# Munin : superviser simplement la machine à café… enfin une réalité !

**ALEXANDRE SIMON**
Équipe Réseau Lothaire / Université de Lorraine / Direction du Numérique / Infrastructures
Plate-forme du Montet - Rue du Doyen Roubault
54500 VANDŒUVRE-LÈS-NANCY

**LUC DIDRY**
Équipe Réseau Lothaire / Université de Lorraine / Direction du Numérique / Infrastructures
Plate-forme du Montet - Rue du Doyen Roubault
54500 VANDŒUVRE-LÈS-NANCY

## Résumé

*Il n'existe pas de solution idéale qui traiterait parfaitement toutes les problématiques de supervision et de métrologie. Ce constat n'est pas un échec : il faut raisonner selon la stratégie « à chaque besoin sa solution ». L'administrateur ne doit pas hésiter à se doter des outils les mieux adaptés à ses problématiques, quitte à en multiplier le nombre. Munin fait naturellement partie de l'arsenal disponible des solutions de surveillance.*

*Munin est construit autour d'une architecture modulaire permettant de déporter sur des hôtes différents la collecte et le stockage des résultats. Il utilise un protocole de communication simple permettant un déploiement rapide et des facilités de diagnostic. Des sondes de collecte (scripts, appelés plugins, écrits dans n'importe quel langage) permettent l'extension de ses capacités.*

*Le rapport coût/investissement très faible de Munin, la multiplicité de ses plugins et la simplicité de développement d'une brique maison en font le compagnon idéal de l'administrateur et le complément parfait à d'autres solutions de supervision/métrologie. Munin se positionne au plus proche des services avec un niveau de détail poussé autorisant un diagnostic fin des problèmes. On l'utilisera également pour des campagnes de surveillance ponctuelles.*

*Nous nous concentrerons sur les particularités des scripts, la mise en œuvre de l'outil, ses capacités d'adaptation et sa complémentarité vis-à-vis des autres outils disponibles. Des anecdotes réelles et des exemples concrets viendront "démystifier" l'écriture de scripts spécifiques et illustreront comment aller là où les autres solutions ne vont pas.*

## Mots-clefs

*Munin, Monitoring, Surveillance, Métrologie*

## 1 Contexte

Tout administrateur systèmes et réseaux se doit de surveiller ses infrastructures, mais les domaines d'application ne s'appréhendent pas tous de la même manière. Par exemple, on ne portera pas la même attention au serveur Apache de développement qu'au serveur LDAP central de l'infrastructure. Il existe également des situations où l'on se contentera d'un simple voyant d'état du service vert/jaune/rouge alors que, pour un service équivalent mais avec un niveau de criticité différent, on souhaitera une visualisation plus détaillée sur un temps variable.

Les différentes solutions de supervision disponibles peuvent couvrir presque tous les besoins évoqués ci-dessus, mais au prix d'une complexité accrue d'installation, de déploiement et de maintenance. Plutôt que de se reposer sur une solution qui offrira une vue "tout de loin, rien de près", il vaut mieux adopter l'adage "à chaque problème sa solution". En effet, les inconvénients de maintenance de plusieurs logiciels de supervision/métrologie complémentaires sont bien moindres que les avantages de cette approche : souplesse, logiciels plus légers et plus simples à configurer, débogage plus aisé… On se rapproche de la philosophie Unix : « des programmes qui font une chose et qui la font bien ».

# 2 Présentation de Munin

## 2.1 Généralités

Munin est un outil qui existe depuis plus de 10 ans. Initialement connu comme LRRD (*Linpro RRD*), il a été renommé Munin à la sortie de la version 1.0.0pre1 en février 2004.

Le nom Munin fait référence à la mythologie nordique dans laquelle Munin et Hugin sont les deux corbeaux messagers du dieu Odin. Ces oiseaux ont pour mission de collecter les informations du royaume et de les rapporter à leur maître. Hugin signifie pensée, ou esprit et Munin mémoire. Les prémices de la supervision et de la métrologie modernes en quelque sorte !

Munin est un outil typé métrologie car sa fonction première est de collecter et de grapher des métriques (charge CPU, trafic réseau, occupation mémoire…). Si les graphes sont la fonction première de Munin, il possède toutefois des capacités d'alerte. Ces alertes peuvent être envoyées par n'importe quel canal de communication (*mail*, SMS, XMPP, IRC, signaux lumineux…) à travers l'utilisation des outils disponibles en ligne de commande sur le serveur.

Munin est écrit dans le meilleur langage qui soit, à savoir Perl[1], et ne nécessite que peu de dépendances : uniquement quelques modules Perl et GNU `make` pour le construire.

De par son âge et son utilité depuis longtemps reconnue, on retrouvera Munin dans toute "bonne" distribution Linux , Hurd (comme Debian GNU/Hurd ☺) ou BSD.

Le déploiement est extrêmement rapide et permet de bénéficier rapidement d'un Munin fonctionnel et adapté au système sur lequel il est installé. En effet, à l'installation de Munin, la commande `munin-node-configure` permet — entre autres — d'activer les *plugins* (terme désignant les sondes dans Munin) pertinents pour le système hôte grâce à la fonction d'auto-configuration de certains *plugins*. Cette fonction détecte la présence de composants spécifiques sur le système et propose d'activer les *plugins* correspondants. Par exemple, un *plugin* surveillant Apache ne s'activera que si celui-ci est installé.

## 2.2 Interface


*Figure 1: Interface de Munin : page d'accueil (à g.) et la vue détaillée d'un serveur (à d.)*

Le mode de visualisation de Munin est basique, mais efficace :

- sur la page d'accueil sont listés les serveurs surveillés (repère 1 sur la Figure 1) ;
- ils sont suivis des différentes catégories de *plugins* pour chaque serveur (repère 2) ;
- ils peuvent être rassemblés en groupes, si cette possibilité a été utilisée dans la configuration (repère 3) ;
- une liste des groupes permet d'énumérer uniquement les serveurs de chaque groupe (repère 4) ;

1. Comment ça, troll inside ?

- une liste des catégories des sondes amène à l'affichage des graphes des *plugins* de la catégorie choisie de tous les serveurs (repère 5). Un clic sur la catégorie "apache" présentera les graphes Apache des serveurs surveillés.

Par défaut, on obtient deux graphes pour chaque *plugin* : les valeurs sur les dernières 24 heures (repère 6) et sur les 7 derniers jours (repère 7). Un clic sur l'un ou l'autre de ces graphes amène une page présentant les graphes des dernières 24 heures, des 7 derniers jours, des 4 dernières semaines et des 12 derniers mois.

## 2.3 Modèle master/node

Munin se divise en deux fonctions : maître (*master*) et nœud (*node*). Dans un modèle client/serveur, le nœud est le serveur et le maître le client.

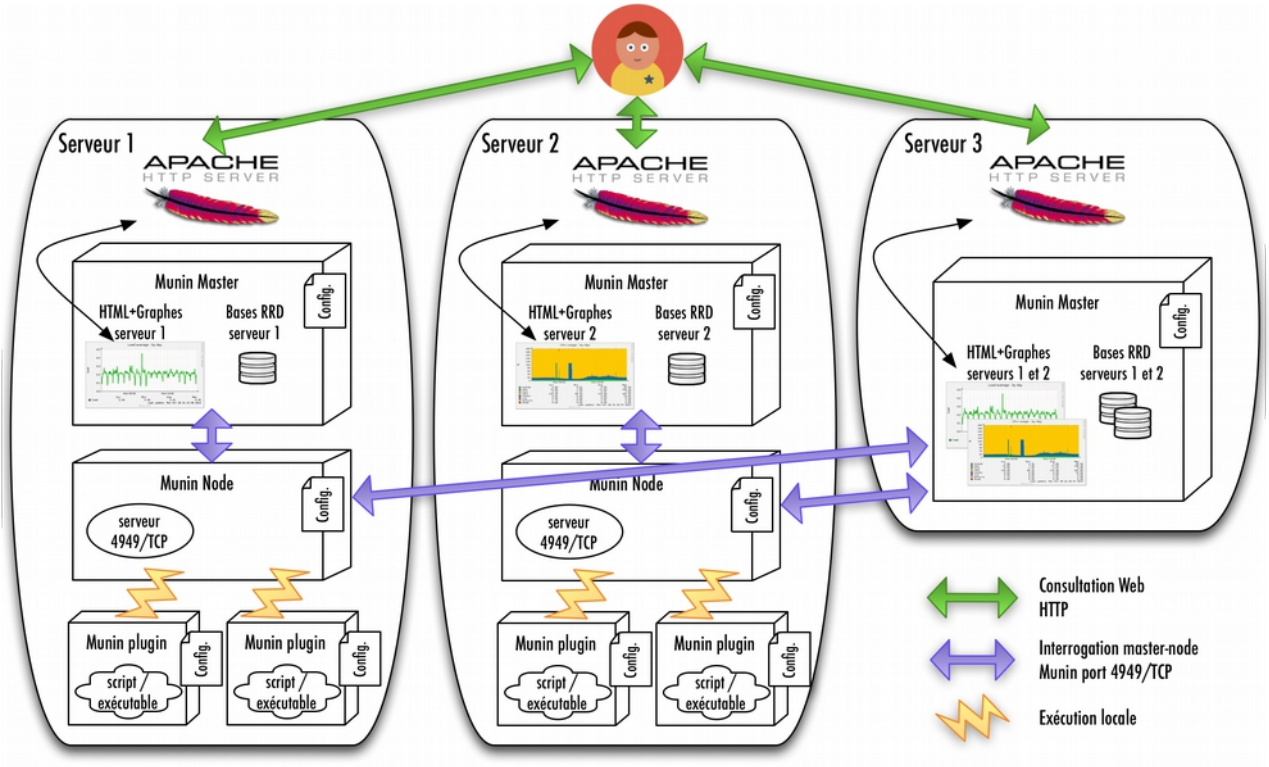


*Figure 2: Modèle master/node*

Le nœud, installé sur toutes les machines supervisées, écoute par défaut sur le port 4949/TCP, exécute les *plugins* à la demande du maître et lui renvoie les résultats. Le maître contacte les nœuds, récupère les données, les stocke dans des bases RRD et produit des graphes.

Il est à noter qu'un nœud peut être interrogé par plusieurs maîtres, qui pourront le contacter à des intervalles différents. De plus, il est possible de surcharger certaines options de configuration[2] d'un *plugin* depuis la configuration d'un maître, pouvant ainsi aboutir à des graphes différents pour un même *plugin* selon le maître (échelle, couleurs…). Il est même possible de créer un graphe agrégeant les valeurs de plusieurs *plugins* (cf. section 5.3).

On peut avoir, par exemple, un nœud et un maître sur chaque machine et un maître qui surveillera plusieurs nœuds. On aura alors des machines autonomes, se surveillant elles-mêmes et une machine centralisant les données. La Figure 3 illustre l'exemple d'une telle configuration.

Le protocole de communication entre le maître et le nœud est très simple et ne repose que sur quelques commandes. Ce protocole est en pur texte et est utilisable avec `telnet` (cf. section 4.2), permettant de facilement tester l'installation ou de diagnostiquer les anomalies.

## 2.4 Système de *plugins*

Les *plugins* sont les sondes utilisées par Munin. Ce sont des exécutables écrits dans n'importe quel langage.

2. Voir la liste des options modifiables sur http://munin-monitoring.org/wiki/protocol-config#Datasourceattributes

Les seules contraintes d'un *plugin* sont d'une part de renvoyer les métadonnées de génération de graphe lorsqu'il est appelé avec l'argument `config` (c.-à-d. comment les données du *plugin* doivent être graphées par le maître). D'autre part, un *plugin* doit renvoyer les données à collecter lorsqu'il est appelé sans argument (c.-à-d. la valeur mesurée à l'instant *t* de l'interrogation).

Il n'existe aucune limite quant à l'objet de surveillance d'un *plugin*, pourvu que les données renvoyées soient numériques. On pourra par exemple surveiller la qualité de réception d'un boîtier SMS, le nombre de fichiers dans un répertoire, le nombre d'occurrence d'un motif dans un fichier de *log*… Les possibilités sont infinies.

L'écriture d'un *plugin* Munin est très aisée. En effet, le format de sortie des deux instructions minimales est extrêmement simple. Exemple de métadonnées de génération de graphe :

```
graph_title Niveau de café
cafetiere.label % de remplissage de la cafetière
```

Si ces deux instructions suffisent à définir un graphe, les options disponibles pour cette définition sont bien plus nombreuses (échelle, couleur, taille…). On pourra les retrouver sur la page *Plugin reference*[3] de la documentation officielle de Munin.

Le format de sortie des données ne possède par contre qu'une seule instruction :

```
cafetiere.value 42
```

Le *plugin* indique ici que la cafetière est remplie à 42%.

# 3 Cas d'usage

## 3.1 Comment et où se positionne munin ?

Munin est un outil de métrologie avant tout. S'il ne remplace pas un Nagios (ou autre solution du même type), il en sera le compagnon idéal, fournissant un historique et un niveau de détail des plus appréciables.

Cet historique, sous la forme de graphes, servira tout aussi bien au débogage d'une machine (« Que s'est-il passé sur l'ensemble de la machine à l'heure où Apache a planté ? ») qu'à la prédiction de futurs problèmes (« Tiens, l'espace disque utilisé ne cesse d'augmenter de plus en plus rapidement. Je vais bientôt avoir des soucis. »).

L'utilité de Munin réside aussi dans un de ses plus grands avantages : sa simplicité. Car il est aisé de créer un *plugin* Munin pour… à peu près n'importe quoi ! S'il est besoin de grapher quelque chose, il est sans doute possible d'utiliser Munin pour cela.

## 3.2 Exemples

Comme cela est indiqué dans la section 4.3, Munin dispose en standard de nombreux *plugins* qui peuvent être complétés par des sources extérieures (dépôt Github…). Avec ce panel de *plugins*, il est possible de mettre sous supervision très rapidement :

- les fonctions vitales d'un serveur : charge CPU, entrées-sorties, interruptions, occupation mémoire, utilisation des interfaces réseau, disponibilité d'entropie, températures diverses…
- le fonctionnement du système d'exploitation : nombre de processus / activités, occupation du système de fichier (taille des partitions, nombre de fichiers, nombre *d'inodes*…), trafic réseau généré (nombre de *sockets*, état du suivi de connexion…), *uptime*, état de la synchronisation NTP…
- certains tests : connectivité IP (*ping*) vers certaines destinations, recherche d'un motif dans un fichier (`grep` dans un fichier de traces), présence d'un processus…

3. http://munin.readthedocs.org/en/latest/reference/plugin.html

- des services applicatifs à valeur ajoutée : état et utilisation des services BIND, DHCP, MySQL, Postgres, OpenLDAP, Postfix, Exim, Courrier, Freeradius, Apache…

La plupart des *plugins* cités ci-dessus sont activés sur les serveurs GNU/Linux opérés par l'Équipe Réseau Lothaire. Pour des besoins plus spécifiques et pour avoir une visibilité plus fine des services opérés, l'Équipe Réseau Lothaire a été amenée à développer ses propres *plugins* selon les techniques présentées dans les chapitres qui vont suivre. Dans la liste des *plugins* "faits maison" on notera plus particulièrement :

- ***iprocess*** : ce *plugin* utilisant les fonctions *multigraph* de Munin (cf. 5.2) permet de regrouper les processus composant un même service applicatif (par ex. pour proposer un ENT il faut à minima un serveur Apache, un annuaire LDAP, un serveur CAS…). Il surveille de manière agrégée (addition des valeurs de chaque processus) et individuellement (valeurs propres à chaque processus) les métriques suivantes : pourcentage d'occupation en CPU, nombre de descripteurs de fichiers ouverts, utilisation de la mémoire, nombre de processus et d'activités. Avec ce *plugin*, l'administrateur a une vision globale de l'utilisation des ressources du service opéré (« Comment l'ENT charge le serveur ? ») et il peut en cas de problème descendre jusqu'au processus pour en déterminer les causes (« L'ENT prend trop de mémoire, quel est le processus qui consomme trop ? »).

- ***iproute2_ss*** : ce *plugin*, directement dérivé de la commande `ss` du logiciel *iproute2*[4], permet de grapher d'une manière globale et agrégée l'utilisation et l'état de tous les types de *sockets* (*dccp*, *raw*, *tcp*, *udp*, *unix*) en IPv4 et IPv6. Le fonctionnement *multigraph* du *plugin* permet à l'administrateur de passer très facilement de la vision globale à une vision très fine de chaque type de *socket* pour chaque protocole IP. Ce *plugin* est très intéressant pour le suivi des utilisations réseaux (ressources en *socket*) d'hôtes proposant des services client/serveur très sollicités.

- ***yacap_chronos*** : ce *plugin* a été mis au point dans le cadre du portail captif mutualisé (YaCaP, présenté aux JRES2007[5]) utilisé sur le réseau régional Lothaire. Il faut rappeler que ce portail s'appuie sur les différents systèmes d'informations de chaque établissement lorrain pour authentifier et autoriser les utilisateurs. Le code du portail a été modifié afin "d'entourer" les fonctions d'authentification et d'autorisation de points de mesure temporels (« Combien de temps prend l'authentification CAS ? Combien de temps prend la recherche d'autorisation dans LDAP ? »). Ces mesures de temps (les *chronos*) sont tracées et utilisées par le *plugin* qui mettra en forme les graphes pour chaque établissement partenaire et pour chaque type de SI (CAS, LDAP, AD, Radius…). Ce *plugin* est un indicateur fondamental dans la recherche de lenteurs signalées par les utilisateurs (« Est-ce le portail captif ou le LDAP qui est lent ? »). Ces mesures ont également permis aux établissements d'ajuster leurs configurations (indexation, nombre de serveurs…).

- ***traffic_ipt*** : ce *plugin* utilise le compteur d'utilisation des chaînes *IPTables* du système GNU/Linux. L'administrateur peut déclarer des *IPTables* (non filtrantes) servant à qualifier le trafic géré par le serveur en IPv4 et IPv6 : chaînes ACC-IN-HTTP, ACC-OUT-HTTP, ACC-IN-RSYNC, ACC-OUT-RSYNC, ACC-IN-FTP, ACC-OUT-FTP. Le *plugin* se charge de consulter les compteurs d'utilisation de chacune de ces chaines et permet de grapher la répartition en IPv4 et IPv6 de chacun des protocoles surveillés. Ce *plugin* est notamment utilisé sur le serveur de miroirs (distributions Linux, CPAN…) opéré par l'Équipe Réseau Lothaire et il permet de suivre de manière très fine les vecteurs de distribution des différents fichiers.

# 4 Les *plugins* au centre du monde

## 4.1 Cycle de vie

Munin est constitué de multiples composants, chacun ayant une tâche bien précise :

- ***munin-cron*** (sur le maître), lancé par le démon `crond` toutes les 5 minutes par défaut, exécute une simple tâche, lancer séquentiellement les programmes suivants :
  - ***munin-update***

    contacte tous les nœuds configurés (dans la configuration du maître) sur le port 4949/TCP et récupère leurs données ;

    enregistre ces données dans des bases RRD (une par métrique de chaque *plugin*).

4. http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2

5. http://2007.jres.org/planning/paperf5f5.html?pid=70

- ***munin-limits***

  lit les deux dernières valeurs de chaque métrique de chaque *plugin* et les compare aux seuils *warning* et *critical* de la configuration si ces seuils existent ;

  si ces seuils sont atteints et qu'une valeur passe par exemple de *ok* à *warning*, ou de *warning* à *critical*, ***munin-limits*** déclenche une alerte pour les contacts configurés ;

  cette alerte peut prendre n'importe quelle forme : *mail*, SMS, alerte Nagios[6]…

- ***munin-html***

  génère les métadonnées utilisées par le composant ***munin-cgi-html*** ;

  si la stratégie de créations des pages HTML (`html_strategy`) est égale à `cron`, ce composant génère les pages statiques.

- ***munin-graph***

  génère les métadonnées utilisées par le composant ***munin-cgi-graph*** ;

  si la stratégie de créations des graphes (`graph_strategy`) est égale à `cron`, ce composant génère les graphes à partir de toutes les bases RRD de Munin.

- ***munin-cgi-html*** et ***munin-cgi-graph*** (sur le maître), responsables de la génération "à la volée" des pages HTML et des graphes de l'interface Web de Munin, si les options de configuration idoines — `html_strategy` et `graph_strategy` — sont égales à `cgi` ;

- ***munin-asyncd*** et ***munin-async*** (sur le maître et le nœud… ou pas), permettent de surveiller les nœuds de façon asynchrone.

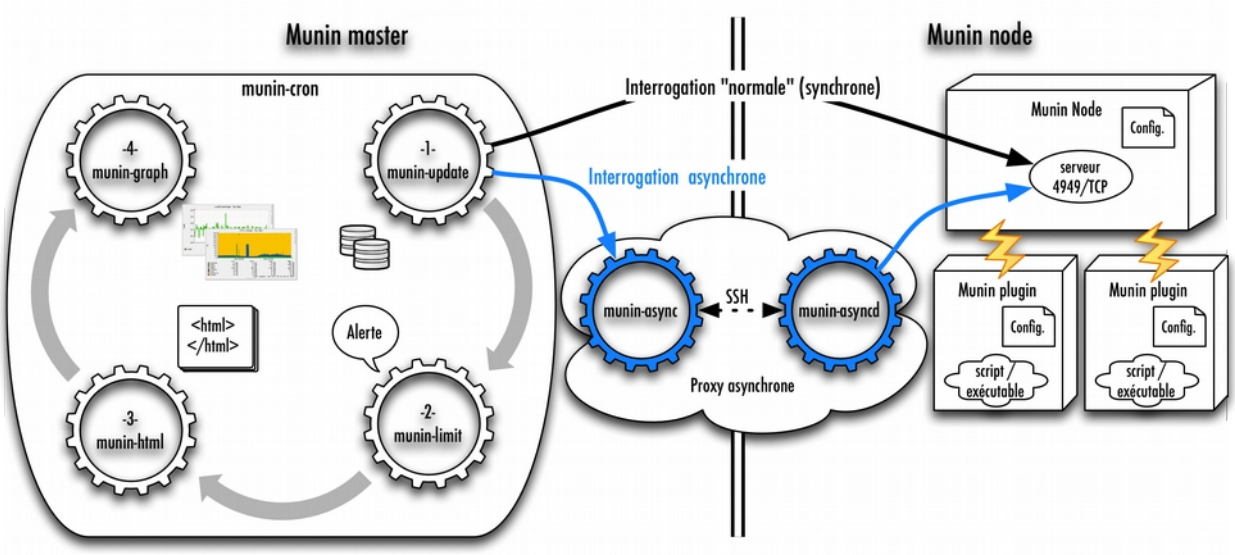


*Figure 3: Cycle de vie et composants Munin*

Dans les versions 1.x de Munin, les mises à jour des valeurs sont synchrones : la date de vérification et la valeur de chaque service sont celles récupérées par ***munin-update*** à chaque lancement. Le problème est que ***munin-update*** demande à récupérer chaque valeur de chaque *plugin* de chaque nœud à chaque cycle, et donc le calcul de chacune de ses valeurs sur le nœud, ce qui peut entraîner des lenteurs. Le temps d'exécution d'un cycle ***munin-update*** était donc directement lié au nombre de serveurs, de *plugins* à traiter et de leur temps d'exécution respectif.

Deux solutions ont vu le jour, la première est l'implémentation de la parallélisation des communications entre le maître et les différends nœuds (version 1.4). La seconde est la technique appelée *supersampling* où les *plugins* s'exécutent d'eux-mêmes à intervalles réguliers et renvoient des données déjà calculées au maître (version 2.0).

Bien que prometteuses, ces solutions amenaient d'autres problèmes : problèmes d'accès disques parallèles sur les bases RRD du maître pour la première, et adaptation lourde et difficile des *plugins* existants pour la seconde, compromettant par ailleurs une des forces de Munin, sa simplicité. Une nouvelle approche à préférer est d'intercaler un proxy asynchrone entre le maître et le nœud.

6. Avec Nagios configuré pour considérer Munin comme une source de résultats de contrôles passifs

***munin-asyncd***, installé de préférence sur chaque nœud (bien qu'il puisse contacter plusieurs nœuds) interrogera régulièrement le nœud mais ne rencontrera pas les mêmes problèmes d'accès disques qu'un maître car il stocke les données en format texte, et non en base RRD.

***munin-async*** se connectera au nœud en SSH pour récupérer les données stockées par ***munin-asyncd*** et les enregistrer dans son propre *spool* de données. Il les renverra enfin telles quelles au processus ***munin-update*** lorsque le maître le contactera.

## 4.2 Protocole de communication

La communication entre un maître et un nœud repose sur l'appel de certaines fonctionnalités des *plugins*. Voici quelques commandes possibles, parmi les plus importantes :

- ***list*** : cette directive ne s'applique pas aux *plugins* mais au nœud qui fournira la liste des *plugins* disponibles.
- ***config*** (la configuration d'un graphe est dans son *plugin*) : encore une fois, Munin joue la carte de la simplicité. Si certains *plugins* ont besoin d'options de configuration — que l'on devra fournir dans un fichier de configuration sur le nœud —, le maître n'a, lui, aucun effort à faire pour connaître le type de graphe nécessaire à chaque *plugin*. En effet, l'appel du *plugin* avec la directive ***config*** renverra au maître les directives nécessaires (cf. section 2.4). Les valeurs d'alerte *warning* et *critical* seront elles aussi communiquées au maître.
- ***fetch*** : récupère simplement les valeurs du *plugin* (cf. section 2.4).

On le voit, peut importe que le maître ait connaissance des *plugins* installés puisque tout lui sera fourni par les nœuds interrogés. Voici un exemple de test avec `telnet` :

```
foo@coffee : ~ % telnet localhost munin
Trying 127.0.0.1...
Connected to localhost .
Escape character is '^]'.
# munin node at coffee.addict.fr
> config cafe
< graph_title Niveau de cafe
< cafetiere.label % de remplissage de la cafetiere
.
> fetch cafe
> cafetiere.value 42
```

## 4.3 Création ou adaptation d'un *plugin*

Une des grandes forces de Munin est sa bibliothèque de *plugins* très complète. En dehors des *plugins* fournis officiellement, le dépôt *git* https://github.com/munin-monitoring/contrib/ propose un nombre important de *plugins* écrits par des tiers et reversés à la communauté. Il est rare de ne pas y trouver son bonheur et encore plus rare de ne pas trouver un *plugin* qui pourra servir de base pour l'écriture d'un *script* dédié à un problème particulier.

Munin fournit de base un module Perl[7] simplifiant l'écriture de *plugins* dans ce langage. Pour Python, il existe la bibliothèque `python-munin`[8]. Si elles ne sont pas indispensables, ces bibliothèques permettent d'écrire un *plugin* Munin de façon aisée et plus facilement lisible pour un tiers.

Les fichiers de configuration des *plugins* sont habituellement situés dans le répertoire `/etc/munin/plugin-conf.d/`[9]. Les différents fichiers du répertoire seront lus dans l'ordre alphabétique et la dernière directive s'appliquant au *plugin* prendra le pas sur celles rencontrées avant. Voici un exemple de configuration de *plugin* :

7. Page de manuel : http://manpages.ubuntu.com/manpages/raring/man3/Munin::Plugin.3pm.html

8. Page du projet : http://python-munin.readthedocs.org/en/latest/

9. http://munin-monitoring.org/wiki/plugin-conf.d

```
# fichier /etc/munin/plugin-conf.d/sql
[mysql]
user root
env.mysqlopts --defaults-extra-file =/etc/mysql/debian.cnf
```

- `[mysql]` initie une section dont le titre sera comparé aux noms des *plugins*. Ce titre peut être précédé ou suivi d'un `*` créant un système de *wildcarding* permettant l'application d'une même configuration à plusieurs *plugins*. Le `*` ne pourra pas se situer au milieu du titre, seulement au début ou à la fin de celui-ci (cf. section 5.1).
- `user root` est une directive de configuration Munin et non du *plugin* : ici le *plugin* sera lancé avec l'*uid* `root`.
- `env.*` est une directive d'environnement qui sera accessible dans le *plugin* :"véritable" configuration du *plugin*.

La commande `munin-run` permet d'exécuter un *plugin* dans les mêmes conditions que lors d'une interrogation par le maître. Une fois le *plugin* testé avec succès, il est toujours bon de le proposer sur le dépôt *Git* cité ci-dessus : la connaissance n'a de valeur que si elle est partagée ☺.

### 4.4 Activation

L'activation d'un *plugin* est, comme souvent avec Munin, fort simple. Il suffit que le *plugin* ou — de préférence — un lien vers celui-ci soit présent dans le dossier `/etc/munin/plugins/` et de relancer le service `munin-node`. Lors de sa prochaine interrogation, le maître prendra connaissance de ce nouveau *plugin*, récupérera sa configuration et créera la base RRD qui lui sera dédiée. Il est possible d'interroger "à la main" le nœud avec la commande `telnet` (voir l'exemple section 4.2). Cela permet de vérifier que le nouveau *plugin* a bien été pris en compte et que celui-ci s'exécute correctement.

## 5 *Plugin* avancé

### 5.1 Un script pour plusieurs *plugins*

Un *plugin* ne fournit habituellement qu'une métrique, mais il est bien évident qu'il ne sert à rien de déployer (et maintenir !) deux fois le même *plugin* pour surveiller par exemple deux interfaces d'une machine : le code reste le même, seul le paramètre de l'interface à superviser change.

Pour ce genre de cas, l'administrateur doit placer dans le dossier `/etc/munin/plugins/` des liens symboliques différents (par ex. `if_eth0` et `if_eth1`) pointant vers le même *plugin* (par ex. `if_`). En fonction de l'approche adoptée par le *plugin* deux déploiements sont possibles :

- si le *plugin* a été conçu pour agir différemment selon le nom par lequel il est appelé, les noms des liens symboliques seront dépendants de la nomenclature permise par le *plugin* et lui indiqueront le paramètre à utiliser pour s'exécuter contextuellement (dans notre exemple, le nom de l'interface à superviser) ;
- si le *plugin* a été conçu pour agir différemment uniquement par le biais de sa configuration, l'administrateur devra adapter celle-ci (configurations dans `/etc/munin/plugin-conf.d/`) afin de préciser le paramètre à utiliser. L'exemple suivant illustre ce cas avec des liens symboliques nommés `if_principale` et `if_backup`.

```
# fichier /etc/munin/plugin-conf.d/interfaces
[if_*]
user root
[if_principale]
env.interface eth0
[if_backup]
env.interface eth1
```

## 5.2 Les plugins multigraph

Certains *plugins* peuvent être très semblables, comme par exemple un *plugin* appelant la commande `sensors` pour fournir des données de température du serveur et un *plugin* appelant lui aussi `sensors` pour surveiller la vitesse de rotations des ventilateurs. Les deux *plugins* ne se différenciant que par la façon d'interpréter la sortie de `sensors`.

La maintenance de deux — ou plus — *plugins* est toujours plus fastidieuse que celle d'un *plugin* unique. Munin, à partir de la version 1.4, permet de mutualiser du code grâce à la capacité *multigraph*. Cette fonctionnalité permet de fournir plusieurs informations différentes à partir d'un *plugin*. Un unique appel à celui-ci fournira plusieurs réponses correspondantes aux différentes informations proposées par le *plugin*.

Le *plugin* pourra aussi fournir les métadonnées de génération d'un graphe "global", reprenant toutes les données fournies par le *plugin*, qui pointera vers une version éclatée de ce graphe. Cette vue éclatée dissocie les données graphées et donc apporte une vue plus précise de celles-ci. Leur échelle sera propre à chacune et non l'échelle la plus importante de toutes les données. Le passage de la vue globale à la vue éclatée est aussi simple que de cliquer sur le graphe de la vue globale.

## 5.3 Les graphes agrégés par le maître

On pourra, sur un serveur maître, agréger les données de plusieurs *plugins* et de plusieurs nœuds en un seul graphe.

On créera pour cela dans la configuration du maître[10] un serveur virtuel contenant la configuration d'un graphe (titre, taille…) ainsi que les noms des *plugins* et des nœuds dont les valeurs serviront de sources de données pour ce graphe. Les valeurs de ces *plugins* pourront, au choix, être ajoutées les unes aux autres ou bien séparées.

L'exemple typique de cette fonctionnalité sur le maître est l'agrégation en un seul graphe de métriques similaires d'un ensemble de nœuds. On peut par exemple faire apparaître sur un même graphe l'ensemble des taux de charge (*load*) d'une ferme de serveurs. On peut également concaténer sur un même graphe l'occupation d'un répertoire spécifique de certains nœuds (par ex. répertoire `/zimbra/var/spool/mail/` des serveurs stockant les boites aux lettres).

# 6 Conclusion

Munin, bien que d'une simplicité déconcertante dans tous ses aspects (installation, configuration, interface, extensibilité des capacités par les *plugins*) est un outil puissant qui fera la joie de l'administrateur systèmes et réseaux avisé qui l'aura installé. Son coût de mise en œuvre quasi nul et les bénéfices apportés sont autant de bonnes raisons de le classer dans la boîte à outils minimale à installer sur chaque nouveau serveur.

Les *plugins* n'étant limités que par l'imagination et/ou les talents de codeur de l'administrateur, il ne fait nul doute que tout nouvel utilisateur pourra lui trouver une utilité encore inexplorée aujourd'hui.

Cet article, complété de sa présentation, ne traitent pas exhaustivement de tous les aspects de Munin. Il conviendra d'approfondir les thématiques liées à un déploiement de masse sur de nombreux nœuds centralisés par un seul maître, architecture dans laquelle la fonction de *proxy* asynchrone sera sans nul doute obligatoire (cf. problèmes de performances). Il faudra également s'attarder sur la compréhension des fonctions avancées des *plugins* et à leur bonne utilisation lors de l'écriture de *scripts* maison.

L'Équipe Réseau Lothaire utilise Munin en complément de ses trois autres solutions de supervision/métrologie (Icinga, iSup, netMET/netMAT) depuis 4 ans. Cet outil à la fois discret et silencieux mais toujours présent a permis de répondre à des besoins spécifiques de supervision (cf. section 3.2) ou à des résolutions de problèmes complexes. L'investissement dans Munin et dans l'écriture de *plugins* spécifiques a toujours été valorisé et recapitalisé pour des projets similaires.

Nous ne pouvons qu'encourager les administrateurs systèmes et réseaux à adopter Munin !

« Prenez un café et libérez le corbeau messager qui est en vous ! »

10. Exemple de configuration sur https://munin.readthedocs.org/en/latest/example/graph/aggregate.html