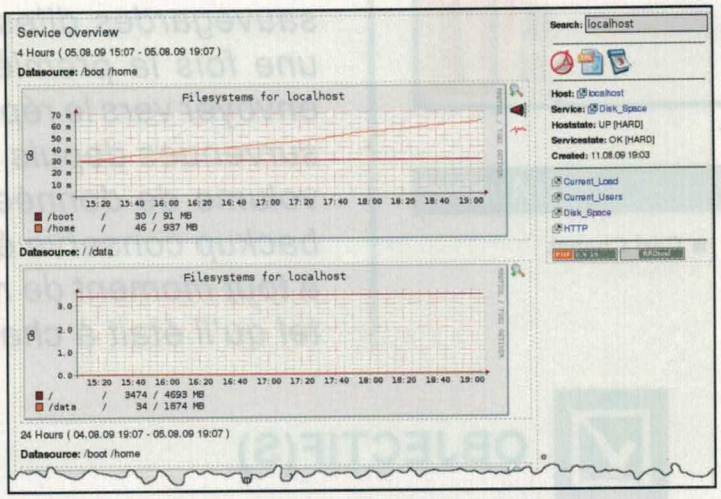


Fig. 3 : Extrait de la page des graphes du service Disk Space avec notre modèle

Remarque

La figure avec notre modèle a un seuil à 1 Go et non 10 Go, car notre serveur de test a de petits disques...

CONCLUSION

PNP4Nagios est un outil très facile à mettre en œuvre conjointement à Nagios et, somme toute, très configurable. Nous avons vu comment modifier les pages de graphes pour les adapter à nos besoins. Mais, dans la version 0.4, PNP4Nagios nous permet de définir des *pages*, à savoir des pages de graphes complètement personnalisées, où vous indiquez quels graphes de quelles bases tourniquet vous souhaitez. Cette fonctionnalité, combinée à la possibilité de créer des résultats au format PDF, vous permet de générer des rapports qui ne manqueront pas de donner le sourire à votre hiérarchie.

La version 0.6 encore en développement propose quelques nouvelles fonctionnalités, ainsi qu'une réécriture de quantités de code. Cette nouvelle mouture devrait apporter plus de robustesse, la correction de petits problèmes gênants de la version 0.4, ainsi qu'une plus grande ergonomie à l'utilisation. Cet article devenant long, nous n'avons pas abordé ce sujet.

Enfin, signalons le projet Icinga, un *fork* récent de Nagios auquel participe entre autres Jörg Linge, l'auteur de PNP4Nagios. En effet, Icinga devrait proposer PNP4Nagios comme extension (ainsi que d'autres) facile à intégrer. ■

Références

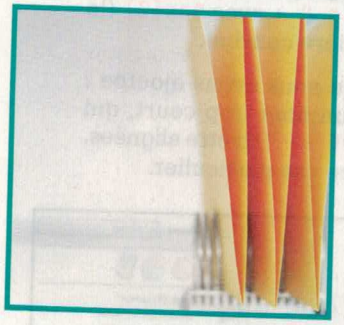
- Site de PNP4Nagios : http://www.pnp4nagios.org/
- *Nagios Plugins Developer Guidelines* : http://nagiosplug.sourceforge.net/developer-guidelines.html
- Site de Nagios : http://www.nagios.org/
- Site de Icinga : http://www.icinga.org

Auteur : Yves Mettier

Auteur de « C en action » paru chez O'Reilly

Auteur du guide de survie « Langage C » paru chez Pearson

Vos sauvegardes incrémentales



Auteur

- Carl Chenet

Le programme Rdiff-backup permet de copier l'intégralité d'un répertoire vers un répertoire de sauvegarde, que ce dernier soit sur votre machine locale ou sur un autre ordinateur accessible à travers le réseau. Rdiff-backup produit des sauvegardes dites « incrémentales », ce qui signifie que, une fois la première copie effectuée, Rdiff-backup ne va envoyer vers le répertoire de sauvegarde que les différences survenues depuis la dernière sauvegarde, rendant ainsi le volume de données à sauvegarder plus léger. Mais, rdiff-backup conserve également ces différences, ce qui permet à tout moment de retrouver l'état de votre répertoire source tel qu'il était à chaque sauvegarde effectuée.

OBJECTIF(S)

Les objectifs sont ici nombreux. Se débarrasser de sa culpabilité. Passer des nuits calmes et sereines. Sauver le monde... Tout cela en appliquant le seul et unique principe qui permet de garantir la sécurité de ses données et celles de son entreprise : faire des sauvegardes et le faire de manière intelligente. Rappelons-le, une sauvegarde est la seule solution pour se mettre en sécurité et « penser à mettre en place un système de sauvegarde » n'est pas suffisant : il faut le faire et éprouver sa solution.

OUTIL(S) UTILISÉ(S)

Rdiff-Backup est une solution éprouvée de plus de 7 ans d'âge. Celle-ci est, depuis peu, souvent vue comme une solution Windows, car largement utilisée sur cette plateforme depuis l'arrivée d'une version native pour le système. Cependant, Rdiff-Backup est bien plus testé pour les environnements POSIX comme GNU/Linux et permettra de procéder efficacement à des sauvegardes incrémentales. Il s'agit d'une solution tout-en-un contrairement aux « scripts maison » utilisant Unison ou encore Rsync.

1 INSTALLATION DE RDIFF-BACKUP

Sous Debian, rien de plus facile :

```
$ aptitude install rdiff-backup
```

Votre machine est maintenant équipée d'un système fonctionnel de sauvegarde incrémentale.

2 EFFECTUER SES SAUVEGARDES

Prenons l'exemple suivant :

```
$ tree source/
source/
|-- bar
|   `-- foo3
|-- foo1
`-- foo2

1 directory, 3 files
$
```

Nous avons donc un répertoire nommé source qui contient lui-même un répertoire et des fichiers.

Nous allons sauvegarder ce répertoire source vers un répertoire nommé **backup** :

```
$ rdiff-backup source backup
$ tree backup/
backup/
|-- bar
|   `-- foo3
|-- foo1
|-- foo2
```

avec Rdiff-backup

```
`-- rdiff-backup-data
    |-- backup.log
    |-- chars_to_quote
    |-- current_mirror.2009-09-03T16:29:03+02:00.data
    |-- error_log.2009-09-03T16:29:03+02:00.data
    |-- extended_attributes.2009-09-03T16:29:03+02:00.snapshot
    |-- file_statistics.2009-09-03T16:29:03+02:00.data.gz
    |-- increments
    |   `-- bar
    |-- mirror_metadata.2009-09-03T16:29:03+02:00.snapshot.gz
    `-- session_statistics.2009-09-03T16:29:03+02:00.data

4 directories, 11 files
$
```

Nous constatons que notre sauvegarde a été correctement effectuée. Un répertoire dont le contenu est géré par Rdiff-backup a également été créé.

Nous allons maintenant modifier notre répertoire **source** :

```
$ echo "Linux Mag" > source/foo1
$ echo "Python c'est bon" > source/foo2
$ mkdir source/bar2chocolat
$ rdiff-backup source backup
$
```

Nous avons modifié deux fichiers et créé un répertoire supplémentaire. Les changements ont dû s'effectuer dans notre répertoire de sauvegarde, mais nous allons vérifier :

```
$ rdiff-backup --compare-at-time now source backup
No changes found.  Directory matches archive data.
$
```

Le programme nous annonce qu'il n'a pas constaté de différence entre les deux répertoires. La sauvegarde s'est donc bien effectuée.

Rdiff-backup est également utilisable à travers le réseau. Il tente par défaut d'établir une connexion SSH vers la machine distante afin d'y déposer les données. Nous pouvons ainsi sauver nos données sur une machine de notre réseau à travers un tunnel chiffré :

```
$ rdiff-backup source user@10.10.10.45::/home/user/backup
user@10.10.10.45's password:
$
```

La syntaxe reste très proche de ce que l'on a vu pour le fonctionnement en local.

Une option assez utile lors de vos phases de test permet de s'assurer de la compatibilité du serveur distant avec les données fournies :

```
$ rdiff-backup --test-server source carl@10.10.10.45::/home/carl
carl@10.10.10.45's password:
Testing server started by:  ssh -C carl@10.10.10.45 rdiff-backup
--server
Server OK
$
```

3 RESTAURER UNE SAUVEGARDE

Nous entrons maintenant dans le cœur du sujet. Notre fichier **foo1** ne nous plaît pas. Nous préférions sa première version. Comment retrouver l'état de notre première sauvegarde ?

```
$ rdiff-backup --force --restore-as-of 1B backup/foo1 source/foo1
$ cat source/foo1
$
```

Notre fichier était en effet vide à la première session de sauvegarde. Le résultat est cohérent.

L'option **--restore-as-of** prend ici une date ou un numéro de session de sauvegarde. **0B** étant l'état courant, **1B** représente l'état précédent. L'option **--force** est ici utilisée, car le fichier d'origine existe et Rdiff-backup ne veut pas prendre le risque d'effacer des données.

Vous n'êtes pas convaincu ? Bien, alors laissons le clavier à un stagiaire.

```
$ whoami
stagiaire
$ rm -rf source/
$
```

C'est le drame, votre répertoire source est effacé. Pas de panique :

```
$ rdiff-backup --restore-as-of now backup source
$ tree source/
source/
|-- bar
|   `-- foo3
|-- bar2chocolat
|-- foo1
`-- foo2

2 directories, 3 files
$ cat source/foo1
Linux Mag
$ cat source/foo2
Python c'est bon
$
```

Impeccable, notre arborescence est de retour et son contenu aussi. Mais, en fin de compte, vous n'êtes pas tout à fait content. Vous préfériez l'arborescence de la première sauvegarde ? Qu'à cela ne tienne :

```
$ rdiff-backup --force --restore-as-of 1B backup source
$ tree source/
source/
|-- bar
|   `-- foo3
|-- foo1
`-- foo2

1 directory, 3 files
$
```

L'arborescence telle qu'elle était lors de la première sauvegarde a été reconstituée.

4 OBTENIR DES INFORMATIONS SUR SES SAUVEGARDES

Vous êtes très occupé et ne surveillez donc pas vos sauvegardes en permanence. Rdiff-backup peut vous fournir une liste des incréments ajoutés au répertoire de sauvegarde :

```
$ rdiff-backup --list-increments backup
Found 1 increments:
    increments.2009-09-03T16:29:03+02:00.dir   Thu Sep  3 16:29:03 2009
Current mirror: Thu Sep  3 16:34:09 2009
$
```

L'option **--list-increments** nous permet de connaître le nombre, mais aussi la date et l'heure de réalisation des différents incréments dans notre répertoire de sauvegarde.

Rdiff-backup va également nous informer des événements survenus à partir d'une date donnée :

```
$ rdiff-backup --list-changed-since 1D backup
changed .
new     bar2chocolat
changed foo1
changed foo2
$
```

Nous obtenons ici l'ensemble des changements survenus sur les fichiers et répertoires depuis maintenant jusqu'à une journée en arrière (**1D** pour *one day*). D'autres syntaxes de date peuvent être fournies à Rdiff-backup. Pour les connaître, reportez-vous à la section **TIME FORMATS** de la page de manuel de Rdiff-backup.

Une option également très utile va nous permettre de trouver les modifications effectuées dans notre répertoire source qui n'ont pas encore été sauvées :

```
$ echo "Tux is cool" > source/bar/tux
$ rdiff-backup --compare source backup
changed: bar
new: bar/tux
$
```

La commande nous renvoie bien le nom du fichier nouvellement créé.

Rdiff-backup permet également d'inclure ou d'exclure explicitement des fichiers de votre sauvegarde. Dans l'exemple qui suit, nous souhaitons ignorer le fichier **source/bar/tux** :

```
$ rdiff-backup --exclude source/bar/tux source backup
$
```

Vous pouvez vérifier en parcourant le répertoire de sauvegarde que le fichier **tux** n'a pas été sauvegardé.

Rdiff-backup peut également nous fournir des statistiques à chaque session de sauvegarde :

```
$ rdiff-backup --print-statistics source/ backup
-------------[ Session statistics ]-------------
StartTime 1251992279.00 (Thu Sep  3 17:37:59 2009)
EndTime 1251992279.15 (Thu Sep  3 17:37:59 2009)
ElapsedTime 0.15 (0.15 seconds)
SourceFiles 7
SourceFileSize 39 (39 bytes)
MirrorFiles 1
MirrorFileSize 0 (0 bytes)
NewFiles 6
NewFileSize 39 (39 bytes)
DeletedFiles 0
DeletedFileSize 0 (0 bytes)
ChangedFiles 1
ChangedSourceSize 0 (0 bytes)
ChangedMirrorSize 0 (0 bytes)
IncrementFiles 0
IncrementFileSize 0 (0 bytes)
TotalDestinationSizeChange 39 (39 bytes)
Errors 0
--------------------------------
$
```

Vous obtenez ainsi un rapport détaillé et synthétique des événements survenus dans votre répertoire **source**.

Il faut également savoir que Rdiff-backup fournit un utilitaire spécialement dédié au traitement des statistiques de votre répertoire de sauvegarde, **rdiff-backup-statistics**, vous permettant ainsi d'affiner vos recherches de statistiques :

```
$ rdiff-backup-statistics backup
Processing statistics from session 1 of 6
Processing statistics from session 2 of 6
Processing statistics from session 3 of 6
Processing statistics from session 4 of 6
Processing statistics from session 5 of 6
Processing statistics from session 6 of 6
Session statistics:
-------------[ Average of 6 stat files ]-------------
ElapsedTime 0.41 (0.41 seconds)
SourceFiles 4.83333333333
SourceFileSize 22.3333333333 (22 bytes)
MirrorFiles 3.83333333333
```

```
MirrorFileSize 15.8333333333 (16 bytes)
NewFiles 2.16666666667
NewFileSize 14.6666666667 (15 bytes)
DeletedFiles 1.16666666667
DeletedFileSize 8.16666666667 (8 bytes)
ChangedFiles 1.16666666667
ChangedSourceSize 0.0 (0 bytes)
ChangedMirrorSize 0.0 (0 bytes)
IncrementFiles 3.33333333333
IncrementFileSize 89.3333333333 (89 bytes)
TotalDestinationSizeChange 95.8333333333 (96 bytes)
Errors 0
-------------------------------------------------------
Top directories by source size (percent of total)
-------------------------------------------------------
foo2 (50.7%)
bar/tux (26.9%)
foo1 (22.4%)

Top directories by increment size (percent of total)
-------------------------------------------------------
foo1 (39.6%)
foo2 (21.1%)
bar/tux (20.7%)
bar/foo3 (18.7%)
Top directories by number of files changed (percent of total)
-------------------------------------------------------------
foo1 (23.8%)
bar (14.3%)
bar/foo3 (14.3%)
bar/tux (14.3%)
foo2 (14.3%)
bar2chocolat (14.3%)
$
```

La commande **rdiff-backup-statistics** présente un rapport extensif des informations liées à vos différentes sauvegardes. Par défaut, toutes les sessions sont prises en compte pour produire le rapport. Vous pouvez bien sûr modifier ce comportement.

J'espère que cette rapide introduction à Rdiff-backup vous encouragera à le mettre en place sur vos machines. Les sauvegardes incrémentales s'avèrent très utiles pour garder une trace des modifications survenues sur un ensemble de fichiers. Citons la possibilité de sauvegarder le répertoire de fichiers de configuration **/etc** de votre serveur de production par exemple. ■

Auteur : Carl Chenet

Carl Chenet, ingénieur système GNU/Linux dans le monde bancaire, utilisateur de GNU/Linux depuis 1999. Contributeur Debian et développeur du projet Bélier.

Référence

Le site du projet Rdiff-backup : http://rdiff-backup.nongnu.org/

Quick sysadmin's tip : Un historique Bash unique pour plusieurs systèmes

Vous est-il déjà arrivé d'oublier une ligne de commande, tout aussi utile que complexe, simplement en vous disant qu'elle restera dans l'historique du shell et que vous n'avez donc pas besoin de la copier quelque part ? Que diriez-vous de partager cet historique de commandes entre vos machines et de disposer, en plus, d'un historique sans limite de taille ? C'est ce que propose Shell Sink.

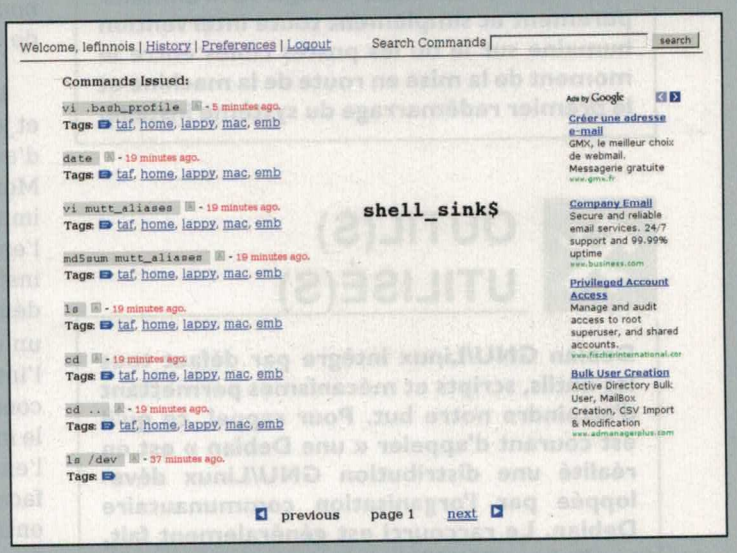
Cet outil n'est pas encore intégré dans Debian/Ubuntu, mais des paquets sources et binaires existent. Le plus simple est de récupérer le fichier en question directement sur http://ppa.launchpad.net/shellsink/ppa/ubuntu/pool/main/s/shellsink/. Bien entendu, vous pouvez également ajouter le dépôt dans votre configuration APT.

Pointez ensuite votre navigateur sur http://history.shellsink.com/. Comme vous pourrez le constater, le service utilise le web et l'API Google en particulier. Vous utiliserez donc votre compte Google (Gmail et autres) pour créer un dépôt d'historique unique. Dès l'ouverture du compte, vous obtenez un identifiant de 32 caractères hexadécimaux.

Il ne vous reste plus, ensuite, qu'à modifier votre **~/.bash_profile** d'une première machine en ajoutant :

```
shopt -s histappend
export SHELL_SINK_COMMAND=shellsink-client
export SHELL_SINK_ID=XXXXXXXXXXXXX
PROMPT_COMMAND="history -a;$SHELL_SINK_COMMAND"
export SHELL_SINK_TAGS=des:tag
```

Dès le prochain login, toutes les commandes shell que vous utiliserez seront envoyées au dépôt désigné par l'identifiant (**XXXXXXXXXXXXX**). Vous pourrez retrouver l'historique sur l'URL spécifiée précédemment. Les entrées de l'historique sont taguées en fonction du contenu de votre **SHELL_SINK_TAGS**, datées et accessibles via le web, mais également sous forme de flux RSS ou Atom.



Remarquez que ce projet est relativement récent et que bon nombre de points mériteraient d'être audités. Je pense naturellement à la sécurité de l'ensemble et l'aspect privé de la liste de commandes saisies. N'hésitez pas à faire un tour sur le site du développeur pour vous tenir au courant et apporter vos commentaires, correctifs et signalements de bugs : http://shell-sink.blogspot.com/

Une installation de Debian

Auteur

- Stephane (stephbul) Bulot

Lecteur de ce magazine, vous avez probablement eu l'occasion d'installer une Debian et, si ce n'est pas le cas, ceci ne vous effraie pas. Par conséquent, loin de moi l'idée de vous expliquer comment le faire, mais plutôt de vous raconter mon expérience d'installation d'une petite appliance que je souhaitais, si ce n'est industrielle, au moins automatique. Je précise par automatique que celle-ci se veut sans interaction de l'allumage de la machine jusqu'à son arrêt, installée, configurée. J'ouvre les cartons du hard reçu à cette occasion et je me lance.

OBJECTIF(S)

Mettre en place une solution permettant d'installer et réinstaller des systèmes Debian pour une plateforme donnée. Derrière cet objectif qui peut paraître simple, se cache en réalité toute une problématique pour peu que l'on souhaite automatiser pleinement les installations. En effet, il s'agit ici d'éliminer purement et simplement toute intervention humaine sur le ou les postes cibles entre le moment de la mise en route de la machine et le premier redémarrage du système installé.

OUTIL(S) UTILISÉ(S)

Debian GNU/Linux intègre par défaut tous les outils, scripts et mécanismes permettant d'atteindre notre but. Pour rappel, ce qu'il est courant d'appeler « une Debian » est en réalité une distribution GNU/Linux développée par l'organisation communautaire Debian. Le raccourci est généralement fait, car Debian GNU/Linux, la distribution, est la branche la plus fonctionnelle et finalisée du projet. Il faut noter cependant que Debian c'est aussi Debian GNU/Hurd ou encore Debian GNU/kFreeBSD.

1 LA MACHINE À INSTALLER

Tout d'abord, j'ai décidé d'investir dans un matériel relativement sobre : un boîtier mini-atx, une carte mère Jetway J7F4, avec un processeur C7 et deux contrôleurs Ethernet, une barrette de 1 GB de RAM et un HDD 2.5 pouces de 160 GB. Je n'ai pas encore envie de vous dire ce que j'en ferai (en ai-je vraiment un idée précise ?) et là n'est pas l'important. J'avoue toutefois que cette configuration me paraît très opportune et très peu coûteuse pour nombre d'applications diverses. On peut la trouver à l'unité autour de 260€ sans trop d'effort.

Le montage de ce type de machine est pour moi une première et je ne saurais trop, si c'est aussi le cas pour vous, vous conseiller d'avancer progressivement et de vous armer d'un multimètre. Mon enthousiasme m'a poussé à monter l'ensemble du système immédiatement et j'ai sans surprise dû me raviser et démonter l'ensemble pour progresser pas à pas. Pour ma part, la carte mère installée dans le boîtier connectée uniquement à l'alimentation démarrait sans souci. Il suffisait avec un petit tournevis de faire un court-circuit sur le connecteur de *front panel* afin de simuler l'interrupteur, mais une fois la nappe de câbles de front panel connectée, il m'était devenu impossible de démarrer le système. C'est le multimètre qui m'a permis de trouver que celle-ci était montée à l'envers et que le court-circuit généré à l'appui de l'interrupteur de face-avant se produisait au mauvais endroit. En testant l'impédance entre les broches, j'ai pu trouver où se faisait le court-circuit. Beaucoup de temps perdu à aller vite. Pour information, je peux vous donner hors antenne le nom du magasin, basé outre-manche, dont je salue la réactivité dans le support client.

Mis à part ce petit souci, le système correspond parfaitement à mes attentes. Il est maintenant temps de l'installer.

automatique

2 INSTALLATION EN RÉSEAU

Au démarrage d'une machine est exécuté un premier bout de code, le bios. Celui-ci donne la main au média sur lequel est stocké le programme à exécuter. Cela peut être un disque dur, un CD-ROM, une clé USB ou un accès réseau. En l'occurrence pour Linux, après le BIOS, est chargé le *kernel* suivi du lancement du process père : init. C'est par l'intermédiaire du réseau que je vais démarrer le processus d'installation de la machine. L'installation se fait en deux parties :

- *Boot* en réseau pour télécharger le noyau et l'initrd d'installation fournis par Debian pour réaliser une installation en réseau.
- Installation par le réseau en téléchargeant les paquets sur les miroirs Debian.

Je rappelle qu'un initrd est un système de fichiers dans lequel on trouve l'arborescence *root* et les applications minimums, ainsi que l'exécutable init lancé par le kernel.

2.1 Boot en réseau

Pour booter en réseau, il me faut configurer deux paramètres du bios, mettre le « legacy LAN » dans les médias de boot et activer le *bootrom* des Realtek Semiconductor Co., Ltd. RTL-8110SC/8169SC Gigabit Ethernet, les contrôleurs Ethernet de la machine.

Au démarrage de la machine, un client DHCP démarre pour obtenir une configuration réseau et obtenir les informations pour télécharger par TFTP les fichiers d'amorçage du système. Il est donc nécessaire d'avoir un serveur DHCP et un serveur TFTP, dont voici les configurations respectives.

J'installe **dhcp3-server**, le serveur disponible dans l'environnement Debian :

```
apt-get install dhcp3-server
```

J'applique ensuite cette configuration :

```
# Masque de sous-réseau du réseau que je vais couvrir
option subnet-mask 255.255.255.0;
# routeur à utiliser pour sortir (la freebox en fait)
option routers 192.168.0.254;
# les serveurs de DNS utilisés, trouvés dans /etc/resolv.conf
option domain-name-servers 212.27.40.241, 212.27.40.240;
default-lease-time 600;
max-lease-time 7200;
# le nom de mon serveur (mon laptop)
server-name ladymadonna;
# Description du sous-réseau adressé, option générale pour toutes les
machines utilisant ce dhcp
subnet 192.168.0.0 netmask 255.255.255.0 {
   # la range d'adresses ip que le serveur va distribuer
   range 192.168.0.100 192.168.0.200;
   option subnet-mask 255.255.255.0;
   # le fichier d'amorçage de boot en réseau qui va permettre d'aller
chercher le noyau et l'initrd à charger.
    filename "pxelinux.0";
    server-name ladymadonna;
    # le serveur suivant (tftp utilisé, toujours mon laptop)
    next-server 192.168.0.7;
    # l'adresse de broadcast
    option broadcast-address 192.168.0.255;
}

#Configuration spécifique par host, ici scalie est le nom de la machine
à installer
host scalie {
    #fichier d'amorçage, pas obligatoire puisque déjà plus haut
    server-name ladymadonna;
    next-server 192.168.0.7;
    # Adress MAC de la machine, afin que le serveur DHCP puisse la
reconnaître et appliquer cette configuration
    hardware ethernet 00:30:18:A7:89:E0;
    # adresse donnée par ce serveur dhcp
    fixed-address 192.168.0.101;
}
```

Attention à bien veiller à ne pas avoir d'autre serveur DHCP sur le réseau. Par conséquent, j'ai désactivé celui de ma Freebox. De plus, pour s'assurer de ne pas perturber le réseau une fois celui-ci réactivé, je supprime l'activation de **dhcp3-server** au démarrage :

```
update-rc.d -f dhcp3-server remove
```

J'installe un serveur tftp. Celui conseillé par Debian est **tftpd-hpa** :

```
apt-get install tftpd-hpa
```

La configuration, très simple, est la suivante :

```
RUN_DAEMON="yes"
OPTIONS="-l -s /var/lib/tftpboot "
```

Ceci signifie que je mettrai tous mes fichiers d'installation dans **/var/lib/tftpboot**.

2.2 Un kernel et un initrd d'installation par le réseau

Debian fournit une méthode d'installation par le réseau composée d'un noyau et d'un initrd spécifiques. Ceux-ci sont en charge de démarrer l'installation, de télécharger les paquets sur les serveurs de Debian et de finalement les installer dans l'arborescence *chrootée* **/target**. Cette méthode nécessite le réseau, mais a le gros avantage d'être très légère. A titre d'exemple, dans le cas de l'utilisation d'un CD d'installation, l'ISO ne pèse que 40 Mo.

Je vais donc récupérer les fichiers nécessaires à une installation par le réseau que je place dans l'arborescence de mon serveur tftp :

```
cd /var/lib/tftpboot
wget http://ftp.debian.org/debian/dists/lenny/main/installer-i386/current/
images/netboot/netboot.tar.gz
tar zxvf netboot.tar.gz
cp -r /var/lib/tftpboot/debian-installer/i386/pxelinux.* /var/lib/tftpboot
```

On retrouve le fichier **pxelinux.0** pour l'amorçage et, dans le répertoire **pxelinux.cfg**, la configuration du boot pxe pour pouvoir installer différents systèmes d'exploitation. J'ai modifié le fichier **default** comme suit :

```
DEFAULT debian-i386
LABEL debian-i386
   kernel debian-installer/i386/linux
    append vga=normal initrd=debian-installer/i386/initrd.gz root=/dev/rd/0 rw --
PROMPT 0
TIMEOUT 0
```

Dans mon cas, je n'ai qu'un mode d'installation que j'appelle **debian-i386**. Les tags sont assez explicites. Dans **kernel**, on donne le chemin vers le noyau et dans **append**, on ajoute les autres options :

- **vga** : gestion de l'affichage ;
- **initrd** : chemin d'accès vers l'initrd sur le serveur TFTP ;
- **root** : point de montage du *root file system* du *netboot* sur la machine en cours d'installation.

Afin d'effectuer l'installation, il suffit de démarrer le serveur DHCP et le serveur TFTP, et d'allumer la machine. C'est une solution classique souvent utilisée nécessitant la présence devant l'écran et le clavier, et je crois même déceler une pointe d'ennui chez certains, si, si, là, au fond de la rame, oui, vous ! Regardons donc maintenant comment automatiser la procédure.

3 LA CONFIGURATION AUTOMATIQUE OU LA PRÉ-CONFIGURATION

Mon objectif est de pouvoir installer et réinstaller un certain nombre de machines en ayant juste à les allumer, l'installation de celles-ci se terminant par leur extinction, sans avoir à répondre aux questions posées sur la langue, le fuseau horaire et autres.

Ce n'est ni plus ni moins que le pendant du *kickstart* de Redhat offert par Debian que je me propose de vous présenter. Cette méthode se nomme le *preseeding* et utilise le fichier **preseed.cfg**. Le principe de base de ce fichier est de répondre aux questions posées par l'installeur et par les différents logiciels installés éventuellement.

Cette méthode est disponible dans les différents modes d'installation que ce soit par clé USB, par CD-ROM, à condition de recréer soi-même l'ISO, ou par le réseau. Par ailleurs, l'installeur permet de récupérer de façon différente ce fichier **preseed.cfg**, soit en utilisant le paramètre **append** de l'amorçage, comme vu plus haut, soit, et c'est la méthode que j'utilise, en le plaçant directement dans l'initrd.

Nous verrons plus tard comment faire, mais regardons comment on écrit ce fichier **preseed.cfg**.

3.1 Le fichier de pré-configuration preseed.cfg

C'est un simple fichier texte de réponse aux questions d'installation. Cela recouvre à titre d'exemple :

- La localisation de l'installation, c'est-à-dire, la langue et le pays de l'installation. La possibilité de configurer automatiquement ce paramètre n'est disponible qu'en plaçant le **preseed.cfg** dans l'initrd, car cette option ne fonctionne pas avec les autres méthodes, le fichier **preseed.cfg** n'étant alors chargé qu'après ces questions. Par conséquent, bien que légèrement moins souple dans la phase de test et de conception du fichier de pré-configuration, la méthode de l'initrd est la seule qui me permet d'être pleinement automatique. Dans la suite de l'article, nous verrons comment différencier l'installation en fonction de l'hôte à installer.
- La configuration réseau.
- Le choix des miroirs d'installation.
- Le partitionnement.
- Le fuseau horaire.
- La configuration des *users* et de leurs mots de passe.
- Le choix du type d'install (standard, *desktop*, serveur d'impression, serveur web).

Chaque option a ce format

```
d-i netcfg/get_hostname string xxxxx
```

où les champs s'expliquent ainsi :

- **d-i** signifie **debian-installer**. Nous verrons que cela peut être aussi un autre logiciel.
- **netcfg/get_hostname** indique qu'il s'agit de la configuration réseau, et de donner le *hostname* de la machine.
- **string** indique qu'il s'agit d'une chaîne de caractères. Cela peut aussi être un booléen par exemple ou une liste de choix ou encore une simple notification (en installation normale, il faut choisir de continuer).
- **xxxxx** est la chaîne de caractères, correspondant ici au hostname.

Les champs sont espacés d'une seule espace ou d'une seule tabulation.

Pour information, l'installeur charge le fichier en une seule fois. Il n'y a pas d'ordre à respecter dans la séquence. En général, un certain ordre apparaît dans un fichier **preseed**, ordre respectant plus ou moins la séquence des questions, mais cela ne répond en aucun cas à un besoin technique.

Il est maintenant temps de passer aux choses sérieuses. Après quelques hésitations, j'en suis arrivé à un premier fichier **preseed.cfg**. Le voici :

```
#choix de la langue d'installation
d-i debian-installer/locale string fr_FR
# choix du clavier (select indique une liste de choix)
d-i console-keymaps-at/keymap select fr
# choix de l'interface automatique
d-i netcfg/choose_interface select auto
# configuration du hostname
d-i netcfg/get_hostname string scalie
# configuration du domaine
d-i netcfg/get_domain string bulot.org
# configuration du réseau en statique
d-i netcfg/disable_dhcp boolean true
# configuration du serveur de nom
d-i netcfg/get_nameservers string xxxxxx
# configuration du serveur de nom
d-i netcfg/get_ipaddress string 192.168.0.101
# configuration du masque sous-réseau
d-i netcfg/get_netmask string 255.255.255.0
# configuration de la gateway
d-i netcfg/get_gateway string 192.168.0.254
# confirmation de la configuration statique
d-i netcfg/confirm_static boolean true
# désactivation des demande de clé wep
d-i netcfg/wireless_wep string
# type de rpotcole pour accéder aux mirroirs
d-i mirror/protocol string ftp
# choix du mirroir
d-i mirror/ftp/hostname string ftp.fr.debian.org
# choix du répertoire dans le mirroir
d-i mirror/ftp/directory string /debian
d-i mirror/ftp/proxy string
# Type de distribution (stable testing...)
d-i mirror/suite string stable
# configuration du fuseau horaire
d-i clock-setup/utc boolean true
d-i time/zone string Europe/Paris
# Pas de serveur ntp fixé
d-i clock-setup/ntp boolean false
# choix du partionnement par défaut
d-i partman-auto/method string regular
d-i partman-auto/choose_recipe select atomic
d-i partman/confirm_write_new_label boolean true
d-i partman/choose_partition select finish
d-i partman/confirm boolean true
#configuration du compte root
d-i passwd/root-password password bl1blu
d-i passwd/root-password-again password bl1blu
#configuration d'un autre user
d-i passwd/user-fullname string opscale
d-i passwd/username string opscale
d-i passwd/user-password password azerty
d-i passwd/user-password-again password azerty
# ajout de non-free et contrib dans /etc/apt/sources.list
d-i apt-setup/non-free boolean true
d-i apt-setup/contrib boolean true
# Sélection d'une install standard, sans desktop
tasksel tasksel/standard
# ajout des packages supplémentaires
d-i pkgsel/include string openssh-server build-essential
# installation de grub
d-i grub-installer/only_debian boolean true
d-i grub-installer/with_other_os boolean true
d-i finish-install/reboot_in_progress note
# extinction de la machine après installation
d-i debian-installer/exit/poweroff boolean true
```

Afin d'économiser du temps et de l'énergie en installant un **preseed.cfg** dans votre initrd et de le tester lors de l'installation, il est grandement conseillé de vérifier sa syntaxe. C'est la suite Debconf, le logiciel de gestion de la configuration de Debian qui vous aidera en ce sens :

```
debconf-set-selections -c preseed.cfg
```

La syntaxe est vérifiée ainsi que l'utilité de certaines commandes. Sans erreur, il est maintenant temps de le déposer, comme je vous le disais, à la racine de l'initrd. Un fichier d'initrd est un système de fichiers compressé le plus souvent avec les utilitaires **cpio** et **gzip**. Voici comment je procède pour le décompresser

```
cp /var/lib/tftp/tftpboot/debian-installer/i386/initrd.gz /tmp
mkdir /tmp/out
cd /tmp/out
gunzip -dc /tmp/initrd.img.gz | cpio -id --no-absolute-filenames
```

où pour cpio :

- **-i** est la commande d'extraction.
- **-d** crée les répertoires nécessaires.
- **--no-absolute-filenames** crée tous les chemins absolus de l'archive par rapport au chemin actuel.

Ensuite, je copie le fichier **preseed.cfg** dans la racine de l'archive décompressée :

```
cp /home/stephbul/debian-install/preseed.cfg /tmp/out/
```

Et je fais le chemin inverse. Je recompresse le répertoire **/tmp/out** pour en faire un initrd

```
find . | cpio -o -H newc | gzip > /var/lib/tftpboot/debian-installer/
i386/initrd.gz
```

où pour cpio :

- **-o** crée l'archive.
- **-H newc** indique le format d'archive. L'option **newc** permet d'utiliser un file system avec plus de 65536 inodes.

Je vous conseille la doc de cpio sur le site de GNU pour la manipulation de l'initrd avec cpio : **http://www.gnu.org/software/cpio/manual/cpio.html** Je remplace l'initrd du Debian installer par celui-ci, qui est le même enrichi du **preseed.cfg** à la racine.

4 CAFÉ OU THÉ ? EUH, BIÈRE POUR MOI !

Voilà, nous allons démarrer l'installation. On peut éteindre l'écran. Enfin, en théorie seulement, car, je l'avoue, il faut faire parfois plusieurs essais, car il n'est pas rare d'oublier de configurer la réponse à une question. Lors de l'absence d'une réponse dans la liste du fichier **preseed**, l'installeur se comporte comme en mode habituel en interrompant

l'installation le temps qu'on lui réponde pour reprendre ensuite son installation automatique.

Sur le serveur dhcp :

```
/etc/init.d/dhcp3-server start
```

```
/etc/init.d/tftpd-hpa start
```

On peut maintenant appuyer sur le bouton « *on* » de la machine, à une seule condition : savoir si on veut un café ou un thé, car, maintenant, le travail est terminé, jusqu'à ce que la machine s'éteigne, sa façon à elle de dire « je suis prête ».

5 ENRICHISSONS UN PEU NOTRE INSTALLATION

Mon utilisation du preseeding est assez pratique, mais, jusqu'à présent, assez basique. Une fois les différentes manipulations digérées, je regarde le chemin parcouru, les mots de PJ Harvey « *Is that all there is?* » surgissent et je me dis que ce n'était pas bien compliqué. D'ailleurs, je me connecte à la machine fraîchement installée et je constate que je ne suis qu'au milieu du gué, car il me reste un certain nombre de choses à configurer.

Ma machine n'utilise que **fr.UTF-8** comme langue chargée, ce qui est, je trouve, particulièrement désagréable. Il est vrai que, à lire ces mots, vous devez penser qu'une reconfiguration est particulièrement simple :

```
dpkg-reconfigure locales
```

Ceci me permet d'installer les locales iso-8859-1 et iso-8859-15, mais qu'advient-il de ma configuration automatique, si, dès le *login*, je dois déjà reconfigurer quelque chose ? Nous allons le voir.

Nous avons vu qu'il était nécessaire de vérifier la syntaxe du fichier **preseed** avec la suite Debconf, et plus particulièrement avec **debconf-set-selections**. Dans le même esprit, je vais utiliser **debconf-get-selections** fourni par le paquet **debconf-utils** pour retrouver la nouvelle configuration appliquée.

```
apt-get install debconf-utils
```

Et ensuite pour retrouver la configuration associée aux locales :

```
debconf-get-selections | grep locales
```

Ceci me donne :

```
locales locales/default_environment_locale      select  fr_FR@euro
locales locales/locales_to_be_generated multiselect     fr_FR.UTF-8 UTF-
8, fr_FR@euro ISO-8859-15
```

Vous reconnaissez le format du **preseed.cfg** ou, plus précisément, vous avez compris que le preseeding utilisait la formalisation de la suite Debconf de Debian. Ces deux lignes sont à ajouter au **preseed.cfg** précédent.

Un autre exemple. J'installe le serveur dhcp3 (**dhcp3-server**) sur la machine. Comme vu plus haut :

```
apt-get install dhcp3-server
```

Je suis notifié que « la version 3 du serveur DHCP ne fait plus autorité par défaut », et je dois appuyer sur OK pour continuer. Ceci signifie que le simple ajout du paquet à la ligne de sélection de paquets supplémentaires de mon fichier d'origine ne va pas suffire et va rompre la belle automaticité de mon installation.

```
d-i pkgsel/include string openssh-server build-essential dhcp3-server
```

Par conséquent, après avoir installé le serveur dhcp avec apt, je passe la commande suivante :

```
debconf-get-selections | grep dhcp3-server
```

Ceci me donne :

```
dhcp3-server    dhcp3-server/new_auth_behavior  note
dhcp3-server    dhcp3-server/interfaces string
dhcp3-server    dhcp3-server/new_next-server_behaviour  note
dhcp3-server    dhcp3-server/config_warn        note
```

Comme précédemment, j'ajoute ces quelques lignes à mon fichier.

Ces opérations sont à répéter pour chacun des *packages* nécessitant une interaction lors de l'installation. Attention, lecteur attentif, vous percevez peut-être le léger inconvénient de la méthode : en effet, nous sommes en train d'installer des paquets par le réseau. Si la génération du paquet est modifiée et que le mainteneur de celui-ci pose une question qui n'existait pas précédemment, l'automatisation ne fonctionnera plus. Ce type de modification a, fort heureusement, rarement lieu, si ce n'est jamais, dans la branche stable. Dans un contexte industriel de déploiement de masse, cet inconvénient est par conséquent amoindri, car il serait dangereux d'utiliser dans ce cas une branche « *testing* » ou « *unstable* ».

6 PLUSIEURS FICHIERS DE PRESEEDING

Il est possible de chaîner les fichiers de preseeding et ainsi d'en charger différents en fonction de certains paramètres spécifiques. La fonction **preseed/run** permet d'exécuter un script. Il faut que le résultat du script ait en sortie un chemin vers un autre fichier de type **preseed.cfg** Ceci permet d'effectuer d'autres opérations de type **preseed**. Attention, **debian-installer** charge l'ensemble des fichiers **preseed.cfg** et, en cas de redondance d'opération, c'est la dernière qui est prise en compte. Voyons un exemple concret :

```
d-i preseed/run string myscript.sh
```

Dans ce cas, il suffit de placer **myscript.sh** comme le **preseed.cfg** dans l'initrd. L'opération de preseeding exécute **myscript.sh** dont la sortie doit être le fichier à charger. Exemple :

```
#!/bin/sh
#Récupération de la mac et génération d'un nom de fichier preseed
PFILE=`ifconfig -a | grep -E "eth1.*HWaddr" | awk '{print $5}'`.cfg
#Téléchargement de ce fichier preseed.cfg
wget http://192.168.0.7/preseed/$PFILE -o /dev/null
#écriture du nom de fichier à charger
echo $PFILE
```

Ceci va vous permettre de télécharger les fichiers **xx:xx:xx:xx:xx:xx.cfg** où **xx:xx:xx:xx:xx:xx** est la MAC de la machine et ainsi de faire un fichier **preseed** différent en fonction de la MAC de la machine. Ceci est particulièrement intéressant pour affiner une configuration réseau en fonction de cette MAC ou encore de créer des utilisateurs différents.

7 COMMENT RÉALISER LA POST-CONFIGURATION ?

Le fichier **preseed.cfg**, comme nous l'avons vu, permet d'éviter de répondre à des questions. Mais, il est nécessaire, pour avoir un process industriel sérieux, de réaliser la post-configuration de la machine à la fin de l'installation avant le premier boot de la machine.

7.1 Exécution d'un script de post-install

Une dernière fonctionnalité très utile de la méthode **preseed** est l'exécution d'une commande de post-installation, dite **late_command**. C'est un peu la cerise sur le gâteau Debian, car elle parfait l'ensemble du processus. Attention, la différence avec la commande précédente (**preseed/run**) est de taille. La précédente permet de charger un autre fichier **preseed**, celle-ci permet d'exécuter une fonction tout autre, sans rapport avec l'installation à proprement parler des paquets. Je m'en sers personnellement pour faire de la post-installation.

La commande que j'ajoute à mon fichier **preseed** est la suivante :

```
d-i preseed/late_command string in-target wget http://192.168.0.7/post-install.
sh; in-target /bin/sh post-install.sh; in-target rm -f post-install.sh
```

Cette ligne mérite quelques explications :

- **preseed/late_command** : cela signifie que cette commande est effectuée à la fin de l'installation de tous les paquets sélectionnés.
- **string in-target** : en spécifiant la commande avec **in-target**, la commande est effectuée en *chroot* sur le point de montage **/target**. Ce point de montage est créé par le **debian-installer** au début de l'installation et ce point de montage est en fait celui où est déposé le système final. Il est bien entendu possible de ne pas utiliser cette option **in-target**, mais cela signifie que la commande est effectuée dans le répertoire racine du système en cours, qui est en fait la racine de l'initrd de Netinstall.

Toutes les commandes à effectuer en **late_command** sont sur la même ligne, car, comme pour les autres commandes du **preseed**, la dernière écrase les précédentes.

Par ailleurs, pour les deux exemples cités à l'instant, vous aurez noté la nécessité d'utiliser un serveur web. Pour ma part, j'utilise Lighttpd sans configuration spécifique.

```
apt-get install lighttpd
```

Et j'installe dans **/var/www** mon fichier **post-install.sh**, dont le contenu est particulièrement simple pour le moment :

```
#!/bin/sh
#
# ssh post-install
sed -i 's/PermitRootLogin\ yes/PermitRootLogin\ no/' /etc/ssh/sshd_config
```

Cela pour interdire l'accès en ssh de root.

CONCLUSION

Comme je l'expliquais au début de l'article, je n'ai pas encore une idée très précise de ce que je ferai de cet équipement, mais je me sais armé pour faire de l'installation et de la configuration de masse et, si vous ne connaissiez pas cette solution, j'espère vous avoir éclairé.

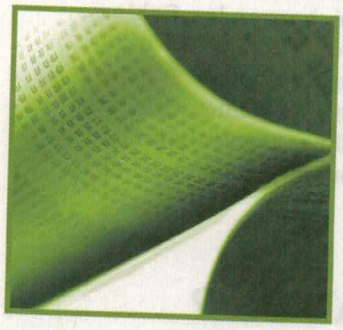
Personnellement, je pense que c'est un outil précieux qui ouvre des horizons non seulement à celui qui souhaite faire du déploiement, mais aussi à celui qui souhaite simplement pouvoir installer une machine plusieurs fois avec la même configuration. Et, au moins dans ce cas, qui ne se sent pas concerné ? ■

Références

- *Debian Installation manual* : http://d-i.alioth.debian.org/manual/en.i386/apbs01.html
- *cpio manual* : http://www.gnu.org/software/cpio/manual/cpio.html
- *Debian installer wiki* : http://wiki.debian.org/DebianInstaller
- *Setting up a server for PXE network booting* : http://www.debian-administration.org/articles/478

Auteur : Stephane (stephbul) Bulot

389 Directory Server as Bind 9



Auteur

- Fabien Dupont

Dans un effort de consolidation de la gestion de configuration au sein d'une équipe d'ingénierie système, je me suis penché sur le cas du service DNS. Et plus particulièrement sur la possibilité de stocker ses données au sein d'un annuaire LDAP. En effet, en y réfléchissant quelques instants, les données d'un service DNS sont principalement une liste de correspondance entre des noms de machines et son adresse IP ; ce qui ressemble fortement à un annuaire téléphonique : des noms en face de numéros de téléphone. Et donc, je me suis lancé dans l'expérimentation avec le serveur Bind 9 et l'annuaire 389 Directory Server. Je vous livre ici les résultats.

OBJECTIF(S)

Installer et déployer un service DNS (Domain Name System, système de noms de domaine) reposant sur un annuaire LDAP. Cette pratique, qui peut paraître atypique, est pourtant parfaitement logique. Lightweight Directory Access Protocol ou LDAP est un protocole permettant l'interrogation et la modification des services d'annuaire. Or, quoi de plus normal que d'utiliser un annuaire et un protocole standardisé pour stocker des correspondances nom/IP comme c'est le cas pour un DNS ?

OUTIL(S) UTILISÉ(S)

Bind pour Berkeley Internet Name Domain est le serveur DNS le plus utilisé sur internet et donc forcément avec les systèmes de type UNIX. Bind est, de fait, un standard pour la mise en œuvre de ce genre de solutions, même si d'autres projets lui font concurrence ces dernières années. La première version de Bind a été conçue sur la base de 4.3BSD en 1988. 389 Directory Server est un serveur LDAP initialement développé par Red Hat par le projet communautaire Fedora. Il s'agit donc d'une solution identique au produit Red Hat Directory Server. Le choix de cet annuaire suggère donc l'utilisation de Fedora Core 11 dans un but de moindre effort.

Pour la réalisation de cet article, je me suis appuyé sur une distribution **Fedora Core 11 (Leonidas)** [1] fraîchement installée avec le minimum de *packages* (en gros le groupe **Core**). Et comme nous sommes dans le monde Fedora, j'ai choisi le serveur d'annuaire proposé par la communauté Fedora : **389 Directory Server** [2]. Pour le serveur DNS, je suis resté sur le très classique, et efficace, **Bind 9** [3].

Note

Dans la suite de l'article, je parlerai de plusieurs produits issus des communautés Fedora et Mozilla. N'y voyez pas un quelconque prosélytisme de ma part, ni le signal d'une chasse au troll des montagnes. Plusieurs produits commerciaux de la société RedHat sont basés sur ces produits communautaires et je travaille en environnement RedHat. Et je ne touche pas le moindre euro de leur part ;)

Pour notre exemple, l'architecture est relativement simple et je me passerai de schéma. Nous aurons 3 machines dans le même réseau **10.0.2.0/24** (les plus rusés auront reconnu le réseau par défaut de VirtualBox) : le serveur LDAP (**10.0.2.101**), le serveur DNS (**10.0.2.102**) et une machine cliente (**10.0.2.15**). Pour se simplifier la vie, l'adressage est statique. Côté nommage, le réseau sera purement interne et aura pour suffixe **localdomain**.

backend

1 389 DIRECTORY SERVER

Pour l'histoire, 389 Directory Server (389DS) est le nouveau nom de *Fedora Directory Server*, projet développé par la communauté Fedora à partir du code source de *Netscape Directory Server*, racheté en 2004, puis libéré par RedHat. Il est aussi la base du produit commercial *RedHat Directory Server*. Cet article reste donc facilement transposable au monde « professionnel » (aucun troll à chercher dans cette phrase) via les produits commerciaux de RedHat.

Quelques aspects différencient 389DS du serveur OpenLDAP, plus connu et répandu dans la communauté *open source* :

- La configuration (à quelques exceptions près) est stockée dans l'annuaire lui-même. Il est dès lors possible d'appliquer des changements de configuration à partir d'un fichier LDIF sans redémarrer le serveur.
- Le serveur gère la réplication multi-maître. Un changement apporté sur l'un des maîtres est répercuté sur les autres. Cela permet de mettre en redondance le serveur maître en quelques lignes de configuration. Une politique d'intégrité des données peut être mise en œuvre pour limiter les risques de collision.
- Le serveur fournit un ensemble de composants complémentaires pour l'administration en mode web, la synchronisation des groupes et utilisateurs avec Active Directory... Je n'utiliserai que la brique de base, l'annuaire LDAP lui-même, pour cet article. De nombreux *howtos* sont disponibles dans la documentation du projet **[4]**.

1.1 Installation et configuration de base

Sur une distribution Fedora, on peut s'attendre à ce que le logiciel soit packagé et donc accessible par une simple commande **yum**. Rassurez-vous, c'est le cas ! La preuve ci-dessous :

```
root# yum install 389-ds-base
```

Le package est installé et a mis à notre disposition les binaires pour la configuration dans l'arborescence système, ainsi que les binaires basés sur les bibliothèques LDAP développées par la communauté Mozilla **[5]** dans **/usr/lib/mozldap**, les fichiers de configuration dans **/etc/dirsrv** et les logs dans **/var/log/dirsrv**.

Nous avons installé l'environnement nécessaire au fonctionnement de 389DS. Il est temps d'appuyer sur le contact pour entendre ronronner le moteur. Pour cela, il faut régler les paramètres de notre instance via le script **/usr/sbin/setup-ds.pl**. Ce script a la bonne idée de pouvoir prendre un fichier de configuration en paramètre pour automatiser l'installation. Je l'utilise donc avec le fichier **/root/389ds/389ds-install.inf** qui suit :

```
[General]
FullMachineName=        ldap.localdomain
SuiteSpotUserID=        nobody
SuiteSpotGroup=         nobody
[slapd]
ServerPort=             389
ServerIdentifier=       localdomain
Suffix=                 dc=localdomain
RootDN=                 cn=Directory Manager
RootDNPwd=              ldapPassword
```

Ce fichier ne doit être accessible en lecture qu'à **root**, et éventuellement supprimé après l'installation, car il contient le mot de passe d'administration de l'annuaire. L'installation se lance avec la commande ci-dessous. En cas de réussite de la configuration, l'instance d'annuaire configurée est démarrée automatiquement.

```
root# /usr/sbin/setup-ds.pl --silent --file=/root/389ds/389ds-
installation.inf
```

Preuve supplémentaire de la collusion de 389DS avec sa version commerciale, la documentation d'un certain nombre de commandes est disponible dans la documentation du produit RedHat **[6]**. Entre autres, vous y trouverez les paramètres possibles pour le fichier utilisé par **/usr/sbin/setup-ds.pl** (à la section 5.5 de l'*Installation Guide*).

Par défaut, et aucun paramètre n'est fourni dans la documentation pour changer cela, le service démarre en écoute sur toutes les adresses IP du serveur et ce n'est pas forcément ce que nous voulons. Pour cela, il faut que le serveur connaisse son adresse IP et que 389DS sache sur quelle adresse écouter. Le premier point est réglé en ajoutant une ligne dans **/etc/hosts**, le second en ajoutant une ligne dans **/etc/dirsrv/slapd-localdomain/dse.ldif** (créé lors de l'installation de l'instance). Cela nécessite l'arrêt du service, mais c'est plutôt compréhensible.

```
root# service dirsrv stop localdomain
root# echo "10.0.2.101    ldap.localdomain ldap" >> /etc/hosts
root# sed -i "/^nsslapd-port:/i\
nsslapd-listenhost: ldap.localdomain" /etc/dirsrv/slapd-localdomain/dse.ldif
root# service dirsrv start localdomain ; chkconfig dirsrv on
```

1.2 Configuration SSL

Pour la gestion de la sécurité réseau, 389DS s'appuie sur les outils du projet *Network Security Services* (NSS), maintenu par la communauté Mozilla **[7]**. Cette suite fournit des outils de gestion des certificats, lesquels sont la base d'un autre produit, en association avec 389DS, DogTag PKI **[8]** développé par la communauté Fedora, et dont le pendant commercial est RedHat Certificate System. Et, une fois encore, la documentation est mixte Fedora/RedHat (**[9]** et chapitre 12 de l'*Administration Guide* **[6]**).

1.2.1 Gestion des certificats

NSS gère ses certificats à l'aide de bases BerkeleyDB **[10]**. La première étape est donc de créer la base de certificats pour notre instance d'annuaire. Pour simplifier les opérations, nous mettons le mot de passe de la base dans un fichier (que nous supprimerons très vite, je vous rassure).

```
root# cd /etc/dirsrv/slapd-localdomain
root# echo 'secretPassword' > /tmp/pwdfile
root# chmod 0400 /tmp/pwdfile
root# certutil -N -d . -f /tmp/pwdfile
```

Nous créons ensuite un certificat racine auto-signé (algorithme de chiffrement RSA avec une clé de 1024 bits) pour la mini-autorité de certification (CA : *Certificate Authority*) locale à notre instance. Vous avez la possibilité d'utiliser une « vraie » CA, mais c'est hors du scope de cet article ; regardez du côté de DogTag à l'occasion.

```
root# certutil -S -n "CA certificate" -s "cn=CA cert,dc=localdomain" \
   -x -t "CT,," -2 -m 1000 -v 120 -d . -k rsa -g 1024 -f /tmp/pwdfile
```

certutil vous posera 3 questions :

- *Is this a CA certificate ?* **y**
- *Enter the path length constraint, enter to skip* : appuyez sur **Enter**
- *Is this a critical extension ?* **y**

L'étape suivante est la création du certificat serveur pour notre instance de serveur d'annuaire, signé par le certificat racine.

```
root# certutil -S -n "LDAP-Cert" -s "cn=ldap.localdomain" -c "CA certificate" \
   -t "u,u,u" -m 1001 -v 120 -d . -k rsa -g 1024 -f /tmp/pwdfile
```

Normalement, ce n'est pas nécessaire, mais on s'assure que tous les fichiers des bases de certificats appartiennent à l'utilisateur propriétaire des processus de l'instance de serveur d'annuaire (**nobody**).

```
root# chown nobody:nobody /etc/dirsrv/slapd-localdomain/*.db
```

Pour être tranquille en cas de crash de la machine et de perte irréversible du répertoire de l'instance, nous exportons nos certificats pour les stocker ailleurs. Cette étape est optionnelle si vous utilisez votre propre Infrastructure à Clé Publique (PKI : *Public Key Infrastructure*).

```
root# pk12util -d . -o cacert.pfx -n "CA certificate"
root# pk12util -d . -o ldapcert.pfx -n "LDAP-Cert"
```

Les certificats principaux ont été générés. Nous n'avons donc plus besoin de stocker le mot de passe de la base de certificats :

```
root# rm -f /tmp/pwdfile
```

1.2.2 Activation du support SSL pour l'annuaire

Comme évoqué précédemment, la configuration de l'instance d'annuaire peut se faire à chaud, car la configuration est stockée dans l'annuaire lui-même. Il en va de même pour la configuration du protocole SSL. Nous créons les fichiers **/root/389ds/389ds-enableSsl.ldif** et **/root/389ds/389ds-addRsa.ldif** ci-dessous.

```
# /root/389ds/389ds-enableSsl.ldif
dn: cn=encryption,cn=config
changetype: modify
replace: nsSSL3
nsSSL3: on
-
replace: nsSSLClientAuth
nsSSLClientAuth: allowed
-
add: nsSSL3Ciphers
nsSSL3Ciphers: -rsa_null_md5,+rsa_rc4_128_md5,+rsa_rc4_40_md5,+rsa_
rc2_40_md5,+rsa_des_sha,+rsa_fips_des_sha,+rsa_3des_sha,+rsa_fips_3des_
sha,+fortezza,+fortezza_rc4_128_sha,+fortezza_null,+tls_rsa_export1024_
with_rc4_56_sha,+tls_rsa_export1024_with_des_cbc_sha,-rc4,-rc4export,-
rc2,-rc2export,-des,-desede3

dn: cn=config
changetype: modify
add: nsslapd-security
nsslapd-security: on
-
replace: nsslapd-ssl-check-hostname
nsslapd-ssl-check-hostname: off
```

Nous avons spécifié à l'instance d'annuaire que nous utilisons le protocole SSLv3 et que nous autorisons l'authentification par certificat utilisateur pour les clients (**nsSSLCientAuth: allowed**), bien que nous ne nous en servions pas ici. Nous avons aussi précisé la liste des algorithmes de chiffrement autorisés pour le protocole SSLv3 (**nsSSL3Ciphers**). Nous avons aussi désactivé la vérification du nom d'hôte dans le certificat, bien que notre exemple ne l'exige pas (je vous renvoie à la documentation pour les subtilités).

```
# /root/389ds/389ds-addRsa.ldif
dn: cn=RSA,cn=encryption,cn=config
changetype: add
objectclass: top
objectclass: nsEncryptionModule
cn: RSA
nsSSLPersonalitySSL: LDAP-Cert
nsSSLToken: internal (software)
nsSSLActivation: on
```

Nous avons ajouté le module de chiffrement RSA à la configuration de l'instance en lui précisant que le certificat est stocké localement (**nsSSLToken**) dans la base liée à l'instance, et que le certificat a pour nom « LDAP-Cert » (**nsSSLPersonalitySSL**).

La modification de la configuration peut maintenant se faire à chaud avec ldapmodify :

```
root# /usr/lib/mozldap/ldapmodify -h ldap.localdomain \
    -D "cn=Directory Manager" -w - \
    -f /root/389ds/389ds-enable_ssl.ldif
root# /usr/lib/mozldap/ldapmodify -a -h ldap.localdomain \
    -D "cn=Directory Manager" -w - \
    -f /root/389ds/389ds-addRsa.ldif
```

Le serveur a maintenant toutes les informations pour utiliser les certificats sauf une : le mot de passe de la base. Au prochain redémarrage du service, il vous demandera donc le mot de passe, ce qui ne vous amusera pas longtemps. 389DS offre la possibilité de stocker le mot de passe dans un fichier aux droits d'accès restreints **pin.txt** :

```
root# cat > /etc/dirsrv/slapd-localdomain/pin.txt << EOF
Internal (Software) Token: secretPassword
EOF
```

Comme nous l'avions fait pour le port 389, nous limitons l'écoute sur le port 636 (protocole LDAPS) à l'adresse IP du serveur.

```
root# service dirsrv stop localdomain
root# sed -i "/^nsslapd-port:/i\
nsslapd-securelistenhost: ldap.localdomain" /etc/dirsrv/slapd-
localdomain/dse.ldif
root# sed -i "/^nsslapd-port:/a\
nsslapd-secureport: 636" /etc/dirsrv/slapd-localdomain/dse.ldif
root# service dirsrv start localdomain
```

Nos machines clientes doivent pouvoir vérifier l'identité du serveur LDAP qu'elles contactent via le protocole LDAPS. Pour cela, nous exportons les informations du certificat racine de la CA associée à l'instance pour pouvoir les déployer sur les machines clientes.

```
root# cd /etc/dirsrv/slapd-localdomain
root# certutil -d . -L -n "CA certificate" -a > /etc/pki/tls/certs/cacert.asc
```

Et pour finir, ou presque, nous testons que la configuration fonctionne. Pour cela, nous précisons que nous utilisons le protocole SSL (**-Z**), le port (**-p 636**) et le fichier contenant la base de certificats (**-P /etc/dirsrv/slapd-localdomain/cert8.db**) :

```
root# /usr/lib/mozldap/ldapsearch -Z -h ldap.localdomain -p 636 \
    -P /etc/dirsrv/slapd-localdomain/cert8.db \
    -D "cn=Directory Manager" -w - \
    -s base -b "" '(objectclass=*)' | head
```

Par défaut, le service **iptables** est démarré sur les machines à base de distribution Fedora. Je ne saurais que trop vous déconseiller de le désactiver. Donc, voici les règles à ajouter dans **/etc/sysconfig/iptables** :

```
# 389DS LDAP Server : LDAP & LDAPS
-A INPUT -d 10.0.2.101 -m state --state NEW -m tcp -p tcp --dport 389 -j ACCEPT
-A INPUT -d 10.0.2.101 -m state --state NEW -m tcp -p tcp --dport 636 -j ACCEPT
```

Authentification des utilisateurs

Je sais que cette thématique est un peu hors sujet, mais c'est un bon exemple pour valider le fonctionnement de l'annuaire et un bon point de départ pour l'authentification centralisée de vos utilisateurs. Et un autre intérêt de cet exemple est que le changement de mot de passe nécessite la sécurisation de la communication par SSL, ajoutant un test grandeur nature de la configuration SSL.

Note

Si la problématique de gestion des utilisateurs vous intéresse, je vous recommande de jeter un œil sur le projet FreeIPA **[11]** dont l'objectif est de fournir un outil de gestion centralisée des comptes utilisateurs/machines et de l'authentification en offrant des fonctionnalités d'audit de la politique mise en œuvre. Il s'agit là aussi d'un projet de la communauté Fedora.

Création des comptes utilisateurs et de groupes

Je ferai simple et rapide : un utilisateur **fdupont** et son groupe associé comme s'il avait été créé par Fedora en local. Le fichier **/root/389ds/389ds-user_fdupont.ldif** au format LDIF ci-dessous crée les entrées dans l'annuaire pour l'utilisateur et le groupe.

```
# /root/389ds/389ds-user_fdupont.ldif
dn: uid=fdupont,ou=People,dc=localdomain
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
uid: fdupont
cn: Fabien Dupont
sn: Dupont
userPassword: test
mail: fdupont@localdomain
description: Fabien Dupont
gidnumber: 500
givenName: Fabien
homedirectory: /home/fdupont
loginshell: /bin/bash
telephonenumber: +33 1 23 45 67 89
uidnumber: 500
departmentNumber: Test Environment

dn: cn=fdupont,ou=Groups,dc=localdomain
objectclass: top
objectclass: groupofuniquenames
objectClass: posixGroup
cn: fdupont
gidNumber: 500
description: Groupe de Fabien Dupont
uniqueMember: dn: uid=fdupont,ou=People,dc=localdomain
```

Nous injectons les données dans l'annuaire :

```
root# /usr/lib/mozldap/ldapmodify -a -h ldap.localdomain \
    -D "cn=Directory Manager" -w - \
    -f /root/389ds/389ds-user_fdupont.ldif
```

Nous positionnons le mot de passe de l'utilisateur que nous venons de créer :

```
root# /usr/lib/mozldap/ldappasswd -Z -h ldap.localdomain -p 636 \
    -P /etc/dirsrv/slapd-localdomain/cert8.db \
    -D "cn=Directory Manager" -w - \
    -S uid=fdupont,ou=People,dc=localdomain
```

Authentification des utilisateurs (suite)

Nous créons le répertoire **HOME** de l'utilisateur. Cette opération est à prévoir dans votre *workflow* de gestion des utilisateurs. Nous utilisons le programme utilisé par le module PAM **mkhomedir** pour la création automatique des **HOME** lors de l'authentification des utilisateurs ; nous aurions d'ailleurs pu laisser cette tâche à PAM.

```
root# mkhomedir_helper fdupont
```

Configuration de l'authentification PAM/LDAP

Pour que nos utilisateurs puissent se connecter au système, nous devons dire à PAM (*Pluggable Authentication Module*) de se connecter au serveur d´annuaire LDAP pour authentifier les utilisateurs. Au passage, nous modifions la configuration de **nsswitch** pour qu´il obtienne les informations sur les utilisateurs et les groupes par l´annuaire.

Mais, avant de lancer la configuration, nous devons installer deux packages supplémentaires : **nscd**, un démon qui gère un cache des données de nom et qui limite donc le volume de communications avec l´annuaire, et **nss_ldap** qui permet au service de gestion des noms (géré par **/etc/nsswitch.conf**) de s´interfacer avec un annuaire LDAP.

```
root# yum install nscd nss_ldap
```

Manuellement, la configuration se fait en modifiant les fichiers suivants : **/etc/nsswitch.conf**, **/etc/ldap.conf**, **/etc/openldap.conf**, **/etc/pam.d/system-auth** et **/etc/sysconfg/authconfig**. Bref, beaucoup de manipulations de fichiers et de syntaxes à connaître. Nous allons donc faire appel à un petit utilitaire fourni par la distribution Fedora : **authconfig** **[12]**. En une ligne de commande, nous sommes sûrs que la configuration sera bonne, y compris si la liste des fichiers à modifier évolue.

```
root# authconfig --disablenis --enablecache --enableldap --enableldapauth \
    --ldapserver=ldap.localdomain --ldapbasedn=dc=localdomain \
    --enableldaptls --ldaploadcacert=file:///etc/pki/tls/certs/cacert.asc \
    --updateall
```

Nous avons donc dit à **authconfig** que nous souhaitons :

- désactiver l´utilisation du protocole NIS (**--disablenis**) ;
- utiliser le démon de cache des données (**--enablecache**) ;
- utiliser un annuaire LDAP (**--enableldap --enableldapauth**) ;
- le serveur d´annuaire LDAP est **ldap.localdomain** ;
- le DN servant de base à la recherche est **dc=localdomain** ;
- la connexion nécessite l´utilisation de TLS (**--enableldaptls**)
- le certificat de la CA est disponible à l´URL **file:///etc/pki/tls/certs/cacert.asc** (je vous laisse explorer les possibilités de diffusion sur les clients) ;
- mettre à jour l´ensemble des fichiers de configuration (**--updateall**).

Nous pouvons tester assez simplement le résultat en nous connectant à la machine avec l´utilisateur que nous avons créé précédemment.

2 INSTALLATION DE BIND 9 AVEC LE BACKEND LDAP

Maintenant que notre serveur d´annuaire est fonctionnel, nous pouvons passer à la seconde phase de notre projet : intégrer les données de notre serveur DNS à base de Bind 9 dans l´annuaire. Pour cela, nous utiliserons l´extension Simple DataBase (SDB) de Bind 9 pour laquelle il existe un *backend* LDAP **[13]** (des backends existent aussi pour MySQL et PostgreSQL).

Nous commençons par simplement installer Bind 9, et plus particulièrement la version *chrootée*, ainsi que l'extension SDB, disponibles sous forme de package RPM.

```
root# yum install bind-chroot bind-sdb
root# echo 'ROOTDIR="/var/named/chroot"' >> /etc/sysconfig/named
```

2.1 Intégration du schéma et arborescence de base

Pour pouvoir ajouter les données DNS à notre annuaire, nous devons expliciter la structure de ces données via un schéma. Celui-ci est disponible sur le site du projet de backend LDAP pour Bind 9 (URL ci-dessous), mais dans un format prévu pour OpenLDAP et qui n´est pas directement compréhensible par 389DS. Mais, heureusement, un convertisseur existe :

```
root# service dirsrv stop localdomain
root# wget http://bind9-ldap.bayour.com/dnszone-schema.txt
root# wget http://directory.fedoraproject.org/download/ol-schema-migrate.pl
root# chmod u+x /root/ol-schema-migrate.pl
root# /root/ol-schema-migrate.pl -b /root/dnszone-schema.txt > /etc/dirsrv/
slapd-localdomain/schema/60dnsZone.ldif
root# service dirsrv start localdomain
```

Nous avons arrêté le service d´annuaire, puis téléchargé le schéma (**dnszone-schema.txt**) et le script de conversion (**ol-schema-migrate.pl**) et installé la version convertie du schéma dans le répertoire où sont installés les schémas de notre instance d´annuaire (**/etc/dirsrv/slapd-localdomain/schema**), et nous avons redémarré le service. Notre instance d´annuaire sait maintenant traiter des données DNS.

Nous créons une arborescence spécifique pour stocker les données DNS, directement sous la racine de l´annuaire,

ainsi qu'un utilisateur spécifique au service DNS. Nous souhaitons limiter les droits sur la consultation des données : nous ajoutons donc une ACI (*Access Control Instruction*) à l'annuaire pour que seul l'utilisateur **cn=dnsadmin,ou=Special Users,dc=localdomain** puisse consulter les données (je vous renvoie au chapitre 6 de l'*Administration Guide* **[6]** pour le fonctionnement des ACI). Cet utilisateur sera utilisé par Bind pour obtenir les données DNS. Nous créons donc le fichier LDIF **/root/389ds/389ds-bind_base.ldif** comme suit :

```
# /root/389ds/389ds-bind_base.ldif
dn: ou=dns,dc=localdomain
objectClass: top
objectClass: organizationalUnit
ou: dns
description: All informations about DNS
aci: (target="ldap:///ou=dns,dc=localdomain")(version 3.0; acl "DNS
Administrator"; deny(all) userdn!="ldap:///cn=dnsadmin,ou=Special
Users,dc=localdomain";)

dn: cn=dnsadmin,ou=Special Users,dc=localdomain
objectClass: top
objectClass: person
cn: dnsadmin
sn: DNS Administrator
```

Nous ajoutons ces données à l'annuaire et positionnons le mot de passe pour l'utilisateur qui gère les données DNS :

```
root# /usr/lib/mozldap/ldapmodify -a -h ldap.localdomain \
    -D "cn=Directory Manager" -w - \
    -f /root/389ds/389ds-bind_base.ldif

root# /usr/lib/mozldap/ldappasswd -Z -h localdomain -p 636 \
    -P /etc/dirsrv/slapd-localdomain/cert8.db
    -D "cn=Directory Manager" -w - \
    -s "dnsPassword" "cn=dnsadmin,ou=Special Users,dc=localdomain"
```

2.2 Configuration de Bind 9

Nous pouvons passer à la configuration du serveur DNS. Cela se fait au travers de l'habituel fichier **/etc/named.conf**. Dans notre exemple, nous gérons l'espace de nommage de la zone **localdomain** sur le réseau **10.0.2.0/24**, ce qui donne le fichier suivant (j'ai supprimé les sections DNSSEC créées par le package RPM, car ce n'est pas l'objet de l'article) :

```
options {
    listen-on port 53 { 127.0.0.1; 10.0.2.102; };
    listen-on-v6 port 53 { none; };
    directory       "/var/named";
    dump-file       "/var/named/data/cache_dump.db";
    statistics-file "/var/named/data/named_stats.txt";
    memstatistics-file "/var/named/data/named_mem_stats.txt";
    allow-query     { localhost; 10.0.2.0/24; };
    recursion yes;
};

logging {
    channel default_debug {
        file "data/named.run";
        severity dynamic;
    };
};

zone "." IN {
    type hint;
    file "named.ca";
};

// Zone pour le domaine localdomain
zone "localdomain" {
    type master;
    database "ldap ldap://10.0.2.101/ou=localdomain,ou=dns,dc=localdom
ain????!bindname=cn=dnsadmin%2cou=Special%20Users%2cdc=localdomain,!x-
bindpw=dnsPassword,x-tls 172800";
};

// Zone reverse pour 10.0.2.0/24
zone "2.0.10.in-addr.arpa" {
    type master;
    database "ldap ldap://10.0.2.101/ou=2.0.10.in-addr.arpa,o
u=dns,dc=localdomain????!bindname=cn=dnsadmin%2cou=Special%20
Users%2cdc=localdomain,!x-bindpw=dnsPassword,x-tls 172800";
};

include "/etc/named.rfc1912.zones";
```

Les sections intéressantes sont celles qui définissent les zones : nous utilisons le backend **database** au lieu du classique **file**. Nous lui précisons ensuite que nous passons par le protocole LDAP avec les paramètres suivants :

- l'URL de recherche : **ldap://10.0.2.101/ou=localdomain,ou=dns,dc=localdomain????** ;
- le login de connexion : **!bindname=...** ;
- le mot de passe de connexion : **!x-bindpw=...** ;
- la connexion doit utiliser le protocole TLS : **x-tls** ;
- le TTL pour les entrées n'ayant pas d'attribut **dNSTTL** : **172800**.

Si vous avez été un peu trop rapide et avez tenté de démarrer Bind, vous constaterez qu'il renvoie des erreurs et que les logs dans **/var/log/messages** indiquent qu'il n'y a pas de SOA pour les zones **localdomain** et **2.0.10.in-addr.arpa**. Ceci est complètement normal, la branche de l'annuaire est vide, et nous allons y remédier dès maintenant.

2.3 Arborescence pour nos zones DNS

Dans un premier temps, nous créons l'arborescence minimale et les SOA. Pour cela, le fichier LDIF ci-dessous crée une entrée de type **organizationalUnit** par domaine : elle est le point d'entrée de l'arborescence de l'annuaire pour chaque domaine et contient au moins une entrée de type **dNSZone** dont l'attribut **relativeDomainName** est le nom du domaine géré, et une entrée de type **dNSZone** dont l'attribut **relativeDomainName** est **@** et qui correspond au SOA du domaine.

```
# /root/389ds/389ds-bind_structure.ldif
dn: ou=localdomain,ou=dns,dc=localdomain
objectClass: top
objectClass: organizationalUnit
ou: localdomain
description: All informations about localdomain zone
```

```
dn: zoneName=localdomain,ou=localdomain,ou=dns,dc=localdomain
objectClass: top
objectClass: dNSZone
zoneName: localdomain
relativeDomainName: localdomain

dn: relativeDomainName=@,ou=localdomain,ou=dns,dc=localdomain
objectClass: top
objectClass: dNSZone
zoneName: localdomain
relativeDomainName: @
dNSTTL: 3600
dNSClass: IN
sOARecord: ns.localdomain. dnsadmin.localdomain. 2006112801 8H 2H 1W 1D
nSRecord: ns.localdomain.
mXRecord: 10 smtp.localdomain.
aRecord: 10.0.2.102
tXTRecord: Zone_Principale_localdomain

dn: ou=2.0.10.in-addr.arpa,ou=dns,dc=localdomain
objectClass: top
objectClass: organizationalUnit
ou: 2.0.10.in-addr.arpa
description: All informations about 2.0.10.in-addr.arpa zone

dn: zoneName=2.0.10.in-addr.arpa,ou=2.0.10.in-addr.
arpa,ou=dns,dc=localdomain
objectClass: top
objectClass: dNSZone
zoneName: 2.0.10.in-addr.arpa
relativeDomainName: 2.0.10.in-addr.arpa

dn: relativeDomainName=@,ou=2.0.10.in-addr.arpa,ou=dns,dc=localdomain
objectClass: top
objectClass: dNSZone
zoneName: 2.0.10.in-addr.arpa
relativeDomainName: @
dNSTTL: 3600
dNSClass: IN
sOARecord: ns.localdomain. dnsadmin.localdomain. 2006112801 8H 2H 1W 1D
nSRecord: ns.localdomain.
mXRecord: 10 smtp.localdomain.
tXTRecord: Zone_Principale_2.0.10.in-addr.arpa+
```

L'injection se fait toujours via **ldapmodify** :

```
root# /usr/lib/mozldap/ldapmodify -a -h ldap.localdomain \
  -D "cn=Directory Manager" -w - \
  -f /root/389ds/389ds-bind_structure.ldif
```

Il ne nous reste plus qu'à créer les enregistrements en tant que tels. Une fois encore, un simple fichier au format LDIF fera l'affaire. Nous créerons ici le minimum : les serveurs DNS et SMTP (indiqués dans le SOA) et le serveur LDAP. Le *provisionning* de l'annuaire est une procédure assez simplement scriptable et adaptable à votre environnement système, alors laissez aller votre imagination...

```
# /root/389ds/389ds-bind_data.ldif
dn: relativeDomainName=ldap,ou=localdomain,ou=dns,dc=localdomain
objectClass: top
objectClass: dNSZone
zoneName: localdomain
relativeDomainName: ldap
dNSTTL: 1800
dNSClass: IN
aRecord: 10.0.2.101
tXTRecord: Serveur_Ldap
```

```
dn: relativeDomainName=ns,ou=localdomain,ou=dns,dc=localdomain
objectClass: top
objectClass: dNSZone
zoneName: localdomain
relativeDomainName: ns
dNSTTL: 1800
dNSClass: IN
aRecord: 10.0.2.102
tXTRecord: Serveur_Dns
dn: relativeDomainName=smtp,ou=localdomain,ou=dns,dc=localdomain
objectClass: top
objectClass: dNSZone
zoneName: localdomain
relativeDomainName: smtp
dNSTTL: 1800
dNSClass: IN
aRecord: 10.0.2.103
tXTRecord: Serveur_Smtp

dn: relativeDomainName=103,ou=2.0.10.in-addr.arpa,ou=dns,dc=localdomain
objectClass: top
objectClass: dNSZone
zoneName: 2.0.10.in-addr.arpa
relativeDomainName: 103
pTRRecord: smtp.localdomain.
tXTRecord: Serveur_Smtp
dn: relativeDomainName=102,ou=2.0.10.in-addr.arpa,ou=dns,dc=localdomain
objectClass: top
objectClass: dNSZone
zoneName: 2.0.10.in-addr.arpa
relativeDomainName: 102
pTRRecord: ns.localdomain.
tXTRecord: Serveur_Dns

dn: relativeDomainName=101,ou=2.0.10.in-addr.arpa,ou=dns,dc=localdomain
objectClass: top
objectClass: dNSZone
zoneName: 2.0.10.in-addr.arpa
relativeDomainName: 101
pTRRecord: ldap.localdomain.
tXTRecord: Serveur_Ldap
```

ldapmodify toujours à l'œuvre :

```
root# /usr/lib/mozldap/ldapmodify -a -h ldap.localdomain \
  -D "cn=Directory Manager" -w - \
  -f /root/389ds/389ds-bind_data.ldif
```

2.4 Finalisation

Nous avons alimenté notre annuaire avec les données DNS, configuré le service Bind pour qu'il utilise un annuaire LDAP comme source de données, limité l'accès à ces données à un utilisateur dédié. La dernière étape est probablement la plus simple : démarrer le service. Pourtant, un petit écueil reste à éviter : par défaut, le service **named** (Bind 9) est configuré pour démarrer avant le service **dirsrv** (389DS), empêchant Bind d'accéder aux données dans l'annuaire. Nous modifions donc l'ordre de démarrage de **named** avant de le démarrer :

```
root# sed -i "/^# chkconfig: /s/: .*/: - 21 79/" /etc/init.d/named
root# service named start ; chkconfig named on
```

De la même façon que nous avions ouvert les ports 389 et 686 sur le serveur LDAP, nous devons ouvrir le port 53 pour que le serveur DNS soit accessible des autres machines. Il suffit d'ajouter les lignes suivantes à **/etc/sysconfig/iptables** et redémarrer le service **iptables** :

```
# Bind9 DNS Server
-A INPUT -d 10.0.2.102 -m state --state NEW -m tcp -p tcp --dport 53 -j ACCEPT
```

Et le moment de vérité est arrivé : notre serveur DNS fonctionne-t-il comme attendu ? Vous me direz que ça dépend de moi et de l´attention que j´ai apportée à la relecture de l´article... Mais, hors coquille, en configurant le serveur de nom dans **/etc/resolv.conf**, les commandes suivantes devraient fournir les bonnes valeurs :

```
root# cat > /etc/resolv.conf << EOF
search localdomain
nameserver 10.0.2.102
EOF
root# nslookup smtp.localdomain
Server:         10.0.2.102
Address:        10.0.2.102#53

Name:   smtp.localdomain
Address: 10.0.2.103
root# nslookup 10.0.2.103
Server:         10.0.2.102
Address:        10.0.2.102#53

103.2.0.10.in-addr.arpa name = smtp.localdomain.
```

Et voilà ! Vous avez maintenant un serveur DNS pleinement fonctionnel et vos données sont stockées dans votre annuaire. Vous pouvez dorénavant mettre à jour vos données sans recharger **named** (sauf si vous ajoutez des zones). Et vous n'avez plus que les données de l'annuaire et **/etc/named.conf** à sauvegarder. Amusez-vous bien... ■

Auteur : Fabien Dupont

Fabien Dupont , Ingénieur Système Senior

Remerciements

Je tiens à remercier Hamza Mokhtari et Sébastien Bonnegent pour leur talent de relecteur et leur amitié.

Références

- [1] http://fedoraproject.org/
- [2] http://directory.fedoraproject.org/
- [3] https://www.isc.org/software/bind/
- [4] http://directory.fedoraproject.org/wiki/Documentation#Howtos
- [5] https://wiki.mozilla.org/LDAP_C_SDK
- [6] http://www.redhat.com/docs/manuals/dir-server/
- [7] http://www.mozilla.org/projects/security/pki/nss/
- [8] http://pki.fedoraproject.org/
- [9] http://directory.fedoraproject.org/wiki/Howto:SSL
- [10] http://www.oracle.com/database/berkeley-db/index.html
- [11] http://freeipa.org/page/Main_Page
- [12] http://fedorahosted.org/authconfig/
- [13] http://bind9-ldap.bayour.com/

Quick sysadmin's tip : Installer un module Perl dans Debian

Perl est un vaste monde. Debian aussi. Cependant, tous les modules Perl ne sont pas intégrés dans Debian pour autant. En revanche, les développeurs ont pensé à une méthode simple permettant de garder sa gestion de paquets cohérente, tout en assurant l'ajout de module à partir des sources téléchargées depuis CPAN (Comprehensive Perl Archive Network).

Pour installer un module Perl non existant sous la forme de paquet, vous aurez besoin de **fakeroot**, **dpkg-dev**, **devscripts**, **build-essential** et **dh-make-perl**. Nous prendrons ici comme exemple le superbe sinon indispensable module **ACME::Buffy**.

Nous commençons donc par le télécharger depuis CPAN et le désarchiver dans un répertoire temporaire :

```
% wget http://search.cpan.org/CPAN/authors/id/L/LB/LBROCARD/Acme-Buffy-1.5.tar.gz
% tar xfzv Acme-Buffy-1.5.tar.gz
```

Nous utilisons ensuite **dh-make-perl** en spécifiant le nom du sous-répertoire pour « debianiser » les sources :

```
% dh-make-perl Acme-Buffy-1.5
Found: Acme-Buffy 1.5 (libacme-buffy-perl arch=all)
Using cached Contents from Tue Nov  3 09:24:16 2009
Using maintainer: Denis Bodor <denis@morgane.ed-diamond.com>
Found changelog: CHANGES
Found docs: README
Using rules: /usr/share/dh-make-perl/rules.dh7.tiny
--- Done
```

Enfin, nous pouvons construire le paquet avec **debuild** après s'être placé dans le répertoire en question :

```
% cd Acme-Buffy-1.5
% debuild
This package has a Debian revision number but there does not seem to be
an appropriate original tar file or .orig directory in the parent directory;
(expected libacme-buffy-perl_1.5.orig.tar.gz or Acme-Buffy-1.5.orig)
continue anyway? (y/n) y
dpkg-buildpackage -rfakeroot -D -us -uc
dpkg-buildpackage| : définir CFLAGS à la valeur par défaut| : -g -O2
dpkg-buildpackage| : définir CPPFLAGS à la valeur par défaut| :
dpkg-buildpackage| : définir LDFLAGS à la valeur par défaut| :
dpkg-buildpackage| : définir FFLAGS à la valeur par défaut| : -g -O2
dpkg-buildpackage| : définir CXXFLAGS à la valeur par défaut| : -g -O2
dpkg-buildpackage: paquet source libacme-buffy-perl
dpkg-buildpackage: version source 1.5-1
```

Vous remarquerez que **dpkg-buildpackage** utilise bon nombre de valeurs par défaut, qui, dans la plupart des cas, conviendront parfaitement :

```
fakeroot debian/rules clean
   dh_testdir
   dh_auto_clean
   dh_clean
dpkg-source -b Acme-Buffy-1.5
debian/rules build
dh build
   dh_testdir
   dh_auto_configure
```

S'en suit la construction du module Perl :

```
Checking if your kit is complete...
Looks good
Writing Makefile for Acme::Buffy
   dh_auto_build
t/buffy.t ......... ok
t/critic.t ........ skipped: Test::Perl::Critic required to criticise code
t/pod.t ........... ok
All tests successful.
Result: PASS
make[1]: quittant le répertoire " /home/denis/P/Acme-Buffy-1.5 "
```

Puis, la construction du paquet à proprement parler :

```
fakeroot debian/rules binary
dh binary
   dh_testroot
source complet)
```

En fin de processus de construction, on retrouve, dans le répertoire parent, un fichier **libacme-buffy-perl_1.5-1_all.deb** qu'il nous suffira d'installer avec **dpkg -i**. Job done !

Migration et ouverture d'une

Si vous êtes un lecteur de longue date, vous en savez déjà pas mal sur ma configuration MTA avec Exim et les ajouts concernant les mécanismes anti-spam. Sur la base de cette configuration plus ou moins standard, je vous propose ici de faire rapidement le tour des modifications permettant l'ajout de l'authentification par mot de passe et les fonctionnalités qui peuvent en découler.

Auteur

- Denis Bodor

OBJECTIF(S)

Migrer une solution classique et courante composée d'un SMTP et d'un service POP3, vers une architecture plus polyvalente par ajout d'IMAP, de stockage en Maildir (plutôt qu'en mbox), de tunneling VPN et de service Webmail. L'objectif principal est d'arriver à composer un système capable de fournir un service privé proche de Gmail (Webmail, IMAP/SSL, POP, etc.) où la messagerie est accessible par plusieurs biais, mais surtout où les actions (suppression, envoi, modification) sont répercutées sur toutes les formes d'accès.

OUTIL(S) UTILISÉ(S)

Exim4 est notre MTA. C'est celui utilisé par défaut avec les distributions Debian et il se prête plutôt gentiment à la mise en place d'un filtrage anti-SPAM pour peu que l'on prenne le temps d'apprécier la syntaxe de configuration. Le couple Fetchmail/Procmail est la solution typique des systèmes Unix permettant la relève et la gestion des messages. Dovecot est un serveur IMAP généralement considéré comme souple et léger. Parmi les outils annexes, nous trouverons également OpenVPN, Mutt (MUA ou client Mail très connu) et Roundcube pour le Webmail reposant sur Lighttpd.

En guise de résumé, voici une brève description de la configuration déjà en place. Le système hôte est une base Debian GNU/Linux utilisant Exim4 en configuration multi-fichier. Cette installation utilise un VPN OpenVPN permettant l'envoi de mails sans authentification depuis n'importe quelle machine du VPN. La solution anti-spam retenue est basée sur MailScanner à laquelle s'ajoute un tri « violent » des hôtes basé sur la géolocalisation IP. Les principaux pays émetteurs de spam sont ainsi directement écartés dès la connexion, ce qui permet d'alléger considérablement la charge système et l'utilisation de la mémoire (SpamAssassin est très gourmand sur ce point).

La nécessité de permettre l'utilisation du MTA avec des périphériques mobiles ne pouvant supporter un client OpenVPN vient à présent imposer une nouvelle modification de la configuration. La solution retenue sera donc l'utilisation d'une authentification par mot de passe. Bien entendu, il est parfaitement hors de question de laisser circuler ce type de données, qu'il s'agisse de mots de passe ou des mails eux-mêmes, au travers d'une connexion non chiffrée. Utilisateur assidu de solutions comme Fon, j'ai déjà eu tout le loisir de constater avec quelle facilité il est possible d'obtenir toutes sortes d'informations sensibles au travers de canaux de communication mal ou non protégés. Eviter que les autres puissent faire ce qu'on est soi-même en mesure de réaliser n'est pas vraiment de la paranoïa...

Première étape donc, ajouter une couche SSL/TLS sous SMTP.

1 AJOUT DU SUPPORT TLS DANS EXIM4

Voici une étape qui ne présente pas de difficulté apparente. Pour entrer dans le vif du sujet avec une installation standard de certificats auto-signés, nous utiliserons le script `exim-gencert` livré avec la documentation du paquet Debian. La génération du certificat serveur et de la clef associée s'en trouve grandement simplifiée. Jugez plutôt :

messagerie Exim4

```
% /usr/share/doc/exim4-base/examples/exim-gencert
[*] Creating a self signed SSL certificate for Exim!
    This may be sufficient to establish encrypted connections but for
    secure identification you need to buy a real certificate!

    Please enter the hostname of your MTA at the Common Name (CN)
prompt!

Generating a 1024 bit RSA private key
.....................++++++
.....................++++++
writing new private key to '/etc/exim4/exim.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Code (2 letters) [US]:FR
State or Province Name (full name) []:Alsace
Locality Name (eg, city) []:Colmar
Organization Name (eg, company; recommended) []:Lefinnois Labs
Organizational Unit Name (eg, section) []:SMTP
Server name (eg. ssl.domain.tld; required!!!) []:mail.lefinnois.net
Email Address []:lefinnois@lefinnois.net
[*] Done generating self signed certificates for exim!
    Refer to the documentation and example configuration files
    over at /usr/share/doc/exim4-base/ for an idea on how to enable TLS
    support in your mail transfer agent.
```

Vous reconnaîtrez là l'utilisation de la commande **openssl** enrobée de manière à limiter au maximum les saisies de l'administrateur. Le point important dans la liste des champs à remplir est, bien entendu le *Server name* qui devra correspondre au FQDN de la machine serveur SMTP. Les fichiers générés tels qu'attendu par la configuration par défaut sont automatiquement placés dans :

```
% ls -l /etc/exim4/exim.*
-rw-r----- 1 root Debian-exim 993 jui 28 21:33 /etc/exim4/exim.crt
-rw-r----- 1 root Debian-exim 887 jui 28 21:33 /etc/exim4/exim.key
```

Il ne vous reste plus qu'à activer l'utilisation de TLS pour vos connexions en définissant une macro dans un fichier placé dans l'arborescence de **/etc/exim4/conf.d**. Veillez simplement à nommer votre fichier de manière à ce que la déclaration de la macro se trouve effectivement placée avant son utilisation dans le fichier final tel qu'utilisé par le démon Exim4 (**/var/lib/exim4/config.autogenerated**) :

```
# /etc/exim4/main/00_exim4-macro_LEFINNOIS
MAIN_TLS_ENABLE = 1
```

Par défaut, Exim demande un certificat au client SMTP afin d'en garder une trace dans les journaux d'activité même s'il ne l'utilise pas pour authentifier celui-ci. Cet état de fait peut poser problème avec certains clients SMTP dont **ssmtp**. Pour désactiver la demande de certificat client, il faut spécifier une chaîne vide pour la macro **MAIN_TLS_TRY_VERIFY_HOSTS** avec :

```
# /etc/exim4/main/00_exim4-macro_LEFINNOIS
MAIN_TLS_ENABLE = 1
MAIN_TLS_TRY_VERIFY_HOSTS =
```

Une autre solution consiste, tout simplement, à produire un certificat client et à spécifier le fichier à utiliser dans votre configuration. En lieu et place de l'erreur **(gnutls_handshake): An unexpected TLS packet was received**, vous verrez apparaître, si tout fonctionne, des informations concernant TLS dans le **mainlog**. Exemple : **P=smtps X=TLS1.0:DHE_RSA_AES_128_CBC_SHA1:16 S=967 id=20090728195455.GA5036@lefinnois.net**.

Puisque nous en sommes à parler de **ssmtp**, la configuration pour le support TLS se résume par l'ajout de deux simples lignes dans votre **/etc/ssmtp/ssmtp.conf** :

```
UseSTARTTLS=YES
UseTLS=YES
```

Notez que, par défaut, Exim n'écoute que le port 25 et non pas, en plus, le 465 comme le font d'autres serveurs (Gmail par exemple). En effet, suite à la commande **EHLO** envoyée par le client SMTP, Exim4 annonce la possibilité d'utiliser TLS via **STARTTLS**. Si le client émet cette commande SMTP, la négociation TLS débute. Sinon, c'est l'échange SMTP classique. Certains clients SMTP considèrent que l'utilisation de TLS doit impérativement se faire via le port 465 et refuseront de négocier du TLS sur le port 25. Il faudra apporter quelques modifications dans la configuration (**/etc/default/exim4**) pour accepter les connexions sur ce port et régler le problème. Les clients en question sont généralement des outils propriétaires fonctionnant avec un système que je n'utilise plus depuis longtemps. Bref, mauvais client, changer de client. Nous en reparlerons dans la suite à propos d'un autre problème concernant la gestion des ports.

L'utilisation de **STARTTLS** permet d'initier une communication SMTP classique et d'utiliser le chiffrement ainsi qu'éventuellement l'authentification par certificat. L'utilisation du port 465 est normalement dédié aux connexions TLS/SSL *on connect*, c'est-à-dire avec un chiffrement immédiat. Dans les faits, étant donné la variété de clients mail existants, toutes les combinaisons sont possibles comme ici, avec le port 465 et l'utilisation de **STARTTLS**.

ssmtp n'est, bien entendu, pas la seule solution pour envoyer des mails bien qu'il s'agisse d'une solution légère pouvant facilement remplacer un SMTP local (Exim, SendMail, Qmail, Postfix, etc.). Si l'on sort un peu des règles d'usage d'UNIX, on peut permettre certaines largesses en demandant au MUA d'envoyer directement les messages via SMTP. C'est quelque chose que propose le très classique Mutt à l'aide d'une simple directive à ajouter dans le fichier de configuration. Il faut cependant faire attention à un petit détail qui peut vous faire perdre un temps précieux : savoir quand utiliser TLS.

Dans les journaux d'Exim, vous pourrez voir apparaître un message comme :

```
2009-08-24 10:31:15 SMTP protocol synchronization error
  (input sent without waiting for greeting):
  rejected connection from H=astrasbourg-753-8-37-173.w88-244.abo.wanadoo.fr
  [186.204.27.33] input="\026\003\001"
```

Ceci dénote un problème de configuration de Mutt et non d'Exim. N'allez donc pas changer la valeur de **smtp_enforce_sync**, puisque cette option est très intéressante dans la lutte contre les spammeurs. En effet, bon nombre de machines zombies utilisées pour l'envoi de spam possèdent la caractéristique suivante : elles n'attendent pas que le MTA présente la liste de ses fonctionnalités pour tenter l'envoi de messages. Cette directive d'Exim permet de s'en tenir à la stricte définition du protocole SMTP et, ainsi, de claquer la porte au nez à une partie du spam. Pour vous assurez que **smtp_enforce_sync** est bien active, utilisez simplement la commande :

```
% sudo exim -bP | grep smtp_enforce_sync
smtp_enforce_sync
```

Le problème vient en réalité de Mutt ou plus exactement de raccourcis dans la configuration. Notre configuration Exim repose sur deux types de connexions :

- A travers le VPN auquel cas SSL/TLS n'est pas très utile.
- Via une connexion « publique » où le chiffrement est indispensable, tout comme l'authentification SMTP (voir plus loin dans l'article).

Par défaut, une connexion provenant d'une adresse IP autorisée pour le *relaying* ne nécessitera pas d'authentification et SSL/TLS ne sera pas utile (si un VPN est utilisé). En revanche, si la connexion pour relais provient d'un hôte non autorisé, l'authentification est demandée et, par conséquent, SSL/TLS aussi. Tout ceci, bien entendu, en respectant le protocole. Or, un problème apparaît si vous utilisez ceci dans la configuration de votre Mutt :

```
set smtp_url="smtps://serveur.mail.net:25"
set ssl_starttls=yes
```

Notez le **smtps** en début d'URL. Ceci fait que Mutt va initier une connexion SSL/TLS sans attendre les *greetings* et sans émettre **STARTTLS** (et pour cause, il n'en aura pas le temps avec **smtp_enforce_sync**). Il faut donc utiliser ceci :

```
set smtp_url="smtp://serveur.mail.net:25"
set ssl_starttls=yes
```

Connexions au port 25 bloquées ?

Vous remarquerez sans doute l'utilisation du port 25, tout ce qu'il y a de plus classique, dans la configuration de Mutt. Rien de spécial me direz-vous, puisque c'est le port SMTP. Depuis des années, sinon des dizaines d'années, le port 25 a toujours été utilisé, entre autres choses, pour qu'un système puisse contacter le *mailhub* et envoyer son ou ses messages. Il est donc parfaitement légitime qu'une machine connectée via une liaison ADSL puisse ainsi contacter son serveur SMTP via ce port. A la charge du MTA d'autoriser le relais ou non. Cet avis ne semble pas être partagé par tout le monde et en particulier l'un des principaux FAI français : Orange. En effet, Orange a tout simplement bloqué le port 25, un simple **netcat** en écoute quelque part et un **telnet** depuis une machine connectée en ADSL Orange vous le confirmera. N'allez donc pas, comme moi, chercher à comprendre où vos règles NetFilter ont un problème...

Orange recommande ainsi l'utilisation du port 587 (*Messages Submission*, RFC 2476). Un port alternatif pour le SMTP. Il nous faut donc configurer Exim de manière à ce qu'il puisse « écouter » également sur ce port. Avec Debian, ceci se fera en modifiant la valeur de la variable **SMTPLISTENEROPTIONS** dans le fichier **/etc/default/exim4** avec, par exemple, **'-oX 587:465:25 -oP /var/run/exim4/exim.pid -odq'**. Notez qu'il faut préciser le fichier PID, car **-oX** en bloque la génération. Ici, nous en profitons pour également spécifier le port d'écoute 465 normalement dédié aux connexions SMTP over SSL/TLS.

Relancez Exim, grommelez contre les FAI peu respectueux des usages et poursuivez.

2 AJOUTER L'AUTHENTIFICATION PAR MOT DE PASSE

A présent, notre MTA est capable de supporter les connexions chiffrées. Nos données sont donc protégées des regards indiscrets, fussent-ils ceux du FAI ou de l'hébergeur de notre serveur. Quoi de mieux alors à faire transiter que nos mails ? Nos identifiants et mots de passe bien sûr !

La configuration par défaut des paquets Exim4 de Debian est fort sympathique de ce point de vue. Elle est, en effet, déjà en place et nous n'avons qu'à éditer le fichier **/etc/exim4/conf.d/auth/30_exim4-config_examples** pour décommenter la partie qui nous intéresse et donc activer un type précis d'authentification à supporter. En ce qui nous concerne et avec un taux de paranoïa digne d'un véritable sysadmin, nous allons éliminer d'office les méthodes **AUTH PLAIN** et **AUTH LOGIN** laissant transiter les informations en clair. C'est là que la configuration par défaut est intéressante, mais qu'il faut bien en comprendre le principe. Pour ces deux méthodes, Exim exige une connexion chiffrée (sauf si la macro **AUTH_SERVER_ALLOW_NOTLS_PASSWORDS** est définie), mais pas pour CRAM MD5. Nous commençons donc par retirer les **#** devant les lignes :

```
cram_md5_server:
  driver = cram_md5
  public_name = CRAM-MD5
  server_secret = ${extract{2}{:}{${lookup{$auth1}lsearch{CONFDIR/
passwd}{$value}fail}}}
  server_set_id = $auth1
```

Et nous ajoutons la condition de chiffrement de connexion utilisée pour les méthodes en texte clair en ajoutant les lignes suivantes :

```
  .ifndef AUTH_SERVER_ALLOW_NOTLS_PASSWORDS
  server_advertise_condition = ${if eq{$tls_cipher}{}{}{*}}
  .endif
```

CRAM MD5

L'authentification CRAM (pour *Challenge-Response Authentication Mechanism*) MD5 contrairement aux deux autres méthodes offre l'avantage de ne faire transiter aucune information en clair dans l'échange d'authentification. Celle-ci est donc incontestablement plus sûr, mais n'est néanmoins pas parfaite. Il s'agit d'une authentification à sens unique. Le serveur authentifie le client, mais non l'inverse. Il est possible de retrouver le mot de passe avec une attaque au dictionnaire (force brute) sur une transaction capturée et, pour finir, on sait maintenant que MD5 ne doit plus être considéré comme un algorithme de hachage cryptographique valable. L'utilisation de CRAM MD5 n'est donc pas suffisant pour assurer notre sécurité. D'où l'utilisation au travers d'une connexion SSL/TLS.

CRAM MD5 dans une connexion SSL/TLS est quelque chose de rassurant. Maintenant, il est toujours possible de faire tout cela au travers d'un tunnel SSH à l'intérieur d'une connexion OpenVPN si vous le souhaitez ;)

Nous pouvons maintenant créer le fichier de mot de passe **/etc/exim4/passwd**. Celui-ci est utilisé pour stocker les couples *login/pass* et peut être généré très facilement avec :

```
% htpasswd -nd utilisateur
New password: <******>
Re-type new password: <******>
utilisateur:NBiBZN/1SSfyc
```

Il suffit alors de copier la dernière ligne dans le fichier. La commande **htpasswd** provient du paquet **apache2-utils** sur Debian. Notez qu'il ne faut pas confondre le fichier **passwd** utilisé pour authentifier les clients avec **passwd.client** utilisé pour qu'Exim puisse s'authentifier comme client auprès d'un autre MTA (relais). Le nom est ambigu et source de confusion. A ce stade, tout est prêt pour le redémarrage d'Exim. Relancez le service et passez aux tests.

Pour **ssmtp**, trois lignes suffiront pour ajouter l'authentification dans le **ssmtp.conf** :

```
AuthUser=utilisateur
AuthPass=mot_de_passe
AuthMethod=cram-md5
```

Enregistrer un mot de passe en clair dans un fichier de configuration est généralement une mauvaise idée. Il est possible de malheureusement faire de même avec Mutt :

```
set smtp_url="smtp://utilisateur:mot_de_passe@serveur.mail.net:25"
```

On préférera cependant ne pas spécifier le mot de passe (et le **:**) afin que Mutt le demande lors de l'établissement de la connexion SMTP sécurisée. La bonne marche de l'ensemble peut être observée dans les journaux d'activité d'Exim. Exemple :

```
2009-08-24 11:03:08 1MfVSF-0002JE-V2 <= utilisateur@domaine.net
   H=astrasbourg-3-5-36-533.w96-204.abo.wanadoo.fr
   (domaine.net) [45.120.17.13] P=esmtpsa
   X=TLS1.0:DHE_RSA_AES_128_CBC_SHA1:16 A=cram_md5_server:utilisateur
   S=938 id=20090824090834.GA31133@domaine.net
```

On voit ici clairement l'utilisation de TLS, ainsi que l'authentification par la méthode CRAM MD5. Tout fonctionne.

3 BONUS : UN PEU D'IMAP ET DE MAILDIR COMME DESSERT ?

Pourquoi s'arrêter en si bon chemin ? Nous avons maintenant un serveur capable de prendre en charge des connexions sécurisées et authentifiées pour l'envoi de messages. En d'autres termes, nous pouvons depuis n'importe quelle connexion IP envoyer des mails en utilisant notre serveur SMTP. C'était le but initial, par exemple, pour rendre possible l'utilisation de Claws-mail sur un Nokia N810 nomade pouvant utiliser une connexion publique non sûre (oui, je sais, OpenVPN existe sur le N810, je l'ai également installé et configuré).

Ma configuration mail ne date pas d'hier et a été mise à jour au fil du temps sans pour autant changer d'architecture. Pour résumer, un serveur Exim4 est le MTA et, très « unixement », le couple Fetchmail/Procmail relève et organise les messages reçus avec POP3 via le VPN. Mes messages sont ainsi relevés et disponibles uniquement sur ma machine de travail personnelle sous forme de fichiers *mbox* manipulés avec Mutt. Comme je n'avais pas vraiment envie de revoir ma configuration, mais que la souplesse d'utilisation de mon compte Gmail commençait à rendre mon Procmail jaloux, j'ai décidé de remixer tout cela.

La stratégie est donc relativement simple. Plutôt que de risquer de changer mes habitudes, j'ai opté pour une transition en douceur :

- reprendre mes messages stockés dans des mbox et les placer dans des *maildirs* ;
- adapter la récupération et surtout le travail du MDA Procmail pour un *dispatch* dans les mbox et les maildirs ;
- créer une configuration de Mutt pour l'utilisation des maildirs ;
- installer un serveur IMAP sécurisé ;
- rediriger depuis le serveur dédié les connexions IMAP vers ma machine de travail où se trouve le serveur et les maildirs.

D'autres stratégies auraient pu être envisagées comme l'installation du serveur IMAP sur le serveur dédié avec une génération Maildir via Procmail (sans toucher à la configuration d'Exim donc). Ceci aurait permis une utilisation distante (IMAP) tout en ayant une copie locale des messages (mbox). Mais, l'objectif était la migration en douceur pour arriver à l'abandon, à terme, des mbox et l'utilisation exclusive des maildirs. Le fait de disposer d'un serveur IMAP est secondaire et uniquement destiné à une utilisation nomade très rare (un SSH+Mutt est préféré dans tous les cas).

La première étape donc consiste à convertir quelques 470 Mo de messages stockés dans une tripotée de fichiers mbox. Pour ce faire, tout en testant la validité de la nouvelle configuration Procmail, le plus simple revient à utiliser **formail**. Une commande comme **formail -s procmail /chemin/nouveau_procmailrc < fichier.mbox** fera le travail assez rapidement. La construction du fichier de configuration de Procmail est relativement simple, puisqu'il suffit de spécifier des répertoires (suffixés **/**) en lieu et place des fichiers mbox dans les règles de filtrage. Ainsi, j'ai pu reprendre très rapidement le fichier **.procmailrc** d'origine et l'adapter à l'utilisation des maildirs. Une première passe avec un petit fichier mbox permet de constater que, même si Procmail crée les répertoires **cur**, **new** et **tmp**, il est nécessaire de créer manuellement les boites maildirs utilisées. L'utilisation couplée de **cat**, **grep** et **sed** permet de faire cela très simplement à partir du fichier de configuration de Procmail.

Une fois les premiers essais effectués avec succès sur un fichier mbox et quelques ajustements faits sur les noms des boîtes, il suffit de concaténer l'ensemble des mbox avec **cat** pour ensuite filtrer le gros fichier résultant avec **formail**. A ce stade et après traitement, on dispose d'une arborescence Maildir contenant tous les messages des mbox.

Vient ensuite la nécessité d'adapter la configuration Procmail existante pour prendre en compte la réplication vers les maildirs. Deux solutions s'offrent à nous. Soit modifier la configuration de manière à remplir à la fois les mbox et les maildirs, soit lancer une autre instance de Procmail sur une copie des messages relevés par Fetchmail. La première possibilité nécessite une modification de l'ensemble du fichier avec l'intégration de Maildir pour chaque règle. De plus, n'oublions pas que le but final est l'abandon des mbox et donc le basculement d'une solution à l'autre. La meilleure solution est donc la seconde qui permet, le moment venu, de remplacer le **.procmailrc** existant par la nouvelle version éprouvée par la période de double traitement. La modification du **.procmailrc** est toute simple, puisqu'il suffit d'ajouter une règle qui sera utilisée sur chaque message :

```
:0c
| procmail /chemin/nouveau_procmailrc
```

Nous renvoyons ainsi une copie des messages à une nouvelle instance de Procmail utilisant un fichier de configuration dédié. Le fichier de configuration de cette instance est celui utilisé pour les tests et la conversion des fichiers mbox. Dès lors, chaque message relevé est stocké comme il se doit. On peut aisément le vérifier en utilisant **multitail** sur les deux fichiers journaux de Procmail.

Nous pouvons maintenant nous pencher sur la configuration de Mutt. Là encore, le plus simple est de reprendre le fichier de configuration **.muttrc** actuellement en place, le copier et l'adapter. Les changements ne sont pas très importants et se résument à :

- spécifier le type de boites : **set mbox_type=Maildir** ;
- les lister : **mailboxes `echo -n "+ "; find /mnt/six5/MAILDIR/ -maxdepth 1 -mindepth 1 -type d -printf "+%f "`** ;
- préciser le répertoire où elles se trouvent : **set folder=/chemin/MAILDIR** ;
- bien définir la boite d'envoi au format Maildir : **set record="+SENT"**.

Si vous avez mis en place des *hooks* spécifiques aux boites (**folder-hook**), il est important de les vérifier en désignant le répertoire des maildirs en question et non plus les fichiers mbox.

La directive **Mailboxes** permet de lister les boites mail afin que Mutt puisse les afficher via l'option **-y** sur la ligne de commande. Dans le cas d'une utilisation de fichiers mbox, c'est généralement la commande suivante qui est utilisée :

```
% sh -c "/bin/ls -1 /chemin/Mailbox | grep '\.mbox$' | sed -n 's#^#=#p' | xargs"`
```

La désignation des fichiers mbox se fait en préfixant les noms de fichier de **=**. Dans les cas des maildirs, c'est le signe **+** qui doit être utilisé et, chose importante, il s'agit de noms de répertoires. C'est donc avec plus de facilité qu'on fera usage de **find** en lieu et place de **ls**, le tout accompagné des options **-maxdepth** et **-mindepth**.

Votre configuration Mutt est maintenant prête pour l'utilisation des boites Maildir locales. Il suffit d'utiliser la commande **mutt -y -F /chemin/nouvelle_config** pour s'en assurer. A vous de choisir ensuite la configuration SMTP à adopter en fonction de l'emplacement du Mutt en question. Pour ma part, sur la machine de travail, l'utilisation de **ssmtp** me semble préférable et parfaitement justifiée. Il n'en va pas de même sur un *lappy* (Thinkpad x61, MSI Wind ou une vieille chose nommée « Mitac ») qui, par définition, est un engin nomade. Là, c'est l'utilisation des fonctionnalités SMTP over TLS de Mutt qui sera préférée.

Il ne reste plus maintenant qu'à installer un serveur IMAP sur la machine où se trouvent les maildirs. C'est Dovecot qui sera choisi pour une installation sans prétention. Avec Debian Lenny, ceci se résume à l'installation des paquets **dovecot-common** et **dovecot-imapd** et à la configuration/création d'un **/etc/dovecot/dovecot.conf** :

```
protocols = imap
disable_plaintext_auth = yes
log_timestamp = "%Y-%m-%d %H:%M:%S "
mail_privileged_group = mail
ssl_disable = no
ssl_cert_file = /etc/dovecot/dovecot.crt
ssl_key_file = /etc/dovecot/dovecot.key
```

Tirées de la configuration par défaut, ces directives permettent de spécifier le ou les protocoles à mettre en œuvre, ainsi que les options réseau à utiliser. Tout comme pour Exim, nous ferons usage de TLS/SSL. La directive **disable_plaintext_auth** nous permet à l'instar de la configuration du MTA de n'autoriser l'authentification en texte clair qu'en cas de connexion chiffrée. Les fonctionnalités offertes par certains MUA IMAP, comme celui livré avec Maemo (Nokia n810), sont parfois limitées et nous obligent à considérer les méthodes d'authentification **LOGIN** ou **PLAIN**. Comme nous allons rediriger les connexions depuis le serveur dédié vers la machine de travail au moyen de NetFilter, nous pouvons parfaitement réutiliser le certificat SSL et sa clef privée déjà en place pour Exim. Il suffit donc de copier et renommer les fichiers. Nous allons également demander à Dovecot de n'écouter que l'interface du VPN pour les connexions entrantes avec :

```
protocol imap {
  listen = 192.168.25.13:143
  ssl_listen = 192.168.25.13:143
}
```

Nous passons ensuite à l'authentification via un fichier de mot de passe **htpasswd** avec :

```
auth default {
  mechanisms = login plain
  passdb passwd-file {
    args = /etc/dovecot/passwd
  }
  userdb passwd {
  }
  user = root
}
```

Notez que **userdb** doit être spécifié dans la configuration afin que les permissions soient correctement gérées, même si aucun argument n'est fourni. Enfin, nous pouvons définir nos maildirs et dossiers (*folder*) IMAP :

```
namespace private {
  separator = /
  prefix =
 location = maildir:/chemin/MAILDIR:LAYOUT=fs:INBOX=/chemin/MAILDIR/Inbox
  inbox = yes
 list = yes
}
```

La configuration du MUA Mutt intégrera alors les paramètres IMAP :

```
set spoolfile=imap://mail.lefinnois.net:143/Inbox
set folder = "imap://mail.lefinnois.net:143/"
set record="imap://mail.lefinnois.net:143/SENT"
mailboxes  +dbodor +debianfr +defaut +denis +Inbox +SENT
```

On n'oubliera pas de spécifier les dossiers IMAP afin de faciliter la navigation avec le *folders browser* de Mutt (**c?[TAB]**) pour passer d'un dossier à l'autre. Tout est maintenant en place pour une utilisation assez proche de celle qu'on a via Gmail avec quelques avantages supplémentaires, à savoir, une accessibilité à distance avec un stockage local et une répercussion des actions entre utilisations IMAP et locales.

Bien entendu, il ne faudra pas oublier la configuration Netfilter sur le serveur dédié/SMTP avec un petit **iptables -t nat -I PREROUTING -i eth0 -p tcp -m tcp --dport 143 -j DNAT --to-destination 192.168.25.13**. On n'oubliera pas non plus d'activer le *masquerading* pour l'interface **tap** du VPN avec **iptables -t nat -A POSTROUTING -o tap1 -j MASQUERADE**, ainsi que le *forwarding* avec **sysctl -w net.ipv4.ip_forward=1** (ou une modification de **/etc/sysctl.conf** pour une configuration permanente).

4 ET POURQUOI PAS UN WEBMAIL AUSSI ?

En général, je ne suis pas très amateur de Webmail pour des raisons évidentes de sécurité. Ce genre d'applications Web est relativement sensible aux attaques XSS et, même si le code est audité par une communauté de développeurs très active, le risque du 0-day est omniprésent. Cependant, il est incontestable que si l'on souhaite pouvoir accéder à sa messagerie coûte que coûte, le Webmail représente la solution ultime. En l'absence de point d'accès, pas de VPN et pas de configuration personnalisée et sécurisé du MUA... Quand rien d'autre n'est disponible, mais qu'on a besoin de garder contact avec sa correspondance, il ne reste que le Webmail. Dans le pire des cas, ce sera l'accès au Webmail depuis un cybercafé en respectant quelques règles de sécurité supplémentaires dont HTTPS, une authentification HTTP supplémentaire et le changement systématique des mots de passe après utilisation.

Le Webmail choisi ici est le très à la mode RoundCube. Codé en PHP, capable d'utiliser IMAP et SMTP over SSL/TLS et pouvant reposer sur des bases MySQL, PostgreSQL ou SQLite, il fera bon ménage avec Lighttpd.

L'installation est relativement simple, puisque après copie des fichiers dans l'arborescence Web sur serveur Lighttpd, il suffira de créer la base de données comme indiqué dans le fichier **INSTALL** pour enfin pointer son navigateur sur **http://site/rep_roundcube/installer**. Là, après quelques tests et renseignements de configuration dûment fournis, on obtiendra le contenu des fichiers de configuration **db.inc.php** et **main.inc.php** à placer dans le sous-répertoire **config**.

Le reste de la configuration se fera via l'interface Web de manière très rapide. RoundCube n'est pas une application très riche en fonctionnalités. L'accès aux dossiers IMAP, une interface simple et un carnet d'adresses sont cependant suffisants pour une utilisation occasionnelle. Le principal avantage est, bien entendu, de garder une synchronicité entre les différentes méthodes d'utilisation de la messagerie.

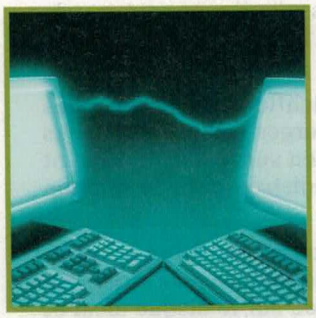
CONCLUSION

Nous voici arrivés au terme de cet article où nous avons transformé une installation classique MTA+MDA+MUA basée sur Exim, ssmtp, Fetchmail, Procmail et mbox en une solution plus accessible. Bien entendu, il reste de petits détails à corriger comme le fait que le serveur Dovecot accédant aux maildirs, Mutt, n'arrive plus à afficher la notification de nouveaux messages en utilisation locale (option **-y**). Ceci n'est cependant pas nouveau, puisque le problème s'était déjà posé lors de la mise en place d'un script d'inspection des fichiers mbox. Mutt utilise les dates d'accès aux fichiers et répertoires en les comparant avec leurs dates de modification. Tout accès aux données par un programme externe perturbe Mutt.

Le durcissement de la configuration est également un point sur lequel travailler. L'utilisation de SSL/TLS permettrait la mise en œuvre de certificats pour authentifier les clients, par exemple. Enfin, l'utilisation du Webmail reste le point noir, mais les stratégies de sécurisation ne manquent pas. ■

Auteur : Denis Bodor

DRBD, la réplication des blocs disque



Auteur

- Guillaume Lelarge

DRBD est un outil capable de répliquer le contenu d'un périphérique bloc. En ce sens, ce n'est pas un outil spécialisé pour PostgreSQL, contrairement aux autres outils vus dans le hors-série 44 sur PostgreSQL. Il peut très bien servir à répliquer des serveurs de fichiers ou de mails. Il réplique les données en temps réel et de façon transparente, pendant que les applications modifient leurs fichiers sur un périphérique. Il peut fonctionner de façon synchrone ou asynchrone. Tout ça en fait donc un outil intéressant pour répliquer le répertoire des données de PostgreSQL.

OBJECTIF(S)

Mettre en œuvre une réplication pour le SGBDR PostgreSQL de façon atypique en travaillant au niveau des périphériques en mode bloc. Il s'agit ici de descendre au plus bas dans les couches du système en ne déléguant le travail de réplication ni à l'application, ni au système de fichiers. Cette solution offre plusieurs avantages parmi lesquels le fait de se détacher des fonctionnalités du SGBD, mais aussi de rendre la solution transposable à de nombreux autres types de machines et services.

OUTIL(S) UTILISÉ(S)

DRBD (Distributed Replicated Block Device) est un système de stockage distribué disponible pour Linux. Il se compose principalement d'une partie noyau et de quelques programmes dans l'espace utilisateur. Cette solution est normalement utilisée dans la mise en œuvre de clusters de haute disponibilité (HA pour high availability). Dans les grandes lignes, on peut considérer DRBD comme un système RAID 1 capable de fonctionner au travers d'un réseau et de prendre en compte les limitations et problèmes que cela implique.

Le cœur de DRBD est implanté sous la forme d'un module pour le noyau Linux. En ce sens, cet outil n'est pas du tout portable. En fait, il agit comme un pilote pour un périphérique bloc virtuel, ce qui le rend très flexible.

En dehors du module, DRBD dispose de trois outils d'administration utilisables en espace utilisateur. Nous n'allons travailler qu'avec **drbdadm**, car il est une interface parfaite pour les deux autres outils (**drbdsetup** et **drbdmeta**).

Pour la démonstration, nous allons utiliser une partition d'un nouveau disque sur les deux serveurs. Cette partition sera montée sur le répertoire **/var/lib/postgresql/8.4/main** avant l'installation de PostgreSQL 8.4. Il faut néanmoins savoir que DRBD ne se limite pas à l'utilisation d'une partition. Vous pouvez lui donner à répliquer un périphérique RAID ou un volume LVM.

De plus, bien qu'il soit recommandé d'utiliser une connexion dédiée pour DRBD, nous n'allons pas le faire pour simplifier la démonstration.

Enfin, contrairement aux autres articles, nous supposerons ici que seule la distribution Debian est installée. Autrement dit, PostgreSQL n'est pas encore installé, et encore moins configuré.

1 INSTALLATION

Il existe des paquets pour Debian 5.0. Ils ne correspondent pas à la dernière version de DRBD, mais ils sont suffisamment récents pour nos besoins. Nous allons donc les utiliser. Cependant, en production, nous vous conseillons d'utiliser la dernière version, quitte à avoir à la compiler. Mais, ici, un simple **aptitude install** doit suffire :

```
debian1:~# aptitude install drbd8-utils drbd8-modules-2.6.26-2-686
```

Ces deux paquets doivent aussi être installés sur **debian2**.

2 CONFIGURATION

Le fichier de configuration se trouve directement dans le répertoire **/etc**. Plutôt que de le modifier, nous allons le renommer et le recréer :

```
debian1:~# cd /etc
debian1:/etc# mv drbd.conf drbd.conf.distro
```

Voici le contenu du nouveau fichier **drbd.conf** :

```
global {
  usage-count yes;
}
common {
  protocol C;
}
resource postgresql {
   on debian1 {
    device    /dev/drbd1;
    disk      /dev/hdb1;
    address   192.168.10.66:7789;
    meta-disk internal;
  }
  on debian2 {
    device    /dev/drbd1;
    disk      /dev/hdb1;
    address   192.168.10.67:7789;
    meta-disk internal;
  }
}
```

La section globale n'existe qu'en un exemplaire. De tous les paramètres possibles, seul un sera utile à tout le monde. **Usage-count** permet au projet DRBD de collecter des statistiques d'utilisation des différentes versions de DRBD. Un serveur web est contacté à chaque fois qu'une nouvelle version de DRBD est installée sur un serveur. Notez que vous pouvez le désactiver en le configurant à **'no'**. Cependant, nous vous conseillons de le laisser à **'yes'**. Cette information est importante pour les développeurs de ce projet. C'est donc une contribution, minime mais respectable, de notre part, surtout que seul le numéro de version est envoyé.

La section **common** regroupe tous les paramètres communs aux différentes ressources. Nous n'indiquons ici que le protocole. Ce paramètre spécifie la façon dont la réplication est faite :

- **protocole A** pour une réplication asynchrone (les écritures locales sont considérées comme terminées une fois que l'écriture a eu lieu sur le disque local et que l'information a été enregistrée dans le tampon TCP d'envoi) ;
- **protocole B** pour une réplication synchrone en mémoire (les écritures locales sont considérées comme terminées une fois que l'écriture a eu lieu sur le disque local et que l'information a été reçue par le deuxième serveur) ;
- **protocole C** pour une réplication synchrone (les écritures locales sont considérées comme terminées une fois que l'écriture a eu lieu sur le disque local et sur le disque distant).

Le protocole A est sans aucun doute le plus rapide, mais aussi le moins sûr. Comme nous aimons nos données (sans quoi nous ne chercherions pas à nous protéger avec de la réplication), nous allons utiliser le protocole C.

Avec DRBD, un terme revient très fréquemment : la ressource. Une ressource est composée de plusieurs propriétés :

- tout d'abord son nom qui sera utilisé pour la différencier des autres ressources ;
- le périphérique DRBD qui, une fois initialisé, pourra être formaté, monté et enfin utilisé ;
- la configuration des disques associés ;
- la configuration du réseau.

Il s'agit donc de la troisième partie du fichier de configuration. Y sont indiqués les différents nœuds à synchroniser et, pour chaque nœud, l'adresse IP, le périphérique disque réel, le périphérique disque virtuel de DRBD et la façon dont les métadonnées sont enregistrées.

Le même fichier de configuration doit se trouver sur **debian2**.

3 MISE EN PLACE DE DRBD

La première chose à faire est de partitionner les deux nouveaux disques (le premier sur **debian1**, le second sur **debian2**).

```
debian1:/etc# fdisk /dev/hdb
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF
disklabel
Building a new DOS disklabel with disk identifier 0x4a30bcd4.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

The number of cylinders for this disk is set to 4161.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)
Warning: invalid flag 0x0000 of partition table 4 will be corrected by
w(rite)

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-4161, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-4161, default 4161):
Using default value 4161
Command (m for help): w
```

La même action doit être entreprise sur le serveur **debian2**.

Maintenant, ajoutons les métadonnées du périphérique du serveur **debian1** :

```
debian1:/etc# drbdadm create-md postgresql
Writing meta data...
initialising activity log
NOT initialized bitmap
New drbd meta data block sucessfully created.
success
```

Avant la suite, nous devons nous assurer que le module **drbd** est bien chargé :

```
debian1:/etc# lsmod | grep drbd
```

Comme il n'est pas chargé, nous nous en occupons :

```
debian1:/etc# modprobe drbd
```

Les trois commandes suivantes vont permettre d'associer la ressource DRBD à son périphérique, de configurer les paramètres de synchronisation pour cette ressource et, enfin, de se connecter :

```
debian1:/etc# drbdadm attach postgresql
debian1:/etc# drbdadm syncer postgresql
debian1:/etc# drbdadm connect postgresql
```

Tout est bon pour **debian1**. Voici ce que nous dit le fichier **/proc/drbd** :

```
debian1:/etc# cat /proc/drbd
version: 8.0.14 (api:86/proto:86)
GIT-hash: bb447522fc9a87d0069b7e14f0234911ebdab0f7 build by phil@fat-
tyre, 2008-11-12 16:40:33

 1: cs:WFConnection st:Secondary/Unknown ds:Inconsistent/DUnknown C r---
    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0
       resync: used:0/61 hits:0 misses:0 starving:0 dirty:0 changed:0
       act_log: used:0/127 hits:0 misses:0 starving:0 dirty:0 changed:0
```

Ce fichier permet de suivre l'état de la réplication. Ce qui nous intéresse surtout est la ligne **1:**. **cs** indique le statut de la connexion, **st** l'état de la réplication et **ds** le statut des disques. Et là, tout ça semble bon. Il n'y a pas de connexion, car nous n'avons encore rien fait du côté de **debian2**. Nous allons donc y faire toutes les étapes précédentes :

```
debian2:/etc# drbdadm create-md postgresql
Writing meta data...
initialising activity log
NOT initialized bitmap
New drbd meta data block sucessfully created.
success
debian2:/etc# modprobe drbd
debian2:/etc# drbdadm attach postgresql
debian2:/etc# drbdadm syncer postgresql
debian2:/etc# drbdadm connect postgresql
```

/proc/drbd nous indique maintenant que la connexion est bien là :

```
debian1:/etc# cat /proc/drbd
version: 8.0.14 (api:86/proto:86)
GIT-hash: bb447522fc9a87d0069b7e14f0234911ebdab0f7 build by phil@fat-tyre,
2008-11-12 16:40:33
1: cs:Connected st:Secondary/Secondary ds:Inconsistent/Inconsistent C r---
    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0
       resync: used:0/61 hits:0 misses:0 starving:0 dirty:0 changed:0
       act_log: used:0/127 hits:0 misses:0 starving:0 dirty:0 changed:0
```

Il ne nous reste plus qu'à faire la synchronisation initiale avant de pouvoir utiliser le périphérique comme n'importe quel autre :

```
debian1:/etc# drbdadm -- --overwrite-data-of-peer primary postgresql
debian1:/etc# cat /proc/drbd
version: 8.0.14 (api:86/proto:86)
GIT-hash: bb447522fc9a87d0069b7e14f0234911ebdab0f7 build by phil@fat-
tyre, 2008-11-12 16:40:33

 1: cs:SyncSource st:Primary/Secondary ds:UpToDate/Inconsistent C r---
    ns:16896 nr:0 dw:0 dr:16896 al:0 bm:1 lo:0 pe:1 ua:0 ap:0
       [>....................] sync'ed:  1.0% (2080148/2097012)K
       finish: 0:57:46 speed: 416 (316) K/sec
       resync: used:0/61 hits:1053 misses:2 starving:0 dirty:0 changed:2
       act_log: used:0/127 hits:0 misses:0 starving:0 dirty:0 changed:0
```

Attention, les chiffres indiquées ne sont pas très représentatifs de ce que vous pouvez obtenir avec des disques réels. En effet, tous ces tests sont fait à partir d'une machine virtuelle (VirtualBox), d'où des performances disque minimales (pour rester poli...).

Il est tout à fait possible de continuer les préparatifs pendant la synchronisation, mais je préfère toujours attendre que cette partie soit terminée avant de continuer. Profitons-en pour regarder deux/trois fonctionnalités de **drbdadm**.

Cet outil peut nous indiquer l'état d'une ressource, à savoir si le nœud où nous sommes est configuré en primaire ou en secondaire (autrement dit maître ou esclave) :

```
debian1:/etc# drbdadm state postgresql
Primary/Secondary
```

debian1 est le nœud primaire/maître. Il est aussi possible de connaître l'état des disques quant à la réplication :

```
debian1:/etc# drbdadm dstate postgresql
UpToDate/Inconsistent
```

On (re-)découvre donc que le disque sur **debian1** est à jour alors que celui sur **debian2** est en cours de synchronisation initiale.

Une fois que la synchronisation est terminée, **/proc/drbd** contient quelque chose comme

```
debian1:/etc# cat /proc/drbd
version: 8.0.14 (api:86/proto:86)
GIT-hash: bb447522fc9a87d0069b7e14f0234911ebdab0f7 build by phil@fat-tyre, 2008-
11-12 16:40:33

1: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r---
    ns:2097012 nr:0 dw:0 dr:2097012 al:0 bm:128 lo:0 pe:0 ua:0 ap:0
       resync: used:0/61 hits:130936 misses:128 starving:0 dirty:0 changed:128
       act_log: used:0/127 hits:0 misses:0 starving:0 dirty:0 changed:0
```

et la commande d'état des disques affiche :

```
debian1:/etc# drbdadm dstate postgresql
UpToDate/UpToDate
```

Parfait ! Il faut maintenant formater et monter ce disque.

4 FORMATAGE ET MONTAGE DU DISQUE

Le disque à considérer n'est plus **/dev/hdb1**, mais **/dev/drbd1**. Le formatage est assez classique :

```
debian1:/etc# mkfs.ext3 /dev/drbd1
mke2fs 1.41.3 (12-Oct-2008)
Étiquette de système de fichiers=
Type de système d'exploitation : Linux
Taille de bloc=4096 (log=2)
Taille de fragment=4096 (log=2)
131072 i-noeuds, 524253 blocs
26212 blocs (5.00%) réservés pour le super utilisateur
Premier bloc de données=0
Nombre maximum de blocs du système de fichiers=536870912
16 groupes de blocs
32768 blocs par groupe, 32768 fragments par groupe
8192 i-noeuds par groupe
Superblocs de secours stockés sur les blocs :
        32768, 98304, 163840, 229376, 294912
Écriture des tables d'i-noeuds : complété
Création du journal (8192 blocs) : complété
 Écriture des superblocs et de l'information de comptabilité du système de
fichiers : complété

Le système de fichiers sera automatiquement vérifié tous les 34 montages ou
après 180 jours, selon la première éventualité. Utiliser tune2fs -c ou -i
pour écraser la valeur.
```

Il faut ensuite passer au montage. Le but est d'y placer le répertoire des données de PostgreSQL. Il existe donc deux possibilités : soit le monter dans **/var/lib/postgresql** et il n'y aura rien de plus à faire, soit le monter sur un répertoire complètement différent et il faudra faire un **initdb** (**pg_createcluster** sous Debian) vers le bon répertoire. Nous allons choisir la première méthode, celle qui nous génère le moins de travail :

```
debian1:/etc# mkdir /var/lib/postgresql
debian1:/etc# mount /dev/drbd1 /var/lib/postgresql
```

5 INSTALLATION DE POSTGRESQL SUR DEBIAN1

Il ne nous reste plus qu'à installer PostgreSQL. Après avoir configuré le fichier **/etc/apt/sources.list** comme indiqué dans l'article sur l'installation dans le hors-série 44, il ne reste plus qu'à faire l'habituel :

```
debian1:/etc# aptitude update
[... message de progression ...]
debian1:/etc# aptitude install postgresql-8.4
[... message de progression ...]
```

Ceci terminé, nous avons le nouveau répertoire des données dans **/var/lib/postgresql** :

```
debian1:/etc# ll /var/lib/postgresql/8.4/main/
total 48
drwx------ 5 postgres postgres 4096 août 29 18:26 base
drwx------ 2 postgres postgres 4096 août 29 18:26 global
drwx------ 2 postgres postgres 4096 août 29 18:25 pg_clog
drwx------ 4 postgres postgres 4096 août 29 18:25 pg_multixact
drwx------ 2 postgres postgres 4096 août 29 18:26 pg_stat_tmp
drwx------ 2 postgres postgres 4096 août 29 18:25 pg_subtrans
drwx------ 2 postgres postgres 4096 août 29 18:25 pg_tblspc
drwx------ 2 postgres postgres 4096 août 29 18:25 pg_twophase
-rw------- 1 postgres postgres    4 août 29 18:25 PG_VERSION
drwx------ 3 postgres postgres 4096 août 29 18:25 pg_xlog
-rw------- 1 postgres postgres  133 août 29 18:26 postmaster.opts
-rw------- 1 postgres postgres   54 août 29 18:26 postmaster.pid
```

6 COPIE DES FICHIERS DE CONFIGURATION DE DEBIAN1 SUR DEBIAN2

Comme nous ne synchronisons « que » le répertoire **/var/lib/postgresql** et que Debian déplace les fichiers de configuration dans **/etc/postgresql**, il nous faut manuellement copier les fichiers de configuration sur le serveur esclave.

```
debian1:/etc# scp -r /etc/postgresql debian2:/etc
```

```
root@debian2's password:
start.conf                                    100%  378     0.4KB/s   00:00
environment                                   100%  316     0.3KB/s   00:00
pg_ctl.conf                                   100%  143     0.1KB/s   00:00
postgresql.conf                               100%   16KB  16.5KB/s   00:00
pg_hba.conf                                   100% 3822     3.7KB/s   00:00
pg_ident.conf                                 100% 1631     1.6KB/s   00:00
```

7 POUR LE REDÉMARRAGE DU SERVEUR

Pour que PostgreSQL puisse démarrer, il faut que le disque soit monté, donc il faut aussi que DRBD soit lancé.

Le script de lancement de DRBD doit être configuré pour démarrer avant celui de PostgreSQL. Or, dans

certains cas, ce n'est pas fait ainsi. Par exemple, sous Debian, il va être nécessaire de modifier le nom du lien symbolique. Nous allons utiliser pour cela l'exécutable **update-rc.d** :

```
debian1:/etc# rm rc*.d/*drbd
debian1:/etc# update-rc.d drbd defaults 18
```

```
Adding system startup for /etc/init.d/drbd ...
   /etc/rc0.d/K18drbd -> ../init.d/drbd
   /etc/rc1.d/K18drbd -> ../init.d/drbd
   /etc/rc6.d/K18drbd -> ../init.d/drbd
   /etc/rc2.d/S18drbd -> ../init.d/drbd
   /etc/rc3.d/S18drbd -> ../init.d/drbd
   /etc/rc4.d/S18drbd -> ../init.d/drbd
   /etc/rc5.d/S18drbd -> ../init.d/drbd
```

8 UTILISATION DE POSTGRESQL

L'utilisation de PostgreSQL ne change en rien. Vous pouvez créer des objets, faire des insertions, des modifications et des mises à jour de données. Vous pouvez aussi les consulter. Bref, une utilisation identique... avec certainement des performances moindres. En effet, chaque modification doit se faire sur les deux machines en même temps. L'aspect synchrone est à ce prix.

Voici quelques modifications effectuées dans le cadre de cette démonstration :

```
postgres@debian1:~$ createdb b1
postgres@debian1:~$ psql b1
psql (8.4.0)
Saisissez " help " pour l'aide.

b1=# CREATE TABLE t1 (id integer);
CREATE TABLE
b1=# INSERT INTO t1 SELECT i FROM generate_series(1, 10000) AS i;
INSERT 0 10000
b1=# INSERT INTO t1 SELECT i FROM generate_series(1, 10000) AS i;
INSERT 0 10000
b1=# \q
postgres@debian1:~$ createdb b2
psql b2
postgres@debian1:~$ psql b2
psql (8.4.0)
Saisissez " help " pour l'aide.

b2=# CREATE TABLE t2 (id integer, autre text);
CREATE TABLE
b2=# INSERT INTO t2 SELECT i, 'Ligne '||i FROM generate_series(1, 100000) AS i;
INSERT 0 100000
b2=# CREATE TABLE t3 (id integer, autre text, autre2 date);
CREATE TABLE
b2=# INSERT INTO t3 SELECT i, 'Ligne '||i, now() FROM generate_series(1, 100000) AS i;
INSERT 0 100000
b2=# DELETE FROM t2 WHERE id BETWEEN 40000 AND 50000;
DELETE 10001
b2=# UPDATE t3 SET autre=upper(autre) WHERE id BETWEEN 20000 AND 25000;
UPDATE 5001
b2=# \q
```

9 TESTONS LE SWITCHOVER

Réaliser un *switchover* prend un peu de temps à cause de toutes les pelures à enlever avant d'arriver au niveau de DRBD. Il faut tout d'abord enlever la « pelure » PostgreSQL :

```
debian1:/etc# /etc/init.d/postgresql-8.4 stop
Stopping PostgreSQL 8.4 database server: main.
```

Ensuite, il faut enlever la « pelure » disque virtuel en démontant le disque :

```
debian1:/etc# umount /dev/drbd1
```

À ce moment-là, DRBD est toujours dans le même état :

```
debian1:/etc# cat /proc/drbd
version: 8.0.14 (api:86/proto:86)
GIT-hash: bb447522fc9a87d0069b7e14f0234911ebdab0f7 build by phil@fat-tyre, 2008-11-12 16:40:33

1: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r---
    ns:192336 nr:0 dw:192336 dr:161 al:49 bm:0 lo:0 pe:0 ua:0 ap:0
        resync: used:0/61 hits:0 misses:0 starving:0 dirty:0 changed:0
        act_log: used:0/127 hits:48035 misses:63 starving:0 dirty:14 changed:49
```

Par contre, comme les deux pelures sont enlevées, on va pouvoir déclasser **debian1** en serveur secondaire :

```
debian1:/etc# drbdadm secondary postgresql
debian1:/etc# cat /proc/drbd
version: 8.0.14 (api:86/proto:86)
GIT-hash: bb447522fc9a87d0069b7e14f0234911ebdab0f7 build by phil@fat-tyre, 2008-11-12 16:40:33

 1: cs:Connected st:Secondary/Secondary ds:UpToDate/UpToDate C r---
    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0
        resync: used:0/61 hits:0 misses:0 starving:0 dirty:0 changed:0
        act_log: used:0/127 hits:0 misses:0 starving:0 dirty:0 changed:0
```

Et enfin, on peut reclasser **debian2** en serveur primaire :

```
debian2:~# drbdadm primary postgresql
 debian2:~# cat /proc/drbd
version: 8.0.14 (api:86/proto:86)
GIT-hash: bb447522fc9a87d0069b7e14f0234911ebdab0f7 build by phil@fat-tyre, 2008-11-12 16:40:33
```

```
1: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r---
   ns:0 nr:192336 dw:192336 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0
        resync: used:0/61 hits:0 misses:0 starving:0 dirty:0 changed:0
        act_log: used:0/127 hits:0 misses:0 starving:0 dirty:0 changed:0
```

Maintenant que **debian2** est primaire au niveau DRBD, il faut remettre en place les pelures. On commence par monter le disque :

```
debian2:~# mkdir /var/lib/postgresql
debian2:~# mount /dev/drbd1 /var/lib/postgresql
debian2:~# ll /var/lib/postgresql/8.4/main/
total 44
drwx------ 7 104 106 4096 sep 27 17:36 base
drwx------ 2 104 106 4096 sep 27 17:38 global
drwx------ 2 104 106 4096 sep 27 17:23 pg_clog
drwx------ 4 104 106 4096 sep 27 17:23 pg_multixact
drwx------ 2 104 106 4096 sep 27 17:38 pg_stat_tmp
drwx------ 2 104 106 4096 sep 27 17:23 pg_subtrans
drwx------ 2 104 106 4096 sep 27 17:23 pg_tblspc
drwx------ 2 104 106 4096 sep 27 17:23 pg_twophase
-rw------- 1 104 106    4sep 27 17:23 PG_VERSION
drwx------ 3 104 106 4096 sep 27 17:36 pg_xlog
-rw------- 1 104 106  133 sep 27 17:23 postmaster.opts
```

Avant de démarrer PostgreSQL, il faut l'installer. **/var/lib/postgresql/8.4/main** et **/etc/postgresql/8.4/main** existant déjà, le paquet ne va pas lancer de nouveau **initdb**. Détectant la présence d'un *cluster* complet, il lance PostgreSQL à partir des données que nous avons déjà grâce à la réplication de DRBD. Par contre, avant de lancer PostgreSQL, il nous faut aussi vérifier que les fichiers de configuration soient lisibles par l'utilisateur **postgres**, et, pour cela, nous devons créer l'utilisateur **postgres** :

```
debian2:/etc# useradd postgres
debian2:/etc# chown -R postgres:postgres /etc/postgresql/8.4/main
debian2:/etc# chown -R postgres:postgres /var/lib/postgresql/8.4/main
debian2:/etc# vim /etc/apt/sources.list
debian2:/etc# aptitude update
debian2:/etc# aptitude install postgresql-8.4
[... messages de progression ...]
Paramétrage de postgresql-8.4 (8.4.0-2~bpo50+1) ...
Starting PostgreSQL 8.4 database server: main.
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Lecture de l'information d'état étendu
Initialisation de l'état des paquets... Fait
Écriture de l'information d'état étendu... Fait
Lecture des descriptions de tâches... Fait
```

Testons maintenant le contenu du nouveau serveur primaire :

```
debian2:/etc# su - postgres
postgres@debian2:~$ psql -l
                          Liste des bases de données
   Nom    |Propriétaire | Encodage |     Tri     |Type caract. |    Droits d'accès
----------+-------------+----------+-------------+-------------+---------------------
 b1       |postgres     |UTF8      | fr_FR.UTF-8 | fr_FR.UTF-8 |
 b2       |postgres     |UTF8      | fr_FR.UTF-8 | fr_FR.UTF-8 |
 postgres |postgres     |UTF8      | fr_FR.UTF-8 | fr_FR.UTF-8 |
 template0 | postgres   |UTF8      | fr_FR.UTF-8 | fr_FR.UTF-8 |=c/postgres
                                                                :postgres=CTc/postgres
 template1 | postgres   |UTF8      | fr_FR.UTF-8 | fr_FR.UTF-8 |=c/postgres
                                                                :postgres=CTc/postgres

(5 lignes)
postgres@debian2:~$ psql b1
psql (8.4.0)
Saisissez " help " pour l'aide.

b1=# \d
         Liste des relations
 Schéma | Nom | Type  |Propriétaire
--------+-----+-------+-------------
 public | t1  | table | postgres
(1 ligne)
b1=# SELECT count(*) FROM t1;
 count
-------
 20000
(1 ligne)
b1=# \q
 postgres@debian2:~$ psql b2
psql (8.4.0)
Saisissez " help " pour l'aide.

b2=# \d
         Liste des relations
 Schéma | Nom | Type  |Propriétaire
--------+-----+-------+-------------
 public | t2  | table | postgres
 public | t3  | table | postgres
(2 lignes)

b2=# SELECT count(*) FROM t2;
count
-------
 89999
(1 ligne)

b2=# SELECT count(*) FROM t3;
 count
--------
 100000
(1 ligne)
```

Faisons quelques modifications avant de tester le *failover* :

```
postgres@debian2:/$ createdb b3
postgres@debian2:/$ psql b1
psql (8.4.1)
Saisissez " help " pour l'aide.

b1=# INSERT INTO t1 SELECT i FROM generate_series(20000, 40000) AS i;
INSERT 0 20001
b1=# \q
```

10 ET MAINTENANT, LE FAILOVER

Vu que nous avons des données, nous allons pouvoir tester le failover. Simulons une panne sur le nœud primaire (l'arrêter suffit). Voici ce que nous avons sur le nœud secondaire (**debian1** actuellement) :

```
debian1:/etc# cat /proc/drbd
version: 8.0.14 (api:86/proto:86)
GIT-hash: bb447522fc9a87d0069b7e14f0234911ebdab0f7 build by phil@fat-
tyre, 2008-11-12 16:40:33

 1: cs:WFConnection st:Secondary/Unknown ds:UpToDate/DUnknown C r---
    ns:0 nr:18260 dw:18260 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0
       resync: used:0/61 hits:0 misses:0 starving:0 dirty:0 changed:0
       act_log: used:0/127 hits:0 misses:0 starving:0 dirty:0 changed:0
```

Il a bien détecté la déconnexion (ou l'absence) de **debian2**. L'état du nœud 2 est inconnu, mais le nœud 1 est à jour. Mettons maintenant le nœud 1 en nœud primaire :

```
debian1:/etc# drbdadm primary postgresql
debian1:/etc# cat /proc/drbd
version: 8.0.14 (api:86/proto:86)
GIT-hash: bb447522fc9a87d0069b7e14f0234911ebdab0f7 build by phil@fat-
tyre, 2008-11-12 16:40:33

 1: cs:WFConnection st:Primary/Unknown ds:UpToDate/DUnknown C r---
    ns:0 nr:18260 dw:18260 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0
       resync: used:0/61 hits:0 misses:0 starving:0 dirty:0 changed:0
       act_log: used:0/127 hits:0 misses:0 starving:0 dirty:0 changed:0
```

debian1 est bien devenu le nœud primaire. Nous pouvons enfin monter le disque :

```
debian2:/etc# mount /dev/drbd1 /var/lib/postgresql
debian2:/etc# ll /var/lib/postgresql/8.4/main/
base/      pg_clog/       pg_stat_tmp/   pg_tblspc/    PG_VERSION    postmaster.opts
global/    pg_multixact/  pg_subtrans/   pg_twophase/  pg_xlog/
```

Tout a l'air d'être là. Il ne nous reste plus qu'à démarrer PostgreSQL :

```
debian1:/etc# chown -R postgres:postgres /var/lib/postgresql/8.4/main
debian1:/etc# /etc/init.d/postgresql-8.4 start
Starting PostgreSQL 8.4 database server: main.
```

Testons si la nouvelle base de données **b3** existe bien :

```
debian1:/etc# su - postgres
postgres@debian1:~$ psql -l
                                  Liste des bases de données
   Nom    | Propriétaire | Encodage |     Tri     | Type caract. |    Droits d'accès
----------+--------------+----------+-------------+--------------+----------------------
 b1       | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 b2       | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 b3       | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 postgres | postgres     | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |
 template0 | postgres    | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |=c/postgres
                                                                 :postgres=CTc/postgres
 template1 | postgres    | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8  |=c/postgres
                                                                 :postgres=CTc/postgres
(6 lignes)
```

C'est bien le cas. Nos dernières modifications sur **debian2** sont bien là.

11 PETIT RÉCAPITULATIF

Avantages majeurs :

- plutôt simple à mettre en place et à configurer ;
- réplication synchrone ;
- documentation excellente ;
- intégration à Heartbeat / Pacemaker très facile.

Inconvénients majeurs :

- pas de granularité sur les objets à répliquer ;
- esclaves inaccessibles ;
- lenteur à prévoir ;
- deux serveurs uniquement (une configuration trois nœuds est disponible à partir de la version 8.3 de DRBD).

Inconvénient « mineur » :

- fonctionne sous Linux uniquement.

CONCLUSION

DRBD est un système simple à mettre en place. Son gros avantage est la possibilité d'avoir une réplication synchrone, son inconvénient direct est sa lenteur. ■

Auteur : Guillaume Lelarge

HAProxy : proxy TCP générique

Auteur

- Fabien Germain

...ou « comment protéger Apache de Slowloris » : j'ai hésité avec ce titre un peu plus racoleur, mais ce serait un peu trop restrictif. HAProxy est un (reverse) proxy travaillant aux niveaux 4 à 7 du modèle OSI, capable de faire de la répartition de charge et de la haute disponibilité sur les services TCP en général, et HTTP en particulier. Tout ceci avec une grande souplesse de configuration, une extrême fiabilité et des performances bluffantes... Il permet en prime de protéger votre serveur web favori (Apache) des attaques à la mode en ce moment. Tout un programme !

OBJECTIF(S)

Mettre en place une répartition de charge de manière simple, efficace et surtout souple. Le cas d'école consistera ici à configurer une machine destinée à équilibrer la charge et la répartition entre plusieurs serveurs HTTP Apache. Par effet de bord, cette configuration permettra d'obtenir une solution plus robuste face à des attaques par déni de service ou les problèmes de charge induits par des vers comme Slowloris dont on connaît malheureusement tous les effets.

OUTIL(S) UTILISÉ(S)

La toute première version de HAProxy, développée par Willy Tarreau, date de la fin de l'année 2000, et a beaucoup évolué en huit ans. HAProxy vient s'ajouter à la liste des reverse proxys et load-balancers libres, qui sont disponibles pour nos OS favoris Linux/BSD. Il est décrit comme étant particulièrement bien adapté pour le support de très fortes charges HTTP. Il est également capable de gérer la notion de « persistance », ainsi que la manipulation des données de la couche application.

Si vous connaissez un petit peu le domaine, des noms comme LVS (*Linux Virtual Server*), Pound ou même Apache (en reverse proxy) vous seront probablement venus à l'esprit. LVS est sûrement le load-balancer libre le plus connu et le plus répandu : Keepalived couplé à Heartbeat, mélangez bien, et vous obtenez un *cluster* haute disponibilité fiable. Que peut donc bien apporter HAProxy dans un domaine qui est déjà bien fourni ? En quelques mots : de la souplesse, de la performance et des fonctionnalités (très) avancées comme le filtrage niveau 7. Nous allons essayer de voir cela ensemble, au travers d'exemples concrets.

1 RÉGNER SUR LE MONDE DU BEIGNET

Pour illustrer cet article, nous allons partir avec une problématique classique dans le domaine du commerce en ligne. Nous gérons un site web de vente de beignets sur Internet, et notre pauvre serveur ne tient plus la charge tant la demande est forte (ça marche du tonnerre les e-beignets !). Pour simplifier le problème et nous concentrer sur la partie qui nous intéressera dans cet article, nous partirons du principe que le site est statique et les problématiques liées aux bases de données ne seront pas abordées. Nous pouvons donc simplement déployer des copies de notre site sur plusieurs machines différentes et rediriger les utilisateurs aléatoirement sur une des machines pour répartir la charge : le site sera identique, seule la machine changera, mais ce sera transparent pour l'utilisateur. Prenons trois machines identiques au niveau matériel, installons-y une Debian Lenny avec un Apache2 configuré pour délivrer au monde notre merveilleux site. Trois bêtes serveurs web, rien de plus compliqué. Durant nos tests, pour bien distinguer le serveur sur lequel nos utilisateurs seront redirigés, plaçons sur la première page du site, sur chaque serveur, un élément nous permettant de distinguer le serveur sur lequel nous nous trouvons (le nom ou l'adresse IP par exemple). En frontal de ces trois serveurs web, une autre machine se

et HTTP

chargera de faire la répartition de charge, grâce à Haproxy, en répartissant le trafic web sur les trois machines. C'est cette dernière machine (le load-balancer) que nous allons étudier, de près, les frontaux web n'ayant pour le moment que peu d'intérêt pour comprendre le fonctionnement. Voici un schéma récapitulatif de l'infrastructure :

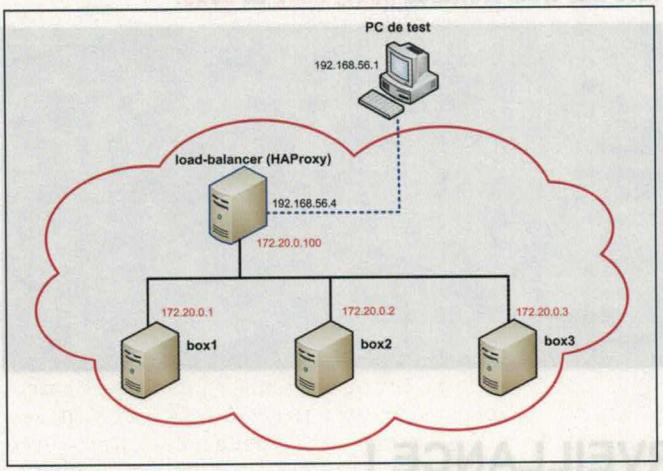


Fig. 1 : Notre infrastructure de test. A noter que le load-balancer est en coupure de ligne, et pourra donc également faire office de firewall si on le souhaite.

Les trois frontaux web ont une IP en 172.20.0.x. Le load-balancer a ici été installé en coupure de ligne, avec une interface réseau en 172.20.0.100 pour pouvoir joindre les frontaux web, et une seconde interface en 192.168.56.4 qui sera l'IP publique (i.e. vue de l'extérieur), de notre site web. Un utilisateur connecté à un PC de test (192.168.56.1) émet une requête avec son navigateur sur notre site **www.e-beignets.local**. Sa requête HTTP arrive sur le load-balancer, HAProxy sélectionne l'un des frontaux vers lequel rediriger la requête, effectue la requête en lieu et place du client, récupère le résultat renvoyé par le frontal, et le renvoie finalement au navigateur de notre utilisateur : la page s'affiche. Maintenant, trois utilisateurs se connectent en même temps. Nous allons pouvoir répartir chaque requête sur un frontal différent ; notre site s'affichera rapidement sur les navigateurs de nos trois utilisateurs. Sans notre petit cluster web, les trois requêtes seraient arrivées sur un seul et même serveur, qui aurait pris plus de temps à renvoyer la page demandée. Imaginez que la page est une page PHP complexe comportant de nombreuses requêtes SQL. Il faut un certain temps pour composer la page. De manière plus générale, si nous prenons l'hypothèse qu'un serveur peut tenir 50 requêtes simultanées, un cluster composé de trois serveurs identiques pourra donc en théorie tenir 150 requêtes simultanées. L'exemple est certes simpliste, mais il est là pour illustrer le principe.

Voilà pour la présentation rapide, passons aux choses sérieuses : comment installer, et configurer HAProxy pour rendre cette magie possible, et pouvoir régner sur le monde du beignet en ligne.

Deux choix s'offrent à nous pour l'installation. La première option : récupérer le **tar.gz** de la dernière version sur le site officiel (**http://haproxy.1wt.eu/**), compiler HAProxy et l'installer sur le système. La compilation est très rapide, HAProxy étant très léger. L'avantage, c'est que cela permet de partir avec la toute dernière version, sans bugs connus et avec les derniers *features* encore tout chaud. Quant à la seconde option, celle que nous allons adopter, puisque nous sommes sur une Debian et que quelqu'un s'est embêté à notre place pour intégrer proprement HAProxy au système de paquets :

```
root@haproxy:~# apt-get install haproxy
 [...]
root@haproxy:~# sed -i /etc/default/haproxy -e "s/ENABLED=0/ENABLED=1/"
```

HAProxy (version 1.3.15.2 au moment de la rédaction de cet article) est installé sur notre Lenny, et sera démarré au *boot* de la machine. La configuration se fait de manière classique, dans le répertoire **/etc/haproxy/**. Voici la toute première version de **haproxy.cfg** que nous allons utiliser :

```
listen beignets4ever 192.168.56.4:80
     mode http
     balance roundrobin
     option httpclose
     option httpchk HEAD /index.html HTTP/1.0
     server box1 172.20.0.1:80 check
     server box2 172.20.0.2:80 check
     server box3 172.20.0.3:80 check
```

Comme vous pouvez le constater, la syntaxe est claire et plutôt explicite. Entrons dans le détail :

- A l'aide de **listen**, nous demandons à HAProxy d'écouter sur l'IP et le port indiqué (à noter que l'on aurait pu simplement préciser **:80**, et HAProxy aurait répondu sur le port 80/tcp de toutes les IP de la machine). Les lignes suivantes sont les paramètres associés à ce couple IP/port écouté.
- Le **mode** indique que l'on va travailler sur des flux de type HTTP. HAProxy fera donc une analyse détaillée des requêtes clientes, avant de les faire suivre aux frontaux. Ce mode permettra plus tard de faire du filtrage de niveau 7. Les autres modes possibles sont **tcp** (connexions TCP génériques) ou **health** (ancien mode utilisé pour la surveillance par des composants externes, plus vraiment utile avec les versions récentes).
- **balance** permet de choisir l'algorithme de la répartition (choix) vers les frontaux. Le **roundrobin** est le plus classique et le plus simple : utilise les serveurs un à un, chacun son tour. A noter qu'il est possible d'affecter des poids particuliers aux ressources, par exemple pour utiliser deux fois plus souvent le frontal qui dispose d'une très grosse CPU et de beaucoup de RAM. Les autres modes possibles sont : **leastconn** (le serveur ayant précédemment reçu le moins de connexions est sélectionné), **source** (algorithme basé sur l'IP source du client), **uri** (basé sur le début de l'URI demandée), **url_param** (fonction de paramètres présents dans l'URL demandée) et **hdr** (fonction d'un champ présent dans l'en-tête HTTP : Host, User-Agent,...).

- L'option **httpclose** force à fermer la connexion HTTP une fois la requête renvoyée au client. On évite ainsi de conserver la connexion HTTP ouverte (« keep-alive ») et donc renvoyer systématiquement cette dernière vers le même frontal tant que la connexion reste ouverte. Cela peut être le comportement recherché, mais, dans le cadre de notre exemple, cela permettra de bien montrer, à chaque [CTRL+R] sur votre navigateur, que l'on tombera sur un frontal différent.
- L'option **httpchck** est suivi d'une requête HTTP qui sera utilisée pour vérifier qu'un frontal web est toujours en vie (activé grâce à l'option **check** de **server**). Si un frontal venait à ne plus répondre (ou pas comme attendu) à cette requête, il serait considéré comme hors service, serait sorti du pool des frontaux et les utilisateurs ne seraient plus redirigés vers ce dernier.
- **server** déclare le ou les serveurs frontaux utilisés pour assurer le service. Chaque « *server* » est nommé, et suivi de son IP/port de connexion. Il peut être suivi de l'option **check**, comme nous l'avons expliqué à l'instant.

Nous voilà paré, et c'est avec une émotion non dissimulée que nous pouvons démarrer HAProxy :

```
root@haproxy:~# /etc/init.d/haproxy start
Starting haproxy: haproxy.
root@haproxy:~#
root@haproxy:~# netstat -tpnl | grep haproxy
tcp        0      0 0.0.0.0:80              0.0.0.0:*
LISTEN      2861/haproxy
root@haproxy:~#
```

Il s'est correctement exécuté et écoute bien sur le port 80 de la machine : parfait ! Faisons pointer notre navigateur sur l'URL **http://192.168.56.4** (i.e. l'IP « publique » de notre site, représenté par le load-balancer, et vers lequel nous ferons pointer **www.e-beignets.local**). Nous constatons avec joie que l'on accède bien à notre site, ce qui est un bon début, mais surtout que l'on tombe à chaque requête sur un frontal web différent : la répartition de charge fonctionne correctement vers nos trois frontaux **box1**, **box2** et **box3**.

```
macbook-pro:~ fabien$ curl http://192.168.56.4
<html><body>
<h1>Les meilleurs beignets du web</h1></body></html>
<h3>box1 (172.20.0.1)</h3></body></html>

macbook-pro:~ fabien$ curl http://192.168.56.4
<html><body>
<h1>Les meilleurs beignets du web</h1></body></html>
<h3>box2 (172.20.0.2)</h3></body></html>

macbook-pro:~ fabien$ curl http://192.168.56.4
<html><body>
<h1>Les meilleurs beignets du web</h1></body></html>
<h3>box3 (172.20.0.3)</h3></body></html>
```

2 LES BEIGNETS SOUS SURVEILLANCE !

Notre cluster web fonctionne, c'est merveilleux. Mais comment savoir ce qui s'y passe, comment sont sollicités les différents frontaux, les éventuels temps d'indisponibilité, et j'en passe ? Là-bas au fond, j'ai entendu parler de Nagios : oui, mais nous n'en n'aurons pas besoin, car HAProxy dispose de sa propre interface de statistiques et de surveillance. Allons un petit peu plus loin dans notre configuration de load-balancer, et modifions le fichier **/etc/haproxy/haproxy.cfg** comme ceci :

```
defaults
      mode http
      option httpclose

frontend public
      bind :80
      default_backend beignets4ever

backend beignets4ever
     balance roundrobin
     option httpchk HEAD /index.html HTTP/1.0
     server box1 172.20.0.1:80 check
     server box2 172.20.0.2:80 check
     server box3 172.20.0.3:80 check
     stats uri /stats
     stats auth fabien:m3s_st4ts
```

Un petit peu plus complexe, mais cette configuration reste lisible et très compréhensible. Dans un premier bloc, nous définissons les paramètres par défaut. Viennent alors deux autres blocs : le *frontend*, et le *backend*. Le frontend représente la partie publique, celle qui est vue par les internautes. On y définit le port d'écoute, et le nom du backend à utiliser (**beignets4ever**) pour gérer les connexions. Le backend, c'est le moteur de la bête, le cluster web à proprement parler. Vous retrouvez toutes les options du premier exemple. Un petit nouveau s'est tout de même glissé : **stats**. Vous l'aurez sûrement deviné, c'est ce qui va activer la page de statistiques. **stats uri** définit l'endroit où les statistiques pourront être consultées (**http://192.168.56.4/stats**), et **stats auth** en sécurise l'accès en le protégeant par un nom d'utilisateur et un mot de passe, séparés par **:**. Mise à part ces deux dernières lignes, cette configuration produira un résultat identique à la précédente. C'est simplement une manière différente de la présenter. Voici à quoi ressemble la page de statistiques :

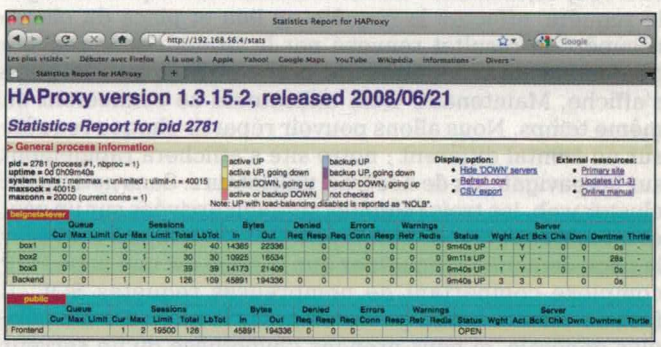


Fig. 2 : Les statistiques HAProxy

On y distingue nos deux blocs frontend (**public**) et backend (**beignets4ever**). Par exemple, dans la colonne « sessions » / « total », on voit que 126 connexions HTTP ont été gérées par HAProxy jusqu'à présent. Le détail de la répartition des requêtes est alors donné sur les lignes **box1**, **box2** et **box3**. D'ailleurs, pourquoi **box2** n'a reçu que 30 connexions, alors que **box1** et **box3** ont en reçues respectivement 40 et 39 ? Regardez un petit peu plus loin sur la ligne, l'*uptime* de

box2 n'est que de 9min11s, c'est moins que les deux autres frontaux. En effet, la colonne « Server » / « Downtime » montre 28 secondes d'indisponibilité, et « Down » nous donne 1 indisponibilité au total. En effet, un test a été effectué sur box2 pour vérifier que HAProxy détecte correctement que le service n'est plus disponible (un simple /etc/init.d/apache2 stop pour simuler un plantage du service web). HAProxy, une fois que le check a échoué sur le frontal box2, l'a sorti du pool, ce qui explique les chiffres observés. D'autres éléments intéressants sont présentés, comme les alertes (« *warnings* ») et les erreurs (nous verrons plus loin que cela peut être très utile), ainsi que les volumes de données ayant globalement transité en entrée et en sortie sur le cluster, mais également dans le détail sur les différents frontaux.

3 BEIGNETS ET COOKIES À LA CONQUÊTE DU MONDE

Finalement, comme je suis nul en web design, je me décide à me limiter à l'administration système et à payer un vrai professionnel pour me faire un vrai site web 2.0 : le luxe ultime et la consécration de toute une vie ! Les utilisateurs peuvent s'enregistrer sur le site, passer des commandes, lire des news sur l'univers très excitant du beignet, discuter entre beignet-addicts sur un forum dédié, etc. Oui mais voilà, il y a un problème : quand l'utilisateur s'authentifie sur le site (sur un des frontaux en réalité), une session est créée pour lui permettre d'accéder à tous les services web que nous proposons. A sa prochaine requête, notre load-balancer fera son travail et redirigera l'utilisateur sur un des trois frontaux, peut-être pas le même que celui sur lequel il s'est authentifié. Il se retrouvera alors déconnecté du site. Il retentera une nouvelle authentification, qui sautera sûrement à sa prochaine requête, et ainsi de suite. Comment faire ? Il faudrait que HAProxy puisse avoir le même fonctionnement (répartition de charge sur les différents frontaux), mais qu'un même utilisateur soit toujours redirigé vers le même frontal, afin de gérer correctement sa session s'il s'authentifie. La solution est la suivante :

```
defaults
      mode http
      option httpclose

frontend public
      bind :80
      default_backend beignets4ever
backend beignets4ever
      balance roundrobin
      cookie QUELSERVEUR insert indirect
      option httpchk HEAD /index.html HTTP/1.0
     server box1 172.20.0.1:80 cookie A check
      server box2 172.20.0.2:80 cookie B check      server box3
172.20.0.3:80 cookie C check
      stats uri /stats
      stats auth fabien:m3s_st4ts
```

Une ligne cookie est apparue, ainsi qu'un paramètre cookie dans les lignes server. Le principe est très simple : à la première connexion de l'utilisateur sur notre site, HAProxy va déterminer, selon l'algorithme sélectionné dans la configuration, vers quel frontal le rediriger. Quand HAProxy récupère auprès du frontal une copie de la page demandée par l'utilisateur, il la renvoie en y insérant un cookie QUELSERVEUR valant A B ou C, selon le frontal utilisé. Ainsi, à la prochaine requête de notre utilisateur, son navigateur renverra également ce cookie avec la requête : HAProxy, en fonction de la valeur, saura déterminer vers quel frontal renvoyer l'utilisateur. Ainsi, plus de problème de session !

Oui, mais :-) si le frontal avait un problème et venait à planter ? L'utilisateur ne pourrait plus se connecter à notre site ! Bien sûr que si. HAProxy saura déterminer que QUELSERVEUR pointe vers un serveur qui n'est plus actif dans le cluster. Il détruira le cookie en lui réassignant une nouvelle valeur pour pointer l'utilisateur vers un nouveau frontal. La session qui était en cours sera alors perdue. Il faudra se ré-authentifier. Mais le service sera toujours rendu. Notre cher client aura toujours accès à son site favori.

4 BIEN FILTRER SES COUCHES

Notre site vit à présent paisiblement sa petite vie, les visiteurs sont de plus en plus nombreux, et un besoin se fait de plus en plus pressant : nous souhaitons mettre en place une liste de diffusion pour les tenir informés de l'actualité du site. Par contre, ce nouveau service sera géré sur un serveur dédié à cette tâche, mais nous ne voulons pas que les utilisateurs aient connaissance de notre infrastructure technique. Le service doit donc être intégré dans notre site directement. HAProxy vient une fois de plus à notre secours, avec ses fonctions avancées de filtrage de la couche 7 (modèle OSI). Concrètement, nous allons lui demander de rediriger les requêtes vers un serveur spécifique lorsqu'une URL particulière est demandée. Nous avons choisi http://www.e-beignets.local/mailing-list/ (ou http://192.168.56.4/mailing-list/ sans utiliser le DNS). On aurait également pu choisir un sous-domaine particulier, par exemple http://mailing-list.e-beignets.local. A vrai dire, à peu près tout est possible. Voici ce que donne notre fichier haproxy.cfg, volontairement simplifié par rapport à avant, pour en faciliter la compréhension :

```
defaults
     mode http
    option httpclose
frontend public
    bind :80
   acl url_mailing_list path_end -i /mailing-list/
     use_backend mailinglist if url_mailing_list
    default_backend beignets4ever
backend beignets4ever
    balance roundrobin
     option httpchk HEAD /index.html HTTP/1.0
    server box1 172.20.0.1:80 check
     server box2 172.20.0.2:80 check
```

```
    server box3 172.20.0.3:80 check
backend mailinglist
    balance roundrobin
    server boxml 172.20.0.15:80 check
```

On a défini une ACL nommée **url_mailing_list**, pour récupérer toutes les URL qui se terminent par **/mailing-list/**. On rajoute ensuite une directive **use_backend** : on utilisera le backend **mailinglist** (plutôt que le défaut : **beignets4ever**) si l'ACL **url_mailing_list** est détectée. Ainsi, un utilisateur qui demande **http://192.168.56.4** sera servi par **box1**, **box2** ou **box3**. Par contre, s'il demande **http://192.168.56.4/mailing-list/**, il sera servi par **boxml**.

Vu de l'utilisateur, il s'agit d'un seul et même site, probablement un seul serveur pour gérer tout ça, puisqu'il ne voit qu'une IP publique. En pratique, quatre machines se répartissent le travail, en fonction de la provenance de l'utilisateur, s'il s'est déjà connecté ou pas, de la partie du site à laquelle il accède, etc. Les possibilités sont très nombreuses, et cet article n'a pas pour vocation d'être exhaustif, loin de là. Les fonctionnalités de HAProxy sont très nombreuses. Nous ne pouvons en présenter qu'une petite partie. Après cet aperçu des bases du fonctionnement, voyons maintenant un dernier exemple, concret et tristement d'actualité.

5 PROTÉGER L'INDIEN DE L'AMOUR SANS FIN DES PETITS ANIMAUX

Une attaque nommée « Slowloris » (**http://ha.ckers.org/slowloris/**) a fait parlé d'elle il y a quelques mois : comment, avec une simple ligne ADSL, écrouler de gros sites web tournant sous Apache. Le principe de l'attaque est très simple : ouvrir de nombreuses connexions en n'envoyant que des requêtes HTTP partielles, et, au compte-goutte de préférence, pour conserver les connexions ouvertes le plus longtemps possible et ainsi saturer les ressources du serveur. Il est ainsi très simple de consommer les *n* connexions simultanées gérées par le serveur, et de rendre le site inaccessible ! Une petite démonstration. Vous n'y croyez pas ? Bien sûr, la voici :

Depuis le serveur HAProxy (qui fait ici office de machine d'attaque), on lance Slowloris vers **box1** :

```
root@haproxy:~# ./slowloris.pl -dns 172.20.0.1
  [..]
Defaulting to port 80.
Defaulting to a 5 second tcp connection timeout.
Defaulting to a 100 second re-try timeout.
Defaulting to 1000 connections.
Multithreading enabled.
Connecting to 172.20.0.1:80 every 100 seconds with 1000 sockets:
    Building sockets.
    Building sockets.
    Building sockets.
    Building sockets.
    Sending data.
  [..]
```

C'est fait, c'est lancé. Mais est-ce que le serveur web (Apache) de **box1** répond toujours ?

```
root@haproxy:~# telnet 172.20.0.1 80
Trying 172.20.0.1...
^C
root@haproxy:~#
```

Eh bien, non. Manifestement, il n'est plus joignable quelques secondes seulement après le lancement de Slowloris ! Redoutable. Un petit tour sur **box1** nous permet de voir rapidement pourquoi :

```
box1:~# netstat -tan | grep 172.20.0.1:80 | wc -l
371
box1:~#
```

Déjà 371 connexions simultanées sur le pauvre Apache de **box1**, et ça ne cesse de grimper, utilisant ainsi toutes les ressources du serveur, qui n'est donc plus en mesure d'accepter d'autres connexions. L'attaque est d'une extrême simplicité et d'une grande efficacité. Il faut se rendre à l'évidence : n'importe qui est capable de faire tomber votre serveur web. Certes, avec un système de monitoring adéquat, vous seriez immédiatement prévenu de l'indisponibilité de ce dernier. Un simple **netstat** vous mettrait rapidement en évidence la cause de ce problème, et un petit coup de **iptables -I INPUT -s IP_FAUTIVE -j DROP** réglerait le problème sans tarder... jusqu'au prochain petit malin qui enverra un Slowloris depuis une autre IP. C'est un jeu sans fin, et, à mon avis, vous vous lasserez vite, surtout face à quelqu'un qui vous en veut vraiment et qui dispose d'un *botnet* de quelques centaines ou milliers de machines ;-)

Comment faire dans ce cas ? Question stupide, vous vous doutez de la réponse : une nouvelle fois, faisons appel à HAProxy pour nous protéger, tel un chevalier filant sur son beignet volant. La configuration ci-dessous est celle qui est fournie sur le site de HAProxy. Nous l'avons honteusement récupérée et adaptée à notre exemple. Mais, nous n'utiliserons ici qu'un seul frontal sur les trois, pour nous trouver dans des conditions identiques aux précédents tests avec Slowloris (à savoir un attaquant vers un seul frontal), et pouvoir ainsi réellement comparer l'apport de HAProxy. Au lieu d'attaquer directement **box1**, nous attaquerons le load-balancer HAProxy, qui fera (ou non) le relais vers **box1**. Voici le nouveau fichier **haproxy.cfg** :

```
# This configuration is meant to be installed in front of an existing web
# server that needs some DoS protection. We assume that the web server has been
# moved to port 8080 on the loopback, and that haproxy 1.3.18 is running on
# port 80. Note that Apache will have to be configured to get the client's IP
# address from the X-Forwarded-For header (mod_rpaf can do that).
# This configuration prevents "slowloris" and "nkiller2" from degrading the
# service. See below for more options.
global
  daemon
  maxconn 20000       # count about 1 GB per 20000 connections
  pidfile /var/run/haproxy.pid
  stats socket /var/run/haproxy.stat mode 600
defaults
  mode http
  maxconn 19500       # Should be slightly smaller than global.maxconn.
  timeout client 60s  # Client and server timeout must match the longest
  timeout server 60s  # time we may wait for a response from the server.
  timeout queue  60s  # Don't queue requests too long if saturated.
  timeout connect 4s  # There's no reason to change this one.
  timeout http-request 5s # A complete request may never take that long.
  # Uncomment the following one to protect against nkiller2. But warning!
  # some slow clients might sometimes receive truncated data if last
  # segment is lost and never retransmitted :
  # option nolinger
  option httpclose
  option abortonclose
```

```
  balance roundrobin
  option forwardfor    # set the client's IP in X-Forwarded-For.
  retries 2
frontend public
  bind :80             # or any other IP:port combination we listen to.
  default_backend apache
backend apache
  # set the maxconn parameter below to match Apache's MaxClients minus
  # one or two connections so that you can still directly connect to it.
  balance roundrobin
  server box1 172.20.0.1:80 maxconn 254
# Enable the stats page on a dedicated port (8888). Monitoring request errors
# on the frontend will tell us how many potential attacks were blocked.
listen stats
  bind      :8888
  stats uri /
```

Nous retrouvons les grandes lignes de notre précédente configuration : un frontend qui écoute sur le port 80/tcp, un backend composé d'un seul frontal web, et les statistiques qui écoutent cette fois-ci sur un port dédié (le 8888/tcp). Mais, surtout, un certain nombre de paramètres par défaut sont définis pour les *timeouts* (en particulier le **http-request** à 5 secondes). Rien de bien complexe à première vue. Vérifions si cela fonctionne comme on l'espère, en lançant une attaque Slowloris contre notre load-balancer fraîchement redémarré sur sa configuration à l'épreuve des balles :

```
MacBook-Pro:~ fabien$ ./slowloris.pl -dns 192.168.56.4
  [...]
 Welcome to Slowloris - the low bandwidth, yet greedy and poisonous HTTP client
Defaulting to port 80.
Defaulting to a 5 second tcp connection timeout.
Defaulting to a 100 second re-try timeout.
Defaulting to 1000 connections.
Multithreading enabled.
Connecting to 192.168.56.4:80 every 100 seconds with 1000 sockets:
    Building sockets.
    Building sockets.
    Building sockets.
    Building sockets.
    Building sockets.
  [...]
```

Vérifions immédiatement comment cela est vécu par le load-balancer (mal, pensez-vous ?) :

```
root@haproxy:~# netstat -tan | grep 192.168.56.4:80.*ESTAB | wc -l
252
root@haproxy:~#
root@haproxy:~# ps aux | awk '/(^USER|sbin\/haproxy)/ {print}'
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      2190  0.3  3.6  12476  4576 ?        Ss   01:40   0:04 /usr/
sbin/haproxy -f /etc/haproxy/haproxy.cfg -D -p /var/run/haproxy.pid
```

A vrai dire, il ne sent pas grand chose : 0.3% de cpu utilisé par le process **haproxy** pendant l'attaque, et quelques Ko de mémoire... alors que cette même attaque dirigée directement contre **box1** quelques minutes auparavant l'avait rendu inaccessible sur le champ ! Plutôt pas mal, n'est-ce pas ? Et pour resituer les choses dans leur contexte, la machine attaquante est une machine disposant d'un Core 2 Duo 2.26 Ghz avec 2 Go de RAM (plutôt puissante donc), alors que le load-balancer qui subit les assauts n'est qu'une simple machine virtuelle (Virtualbox) avec une seule CPU et seulement 128 Mo de RAM. Ça force encore plus le respect, non ?

Et sinon, pendant ce temps, comment va la vie pour **box1** ? Tout va on ne peut mieux pour elle, merci de vous en inquiéter :

```
debian1:~# uptime
 02:09:33 up 29 min,  1 user,  load average: 0.00, 0.00, 0.00
debian1:~#
debian1:~# netstat -tan | grep 172.20.0.1:80.*ESTAB | wc -l
0
```

Avec une charge de 0.00 pendant l'attaque, et aucune des 250 connexions HTTP lancées par Slowloris n'arrivant jusqu'à **box1**, on peut difficilement faire mieux. Et, bien sûr, pour un client légitime essayant d'accéder à notre site web, l'accès est immédiat et passe au travers du filtrage :

```
macbook-pro:~ fabien$ time curl http://192.168.56.4
<html><body>
<h1>Les meilleurs beignets du web</h1></body></html>
<h3>box1 (172.20.0.1)</h3></body></html>
real 0m0.027s
user 0m0.003s
sys 0m0.004s
```

Pour finir de vous convaincre, jetons un œil aux statistiques de HAProxy (en figure 3) après quelques attaques :

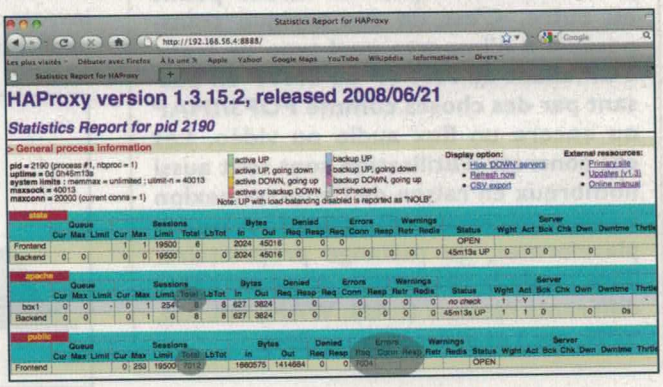


Fig. 3 : Statistiques HAProxy après une attaque Slowloris

Le frontend « public » a reçu au total 7012 connexions, dont 7004 qui sont parties en erreur (car rejetées par HAProxy)... et seulement 8 ont été correctement « proxifiées » vers **box1** : les 8 seules vraies connexions à notre site web. CQFD :-)

CONCLUSION

Cet article vous a rapidement présenté les fonctionnalités de base de HAProxy, un reverse proxy/load-balancer léger, performant et très flexible. Ses possibilités de protection des attaques de type Slowloris contre les serveurs Apache sont extrêmement intéressantes, mais ne sont qu'une des nombreuses choses que Haproxy sache faire. Si vous êtes curieux, je vous invite à consulter l'*Architecture Guide* (http://haproxy.1wt.eu/download/1.3/doc/architecture.txt), qui vous présentera des infrastructures plus complexes que celles évoquées dans cette introduction, et que l'on rencontre couramment sur Internet pour héberger le monde merveilleux du web 2.0. Mais... vous reprendrez bien un petit beignet pour la route ? ■

Auteur : Fabien Germain

TIMTOWTDIHTTPSProxy

TIMTOWTDI [1], ça vous dit quelque chose ? « There's more than one way to do it » ou, en « bon » français, « il y a plus d'une façon de faire ». Cet article va mettre en œuvre l'adage de Perl [2] pour un cas concret de redirection de trafic HTTPS [3].

Auteur

- Laurent Gautrot

OBJECTIF(S)

Rediriger le trafic HTTP over SSL vers une machine ou un serveur qui n'est pas accessible depuis un réseau public ou autre. C'est un cas typique transposable à n'importe quel type de trafic TCP allant de l'HTTP au SMTP en passant par des choses comme POP3/IMAP ou encore un flux audio ou vidéo. Les cas concrets d'utilisation sont tout aussi nombreux en raison de l'interconnexion souvent nécessaire d'applications multiplateformes, du mélange propriétaire/libre ou de la réalisation de ponts entre différents réseaux (LAN, WAN, WLAN, VPN, etc.) et différents services.

OUTIL(S) UTILISÉ(S)

Nous avons ici non pas un seul mais une collection complète d'outils. Comme le précise le titre même de cet article, il y a plus d'une manière de faire, bien plus. Super serveur inetd, tunnel SSH, Netfilter, cat réseau, Perl, Proxy, serveur HTTP, load balancing... toutes ces solutions seront ici avancées et analysées. A vous de choisir celle qui s'adaptera le mieux à votre configuration et surtout à vos besoins.

1 OBJECTIF

Suite à une question sur un cas pratique de la vraie vie, avec des vraies contraintes de la vraie vie, je me suis dit qu'un petit tour d'horizon des manières de rediriger un port TCP pourrait être bienvenu. Compte tenu de tout ce qu'il est possible de faire avec des bits, la liste n'est pas exhaustive, et c'est normal.

Le but de cette opération est de disposer d'une solution pour rediriger du trafic HTTPS vers une machine non atteignable d'un réseau. Il faut donc configurer d'une façon ou d'une autre un serveur mandataire inverse ou une redirection de port TCP. La machine interne héberge une application en boîte noire, qu'il n'est pas possible d'utiliser directement.

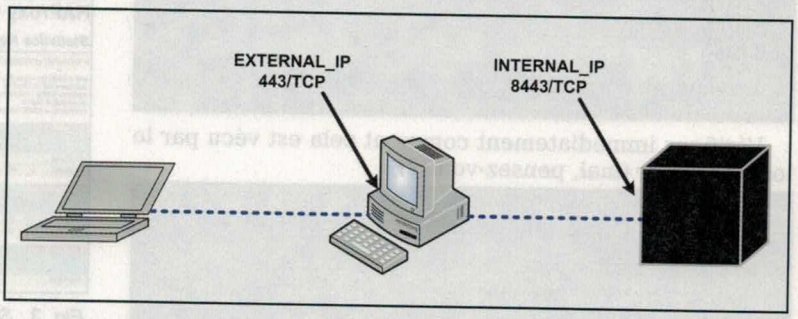


Fig. 1 : Schéma réseau de redirection HTTPS

Dans tous les exemples ci-après, `EXTERNAL_IP` est à substituer avec l'adresse de réseau « publique » utilisable par tout client pour pouvoir utiliser le service. `INTERNAL_IP` correspond à l'adresse du serveur qui héberge la boîte noire sur le réseau interne. Bien que cette adresse ne soit pas publiquement connue, elle est utilisable par le relais pour s'y connecter et transmettre les requêtes et réponses.

2 ROUTAGE SIMPLE

La première approche, qui est aussi la plus simple, consiste à acheminer sans aucune autre forme de procès les datagrammes à destination de l'hôte sur le réseau interne et les retours.

C'est, d'une certaine manière, l'attitude de l'autruche, qui laisse passer un flux, sans se poser de question sur la nature ou la légitimité des informations échangées.

On trouve alors deux manières d'implanter ce routage :

- En espace utilisateur, on trouve, parmi tant d'autres, **rinetd**, **inetd** (avec un petit coup de **nc** au passage), **xinetd**, OpenSSH, **socat** ou encore l'excellent module Perl **Net::Proxy**.
- En espace noyau, même s'il y a potentiellement de nombreuses manières d'obtenir ce résultat, on survolera deux solutions utilisant les pare-feu, à savoir NetFilter et pf.

2.1 rinetd

En espace utilisateur, le plus simple est peut-être d'utiliser, s'il est disponible, un outil comme **rinetd** **[4]**. Ce *daemon* est exactement prévu pour cet usage. Il suffit de préciser les adresses de réseau et ports TCP internes et externes, et le tour est joué.

```
EXTERNAL_IP 443 INTERNAL_IP 8443
```

2.2 inetd

Le vénérable **inetd** **[5]**, issu d'OpenBSD 4.3, a l'avantage de la simplicité. Sans revenir sur son fonctionnement, c'est un super serveur, qui peut être utilisé éventuellement pour des redirections, même si ce n'est pas son rôle. Pour établir les connexions à distance, on utilisera une commande externe comme **netcat** **[6]**, qui fera très bien l'affaire.

```
EXTERNAL_IP:https  stream  tcp  nowait  www-data  /usr/bin/nc INTERNAL_IP 8443
```

L'astuce ici consiste à préfixer le nom du service (il s'agit ici de **https**) par l'adresse d'écoute suivie du caractère « **:** » (deux-points).

2.3 xinetd

xinetd **[7]** réimplémente la logique d'**inetd** en intégrant en standard des notions de sécurité (ACL avec **only_from** et **no_access**, liaison aux *TCP_wrappers* en standard) et des fonctionnalités très intéressantes, comme... la redirection !

La page de manuel de **xinetd** **[8]** est très claire et comporte des exemples très intéressants.

```
service https
{
  flags = REUSE
  socket_type = stream
  wait = no
  user = www-data
  bind = EXTERNAL_IP
  redirect = INTERNAL_IP 8443
}
```

2.4 socat

socat **[9]** est une autre boîte à outils intéressante, qui permet très facilement la redirection de trafic d'un port TCP à un autre. Pas d'intelligence applicative ici. C'est juste aux niveaux réseau et transport que cela se passe.

```
socat TCP-LISTEN:443,bind=EXTERNAL_IP,fork,reuseaddr TCP:INTERNAL_IP:8443
```

Là encore, la documentation de **socat** fournit de précieux exemples.

2.5 OpenSSH

Cette solution est franchement « capillo-tractée ». Lancer un processus **ssh** pour démarrer un nouveau daemon **sshd** pour chiffrer du contenu HTTPS dans du SSH n'est pas ce que l'on fait de plus léger. Pourtant, il faut bien reconnaître que c'est possible. Mais, qui sera autorisé à se connecter sur la boucle locale, et quelle sera la méthode d'authentification choisie ?

Il reste possible de choisir le chiffrement le plus rapide et le plus léger pour cette opération. Pourquoi pas *blowfish* ? (Cf. l'option **Ciphers** de **ssh_config(5)** **[10]**)

```
ssh -L EXTERNAL_IP:443:INTERNAL_IP:8443 localhost
```

2.6 Net::Proxy

Cette dernière solution de routage (pas la moindre) en espace utilisateur utilise le module **Net::Proxy** **[11]** de Philippe « BooK » Bruhat.

```
#!/usr/bin/perl
use strict;
use warnings;
use Net::Proxy;
my $proxy = Net::Proxy->new(
  {
    in  => { type => 'tcp', host => 'EXTERNAL_IP', port => '443' },
    out => { type => 'tcp', host => 'INTERNAL_IP', port => '8443' },
  }
);
$proxy->register();
Net::Proxy->mainloop();
```

2.7 NetFilter

Il est possible dans Linux, en espace noyau, de réaliser cette redirection très simplement avec une règle de NetFilter **[12]**.

```
iptables -A PREROUTING -t nat -p tcp -d EXTERNAL_IP dport 443 -j DNAT
to INTERNAL_IP:8443
```

2.8 pf

pf **[13]** le *packet filter* de *BSD permet très facilement la redirection. La syntaxe (qui ne s'invente pas) ressemblerait à ceci :

```
rdr on rl0 to EXTERNAL_IP port 443 -> INTERNAL_IP port 8443
```

3 LES « VRAIS » SERVEURS MANDATAIRES

Dans cette catégorie, on trouvera des solutions fondamentalement différentes de ce qui a été vu plus haut. C'est au niveau « application » que tout va se jouer.

Le fait est que, à cet endroit, on constate une rupture de protocole au niveau du serveur mandataire inverse. En effet, lorsque l'on configure un proxy de cette façon, le flux HTTPS est décodé au passage dans le mandataire, et est chiffré avant d'être transmis en sortie. C'est très intéressant en pratique pour implémenter des filtres à la volée. Mais, si l'on ne souhaite pas casser le protocole en cours de route, on pourra utiliser une solution plus « opaque » de redirection de port TCP classique.

Les solutions évoquées de manière non exhaustive sont Apache, Squid, Pound, lighttpd et nginx.

3.1 Apache

Évidemment, comme il s'agit de trafic HTTPS, on peut penser immédiatement au serveur web Apache **[14]**, à son **mod_proxy** **[15]** et son **mod_ssl** **[16]**. En pratique, on aura aussi certainement intérêt à utiliser **mod_rewrite** **[17]**.

```
NameVirtualHost EXTERNAL_IP:443
<VirtualHost EXTERNAL_IP:443>
  # ... directives classiques comme ServerName, ServerAlias,
  # ServerAdmin, CustomLog, etc.
  SSLEngine On
  SSLCertificateFile /some/path/to/crt
  SSLCertificateKeyFile /some/other/path/to/key
  SSLCACertificateFile /some/path/to/ca/crt
  SSLProxyEngine On
  RewriteEngine On
  RewriteRule ^/(.*) http://INTERNAL_IP:8443/$1 [P,L]
  ProxyPassReverse / https://INTERNAL_IP:8443/
</VirtualHost>
```

Pour mémoire, la documentation d'Apache contient un tutoriel simple pour générer soi-même ses certificats auto-signés **[18]** à l'aide d'OpenSSL. Je ne suis pas en train de pousser à l'utilisation des certificats auto-signés, je signale juste l'existence de la documentation.

3.2 Squid

Squid **[19]** est avant tout un serveur mandataire et cache web, mais il peut aussi servir de serveur mandataire inverse.

```
https_port EXTERNAL_IP:443
cert=/some/path/to/server.crt
key=/some/other/path/to/server.key
cafile=/some/path/to/ca.pem
cache_peer INTERNAL_IP parent 8443  0  no-query originserver
```

Ce qui précède est un morceau du fichier de configuration.

3.3 lighttpd

lighttpd **[20]** est une étoile montante du web 2.0. *Ultra-fraîme* donc, mais aussi terriblement efficace. La configuration d'un reverse proxy HTTP est enfantine. Il en va de même pour un reverse proxy HTTPS.

```
$SERVER["socket"] == "EXTERNAL_IP:443" {
   ssl.engine      = "enable"
   ssl.pemfile     = "/some/path/to/server.crt"
   ssl.ca-file     = "/some/path/to/ca.pem"
   proxy.server = (
       "" => ( ( "host" => "INTERNAL_IP", "port" => 8443 ) )
   )
}
```

3.4 Pound

Pound **[21]** est un répartiteur de charge web. Sa configuration est plutôt claire.

```
ListenHTTPS
  Address EXTERNAL_IP
  Port 443
  Cert    "/some/path/to/server.pem"

  Service
    BackEnd
```

```
        Address INTERNAL_IP
        Port 8443
    End
  End
End
```

Normalement, l'intérêt de Pound réside dans sa capacité à répartir des charges et à assurer de la haute disponibilité. Dans ce cas, c'est très réducteur. Outre la facilité de configuration, son autre avantage est la validation des requêtes avant transmission au *backend*.

3.5 nginx

nginx **[22]** est un serveur web, qui peut aussi être utilisé pour répondre aux requêtes à destination d'autres hôtes.

Il est très léger, et sa configuration est simple. Il faut juste la trouver... sur le *wiki* **[23]**.

```
server {
    listen              EXTERNAL_IP:443;
    server_name         _;
    ssl                 on;
    ssl_certificate     cert.pem;
    ssl_certificate_key cert.key;
    ssl_session_timeout 5m;
    ssl_protocols SSLv2 SSLv3 TLSv1;
    ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
    ssl_prefer_server_ciphers  on;

    location / {
        proxy_pass  https://INTERNAL_IP:8443;
    }
}
```

CONCLUSION (DÉJÀ ?)

La liste pourrait continuer encore longtemps...

Les solutions sont pratiquement infinies. Il y a beaucoup d'autres solutions, ouvertes ou propriétaires pour rediriger et/ou filtrer du contenu HTTPS.

Vous en connaissez d'ailleurs beaucoup d'autres, qui sont mieux, plus belles, plus efficaces. N'hésitez pas à en faire la promotion !

À suivre, donc... ■

Références

- **[1]** TIMTOWTDI – http://en.wikipedia.org/wiki/There%27s_more_than_one_way_to_do_it
- **[2]** Perl – http://www.perl.org/
- **[3]** HTTPS, RFC2818 – http://tools.ietf.org/html/rfc2818
- **[4]** rinetd – http://www.boutell.com/rinetd/
- **[5]** inetd – http://www.openbsd.org/cgi-bin/man.cgi?query=inetd&sektion=8
- **[6]** GNU netcat – http://netcat.sf.net/
- **[7]** xinetd – http://www.xinetd.org/
- **[8]** xinetd.conf(5)
- **[9]** socat – http://www.dest-unreach.org/socat/
- **[10]** ssh_config(5)
- **[11]** Net::Proxy – http://search.cpan.org/~book/Net-Proxy/lib/Net/Proxy.pm
- **[12]** NetFilter – http://www.netfilter.org/
- **[13]** pf – http://www.openbsd.org/faq/pf/fr/index.html
- **[14]** Apache HTTPd – http://httpd.apache.org/
- **[15]** mod_proxy – http://httpd.apache.org/docs/2.2/mod/mod_proxy.html
- **[16]** mod_ssl – http://httpd.apache.org/docs/2.2/mod/mod_ssl.html
- **[17]** mod_rewrite – http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html
- **[18]** Tutoriel pour la création de certificats auto-signés – http://httpd.apache.org/docs/2.2/ssl/ssl_faq.html#selfcert
- **[19]** Squid – http://www.squid-cache.org/
- **[20]** lighttpd – http://www.lighttpd.net/
- **[21]** Pound – http://www.apsis.ch/pound/index_html
- **[22]** nginx – http://www.nginx.org/
- **[23]** Documentation de nginx – http://wiki.nginx.org/Main

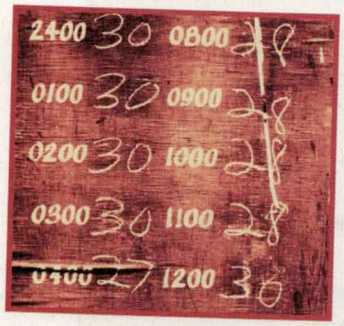
Auteur : Laurent Gautrot

Remerciements

Pour la question de Maxime Goepp, pour la provocation de Nicolas Chuche, merci ! Cela fait un os à ronger.

Un grand merci aux Mongueurs francophones pour la relecture !

Développement avec Bazaar,



Auteur

- Guillaume Mazoyer

Lorsque l'on décide de rendre un projet de programmation public, il y a des tas de manières de faire. On peut simplement mettre à disposition une archive du code source ou on peut aussi utiliser les différents gestionnaires de versions existants. Il est souvent difficile d'en faire le choix lors du début d'un projet. En effet, on ne sait pas toujours si ce dernier profitera mieux d'un système centralisé ou décentralisé. Bien qu'il soit possible de changer en cours de route, bien choisir dès le départ peut permettre de mieux se concentrer sur le code que sur la mise en ligne du projet. C'est dans ce but que j'ai choisi d'utiliser le trio Bazaar/SSH/Trac, parce qu'il mélange à la fois, simplicité, sécurité et polyvalence.

OBJECTIF(S)

Installer une solution permettant le développement collaboratif en privilégiant le développement lui-même et non la maintenance de la solution collaborative. En d'autres termes, il s'agit de trouver et d'utiliser un système simple, efficace et souple où l'énergie reste dans le développement et non l'administration. Il faudra, de plus, que la solution puisse être facilement adaptable si le projet en vient à passer d'une poignée de développeurs à des dizaines, voire plus.

OUTIL(S) UTILISÉ(S)

Bazaar ou bzr est l'un des nombreux systèmes de gestion de versions en logiciel libre disponibles (RCS, CVS, Git, etc.). C'est un système de gestion de versions décentralisée où chaque copie est un dépôt complet (historique inclus). Le développement de Bazaar est focalisé sur la simplicité et la flexibilité. Trac est un système de gestion de projets développé en Python (comme Bazaar). Il intègre notamment un wiki, une gestion de feuilles de route, un historique, un système de suivi de bugs et un explorateur Subversion.

1 BAZAAR, SSH ET TRAC, C'EST QUOI ?

1.1 Bazaar (bzr), le gestionnaire de versions

Bazaar (aussi abrégé « bzr ») est un système de gestion de versions. Il se veut simple d'où la citation : « *Version Control for Human Beings* » qui signifie « Contrôle de version pour les êtres humains ». Il est conçu pour faciliter la collaboration des développeurs dans un projet et aussi conserver un historique de l'évolution d'un projet. Faire le choix d'un gestionnaire de versions, c'est faire le choix d'une manière de gérer le projet. En effet, il y a les systèmes centralisés (tels que CVS et SVN) et il y a les systèmes décentralisés (comme Git et Mercurial). Bazaar, quant à lui, peut faire les deux. Ceci peut donc se révéler être un avantage si, par la suite, le projet veut changer de schéma. Dans cet article, nous l'utiliserons de manière décentralisée. La branche principale du projet sera accessible en écriture seulement par quelques personnes (une seule peut suffire d'ailleurs), alors que n'importe qui aura le droit de récupérer le code. Seuls les développeurs du projet pourront, bien entendu, envoyer leurs branches de développement sur le serveur centralisant les données (voir Fig.1).

1.2 Secure Shell (SSH), le gestionnaire d'authentification

Le protocole *Secure Shell* est connu comme étant un des moyens les plus sécurisés pour contrôler une machine à distance. C'est d'ailleurs son utilisation la plus commune. La connexion par SSH à une machine peut se faire soit à l'aide d'un mot de passe, soit en utilisant une ou plusieurs clefs. La seconde option offre l'avantage de ne pas entrer un mot de passe à chaque connexion, mais elle est

SSH et Trac

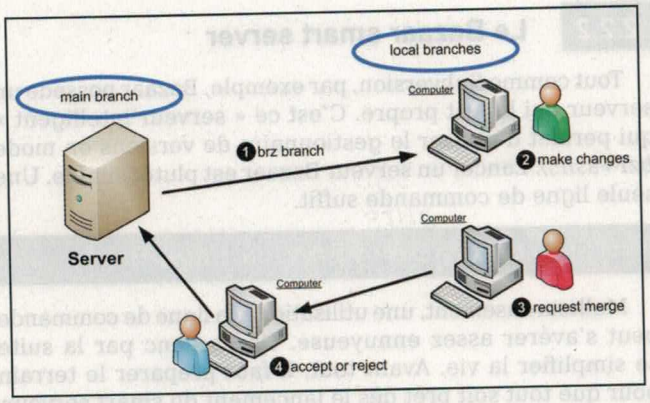


Fig.1 : Principe de Bazaar dans notre cas

surtout plus sécurisée, puisque la clé privée doit se trouver sur la machine d'où l'on se connecte. Pourquoi parler de SSH alors que l'on veut utiliser un système de gestion de versions ? En fait, Bazaar permet d'utiliser SSH comme système d'authentification. En gros, seules les personnes ayant un moyen de se connecter via SSH pourront avoir des accès spéciaux vers les branches Bazaar. Pour utiliser SSH avec Bazaar, on utilise soit des URL de type *sftp://*, soit des URL de type *bzr+ssh://*. C'est cette dernière forme que nous allons garder ici. Mais, pour cela, il faudra également se servir d'un serveur Bazaar (appelé aussi « *Bazaar smart server* »).

1.3 Trac, la gestion du projet par Internet

Trac est un système complet de gestion de projet. Il est développé en Python tout comme Bazaar et intègre un *wiki*, une gestion des feuilles de route, un historique, un outil de rapport de bug et un explorateur du code source. Trac est généralement utilisé en couple avec Subversion, mais il existe des greffons pour le faire cohabiter avec d'autres gestionnaires de versions. Dans notre cas, nous allons donc utiliser le greffon destiné à pouvoir utiliser Trac et Bazaar ensemble. Je reconnais que ce n'est pas toujours simple à prendre en main, en particulier la première fois. Mais, depuis quelque temps maintenant, seule la création du projet Trac requiert une utilisation de la ligne de commande plus ou moins difficile. Une fois le tout installé, il est possible d'administrer le système via l'interface web de Trac.

1.4 Avant de se lancer

L'installation et la configuration que nous allons réaliser n'est pas forcément la plus adaptée pour votre projet. Vous pouvez tout de même vous inspirer de cet article afin de comprendre les principes du fonctionnement de Bazaar. Cet article a été rédigé en utilisant la distribution Ubuntu Server (version 9.04 Jaunty Jackalope). Il se peut donc que les noms des paquets diffèrent d'une distribution à une autre. De plus, au début de l'installation, des dépôts PPA (hébergés grâce à Launchpad) sont utilisés. Il est donc nécessaire de vous rappeler que ce sont des dépôts non officiels et qu'il peut être dangereux de les utiliser. Cependant, ils proposent Bazaar dans sa dernière version. Rien ne vous empêche d'utiliser les dépôts officiels avec la version de Bazaar qui vous est proposée. Enfin, dernière chose : toutes les manipulations effectuées sur le serveur sont faites avec l'utilisateur *root*. Vous devez donc vous assurer que vous pouvez faire de même.

2 MISE EN PLACE DE BAZAAR, SSH ET TRAC

2.1 Configuration du serveur SSH

Lorsque l'on utilise une distribution pour un serveur, on a l'habitude d'installer un serveur SSH (si ce n'est pas déjà fait).

```
# aptitude install openssh-server
```

Par défaut, ce serveur va écouter sur le port 22. Il est conseillé de garder cette configuration pour coupler SSH à Bazaar. La configuration du serveur se fait via le fichier **/etc/ssh/sshd_config**. La configuration recommandée de base est de ne pas permettre de se connecter en root directement. Le fichier de configuration va donc contenir au moins ces quelques lignes :

```
...
# What ports, IPs and protocols we listen for
Port 22
...
PermitRootLogin no
...
```

On peut désormais configurer notre client SSH. L'authentification par clef est à la fois pratique et sécurisée. C'est donc celle qu'il faut préférer. Il faudra générer une clef sur chaque client. Pour cela, il faut exécuter la commande suivante (côté client) :

```
$ ssh-keygen
```

Il faut accepter le fichier proposé par la suite. Ce dernier est le fichier dans lequel la clef sera stockée. Pour finir, une *passphrase* est demandée. C'est un mot de passe. Il faut donc qu'elle reste secrète et qu'elle soit suffisamment longue et complexe. Maintenant que le client est configuré (oui, ce n'est pas très long), il faut simplement en autoriser l'accès sur le serveur. La liste des clefs publiques autorisées est stockée dans le fichier **~/.ssh/authorized_keys** de l'utilisateur. Ainsi, un même utilisateur peut se connecter avec différentes clefs. La clef publique du client est stockée dans le fichier **~/.ssh/id_rsa.pub** ou **~/.ssh/id_dsa.pub**. Il faut alors ajouter le contenu de ce fichier dans le fichier d'autorisation du serveur. On peut le faire grâce à une clef USB ou en le recopiant à la main, mais également par SSH. Dans ce cas, il faut encore pouvoir se connecter avec l'authentification par mot de passe. On entre alors simplement la commande suivante depuis le client :

```
$ ssh login@serveur "echo `cat ~/.ssh/id_rsa.pub` >> .ssh/authorized_keys"
```

La configuration de SSH est alors terminée. Si une modification a été apportée au fichier de configuration, il est alors indispensable de redémarrer le serveur SSH.

```
# /etc/init.d/ssh restart
```

2.2 Configuration de Bazaar

2.2.1 Installation de Bazaar

L'installation de Bazaar se résume à l'installation d'un seul paquet. En effet, **bzr** est une commande comportant un nombre d'arguments plutôt impressionnant du fait de son architecture. La commande **bzr** n'est en fait pas qu'un simple programme. Elle est accompagnée d'un très grand nombre de greffons. En réalité, chaque commande commençant par **bzr** fait appel à un greffon. Pour installer Bazaar, on utilise donc la commande suivante :

```
# aptitude install bzr
```

Les utilisateurs de la distribution **Ubuntu** aimeront peut-être pouvoir profiter des évolutions de la version stable de Bazaar grâce au PPA des développeurs du gestionnaire de versions. Pour ajouter le dépôt non officiel, il faut éditer le fichier **/etc/apt/sources.list** avec les droits d'administrateur et ajouter les lignes qui suivent :

```
deb http://ppa.launchpad.net/bzr/ppa/ubuntu jaunty main
deb-src http://ppa.launchpad.net/bzr/ppa/ubuntu jaunty main
```

Une fois ajouté, on récupère la clef du dépôt qui sert à signer les paquets, puis on met à jour le système.

```
# apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 8C6C1EFD
# aptitude update
# aptitude full-upgrade
```

Attention

La version actuelle de Bazaar est la 2.0.0 (15 octobre 2009).

2.2.2 Le Bazaar smart server

Tout comme Subversion, par exemple, Bazaar possède un serveur qui lui est propre. C'est ce « serveur intelligent » qui permet d'utiliser le gestionnaire de versions en mode *bzr+ssh://*. Lancer un serveur Bazaar est plutôt simple. Une seule ligne de commande suffit.

```
# bzr serve --directory=/chemin/vers/la/racine/des/projets/
```

Malheureusement, une utilisation à la ligne de commande peut s'avérer assez ennuyeuse. On va donc par la suite se simplifier la vie. Avant tout, il faut préparer le terrain pour que tout soit prêt dès le lancement du smart serveur. Comme on peut le voir dans la commande ci-dessus, il faut créer la racine qui va contenir par la suite tous les projets. Ainsi, ces derniers seront tous accessibles via Bazaar. Ici, la racine sera **/var/bzr/**.

```
# mkdir /var/bzr/
```

Ensuite, on pourra lancer le serveur. Cependant, il faut être certain qu'il soit accessible via l'extérieur. La configuration du firewall est donc une étape qui (je l'espère) vous sera indispensable. Pour ceci, il faut savoir que le smart serveur utilise le port **4155** par défaut, mais qu'il est possible de le changer grâce à l'option **--port=LE_PORT**. Dans un script de configuration de **iptables**, on pourra écrire quelque chose comme ceci.

```
# Bazaar server
iptables -t filter -A OUTPUT -p tcp --dport 4155 -j ACCEPT
iptables -t filter -A INPUT -p tcp --dport 4155 -j ACCEPT
echo - Autoriser serveur Bazaar : [OK]
```

Pour simplifier les choses, il peut être intéressant de pouvoir lancer, redémarrer, vérifier le statut et arrêter le serveur Bazaar comme tout autre service. En général, on fait ceci via une commande du type :

```
# /etc/init.d/service start|restart|status|stop
```

Dans notre cas, du fait que le serveur est un greffon rattaché à Bazaar et qu'il n'est pas installé comme un serveur Apache par exemple, il faut créer un script dans le dossier **/etc/init.d/**. Ce script nous permettra donc de gérer notre serveur un peu plus facilement :

```
### BEGIN INIT INFO
# Provides:          bazaar_server
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start daemon at boot time
# Description:       Enable service provided by daemon.
### END INIT INFO
```

LINUX PRATIQUE ÉVOLUE !

LINUX PRATIQUE N°56

GNOME 2.28, 2.30... 3.0 ! EN ROUTE POUR UNE RÉVOLUTION DU BUREAU

Sommaire

ENCORE DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX JUSQU'AU 26 DÉCEMBRE 2009

```
#!/bin/sh

# Path to root of repo tree
BZRROOT=/var/bzr/

# Listening port (default = 4155)
PORT=4155

# Logs file
LOG_FOLDER=/var/log/bzr
LOG_FILE=$LOG_FOLDER/smart_server.log

# Arguments to start the server
ARGS="serve --port=$PORT --directory=$BZRROOT"

bzr_smart_server_process() {
    pgrep -fl "bzr $ARGS"
}

bzr_smart_server_status() {
    process=`bzr_smart_server_process`
    listening=`netstat -nl | grep -e ":$PORT "`

    if [ -z "$process" ]; then
        echo "Bazaar smart server is *not* running."
    else
        echo "Bazaar smart server is running."

        if [ -z "$listening" ]; then
            echo "The server is *not* listening on port $PORT."
        else
            echo "The server is listening on port $PORT."
        fi
    fi
}

bzr_smart_server_start() {
    echo "Starting Bazaar smart server."

    # Make sure the log folder is created
    if [ ! -d $LOG_FOLDER ]; then
        mkdir -p $LOG_FOLDER
    fi
    echo "" > $LOG_FILE

    bzr $ARGS > $LOG_FILE 2>&1 &

    bzr_smart_server_status
}

bzr_smart_server_stop() {
    echo "Stopping Bazaar smart server."

    pkill -f "bzr $ARGS"

    bzr_smart_server_status
}

case "$1" in
    start)
        bzr_smart_server_start
    ;;
    stop)
     bzr_smart_server_stop
    ;;
    status)
     bzr_smart_server_status
    ;;
    restart)
     bzr_smart_server_stop
     bzr_smart_server_start
    ;;
    *)
        echo "Usage: $0 { start | stop | status | restart }"
        exit 1
    ;;
esac
```

L'en-tête du script va permettre de contrôler son exécution lors du démarrage de la machine. En effet, on va également ajouter ce script aux *runlevels*, afin de ne pas être obligé de le relancer à chaque redémarrage de la machine. Pour cela, il faut utiliser la commande ci-dessous :

```
# update-rc.d bazaar_server defaults
```

Les quatre variables principales du script (celles dont les noms sont en majuscules) sont éventuellement à modifier. Elles concernent la racine des projets à utiliser avec Bazaar, le port utilisé par le serveur et le fichier dans lequel seront enregistrés les logs. Maintenant, on rend le script exécutable et il est alors possible de lancer, redémarrer, arrêter le serveur et d'en voir le statut avec les lignes de commandes suivantes :

```
# chmod +x /etc/init.d/bazaar_server
# /etc/init.d/bazaar_server start
# /etc/init.d/bazaar_server restart
# /etc/init.d/bazaar_server stop
# /etc/init.d/bazaar_server status
```

2.2.3 Création du projet

Une fois que le serveur Bazaar est prêt, il suffit de créer un projet. Pour gérer les droits d'accès, on va utiliser un groupe et les utilisateurs ayant accès à SSH. On crée donc un groupe pour le projet.

```
# groupadd mon_projet
```

On peut désormais ajouter à ce groupe les utilisateurs qui auront un accès en écriture aux branches. Si l'utilisateur n'existe pas encore, on utilisera :

```
# adduser --ingroup mon_projet mon_utilisateur
```

Mais, s'il existe déjà, on fera :

```
adduser mon_utilisateur mon_projet
```

Il faut bien faire attention à inverser le nom du groupe et le nom de l'utilisateur d'une commande à l'autre. On peut alors créer le dossier dans lequel toutes les branches seront présentes pour ensuite lui modifier son propriétaire (qui sera donc le groupe précédemment créé).

```
# mkdir /var/bzr/mon_projet
# chown :mon_projet /var/bzr/mon_projet
```

Toutes les branches appartiendront au même groupe et donc tous les membres du groupe devront avoir un accès en lecture et en écriture, alors que toutes les autres personnes ne devront que pouvoir lire le contenu des branches sans pouvoir le modifier. C'est dans ce but que l'on va préciser les permissions.

```
# chmod ug+rwx,g+s,o+rx,o-w /var/bzr/mon_projet
```

Après avoir déposé la branche **mainline** qui est la branche principale du développement (l'équivalent de **trunk** avec SVN), tous les développeurs du projet pourront la modifier. Il peut être intéressant pour des raisons de contrôle strict du code validé de n'avoir qu'un seul *gatekeeper* et donc qu'une seule personne ayant accès en écriture à **mainline**. Pour cela, on peut simplement réajuster le propriétaire de cette branche (il est possible de le faire pour toutes les branches).

```
# chown mon_utilisateur:mon_utilisateur /var/bzr/mon_projet/mainline
```

Arrivé ici, il doit être possible pour tout le monde de créer une branche locale ou simplement de télécharger le code avec les commandes ci-dessous :

```
$ bzr branch bzr://domaine.tld/mon_projet/branche
$ bzr checkout bzr://domaine.tld/mon_projet/branche
```

En revanche, les développeurs du projet devraient être capables d'envoyer leurs branches et également de *pusher* leurs modifications sur les différentes branches avec les commandes qui suivent :

```
$ bzr push bzr+ssh://domaine.tld/var/bzr/mon_projet/branche
```

Attention

Il est très important d'indiquer le chemin complet, car c'est comme cela que SSH fonctionne (cela ne dépend pas de Bazaar).

Pour terminer sur la création du projet, il faut savoir que Bazaar fonctionne avec et sans *working-tree*. Une branche qui ne possède pas de working-tree ne permet pas à l'utilisateur de voir réellement les fichiers. C'est-à-dire que cette branche ne contient que les informations permettant de connaître les différences avec la branche mère. Il est toutefois possible de construire un working-tree dans une branche qui n'en a pas. Pour cela, on utilise les commandes suivantes :

```
$ cd /chemin/vers/branche/
$ bzr checkout .
```

La branche nommée **mainline** étant la branche mère des toutes les autres (soit par héritage, soit directement), il est impératif que celle-ci possède un working-tree. Cependant, un problème fait son apparition. En effet, l'action de **push** ne met pas à jour le working-tree. On peut le faire à la main en se connectant en SSH sur la machine, puis en exécutant la commande de mise à jour dans la branche.

```
$ bzr update
```

Ou alors, on peut utiliser, pour plus de simplicité, le greffon **push-and-update**. En réalité, ce dernier ne fait qu'exécuter à la suite les commandes de **push** et d'**update**, à savoir :

```
$ bzr push bzr+ssh://domaine.tld/var/bzr/mon_projet/branche
$ ssh domaine.tld bzr update /var/bzr/mon_projet/branche
```

2.3 Configuration de Trac

Par défaut, Trac ne gère pas le gestionnaire de versions Bazaar. Cependant, il existe un greffon qui permet de l'utiliser avec. De plus, dans notre cas, nous allons voir comment utiliser Trac au travers d'Apache pour ne pas avoir à lancer le démon **tracd**. Cela va nous permettre d'accéder à Trac comme on peut le faire avec une simple page web. Pour commencer, on installe donc Trac, Apache (si ce n'est pas déjà fait), et le greffon qui va bien :

```
# aptitude install trac trac-bzr trac-spamfilter apache2 libapache2-mod-python
```

Le paquet **trac-bzr** correspond au greffon Bazaar, **trac-spamfilter**, lui, permet d'éviter les spams sur les rapports de bug et **libapache2-mod-python** va nous être utile pour accéder à Trac via Apache. À présent, il faut créer un dossier qui va contenir toutes les instances de Trac pour que l'on puisse créer une instance par projet.

```
# mkdir /var/trac
```

On peut alors initialiser notre première instance pour notre projet Bazaar.

```
# trac-admin /var/trac/mon_projet initenv
```

Une suite de questions est posée. Il faut y répondre correctement pour indiquer le chemin du projet, le type de gestionnaire de versions utilisé, etc. Voici, un exemple (coupé) de ce qui doit s'afficher dans le terminal.

```
# Le nom du projet
Project Name [My Project]> Mon Projet

# Valider sans rien répondre
Database connection string [sqlite:db/trac.db]>

# Entrer "bzr" pour Bazaar
Repository type [svn]> bzr

# Chemin vers la racine des branches du dépôt
Path to repository [/path/to/repos]> /var/bzr/mon_projet
```

Après ceci, notre instance de Trac n'est ni fonctionnelle, ni accessible. En effet, il faut d'abord activer les greffons installés afin que Trac puisse prendre en compte notre projet Bazaar. Pour cela, on édite le fichier **/var/trac/mon_projet/conf/trac.ini** et on y ajoute, à la fin, les lignes suivantes :

```
[components]
tracbzr.* = enabled
tracspamfilter.* = enabled
```

Pour prendre en compte cette nouvelle configuration, il faut mettre à jour notre instance de Trac.

```
# trac-admin /var/trac/mon_projet upgrade
```

À ce stade, l'instance est fonctionnelle, mais pas encore accessible avec un navigateur web. De plus, aucun compte n'a été créé et donc Trac ne peut savoir si le visiteur est un administrateur, un développeur ou une personne anonyme.

Pour y remédier, on va créer un fichier qui va contenir toutes les personnes autorisées à se connecter sur l'instance de Trac.

```
# htpasswd -c /var/trac/mon_projet/trac.htpasswd mon_utilisateur
```

L'option **-c** permet de créer le fichier. Après avoir validé, le mot de passe de l'utilisateur sera demandé. Il suffit de l'entrer et l'utilisateur sera ajouté. Pour en ajouter un autre par la suite, il faut bien veiller à ne pas utiliser l'option **-c**. Trac a sa propre gestion des groupes et des utilisateurs. La liste des utilisateurs de Trac est enregistrée dans le fichier **/var/trac/mon_projet/trac.htpasswd** que nous avons créé précédemment. Pour davantage de sécurité, il est conseillé de retirer certaines permissions aux personnes non enregistrées et non connectées. Les anonymes ne doivent pas pouvoir modifier le wiki, ni modifier les tickets déjà créés.

```
# trac-admin /var/trac/mon_projet permission remove anonymous WIKI_
CREATE WIKI_MODIFY TICKET_MODIFY
```

En revanche, les développeurs du projet doivent être capables de faire de telles choses. On crée donc un groupe **developer** avec les droits d'édition et on y ajoute les bonnes personnes.

```
# trac-admin /var/trac/mon_projet permission add developer WIKI_CREATE
WIKI_MODIFY TICKET_MODIFY
# trac-admin /var/trac/mon_projet permission add mon_utilisateur developer
```

Enfin, le mainteneur de projet se verra obtenir tous les droits sur Trac, y compris les droits qui lui permettront d'administrer le système pour modifier les types de rapport de bug, les feuilles de route et bien d'autres choses.

```
# trac-admin /var/trac/mon_projet permission add mon_utilisateur TRAC_ADMIN
```

Une fois que l'instance de Trac sera mise en ligne, les personnes enregistrées auparavant pourront se connecter et celles désignées en tant qu'administrateurs auront accès au bouton ***ADMIN*** pour configurer Trac plus facilement. Maintenant, la tâche consiste à utiliser Apache pour pouvoir accéder à Trac depuis n'importe où. Pour cela, on va utiliser le **mod python**. On commence donc par s'assurer qu'il est actif.

```
# a2enmod python
```

Ensuite, on crée un nouveau site qui va pointer vers notre instance de Trac et qui va utiliser le bon fichier pour pouvoir se connecter avec un utilisateur enregistré. On crée et édite alors un fichier nommé **/etc/apache2/sites-available/mon_projet** et on y met le code suivant :

```
<Location /mon_projet>
    SetHandler mod_python
    SetHandler mod_python
    PythonHandler trac.web.modpython_frontend
    PythonInterpreter main
    PythonOption TracEnv /var/trac/mon_projet
    PythonOption TracUriRoot /mon_projet
    SetEnv PYTHON_EGG_CACHE /tmp
</Location>
<Location /mon_projet/login>
    AuthType Basic
    AuthName "mon_projet login"
    AuthUserFile /var/trac/mon_projet/trac.htpasswd
    Require valid-user
</Location>
```

Il faut également penser à donner l'accès à Apache pour qu'il puisse accéder au contenu du projet Trac.

```
# chown -R www-data:www-data /var/trac/mon_projet/
```

Pour finir, on active le nouveau site, puis on recharge la configuration d'Apache pour le prendre en compte.

```
# a2ensite mon_projet
# /etc/init.d/apache2 reload
```

Suite à ces deux dernières commandes, l'instance de Trac doit être accessible à l'adresse **http://domaine.tld/mon_projet/**. Il ne reste plus qu'à se connecter et à configurer le Trac fraîchement installé de manière plus précise et complète.

CONCLUSION

Comme nous avons pu le constater, la configuration d'un trio Bazaar/SSH/Trac n'est pas forcément des plus simples, surtout pour des débutants. Cependant, cela vaut la peine de le faire dans le but de mener son projet correctement le plus longtemps possible. Bazaar offre une très bonne alternative à tous les autres gestionnaires de versions grâce à sa polyvalence. Le coupler avec SSH lui apporte un aspect sécurité, alors que Trac lui apportera la convivialité et la facilité de gestion du projet. De plus, sa mise en valeur, notamment grâce à la documentation, l'aspect du développement, ainsi que la connaissance des objectifs, vous permettront d'attirer le public et donc des contributeurs potentiels. Il existe bien entendu de nombreuses autres combinaisons. Certains préféreront SVN, Git ou encore Mercurial. L'essentiel reste tout de même de faire quelque chose de cohérent et fonctionnel. Les outils importent moins, mais, pour ma part, Bazaar, SSH et Trac semblent être une combinaison de force. ■

Sources

- http://trac.edgewall.org/wiki/TracOnDebian
- http://trac.edgewall.org/wiki/SpamFilter
- http://doc.bazaar-vcs.org/plugins/en/push_and_update-plugin.html
- http://blog.michaelgreenly.com/2008/01/setting-up-shared-bazaar-repository.html

Merci à Andrew Cowie (mainteneur de java-gnome) pour ses nombreux conseils.

Auteur : Guillaume Mazoyer

Utilisateur GNU/Linux (Ubuntu et Ubuntu Server) depuis 2006. Membre du Planet Ubuntu Francophone. Contributeur du projet java-gnome.