N°188 décembre 2015

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

LES «HOW-TO» DU SYSADMIN

Configurez Nginx en tant que reverse proxy et serveur de cache p.26

JASPER / POCKETSPHINX

VOUS SOUHAITEZ ENFIN ÊTRE ENTENDU ET COMPRIS DE VOTRE PC ?

RECONNAISSANCE VOCALE ... AVEC PYTHON ET POCKETSPHINX p.52

OPENSTREETMAP / CARTES

Installez votre serveur de tuiles et utilisez les données d'OpenStreetMap p.42

PYTHON / R

Analysez vos données avec Python et le module Pandas p.72

SYSADMIN / LOGS

Auditez votre système en SQL avec OSQuery p.36

SPECTRE / WIFI

Comprenez les perturbations d'un réseau Wifi p.22

TESTS / JBOSS

Réalisez vos tests unitaires avec Arquillian p.64

ET AUSSI : PostgreSQL et Slony – Attribut tools et vues Android

France METRO : 7,90 € – CH : 13 CHF – BEL/PORT.CONT : 8,90 € – DOM TOM : 8,50 € – CAN : 14 $ cad – TUNISIE : 18 TND – MAR : 100 MAD

L 19275 - 188 - F: 7,90 € - RD

DOMAINES | MAIL | HÉBERGEMENT | E-COMMERCE | **SERVEURS**

TESTEZ LE MEILLEUR CLOUD

TESTÉ ET APPROUVÉ PAR CLOUD SPECTATOR

Powered by intel Cloud Technology

1&1 Serveur Cloud : Easy to use – ready to Cloud*

Les performances des nouveaux serveurs Cloud sont imbattables en termes de CPU, RAM et SSD.

Réalisez vos projets dans le Cloud avec la combinaison parfaite entre flexibilité et performances.

- ✓ **Load balancing**
- ✓ **Stockage SSD**
- ✓ **Facturation à la minute**
- ✓ **Intel® Xeon® Processor E5-2660 v2 et E5-2683 v3**

1 mois gratuit !

Puis à partir de 4,99 € HT/mois (5,99 € TTC)*

1 MOIS POUR ESSAYER

1 CLIC POUR CHANGER DE PACK

1 APPEL UN EXPERT VOUS RÉPOND

0970 808 911
(appel non surtaxé)

1&1

1and1.fr

*Facile à utiliser - prêt pour le Cloud. 1&1 Serveur Cloud : 1 mois d'essai gratuit, puis à partir de 4,99 € HT/mois (pour la configuration du serveur Cloud S). Facturation mensuelle en fonction de la configuration choisie. Pas de durée minimum d'engagement, ni de frais de mise en service. Conditions détaillées sur 1and1.fr. Intel et le logo Intel sont des marques commerciales d'Intel Corporation aux États-Unis et/ou dans d'autres pays. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

GNU/Linux Magazine France
est édité par Les Éditions Diamond

10, Place de la Cathédrale - 68000 Colmar - France
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
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Responsable service Infographie : Kathrin Scali
Exé PAO : Thomas Pichon
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 – v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, 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

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN :** 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,90 €

SUIVEZ-NOUS SUR :

https://www.facebook.com/editionsdiamond

@gnulinuxmag

LES ABONNEMENTS ET LES ANCIENS NUMÉROS SONT DISPONIBLES !

En version papier et PDF :
www.ed-diamond.com

Codes sources sur https://github.com/glmf

ÉDITORIAL

L'informatique est là pour nous simplifier le quotidien et nous pouvons justement nous en rendre compte tous les jours. Vous en voulez une preuve ? Prenons un exemple fort simple : votre lieu d'habitation est identifié à l'aide d'une adresse unique ; suivant où vous habitez cette adresse peut être plus ou moins longue, avec ou sans complément. Quelle joie donc lorsqu'une interface vous propose de compléter automatiquement votre adresse ! Il suffit d'indiquer le code postal, de cliquer sur le type de voie et... et parfois votre rue n'est pas répertoriée. Si vous habitez dans une grande ville, à cheval sur deux arrondissements, votre adresse postale ne sera pas forcément reconnue automatiquement. Pour le service que vous utilisez, vous serez domicilié dans l'arrondissement A et pour la poste dans l'arrondissement B. Et si, cerise sur le gâteau, on vous interdit d'ajouter un complément d'adresse... vous ne recevrez jamais votre courrier !

La reconnaissance automatique d'adresse n'est pas en cause : il s'agit d'une aide à la saisie et cela devrait être considéré comme tel. Le problème se situe au niveau des concepteurs de ces services qui n'arrivent pas à imaginer que l'utilisateur puisse avoir à corriger des données et qui lui interdisent donc de le faire. Sur une page Web, pour ceux qui le peuvent, il suffira parfois d'un peu de hacking pour forcer la saisie (pas toujours, car parfois lesdits développeurs ont quand même des notions de base en sécurité... ou ils utilisent des ***frameworks*** qui gèrent cela pour eux). Mais lorsque l'on doit remplir un document papier (oui, ça existe encore) et que la saisie est faite par une autre personne dans un service administratif, il n'y aura aucun moyen de tenter une correction... Donc la reconnaissance automatique c'est bien, mais tant que ce n'est pas fiable à 100% il faut laisser la possibilité à l'utilisateur de corriger les données !

Pour la reconnaissance vocale, le problème est identique : est-ce que vous allez supposer que la traduction textuelle réalisée par la machine est toujours correcte ? Est-ce que cela vous viendrait à l'idée de dicter un texte (ou une phrase, restons réalistes) sans le relire avant de vous en servir ? Non, bien sûr ! Eh bien nous vous proposons de tester tout cela ce mois-ci en mettant en place vos propres solutions de reconnaissance textuelle ou vocale. Bien que certains moteurs STT (***Speech To Text***) soient plus performants que d'autres, il faut garder à l'esprit qu'ils ne sont pas infaillibles et permettre là encore à l'utilisateur d'avoir le dernier mot...

En parlant de dernier mot, les miens seront pour ce mois-ci de vous souhaiter une excellente lecture !

Tristan Colombo

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°188

LA SUPERVISION CONFORTABLE AVEC OMEGA NOC

Guillaume SEIGNEURET [Architecte sécurité systèmes et réseaux chez Omega Cube]

Dotez votre infrastructure d'un véritable « Network Operating Center » avec Omega NOC, le logiciel de supervision et de métrologie adapté à tous.

Mots-clés : Supervision, Métrologie, Hypervision, Prédiction, Trending, Alerte

Résumé

Cet article va présenter les capacités de la nouvelle version de l'application Omega NOC et détailler son installation et sa configuration.

Nous sommes nombreux à avoir utilisé les classiques et néanmoins excellents **Nagios**, **Cacti**, **Zabbix**, **Observium**, **Centreon** etc., mais aucun d'entre eux ne nous a pleinement satisfaisait. Soit parce qu'ils ne proposent pas la métrologie intégrée, soit parce qu'ils sont lourds à maintenir, soit parce qu'ils sont pénibles à configurer ou encore parce que leur interface aux graphismes des années 90 n'est pas très présentable pour les clients.

De ce constat est né **Omega NOC**, un logiciel d'hypervision open source en AGPL v3 qui embarque la supervision, la métrologie et le *graphing*, et qui inclut également la prédiction et le *trending*. Le tout a été concocté avec un module de configuration intégré à l'interface web et une visualisation graphique de l'infrastructure afin de permettre un confort d'utilisation maximum (figure 1).

Omega NOC est basé sur **Shinken**, une réécriture de Nagios en Python, qui a le mérite d'avoir une architecture logicielle modulaire (comme le montre la figure 2), comparé à Nagios qui est totalement monolithique (tout est compilé dans un seul exécutable nagios).

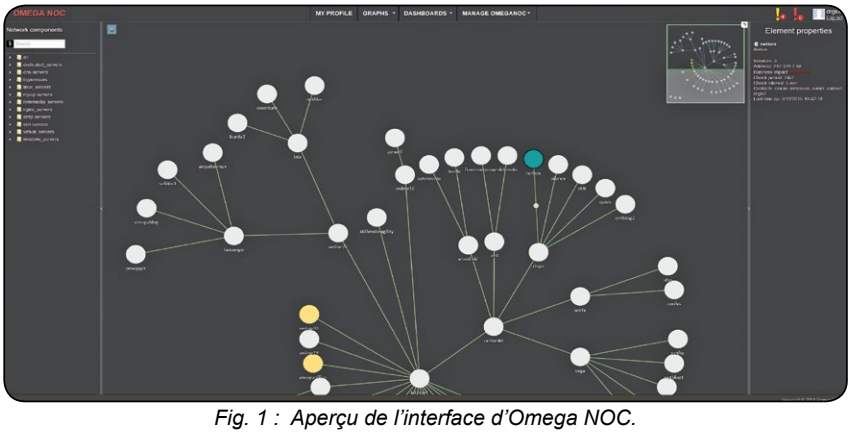


Fig. 1 : Aperçu de l'interface d'Omega NOC.

1 Que fait Omega NOC ?

Omega NOC a été développé dans le but de tout avoir dans une seule et même interface. Il permet de visualiser les nœuds et composants d'une infrastructure tout en mettant en valeur les problèmes.

Une section « vue logique » vous permet de voir votre infrastructure avec

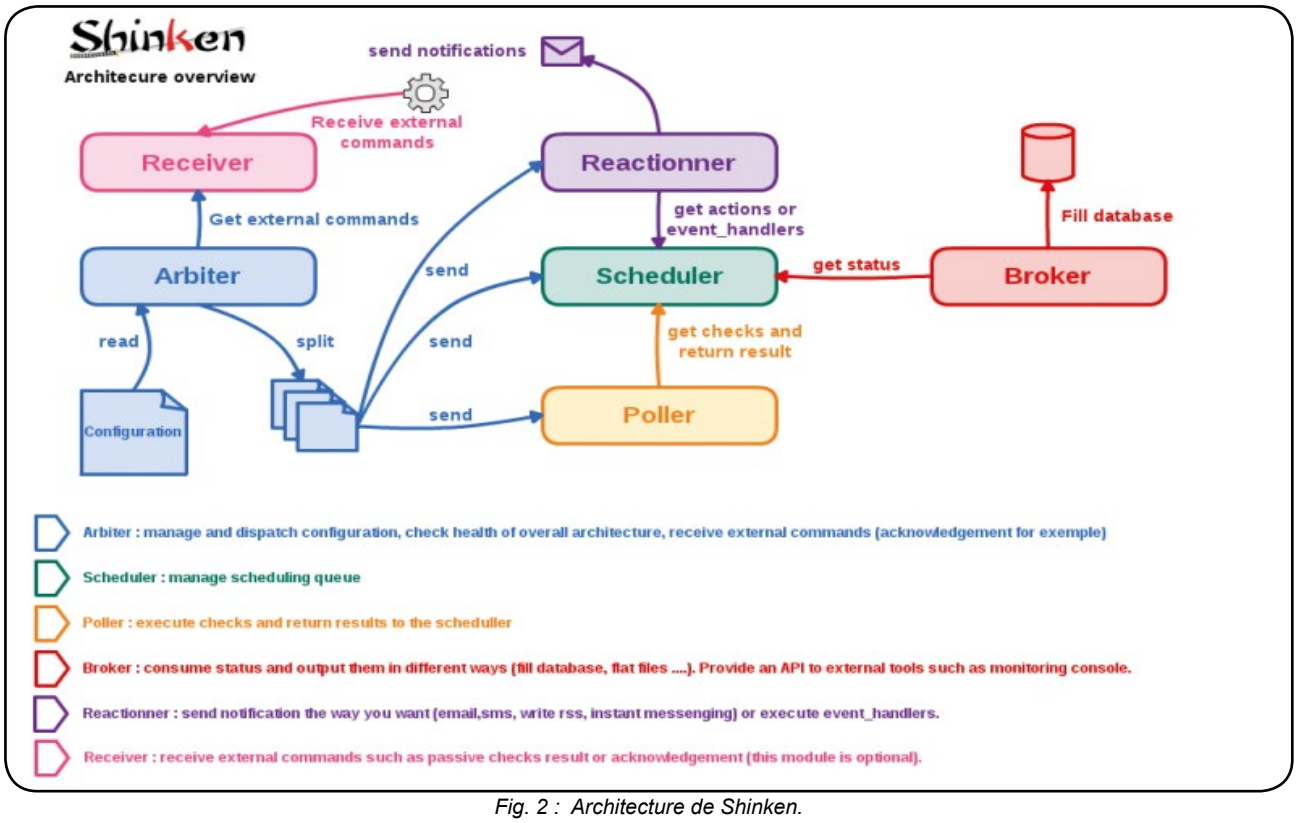


Fig. 2 : Architecture de Shinken.

une perspective logicielle/service. Il est donc possible de voir d'un coup d'œil quel logiciel dialogue avec quel autre logiciel et donc de visualiser en quelques secondes l'impact d'une panne sur l'ensemble de la chaîne applicative (figure 3).

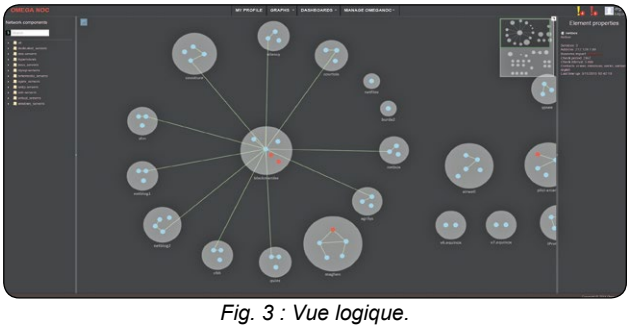


Fig. 3 : Vue logique.

Avoir la métrologie intégrée à la supervision n'a jamais été aussi simple. Dès lors qu'Omega NOC est installé et configuré avec les sondes adéquates, les données de métrologie sont automatiquement traitées et visualisables suivant vos critères d'affichage (figure 4, page suivante).

Les rapports de SLA (*Service Level Agreement* ou Niveaux de Service) sont simples et aisés comme le montre la figure 5 (voir page suivante).

Les tendances et le *forcasting* des données de métrologie sont automatiquement calculés (figure 6, en page suivante).

2 Installation

Comme dit précédemment, Omega NOC a été conçu sur une base de Shinken et il en résulte une compatibilité maximale avec les configurations Nagios et parfaite avec Shinken ! Youpi !

Grosso modo, vous pouvez considérer que 99% de votre configuration Nagios marchera avec Omega NOC.

Omega NOC est dépendant des programmes et librairies suivantes : **`shinken`** >= 2.0, **`sqlite3`**, **`graphviz`**, **`graphviz-devel`**, **`python`** >= 2.6, **`python-devel`**, et **`libxml2-devel`**.

Pour commencer, préparez votre système en installant les dépendances nécessaires au bon fonctionnement d'Omega NOC :

- Sur Debian/Ubuntu :

```
$ apt-get install python-pip python-pycurl sqlite3 graphviz
graphviz-dev pkg-config python-dev libxml2-dev
```

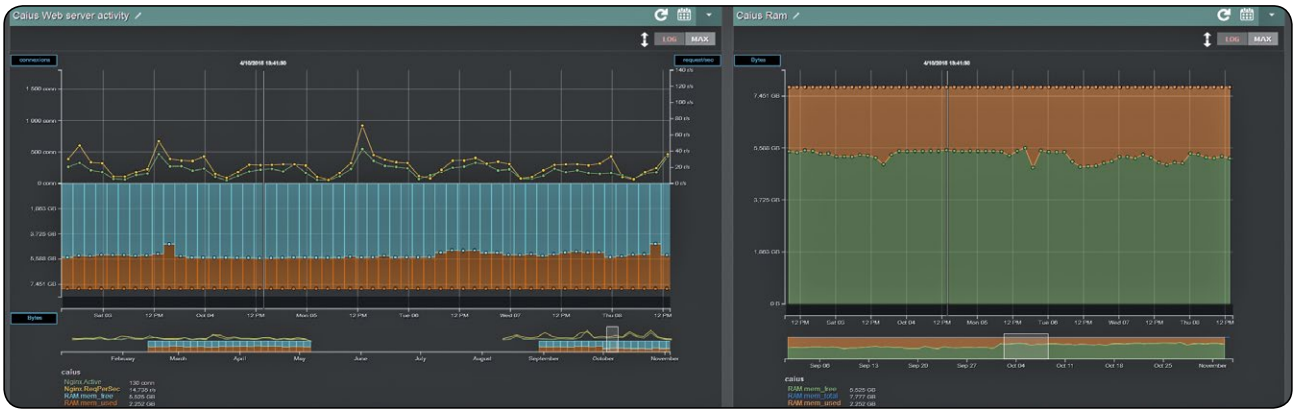


Fig. 4 : Données de métrologie.

- Sur CentOS :

```
$ install setup tools
$ curl https://bitbucket.org/pypa/setuptools/raw/bootstrap/
ez_setup.py | python -
$ install pip  curl https://raw.github.com/pypa/pip/master/
contrib/get-pip.py | python -
$ easy_install pip

$ yum install sqlite graphviz graphviz-devel gcc gcc-c++ python-
devel libxml2-devel
```

Il faut ensuite récupérer les sources du projet sur GitHub :

```
$ git clone https://github.com/Omega-Cube/omeganoc.git
```

La récupération des sources de Shinken va se faire facilement grâce à un git submodule :

```
$ cd omeganoc
$ git submodules init && git submodules update
```

Il s'agit maintenant d'installer Shinken avec les quelques commandes qui suivent :

```
$ useradd --user-group shinken
$ make shinken-install
$ shinken --init
```

Terminons l'installation d'Omega NOC avec la commande suivante :

```
$ make install
```

Note

Si Shinken a déjà été installé auparavant, il suffit de lancer la simple commande :

```
$ make shinken-install
```

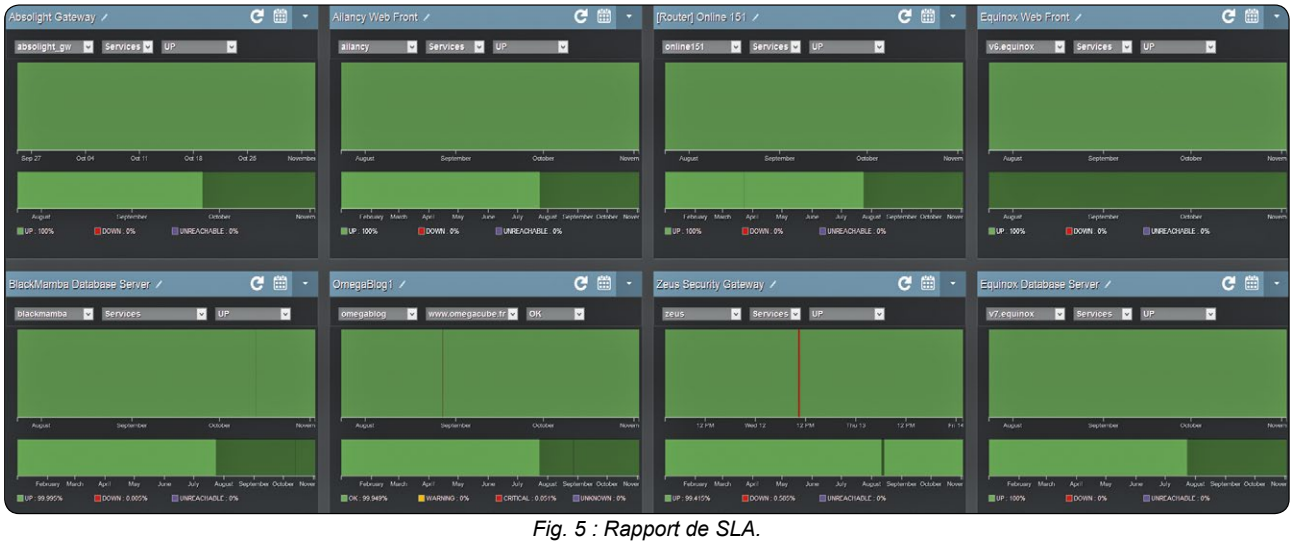


Fig. 5 : Rapport de SLA.

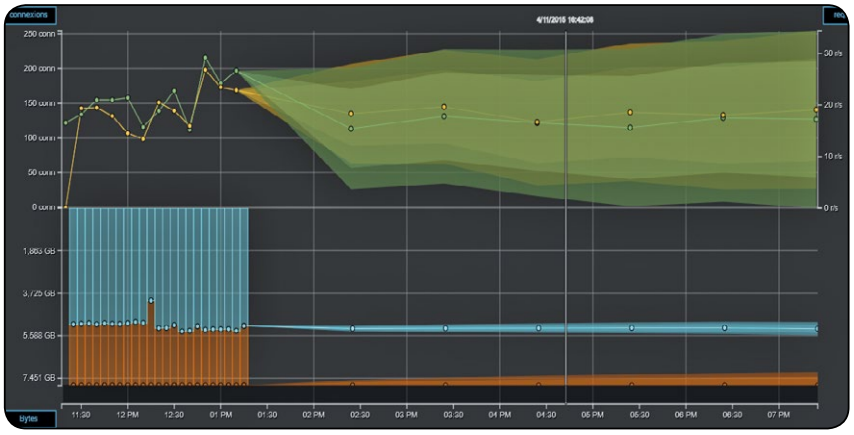


Fig. 6 : Tendances et forecasting.

Omega NOC est maintenant installé sur votre système et prêt à l'emploi. Vous souhaitez néanmoins sûrement régler quelques petits détails avant le lancement...

3 Configuration

Deux composants essentiels viennent se greffer à Shinken : l'interface graphique **Hokuto** et le moteur de prédiction **Nanto**.

L'interface graphique se présente sous la forme d'un *daemon* se connectant au *broker* Shinken. Il se nomme Hokuto, en référence à Hokuto Shinken, l'école d'art maritial dont provient Ken le survivant : *« Tu es déjà mort, mais tu ne le sais pas encore ! »*.

Cette architecture permet à l'interface graphique de fonctionner sur un autre serveur que le serveur de supervision. Elle est donc très pratique pour faciliter l'accès externe, sans compromettre la sécurité de l'ensemble de la chaîne de supervision, se trouvant en général dans une zone privilégiée de l'infrastructure.

Pour configurer Hokuto, éditez le fichier **`/etc/hokuto.cfg`** :

```
# Paramétrez le port et l'adresse IPv4 d'écoute
GUNICORN_BIND = '0.0.0.0:5555'

# Nombre de worker de l'application, par défaut 1 worker
# Changez cette valeur si vous avez un nombre important d'utilisateurs
#GUNICORN_WORKERS = 4

# Si vous souhaitez binder l'interface sur un nom de domaine en particulier,
# utilisez en plus de la ligne précédente :
#SERVER_NAME = omeganoc.myserver.com:5000

# Clé secrète servant à chiffrer les données entre vous et l'application.

# Changez-la pour plus de sécurité.
SECRET_KEY = 'blah blah blah'

# Adresse et port d'écoute du Broker Shinken, changez ces valeurs seulement
# si vous l'avez fait préalablement sur votre installation de Shinken
LIVESTATUS_HOST = '127.0.0.1'
    LIVESTATUS_PORT = 50000
```

Finalement, lancez l'interface par la commande **`service hokuto start`** ou si vous préférez **systemd** (nul n'est parfait) **`systemctl start hokuto.`** Démarrez également Shinken par la commande **`service shinken start`**.

Vous pouvez alors accéder à l'interface via votre navigateur préféré. Les identifiants par défaut sont **`admin/admin`**.

Nous vous invitons à changer ces informations dans le menu ***Manage Omega NOC > Manage Users***. Vous constaterez alors que les utilisateurs Omega NOC sont indépendants des utilisateurs Shinken, mais qu'ils peuvent y être associés (figure 7).

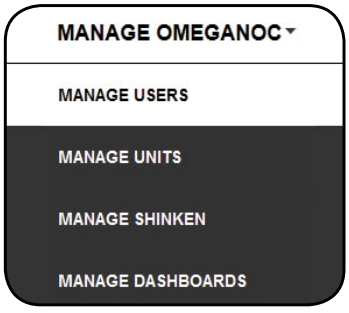


Fig. 7 : Gestion des utilisateurs.

Conclusion

Vous êtes maintenant prêts à configurer Shinken pour avoir vos *babasses monitorées*. Nous vous invitons à consulter sur le site http://shinken.io les articles « *online course* » pour aller plus loin dans la configuration.

Vous observerez également que dans les sources, il y a tout ce qu'il faut pour lancer une machine virtuelle avec **Vagrant**. Un **`vagrant up`** dans le répertoire racine vous lancera toute l'installation en mode automatique et vous fournira un environnement totalement opérationnel.

De nombreuses fonctionnalités sont en cours de développement, principalement axées sur la prédiction. N'hésitez pas à consulter régulièrement http://www.omeganoc.org pour plus de nouveautés ! ■

DE L'ÉLITISME À LA FUMISTERIE : NOS MENTALITÉS DOIVENT CHANGER

Tris ACATRINEI [Projet Arcadie]

Il est de plus en plus facile de pointer les approximations de nos dirigeants et c'est quelque chose que je fais d'ordinaire avec beaucoup de facilité et de plaisir. Mais à l'approche de la fin de l'année et de ses éternelles bonnes résolutions, il est peut-être temps de revenir sur nos propres comportements et d'apprendre de nos erreurs.

Mots-clés : Numérique, Prise de conscience, Recul, Éducation

Résumé

Depuis quelques années, des articles et des conférences se multiplient pour parler de l'inclusion de nouvelles personnes dans l'univers du logiciel libre et plus globalement de l'informatique. Si les difficultés relatives au sexisme et au racisme sont régulièrement soulignées, d'autres aspects sont pour le moment passés sous silence. C'est justement à ces aspects que je vais m'intéresser.

Pour une fois, je vais partiellement laisser de côté nos amis les politiques pour m'arrêter sur quelque chose qui m'a interpellé. Dans ma dernière chronique, je citais un développeur qui disait trouver scandaleux d'être considéré comme une simple unité de production par les organisateurs de *hackathons*. Sur le moment, son propos me semble juste et justifié. Mais cette phrase me travaillait, alors, je m'y suis attardée.

La question de base est la suivante : le technicien informatique – quel qu'il soit – est-il une unité de production ou une entité à part entière ? Je vais utiliser le terme de technicien pour désigner indifféremment le développeur, l'administrateur-système, le graphiste, l'ingénieur réseau, etc. Humainement parlant, on dirait que le technicien est une entité à part entière, avec des sentiments, des émotions, des idées, un raisonnement intellectuel. Alors pourquoi le monde de l'informatique continue-t-il à se comporter comme s'il n'était que la somme d'unités de production ? Vous ne voyez pas où je veux en venir ? Je vais vous éclairer.

1 Un snobisme paradoxal

Même si l'informatique a envahi nos vies, pour une majorité de personnes, l'informatique et ce qui va avec doit être aussi simple d'accès et d'usage qu'un banal micro-ondes : mettre la barquette dans le micro-ondes, appuyer sur un bouton, attendre 30 secondes et manger. Ce n'est pas pour autant de la paresse intellectuelle. Rares sont les personnes qui s'amusent à décortiquer les micro-ondes sur leur temps libre pour

en comprendre le fonctionnement. La plupart des objets qui entourent notre quotidien ont pour mission principale de nous le simplifier et de nous faire gagner du temps : la télévision, les journaux, le réfrigérateur, le four, la machine à laver, etc. Ce temps peut alors être utilisé à faire autre chose, cette autre chose étant laissée à la discrétion de chacun d'entre nous. Pourtant, cet axiome valable pour l'électroménager ne semble pas avoir fait son cheminement intellectuel en matière d'informatique. La conséquence est simple : les appareils et les systèmes les plus intuitifs et les plus simples trustent les ventes alors que les systèmes – certes plus respectueux des libertés fondamentales et du choix des utilisateurs – plus complexes n'occupent que de très maigres pourcentages de parts de marchés. On objectera que je me focalise sur les aspects commerciaux et marketing, mais au-delà des conceptions mercantiles, il y a des évidences qui s'imposent à nous. Prenons un exemple très simple : les mises à jour. Si vous êtes utilisateur de Windows, les mises à jour se font toutes seules, en tâche de fond. Si vous êtes utilisateur d'un système basé sur GNU/Linux, c'est à vous de vous imposer une routine de vérification. En théorie, c'est l'affaire de quelques minutes. En pratique, on oublie purement et simplement parce que le quotidien et les tâches à accomplir quotidiennement prennent le pas et qu'on a envie de faire autre chose que de vérifier les mises à jour sur nos ordinateurs, nos téléphones, nos tablettes et nos objets connectés.

Quand on discute de ces questions avec certaines personnes, très engagées dans le monde du logiciel libre, on est parfois confronté(e) à des réponses assez abruptes, laissant sous-entendre que si les personnes ne souhaitent pas en apprendre davantage sur le fonctionnement de leurs machines personnelles, elles sont idiotes et paresseuses.

Or, le salut du logiciel libre viendra des néophytes et des amateurs. Mais pour susciter leur intérêt, il faut être capable de transmettre. Cette transmission d'informations ne se fait pas sans compromis et sans mise à niveau. Ce n'est pas à l'apprenti de se mettre au niveau du praticien pour apprendre, mais bien au sachant de se mettre au niveau de celui qui veut apprendre. Sur ce point, nous avons encore énormément de travail et quand on voit certaines personnes – se moquant d'ailleurs ouvertement des débutants – s'improviser formateurs, on est, au mieux dubitatif, au pire terriblement inquiet.

En matière de sécurité informatique, le mépris est encore plus criant et il faut avoir beaucoup de curiosité pour avoir envie de persévérer dans l'étude de cette matière fascinante après avoir essuyé insultes, commentaires péjoratifs et autres noms d'oiseau qui fleurissent ici et là.

Cette attitude n'est pas l'apanage du seul monde de l'informatique, mais c'est le seul domaine pour lequel deux injonctions contradictoires sont systématiquement mises en œuvre : utilisez l'informatique pour tout (et parfois n'importe quoi) et débrouillez-vous seul pour comprendre comment ça fonctionne. Si on transposait cela dans la vie quotidienne, cela donnerait ceci :

Bonjour, je voudrais une baguette pas trop cuite ;

Bonjour, donc pour faire une baguette, on doit utiliser de la farine. Cette farine est produite dans...

Non, mais je veux juste acheter une baguette de pain, s'il vous plaît ;

Avant d'avoir l'honneur d'acheter cette baguette de pain, vous devez apprendre et comprendre comme elle a été faite.

Ce dialogue complètement imaginaire vous paraît tiré par les cheveux ? Lisez plus bas :

Bonjour est-ce qu'il existe un module de suggestion orthographique ne nécessitant pas Apache SoLr ?

search_api_autocomplete avec search_api_db ?

C'est ce que j'avais vu, mais j'avais cru comprendre qu'il fallait obligatoirement Apache SoLr. Mais je ne suis peut-être pas assez réveillée.

"Autocompletion only works on backends supporting it – currently, this is the case only for SoLr and the database."

...

Et en français ?

t("Autocompletion only works on backends supporting it – currently, this is the case only for SoLr and the database.");

Et c'est justement parce que je ne suis pas assez réveillée pour comprendre ce que je lis sans ambiguïté que je posais la question...

Dans ce cas merci de venir réveillée poser des questions pour ne pas nous faire perdre notre temps !

Ce dialogue n'est pas imaginaire et a eu lieu sur un canal IRC, théoriquement dédié à l'aide et au support des utilisateurs d'une solution de gestion de contenus et ce n'est qu'un exemple parmi tant d'autres. Le problème étant que lorsqu'on se fait accueillir avec un certain mépris, il est difficile de rester courtois et d'avoir envie de persévérer, surtout quand la documentation, censée exister, est lacunaire, voire absente.

Une des questions communément posées en entretien d'embauche de chef de projet concerne la production de code et dans le cas de Drupal, des modules Drupal. Faisant preuve d'une franchise à toute épreuve en la matière, j'ai toujours répondu que je n'avais jamais développé de modules Drupal, car les contributions sont suffisantes pour ne pas avoir besoin de réinventer la roue. J'aurais également pu préciser que si le premier pisseur de code était capable de faire un module Drupal, faire une documentation claire, sans ambiguïté ainsi que des tutoriels ça se saurait.

Cette absence d'empathie, de considération ou de mise en situation est tellement criante dans certains sous-domaines de l'informatique qu'il n'est pas du tout étonnant que certaines personnes extérieures considèrent les techniciens comme des unités de production et non pas comme des personnes. En ne nous mettant pas au niveau de nos interlocuteurs, nous les avons incités à nous considérer comme des unités de production.

Mieux encore : non seulement nous avons incité les personnes extérieures à l'informatique à nous considérer comme des unités de production, mais nous leur avons fait comprendre que nous étions des unités de production peu onéreuses et totalement interchangeables.

2 La tentative de génération de revenus sans prestation de services

La multiplication des événements autour de la technique et de l'informatique donne une impression de masse, ce qui est paradoxal pour un domaine au sujet duquel, tout le monde semble s'accorder à dire qu'il y a pénurie de compétences. Pourtant, il suffit de se rendre sur n'importe quel site de recrutement et/ou de mise en relation avec des indépendants en informatique pour voir que s'il y a effectivement des catégories dans lesquelles il y a des pénuries, d'autres sont déjà bien fournies. À partir de là, la sélection peut se faire sur l'offre la mieux-disante ou la plus intéressante commercialement parlant.

En matière de logiciel théoriquement prêt à l'emploi, on peut également constater quelques libertés avec la réalité technique. Il y a quelque temps de cela, je recherchais un logiciel – pas nécessairement gratuit ni même open source ou libre (quelle hérésie n'est-ce pas) capable de monitorer les changements, non pas sur une page Web, mais sur l'ensemble d'un site. Une sorte de flux RSS, sans flux RSS avec une représentation graphique des changements. L'objectif étant de suivre des modifications mineures, mais dont les conséquences peuvent s'avérer majeures à partir d'une liste de sites Web prédéterminés par mes soins, lesquels ayant chacun quelques milliers de pages, sur la base d'une liste d'exclusion. Idéalement, je souhaitais que le logiciel m'envoie un e-mail m'alertant d'une modification ou – à défaut – que le logiciel s'exécute en tâche de fond. À ce stade de l'histoire, je tiens à préciser que je préférais un logiciel compatible avec Windows, installé sur une de mes machines personnelles et non pas sur un serveur qui me serait inconnu. Au bout de trois jours de recherches infructueuses et de tests décevants, je devais me rendre à l'évidence : aucune des solutions proposées que j'ai trouvées ne surveillait les changements de sites Web.

Au-delà de ma déception personnelle – qui n'est pas l'objet du débat – quelque chose m'a frappé. Non seulement quasiment aucune des solutions n'annonçait clairement la couleur sur ses capacités réelles, mais le soi-disant support commercial et technique était fantaisiste. Ainsi, pour un logiciel conçu par des Allemands, qui semblait répondre à mes besoins, il a fallu sept e-mails avec le support et des petits bricolages de mon côté pour me rendre compte que le produit n'était pas à la hauteur de mes espérances. Or, les questions posées étaient très simples et auraient dû nécessiter des réponses tout aussi simples, surtout pour un logiciel payant. Étant de nature particulièrement têtue, je suis allée au fond des choses, mais un consommateur – puisque nous parlons de consommation – n'aura pas cette patience. Comme pour le micro-ondes, il veut quelque chose de rapide et de simple. Un ami qui avait rencontré un problème avec une solution logicielle payante et payée s'est vu répondre par le support technique « si vous cherchez dans Google, vous trouverez la solution ».

La grande force d'Apple, au-delà des aspects purement marketing et de mise en scène qui étaient la marque de

NE MANQUEZ PAS
HACKABLE N°9 !

ENFIN UNE SOLUTION D'AFFICHAGE 7 POUCES DE QUALITÉ POUR LA PI !

ÉCRAN OFFICIEL RASPBERRY PI

DISPONIBLE

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

fabrique de Steve Jobs, est d'avoir su se mettre au niveau des utilisateurs qui voulaient quelque chose de simple, d'intuitif et d'élégant. Jusqu'à présent, aucune marque n'arrive réellement à détrôner la firme à la pomme. S'il est évident qu'il y a un phénomène de mode voire d'endoctrinement et d'enfermement de l'utilisateur dans un écosystème spécifique, le fait est que même les libristes les plus aguerris finissent par céder ou par revenir vers Apple, soit en raison de la solidité du matériel soit en raison de la qualité du service.

Le technicien n'est pas une machine qui sait tout, mais l'utilisateur final et/ou le client n'est pas non plus systématiquement un abruti fini or le second paramètre ne semble pas toujours être pleinement intégré. À partir de là, difficile pour l'utilisateur « normal » de ne pas considérer le technicien comme une unité de production. Mieux encore, il creuse lui-même sa tombe et est aidé dans ce sabordage par l'État.

3 Quand l'État creuse la tombe des techniciens … avec leurs encouragements

Nous avons été quelques-uns à soulever un sourcil interrogateur devant la déclaration pour le moins délirante de la Ministre de la Culture concernant le « codage » à l'école et surtout son apprentissage en trois jours destinés aux enseignants, qui devraient eux-mêmes l'enseigner à leurs élèves. Soulevant un certain paradoxe dans cette démarche, une journaliste de 20 minutes **[1]** lui a posé la question suivante : *« Ne faudrait-il pas créer un CAPES et une agrégation d'informatique pour avoir de vrais experts du sujet ? ».*

La question est légitime puisqu'une agrégation est nécessaire pour enseigner l'anglais, par exemple, qui est une discipline à part entière, tout comme la littérature, les mathématiques ou l'histoire. La réponse de la ministre fut la suivante : *« Cela reviendrait à créer une discipline à part entière. Or, le numérique traverse toutes les disciplines et nous voulons que tous les enseignants puissent l'utiliser dans leurs cours. Un enseignant d'anglais doit pouvoir se servir d'un laboratoire numérique de langues. Et un professeur de musique doit avoir la possibilité d'initier ses élèves au piano sur tablette. »*

Curieusement, cette interview, que je considère comme une insulte, à la fois à l'informatique, aux matières énoncées dans cette réponse et à l'enseignement en général, n'a pas soulevé d'indignations, si ce n'est quelques tweets moqueurs ici et là. Pourtant, si on pense réellement que l'initiation au numérique peut s'enseigner en trois jours à des adultes pour qu'ils le distillent quotidiennement à des enfants, c'est soit qu'on ne croit pas ce qu'on raconte soit qu'on n'a plus toutes les frites dans le même sachet. Là où justement, les techniciens auraient dû être vent debout contre la dévalorisation de leur matière, ils sont restés très silencieux, accréditant malheureusement l'idée que le premier crétin qui passe peut s'improviser formateur ou enseignant en informatique, réduisant ainsi le technicien à une simple unité de production, interchangeable, sans aucune valeur ajoutée.

Cette attitude paradoxale est finalement la cristallisation de l'ambiguïté des techniciens : ils veulent être considérés comme des personnes à part entière, avec des compétences propres, mais lorsqu'il s'agit de défendre leurs professions ou de gérer leurs clients et/ou leurs utilisateurs, afin de les fidéliser ou de concrétiser un éventuel acte d'achat, ils sombrent dans le mutisme intense. Qu'il soit clair pour le lecteur que je fais une généralité et que je suis parfaitement consciente qu'il existe des techniciens informatiques qui aident les utilisateurs, leur répondent gentiment, se mettent à leurs niveaux. Malheureusement, ils ne semblent pas constituer une majorité.

4 À quand une prise de conscience ?

On pourrait expliquer cette attitude par l'aspect novateur de cet écosystème. Pourtant, depuis les premiers ordinateurs, les bidouillages d'imprimantes de R.M. Stallman, les téléphones portables intelligents jusqu'aux révélations de Snowden, on aurait pu penser qu'était justement venu le temps du recul, de la réflexion et de la prise de distance.

Alors que la technologie prend une place de plus en plus grande dans nos vies, on a parfois l'impression qu'il n'y a pas de recul, pas de prise de conscience sur ces nouveaux éléments alors même qu'ils posent des questions fondamentales sur nous, notre mode de vie et dans certains cas, nos limites. Qu'est-on prêt à accepter ? Jusqu'où sommes-nous capables d'aller ? À quel point sommes-nous devenus dépendants ? Comment ferons-nous si un jour, nous sommes privés d'électricité et que nous devions revenir aux fondamentaux alors qu'il y a au moins trois générations dans nos sociétés occidentales qui ne connaissent absolument rien de la nature ?

De la même façon, il y a autre chose qui devient de plus en plus dérangeant, notamment pour la génération actuellement à l'école. Les enfants sont de plus en plus tôt confrontés à des écrans, à des machines, à des appareils électroniques, largement encouragés par l'Éducation Nationale. Mais sommes-nous en train de former des utilisateurs aguerris qui sauront remettre en question les dogmes ou une nouvelle génération de consommateurs, qui ne sera pas capable de se désintoxiquer de l'écran ? De récentes études montrent que des adultes, qui ne sont pas nés avec l'outil informatique, sont incapables de rester éloignés de leurs téléphones portables et/ou de leurs ordinateurs. Comment des enfants qui ont grandi à l'ère du tout numérique pourront-ils en être capables, surtout s'ils n'ont pas été éduqués en ce sens. La question se pose d'autant plus largement que les grands patrons du numérique américains scolarisent leurs enfants dans des établissements scolaires où l'informatique est quasiment absente et où les téléphones portables et tablettes sont interdits. Par ailleurs, en cherchant à tout prix à initier les enfants au « codage », nous risquons de créer un nouveau facteur discriminant entre ceux qui seront théoriquement coder et ceux qui ne le sauront pas.

Depuis quelques années, l'informatique a été conçue et pensée pour être une fin en soi et non plus un moyen et là encore, c'est Apple qui en est la plus parfaite démonstration. On n'achète pas un téléphone portable pour s'en servir, on achète un iPhone pour avoir l'illusion d'appartenir à une communauté de *happy few*, dont la seule sélection serait la capacité financière à acquérir très régulièrement un nouveau modèle et dont les détenteurs seraient auréolés d'une sorte de divine grâce qui embellirait chaque instant de leur vie. C'est du moins le message transmis dans chacun des spots publicitaires de la marque. Or, à partir du moment où un instrument est pensé comme une fin en soi, le mouvement naturel veut que celui qui provient à cette fin soit lui-même considéré comme un moyen, par un effet d'inversion intellectuelle. Dans l'Histoire récente, on l'a vu avec l'industrie. Initialement créée pour améliorer les conditions de vie et développer la croissance – et les bénéfices des propriétaires d'usines, ne nous voilons pas la face – elle est devenue un secteur d'activité à part entière pour finir comme étant une sorte de totem intouchable. Mais si l'Industrie est devenue la marotte de nos différents gouvernements depuis la fin des Trente Glorieuses, le sort des ouvriers de ce secteur est devenu une vue de l'esprit et non plus une réalité, sauf pour les ouvriers eux-mêmes.

Conclusion

Il ne s'agit pas de démolir en bloc tout ce que l'informatique nous a apporté et continuera à nous apporter, mais de commencer à nous poser les bonnes questions sur notre devenir.

L'informatique ou le numérique – peu importe comment on le nomme – change notre façon de penser, notre façon de voir le monde, les choses, mais nous devons, surtout nous les geeks, prendre garde à ne pas scier la branche sur laquelle nous sommes assis, au risque de nous voir réduits au rang d'objet, incapable de couper ce lien de dépendance que nous avons avec la technologie. ■

Référence

[1] Numérique à l'école : « En 2018, 100% des collèges seront connectés » http://www.20minutes.fr/societe/1710175-20151015-numerique-ecole-2018-100-colleges-connectes


ILS SONT FOUS CES ROMAINS !

Tristan COLOMBO

Vous aurez reconnu l'une des phrases favorites d'Obélix... mais derrière ce clin d'œil d'amateur de bandes dessinées, se cache une question plus mathématique : comment passer de la numération décimale à la numération romaine et inversement ?

Mots-clés : Numération décimale, Numération romaine, Conversion, Algorithme, Python

Résumé

La conversion de la numération décimale vers la numération romaine nous sert de prétexte pour développer des algorithmes simples faisant intervenir la manipulation de listes et de chaînes de caractères.

Tout part d'une question toute bête de mon fils de sept ans qui s'amusait à convertir des nombres en numération romaine : « Papa, comment on fait quand on arrive à 5000 ? ». Jusqu'à 4999 j'arrive à me débrouiller... mais là, 5000 ? Est-ce qu'il existe un symbole particulier pour représenter cette quantité ? La question première était donc de chercher la lettre représentant 5000 puis, tant qu'à faire, j'ai proposé de rechercher une méthode permettant de trouver mécaniquement l'écriture romaine d'un nombre décimal. Pour cela, j'ai commencé par remettre à plat mes connaissances, histoire de ne pas me tromper...

1 Numération romaine

La numération romaine utilise un ensemble de symboles ou lettres dont la valeur peut être additionnée ou soustraite suivant leur position les unes par rapport aux autres.

1.1 Jusqu'à 4999, pas de problème

Les chiffres romains « standards » sont au nombre de sept et, même si vous ne les avez pas tous employés depuis longtemps, vous devriez vous en souvenir. Ils sont rappelés dans le tableau ci-dessous :

Chiffre romain	Valeur décimale
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Il y a ensuite un nombre très restreint de règles à respecter :

- Les symboles s'écrivent de gauche à droite par ordre de valeur décroissante (sauf cas particulier du XXX) ;
- On ne peut pas employer un symbole plus de trois fois de suite (sauf **M** qui pourra être utilisé quatre fois) ;
- Tout symbole suivant un symbole de valeur supérieure ou égale s'ajoute à la valeur précédente. Par exemple, **VI = 6**, **XX = 20**, **LXV = 65**, etc. Par contre, les symboles **V**, **L** et **D** ne peuvent se suivre.
- Un symbole qui précède un symbole de valeur supérieure doit être soustrait à ce dernier suivant les règles suivantes :
  - → **I** peut être retranché à **V** ou **X** ;
  - → **X** peut être retranché à **L** ou **C** ;
  - → **C** peut être retranché à **D** ou **M**.

Connaissant ces règles, on peut effectivement compter de **1** à **4999** soit **I** à **MMMMCMXCIX** (le zéro n'était pas connu des Romains). De nos jours, on va rarement au-delà de **XXI** en numération romaine puisque c'est l'écriture employée couramment pour indiquer un siècle (ou aussi les noms de rois parmi lesquels on peut citer « Louis croix-V-bâton »).

1.2 À partir de 5000

Il n'y a pas de symbole pour représenter **5000** ! Une autre règle est alors appliquée :

- Tout symbole se trouvant sous une barre horizontale est multiplié par un facteur mille (et pour **n** barres par un facteur 1000^n).

5000 s'écrit donc $\overline{V}$, **14022** s'écrit $\overline{XIV}$**XXII** (**14 x 1000 + 22**) et ainsi de suite. Pour des nombres très élevés, on peut même avoir des écritures du type $\overline{\overline{XIV}}\overline{CXX}$**IV** (**14 x 1000^2 + 120 x 1000 + 4 = 14 120 004**).

2 Conversion en numération romaine

Les règles énoncées précédemment ne sont pas toujours strictement appliquées ; il peut exister des variantes... Nous nous contenterons de ces règles pour ne pas alourdir inutilement notre algorithme et nous allons commencer par convertir un nombre décimal dans sa représentation en numération romaine.

Nous allons utiliser ici simplement des tableaux de conversion pour remplacer les valeurs des unités, dizaines, centaines, etc. Le problème principal va être de segmenter l'entier en centaines, centaines de milliers, etc. de manière à pouvoir appliquer un traitement répétitif. Ainsi, pour **4025** nous souhaitons obtenir la liste **[4, 25]**, ce qui nous permettra de calculer la représentation de **4** (**MMMM**) puis de **25** (**XXV**). Pour **1125643**, nous voulons **[1, 125, 643]**, etc.

Une fois l'idée du découpage trouvée, il n'y a plus grand-chose à faire si ce n'est de réaliser une implémentation lisible. Voici le code du fichier **Roman.py** permettant de calculer un nombre romain d'après un nombre décimal :

```
01: #!/usr/bin/python3
02:
03: class Roman:
04:     units = ['', 'I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX', 'X']
05:     decades = ['', 'X', 'XX', 'XXX', 'XL', 'L', 'LX', 'LXX', 'LXXX', 'XC']
06:     hundreds = ['', 'C', 'CC', 'CCC', 'CD', 'D', 'DC', 'DCC', 'DCCC', 'CM']
07:     dicos = [units, decades, hundreds]
```

Nous créons une classe **Roman** qui contient quatre attributs statiques : les trois premiers (lignes 4 à 6) sont les listes de conversion et le dernier, en ligne 7, est une liste de pointeurs vers les listes précédentes qui nous servira plus tard pour économiser des lignes de code. Le fonctionnement des listes de conversion est très simple : pour **234** nous avons **2** centaines, **3** dizaines et **4** unités donc pour la conversion on utilise **hundreds[2]** soit **CC**, **decades[3]** soit **XXX** et **units[4]** soit **IV** ce qui donne **CCXXXIV**. C'est pour cela que le premier élément de chaque liste contient une chaîne vide : l'index **0** ne sert pas.

```
10:     def __init__(self, decimal_number):
11:         self.__decimal = decimal_number
12:         self.__roman = self.__convert()
```

Au niveau du constructeur, nous définissons deux attributs privés permettant de conserver l'entier passé en paramètre (**decimal_number** en ligne 11) et la conversion en numération romaine en ligne 12 en faisant appel à la méthode privée **__convert()** qui sera définie dans la suite.

```
15:     def __getDecimal(self):
16:         return self.__decimal
17:     decimal = property(__getDecimal)
18:
19:
20:     def __getRoman(self):
21:         return self.__roman
22:     roman = property(__getRoman)
```

Pour que la classe soit plus simple à utiliser, j'ai défini les propriétés **decimal** et **roman** pour accéder aux attributs privés.

```
25:     def __partition(self):
26:         def calc_partition(n):
27:             while len(n) > 3:
28:                 n = n[:-3]
29:             return int(n)
30:
31:         strDecimal = str(self.__decimal)
32:         return list(map(calc_partition,
33:             [strDecimal[i:] for i in range(-3,
-len(strDecimal)-4, -3)]))
```

Comme dit précédemment, nous devons segmenter notre entier en centaines, centaines de milliers, etc. Donc pour faire simple, si nous considérons notre entier comme une chaîne de caractères nous voulons une liste de blocs de trois caractères en partant de la fin de la chaîne. Nous commençons donc par convertir l'entier en chaîne de caractères en ligne 31. Les lignes 32 et 33 réalisent beaucoup d'opérations et nous allons les analyser pas à pas :

- **[strDecimal[i:] for i in range(-3, -len(strDecimal)-4, -3)]** : On crée une boucle dans laquelle **i** varie de **-3** à **-longueur_chaîne - 4** (si on dépasse la taille de la chaîne il n'y a pas d'erreur et on évite d'oublier des caractères à cause du pas) par pas de **-3**. Ceci utilise le fait qu'en Python un index négatif est calculé depuis la fin de la chaîne (**-1** est le dernier caractère et ainsi de suite). **strDecimal[i:]** vaut donc successivement **strDecimal[-3:]**, **strDecimal[-6:]**, etc. Donc, en prenant pour exemple **1234567**, la liste produite est **[567, 234567, 1234567]**.
- **list(map(calc_partition, ...))** : La fonction **map()** appelle la fonction **calc_partition()** sur l'ensemble des éléments de la liste qui lui est passée en paramètre (la liste calculée précédemment). En Python 3, **map()** renvoie un générateur, c'est pour cela que l'on appelle **list()** sur son résultat. La fonction **calc_partition()** des lignes **26** à **30** est ici primordiale : elle retire progressivement tous les trois caractères les plus à droite de manière à ne conserver que les trois caractères les plus à gauche (ou deux, ou un). En reprenant notre exemple de **[567, 234567, 1234567]**, nous obtenons **[567, 234, 1]**. On pourrait optimiser ce calcul en utilisant une expression telle que **((len(n) - 1) // 3 * -3) - 1** qui donne directement l'indice du premier caractère à conserver... mais qui est nettement moins lisible !

```
37:     def __translateGroup(self, group, rank):
38:         if rank == 1 and group <=4 and group >= 1:
39:             return (group * 'M', 0)
40:
41:         group = str('{:03d}'.format(group))
42:
43:         symbol = ''
44:         for i in range(3):
45:                 symbol += Roman.dicos[2-i][int(group[i])]
46:
47:         return (symbol, rank)
```

Une fois les groupes obtenus, il faut donner leur représentation en numération romaine. Pour cela, il faut savoir à quel rang (ici paramètre **rank**) se trouve le groupe à traduire : rappelez-vous que jusqu'à **4000** on peut utiliser des **M**, mais plus après. C'est l'objet du test de la ligne 38. On renvoie alors un tuple contenant la traduction et un indice indiquant le coefficient de multiplication par mille (nombre de barres de surlignage). Ensuite, pour pouvoir tout automatiser, on s'assure que le groupe est une chaîne de trois caractères en ajoutant le cas échéant des zéros devant les nombres trop petits (ligne 41). Il n'y a plus qu'à parcourir ces caractères et utiliser le dictionnaire associé à leur position pour effectuer la traduction. C'est en ligne 45 qu'intervient la liste **Roman.dicos** qui contient des pointeurs vers les listes de traduction : en fonction de la position du caractère, on utilise le bon dictionnaire.

```
50:     def __convert(self):
51:         result = ''
52:
53:         for i, p  in enumerate(self.__partition()):
54:             conv = self.__translateGroup(p, i)
55:             if conv[1] == 0:
56:                 result = conv[0] + result
57:             elif conv[0]!= '':
58:                 result = '(' + conv[0] + ')(' + str(conv[1])
+ ')' + result
59:
60:         return result
```

La méthode de conversion n'a plus qu'à appeler les méthodes précédentes : **__partition()** pour obtenir les

NE MANQUEZ PAS

MISC N°82 !

PROTÉGEZ VOS CODES À TOUS LES NIVEAUX !

DISPONIBLE

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

groupes (ligne 53) et **__translateGroup()** pour convertir ces groupes (résultat stocké dans **conv** en ligne 54). **conv** contient un tuple avec la représentation en numération romaine du groupe (**conv[0]**) et le nombre de barres multiplicatives (**conv[1]**). Comme vous pouvez le voir en ligne 58, pour simplifier la représentation, j'ai choisi d'indiquer les groupes multipliés à l'aide de parenthèses : **(symboles romains)(multiplications par 1000)**.

```
63:     def __str__(self):
64:         return self.__roman
```

La définition de **__str__()** permet simplement de rendre la classe plus facile à utiliser.

```
67: if __name__ == '__main__':
68:     d = Roman(14022)
69:     print('{} => {}'.format(d.decimal, d.roman))
```

Dans le programme principal, nous affichons le nombre **14022** et sa représentation en numération romaine :

```
14022 => (XIV)(1)XXII
```

Je considère volontairement ici que l'utilisateur fournit toujours en paramètre du constructeur de la classe un entier et je ne traite donc pas les cas d'erreur. Si vous le souhaitez, vous pouvez ajouter le déclenchement d'une exception si **decimal_number** n'est pas un entier en ligne 11.

3 Conversion en numération décimale

Pour la conversion inverse, nous allons reprendre le même schéma de classe en créant une classe **Dec** (**Decimal** existe déjà et le fait de créer une classe de même nom pourrait induire une erreur si l'on voulait l'utiliser). Il va falloir parcourir le nombre de la droite vers la gauche et appliquer les règles 3 et 4 vues précédemment :

```
01: #!/usr/bin/python3
02:
03: class Dec:
04:     symbols = {'I' : 1, 'V' : 5, 'X' : 10, 'L' : 50, 'C' :
100, 'D' : 500, 'M' : 1000}
```

Un attribut statique **symbols** permet de connaître l'ordre et la valeur numérique des symboles employés.

```
06:     def __init__(self, roman_number):
07:         self.__roman = roman_number
08:         self.__decimal = self.__convert()
```

Le constructeur est le même que dans la classe **Roman**... mais l'attribut directement stocké et l'attribut calculé sont bien sûr inversés.

```
10:     def __getDecimal(self):
11:         return self.__decimal
12:     decimal = property(__getDecimal)
13:
14:     def __getRoman(self):
15:         return self.__roman
16:     roman = property(__getRoman)
```

On retrouve les méthodes d'accès aux attributs avec les propriétés associées.

```
18:     def __explode(self):
19:         result = []
20:         save = None
21:
22:         for elt in self.__roman.split(')'):
23:             if elt.startswith('('):
24:                 if save is None:
25:                     save = elt[1:]
26:                 else:
27:                     for symbol in save:
28:                         result.insert(0, (symbol,
int(elt[1:])))
29:                     save = None
30:             else:
31:                 for symbol in elt:
32:                     result.insert(0, (symbol, 0))
33:         return result
```

La méthode privée **__explode()** va lire la chaîne de caractères et en extraire chaque symbole pour former des couples indiquant le facteur multiplicatif. Par exemple pour **(XIV)(1)XLII** nous obtiendrons : **[('I', 0), ('I', 0), ('L', 0), ('X', 0), ('V', 1), ('I', 1), ('X', 1)]**. En employant la méthode **insert()** pour ajouter les éléments et en indiquant à chaque fois l'index **0** (lignes 28 et 32) nous inversons l'ordre des symboles pour pouvoir réaliser ensuite une lecture de la droite vers la gauche du nombre. En ligne 32, pour minimiser le nombre de lignes de code et conserver le même traitement pour tous les symboles, le facteur **0** est utilisé pour les symboles non surlignés (**$1000^0 = 1$**).

```
35:     def __convert(self):
36:         value = 0
37:         pred = (None, None)
38:         partition = self.__explode()
39:
40:         for elt in partition:
41:             if pred[0] is None and pred[1] is None:
42:                 value = Dec.symbols[elt[0]] * 1000 ** elt[1]
43:             else:
44:                 if Dec.symbols[elt[0]] < Dec.symbols[pred[0]] and
elt[1] == pred[1]:
45:                     value -= Dec.symbols[elt[0]] * 1000 ** elt[1]
46:                 else:
47:                     value += Dec.symbols[elt[0]] * 1000 ** elt[1]
48:             pred = elt
49:         return int(value)
```

La conversion est effectuée en travaillant sur la liste obtenue après appel à **`__explode()`** (ligne 38). La variable **`value`** de la ligne 36 permettra de stocker la valeur du nombre et la variable **`pred`** de la ligne 37 conservera les informations sur le symbole précédent (pour l'application des règles d'addition ou de soustraction). Nous parcourons ensuite l'ensemble des symboles de la liste **`partition`** (ligne 40) et en fonction du symbole précédent (**`pred[0]`**) et du rang du symbole courant (**`elt[1]`**), nous calculons la valeur de **`value`** (lignes 42, 45 et 47).

```
51:     def __str__(self):
52:         return self.__roman
```

La méthode **`__str__()`** est définie de manière classique pour utilisation directe avec **`print()`**.

```
54: if __name__ == '__main__':
55:     d = Dec('(XIV)(1)XLII')
56:     print('{} => {}'.format(d.roman, d.decimal))
```

Comme précédemment, je considère que la chaîne transmise au constructeur de **`Dec`** est bien formée. Il faudrait ajouter une méthode permettant de valider le paramètre **`roman_number`** en ligne 7. En exécutant ce code nous obtenons :

```
(XIV)(1)XLII => 14042
```

Conclusion

Objectif atteint : nous avons pu développer deux petits programmes de conversion ! Il reste maintenant à expliquer tout ça à un enfant de sept ans qui ne sait programmer qu'en Logo et EV3 (programmation graphique Lego Mindstorm). Je pense avoir été peut-être un peu trop ambitieux... Finalement je vais me limiter à une explication de la partie algorithmique, mais la question était intéressante ! ■

ANALYSE SPECTRALE WIFI

Arnaud FÉVRIER [Maître de conférences en Informatique, Aix Marseille Université, laboratoire I2M, équipe eRISCS]

La configuration d'un réseau wifi est assez facile. Mais parfois, après déploiement, les utilisateurs se plaignent que « le réseau ne marche pas ». Il existe des outils informatiques qui permettent de corriger quelques problèmes. Mais quand ceux-ci ne sont pas suffisants, il faut vérifier ce qui se passe sur les fréquences.

Mots-clés : Wifi, Spectre, Perturbations électromagnétiques, Analyseur de spectre USB, Spectools, Plan de fréquences

Résumé

L'installation d'un réseau wifi pose souvent des problèmes. Au-delà de la mise en place du réseau par des techniques informatiques et réseaux, l'utilisation de transmission hertzienne peut générer des défaillances non prises en compte par les interfaces wifi et les systèmes d'exploitation.

Nous présentons donc les techniques de transmission utilisées par le wifi et l'utilisation d'un analyseur de spectre pour détecter et corriger les perturbations. Nous montrons trois perturbations classiques et leur mise en évidence.

Nous présentons dans cet article l'utilisation d'un analyseur de spectre pour aider à résoudre certains problèmes de déploiement de réseaux wifi.

Le wifi est constitué d'un ensemble de standards industriels (IEEE 802.11). La Wi-Fi Alliance regroupe les entreprises qui souhaitent certifier leurs produits. Il existe plusieurs techniques de transmission utilisées par le wifi. Dans les interfaces de configuration, plusieurs sont mentionnées (a, b, g, n). Nous n'allons pas détailler chacune de ces techniques mais montrer comment une utilisation simple d'un analyseur de spectre bon marché peut permettre de détecter et corriger des perturbations causées par d'autres sources que le wifi (moteurs, micro-ondes, Bluetooth, etc.).

Nous allons d'abord présenter la transmission wifi de manière succincte et présenter l'utilisation élémentaire d'un analyseur de spectre. Ensuite, nous montrerons le spectre d'un signal wifi normal sans perturbation, après quoi nous montrerons l'effet de quelques éléments perturbateurs. Enfin, nous donnerons quelques indices pour détecter et corriger les défauts.

1 Comment ça marche ?

Nous présentons dans cette partie le fonctionnement radio du wifi et l'utilisation d'un analyseur de spectre.

1.1 Les fréquences

Le signal wifi est un signal radio qui utilise une bande de transmission disponible pour cet usage.

Pour analyser un spectre, il faut avoir quelques notions de transmission et de traitement du signal. L'espace des fréquences électromagnétiques est divisé en bandes. Chaque pays définit quel usage sera affecté à chaque bande **[1]**.

En France, c'est l'Agence Nationale des Fréquences **[2]** qui gère les utilisations de bandes de fréquence. Il y a ainsi la bande FM (87,5 – 108 MHz) **[3]**. La bande FM est subdivisée en canaux. Les stations de radio obtiennent, éventuellement, l'autorisation d'émettre sur un canal à partir d'une antenne donnée. Cette bande n'est pas considérée comme libre ; la station de radio doit demander, et obtenir, l'autorisation et vraisemblablement s'acquitter d'un droit d'utilisation.

Le principe technique est le même pour le wifi mais la bande wifi est considérée comme libre. Chacun peut donc émettre, dans les limites légales, sur ces canaux. La bande de fréquence du wifi est principalement la bande 2,4GHz. Cette bande fait partie des bandes ISM (industrielle, scientifique et médicale). D'autres bandes peuvent ou pourront être utilisées, comme la bande des 5GHz déjà en service. L'utilisation des bandes hautes suppose que l'ensemble des équipements peut l'utiliser, ce qui n'est pas forcément le cas des anciennes cartes wifi.

Nous allons principalement présenter la bande 2.4GHz. Celle-ci est encore la plus utilisée, et les analyseurs de spectre pour cette bande sont beaucoup moins chers que pour les bandes de plus haute fréquence.

1.2 La bande WIFI 2.4GHz

La bande 2.4GHz est divisée en 14 canaux espacés de 5MHz, en commençant par le canal 1, centré sur 2,412 GHz. Le canal 14 est plus écarté du canal 13 (12 MHz du précédent). Encore une fois, tous les équipements ne sont pas compatibles avec tous les canaux.

Le signal wifi, centré dans un canal, est plus large que la largeur de ce canal (environ 22MHz, soit cinq canaux). Deux émetteurs dans deux canaux voisins vont créer des interférences qui vont limiter la communication. Pour éviter les perturbations entre des canaux adjacents, seuls trois canaux sont utilisés : 1,6,11 ou 1,7,11.

1.3 Les outils

L'outil élémentaire est un analyseur de spectre. Le coût d'acquisition de ces équipements croît avec la fréquence maximum. Un analyseur qui peut mesurer les signaux jusqu'à 3GHz coûte plus de 3000 euros. Pour de plus hautes fréquences, c'est encore plus cher. Ceux-ci sont assez complets et permettent d'obtenir des mesures précises avec beaucoup de paramètres. Ces paramètres ne sont pas forcément utiles pour détecter les problèmes causés aux transmissions wifi. De plus, il faut acheter et connecter une antenne wifi.

J'ai choisi, il y a quelques années un analyseur de spectre spécialisé wifi à bas coût. Il ne couvre pas la bande 5GHz, mais la technique serait la même. Cet analyseur est un Wi-Spy 2.4x qui se connecte sur un port USB. Un des avantages de cet équipement, en dehors de son prix, c'est qu'il est déjà prêt pour le wifi. Les réglages non triviaux d'un analyseur de spectre classique sont déjà faits, il n'y a plus qu'à regarder et interpréter.

Le paquet **spectools** (**apt-get install spectools**) fournit quatre utilitaires. Nous utiliserons dans cet article **spectool_gtk** qui permet de visualiser le spectre de la bande wifi. L'autre outil utile est **spectool_raw** qui affiche sur la sortie standard le résultat des mesures. Il effectue 256 mesures sur la bande wifi et fournit donc la puissance en dBm pour chaque point.

```
$ spectool_raw
Found 1 spectool devices...
Initializing WiSPY device Wi-Spy 24x USB 664339890 id 664339890
Configured device 1523123681 (Wi-Spy 24x USB 1523123681)
    2399MHz-2483MHz @ 327.00KHz, 256 samples
Wi-Spy 24x USB 1523123681: -107 -109 -111 -111 -111 ...
Wi-Spy 24x USB 1523123681: -108 -110 -111 -111 -114 ...
Wi-Spy 24x USB 1523123681: -108 -110 -110 -110 -108 ...
```

Ceci peut être utile pour enregistrer en continu le spectre et pouvoir ainsi avoir une chance de détecter pourquoi, dans le passé, il y a eu un problème. Pour manipuler ces données, j'ai utilisé le logiciel **octave** (**apt-get install octave**). GNU Octave est un logiciel de calcul scientifique qui permet de faire tous les calculs liés au traitement du signal. Nous n'utiliserons pas Octave directement dans cet article.

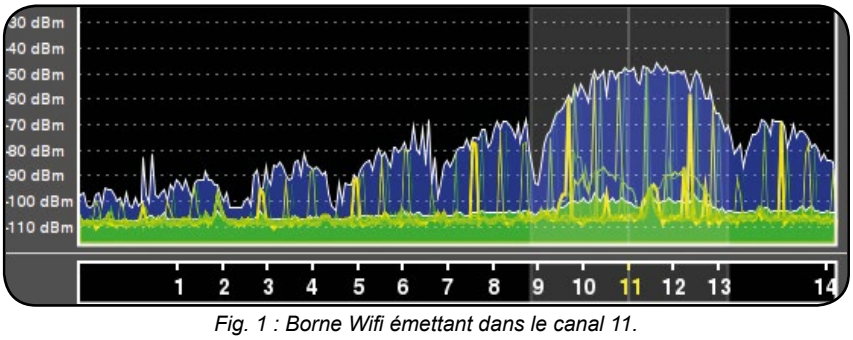


Fig. 1 : Borne Wifi émettant dans le canal 11.

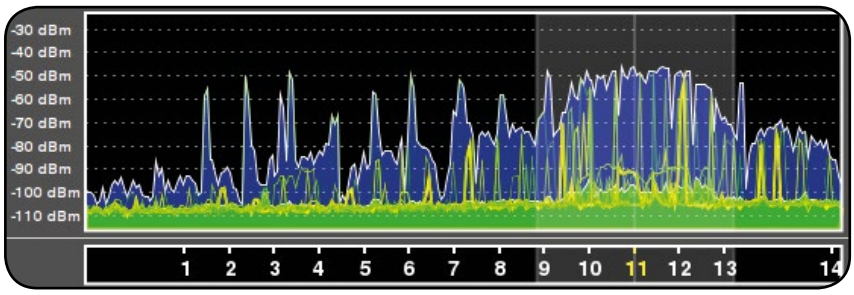


Fig. 2 : Borne Wifi émettant dans le canal 11, avec des perturbations wifi.

2 Le cas normal

Dans la figure 1, nous captons l'émission d'une borne centrée sur le canal 11. Le logiciel spectool_gtk grise les fréquences occupées par le signal centré sur le canal 11. Il est aisé de voir que l'utilisation des canaux adjacents va créer des problèmes.

Sur ce graphique, le sommet de la courbe se situe entre -40 et -50 dBm. La commande **`iwlist wlan0 scan`** indique une puissance de -43 dBm. Les valeurs sont compatibles.

```
Cell 04 - Address: F8:D1:11:0D:86:4D
          Channel:11
          Frequency:2.462 GHz (Channel 11)
          Quality=67/70  Signal level=-43 dBm
          Encryption key:on
          ESSID:"HermesAP"
```

Nous avons donc ici l'exemple normal d'une borne wifi. Pour être complet, il conviendrait de tester la borne dans chacune des configurations de transmission du wifi et de regarder l'influence du trafic sur le spectre de la transmission. Mais cela sort du cadre de cet article.

Après le comportement normal, nous allons maintenant étudier quelques éléments perturbateurs.

3 Quelques perturbations

Nous présentons dans cette section quelques perturbations courantes : les autres réseaux wifi ; d'autres transmissions utilisant la même bande, comme le Bluetooth ou un four, un micro-ondes.

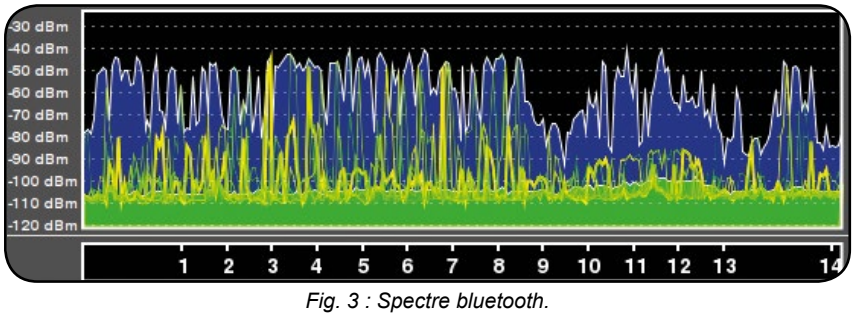


Fig. 3 : Spectre bluetooth.

3.1 Les autres wifi

Une des perturbations élémentaires consiste à utiliser le réseau wifi pendant les mesures. Dans la figure 2, qui correspond au même endroit que la figure 1, mais à cinq minutes d'intervalle, les Android étaient allumés. Il est donc vraisemblable que ce sont eux qui ont généré les pics situés entre les canaux 1 et 9.

De même, l'activation de l'interface wifi sur le portable de mesure a fait apparaître un niveau beaucoup plus élevé sur l'analyseur de spectre. L'interface réseau est très proche de l'interface de l'analyseur de spectre.

3.2 Le Bluetooth

Le Bluetooth utilise la même gamme de fréquences que le wifi. Pour le test, j'ai échangé des fichiers entre deux Android avec l'analyseur de spectre à proximité. La puissance reçue, donc perturbante, diminue avec le carré de la distance, certes, mais c'est parfois la même puce qui fournit les interfaces wifi et Bluetooth. Avec une distance de 20cm entre le portable Android et l'antenne de l'analyseur de spectre, le spectre est affiché dans la figure 3.

En plaquant l'android contre l'antenne de l'analyseur de spectre, le niveau de puissance dépasse celui du wifi. Grâce à **`spectool_raw`** et **`octave`**, le maximum sur quelques instants est de -23dBm.

3.3 Le micro-ondes

Le four à micro-ondes émet dans la même gamme de fréquence. Nous avons placé une box Internet près d'un four à micro-ondes et placé le détecteur derrière un mur. La capture du spectre de la box est montrée figure 4. Nous recevons une puissance d'environ -42dBm.

Quelques instants plus tard, nous faisons chauffer une tasse d'eau.

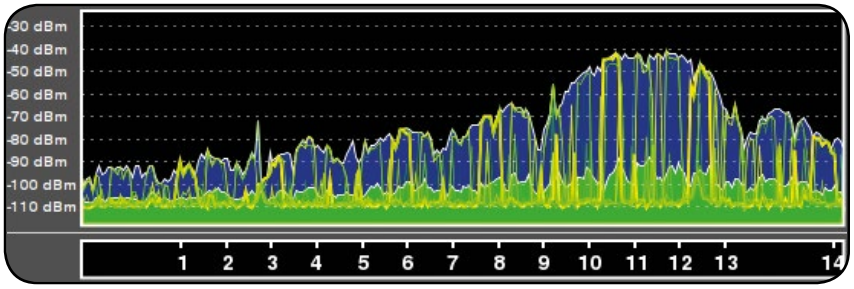


Fig. 4 : Box internet, avant le four à micro-onde.

Le spectre est affiché dans la figure 5. Le niveau de radiations émises par le four dépasse les -20dBm. Le spectre de ce four-là perturbe les canaux 6 à 13. L'utilisation du canal 1 est donc plus efficace (sauf pour montrer les problèmes).

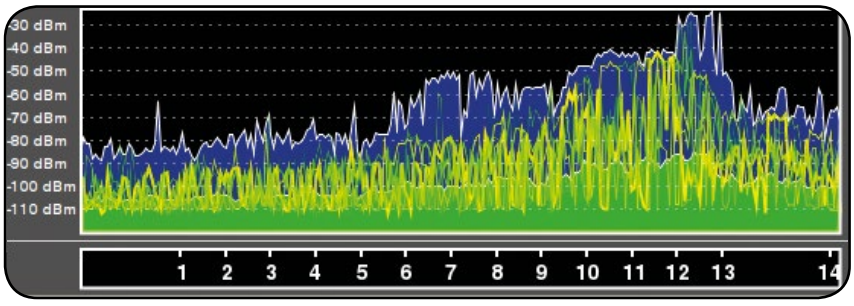


Fig. 5 : Box Internet à côté d'un micro-ondes.

Nous avons montré quelques perturbations, nous allons maintenant montrer quelques pistes pour diagnostiquer et résoudre les problèmes.

4 Expertise

L'utilisation simple de cet outil permet de visualiser facilement si la bande utilisée est déjà chargée ou non.

Le premier réglage consiste à positionner sa borne wifi dans une partie peu chargée du spectre. Au lancement, une box va scanner la bande wifi et choisir le meilleur canal. La documentation ne dit pas si elle vérifie les fréquences disponibles (analyse spectrale) ou les canaux utilisés (scan réseau). Tant que la borne reste active, elle ne vérifie pas s'il serait plus judicieux d'utiliser un autre canal. L'outil **`iwlist`** permet de détecter les émetteurs wifi à proximité et il affiche les puissances émises pour chaque SSID. D'autres outils graphiques permettent de visualiser cela sur un graphique. Néanmoins les perturbateurs autres que le wifi ne seront pas détectés par le système d'exploitation.

Si des perturbations autre que wifi existent, il est possible d'utiliser un portable et la clef pour se rapprocher de l'émetteur et identifier la source. Il est alors possible d'éviter la zone perturbée. En principe, le niveau de puissance de l'élément perturbateur ne devrait pas mettre en danger l'analyseur ni l'humain (enfin, cela, ça reste à voir).

Enfin, il est possible que des perturbations temporaires perturbent le wifi. La solution consiste à laisser le programme **`spectools_raw`** tourner pour visualiser (avec octave, par exemple) quels canaux sont perturbés et à quel rythme.

Conclusion

Nous avons vu l'utilisation d'un analyseur de spectre simple et directement interfacé avec un ordinateur. L'utilisation d'un analyseur de spectre plus traditionnel est néanmoins possible. Cependant ce genre de matériel est le plus souvent vendu sans documentation technique pour interfacer le matériel avec un système GNU/Linux. De plus, il est pratique pour ce genre de manipulation d'avoir un équipement léger, transportable.

La société fabricant le Wi-Spy favorise aussi les outils pour système privatif. Néanmoins la clef dont je dispose est utilisable sous GNU/Linux. Ce serait bien qu'un projet crée une version libre et low-cost de ce genre d'équipement.

Enfin, pour une utilisation pour faire un audit réseau, je n'ai pas trouvé de logiciel libre s'interfaçant avec mon analyseur qui pourrait préparer le rapport d'interventions. ■

Références

[1] Wikipédia, attribution des fréquences : https://fr.wikipedia.org/wiki/Attribution_des_frC3%A9quences

[2] Site officiel de l'Agence Nationale des Fréquences : http://www.anfr.fr

[3] Wikipédia, radio FM : https://fr.wikipedia.org/wiki/Radio_FM

CONFIGUREZ NGINX POUR ACCUEILLIR VOS SERVICES

Stéphane MOUREY [Mousse sur le Seeraiwer]

Nginx est le deuxième serveur Web le plus populaire après Apache dans le monde libre, et non sans raison. Plus jeune que son concurrent, Nginx a tiré les leçons de son aîné et résiste mieux que lui à de fortes charges. Mais Nginx peut faire plus. Voyons comment au travers d'un cas pratique.

Nginx Reverse proxy Services Docker Cache

L'OBJECTIF

Outre ses fonctions de serveur Web, **Nginx** peut également jouer le rôle de *reverse proxy* et de cache côté serveur pour les données statiques. La situation que je vous propose d'étudier va utiliser ces trois fonctions.

Imaginons que vous ayez mis en place des conteneurs **Docker** pour faire tourner, sur la même machine, des services Web de manière autonome. Tout marche bien, mais pour accéder à un service ou à un autre, vous êtes obligé de compléter l'adresse du serveur par un numéro de port, ce qui n'est pas très mnémotechnique. L'idéal pour vous serait d'accéder à chaque service en sous-domaine ou en sous-répertoire du nom de serveur principal, de laisser une simple page d'accueil sur le nom principal indiquant la nouvelle adresse de chaque service. Pour compléter le tout, tant qu'à faire, Nginx pourrait également mettre en cache les données statiques de façon à ne pas solliciter inutilement les services.

Pour résumer et concrétiser, si l'adresse du serveur est exemple.com, les missions Nginx seront :

- de permettre l'accès aux services accessibles par les adresses http://exemple.com:81 et http://exemple.com:82 respectivement par les adresses http://service1.exemple.com et http://service2.exemple.com ;
- d'offrir une page d'accueil sur l'adresse http://exemple.com proposant des liens vers les nouvelles adresses de ces deux services ;
- de fournir un cache pour les données statiques de ces mêmes services.

Remarquez que vous n'avez pas besoin d'être propriétaire d'un nom de domaine pour mener à bien les manipulations décrites dans cet article : vous pouvez utiliser à tous les endroits l'adresse IP de votre serveur à la place, au moins jusqu'en phase 5 ; à partir de là, nous verrons comment définir un nom de domaine en local valable pour notre seule machine...

LES OUTILS

- un système GNU/Linux Debian (version utilisée dans l'article Raspbian)
- **apt-get** (paquet **apt-get**)
- **Nginx** (paquet **nginx-light**)
- **Docker** (paquet **docker.io**)

PHASE 1

Sous Debian, sur le serveur exemple.com, nous installons le paquet **nginx-light**, ainsi que **docker.io** si ce n'est déjà fait.

```
root@exemple.com# apt-get install nginx-light docker.io
```

Pourquoi *light* ? Debian propose l'installation de Nginx à partir de trois paquets différents : **nginx-light**, **nginx-full** et **nginx-extras**. Chacun de ces paquets inclut un plus ou moins grand nombre de modules, le plus complet étant **nginx-extras**.

Mais **nginx-light**, le plus léger, suffit à nos besoins. Pour connaître le détail des modules en fonction des paquets, consultez https://wiki.debian.org/Nginx#line-34.

PHASE 2

Cet article étant centré sur Nginx, nous n'aborderons l'utilisation de Docker que le minimum nécessaire pour parvenir à nos fins, et en particulier, nous n'expliquerons pas comment conserver les modifications faites dans un conteneur. Je me contenterai de vous donner les commandes nécessaires au lancement des conteneurs dont nous avons besoin pour réaliser l'exercice. Vous pouvez sauter à la section suivante si vous disposez déjà de conteneurs proposant des sites Web.

Il nous faudra deux conteneurs, mais utilisant la même image. Ces conteneurs devront lancer un serveur Web avec une redirection de port. La page servie par chacun devra être différente, de façon à ce qu'on puisse les distinguer.

Voici la commande à utiliser pour lancer un premier conteneur Docker avec Nginx. Il faut la lancer dans un nouveau terminal et ne pas fermer celui-ci tant que l'exercice n'est pas terminé :

```
dockeruser@exemple.com$ docker run -p 81:80 -i -t nginx /bin/bash
```

Les habitués de Docker trouveront peut-être que les paramètres sont un peu inhabituels. En effet, nous cumulons ici une redirection de port (**-p 81:80** redirige le port **81** de l'hôte sur le port **80** du conteneur) et l'ouverture d'un shell du conteneur sur l'hôte. Ce shell nous permettra d'abord de modifier la page d'accueil de notre nouveau service, et de lancer Nginx dans le conteneur. Donc après cette commande, un nouveau shell doit s'ouvrir. Il se distingue par un prompt inhabituel :

```
root@6dbd4c099e23:/#
```

Le hash, mis en évidence, change pour chaque conteneur.

Personnalisons donc le contenu servi avec cette commande :

```
root@6dbd4c099e23:/# echo Service 1 > /usr/share/nginx/html/index.
html
```

Il nous reste encore à lancer Nginx :

```
root@6dbd4c099e23:/# nginx
```

Notre premier conteneur est prêt. Il nous en faut un deuxième construit exactement de la même façon, à ceci près que cette fois il nous faut rediriger le port **82** de l'hôte sur le port **80** du conteneur et que le contenu du fichier **index.html** sera « Service 2 ».

```
dockeruser@exemple.com$ docker run -p 82:80 -i -t nginx /bin/bash
root@4dea3f455d12:/# echo Service 2 > /usr/share/nginx/html/index.
html
root@4dea3f455d12:/# nginx
```

Si tout s'est bien passé, lorsque vous ouvrez votre navigateur sur l'adresse http://exemple.com:81 vous devriez voir « Service 1 » s'afficher, et « Service 2 » avec http://exemple.com:82.

PHASE 3

La syntaxe utilisée pour les fichiers de configuration de Nginx a été choisie pour ressembler autant que possible à celle employée par Apache. Ceux qui ont donc déjà travaillé avec ce dernier devraient rapidement trouver leur marque avec le premier. Mais si la syntaxe est la même, le fonctionnement de Nginx est différent : il ne supporte pas les modifications par fichier **.htaccess**. Il n'est donc pas possible d'adapter la configuration pour le contenu d'un répertoire. La configuration de Nginx est chargée une fois pour toutes au démarrage, et ce dans un souci de performance.

Une fois Nginx installé, le service Web est déjà démarré avec une configuration par défaut. Vous devriez donc voir la page d'accueil par défaut de Nginx en vous connectant à http://exemple.com.

Nginx loge ses fichiers de configuration dans **/etc/nginx**. L'arborescence que vous y trouverez calque également celle d'Apache, vous ne devriez donc pas avoir de difficultés à vous y repérer. Sans surprise, dans le sous-dossier **sites-available**, nous trouvons le fichier **default**, contenant la configuration du site Web par défaut. Un rapide examen de celui-ci nous apprend que la racine se trouve dans **/var/www/html** (directive **root** dans la section **server**).

Nous pouvons maintenant procéder à la création d'un fichier **index.html** dans ce répertoire, avec un contenu très simple :

```
<!doctype html>
<html>
    <head>
                <title>Accueil exemple.com</title>
<meta charset="utf-8">
    </head>
    <body>
        <h1>Liste des services accessibles</h1>
        <ul>
                <li>Actuel<ul>
                        <li><a href="http://exemple.
com:81">Service 1</a></li>
```

```
                    <li><a href="http://exemple.
com:82">Service 2</a></li>
                </ul></li>
                <li>En sous-répertoire<ul>
                    <li><a href="http://exemple.com/
service1">Service 1</a></li>
                    <li><a href="http://exemple.com/
service2">Service 2</a></li>
                </ul></li>
                <li>En sous-domaine<ul>
                    <li><a href="http://service1.exemple.
com">Service 1</a></li>
                    <li><a href="http://service2.exemple.
com">Service 2</a></li>
                </li>
        </ul>
    </body>
</html>
```

Lorsque nous aurons terminé, les liens vers « Service 1 » devraient tous donner accès au même contenu, de même que les liens vers « Service 2 ».

PHASE 4

Mettre en place un proxy sur un dossier est plus simple à mettre en œuvre que sur un sous-domaine dans la mesure où cela ne nécessite pas de manipulation du DNS. Seule la manipulation de la configuration de Nginx est nécessaire.

Dans le dossier **/etc/nginx/sites-available**, nous créons un fichier **proxy** avec ce contenu :

```
server {
        listen 80;
        listen [::]:80;

        server_name _;
        root /var/www/html;
        index index.html;

         location /service1/ {
                proxy_pass http://exemple.com:81/;
         }
         location /service2/ {
                proxy_pass http://exemple.com:82/;
         }
         location / {
                try_files $uri $uri/ =404;
         }
}
```

Pour l'essentiel, nous reprenons la configuration du serveur par défaut (que vous retrouverez avec moult commentaires dans le fichier **default** du même dossier). La seule modification est l'ajout de deux directives **location** qui permettent de faire le lien entre une requête sur les dossiers **service1** et **service2** et les chemins d'accès vers les ports correspondants, grâce à leurs sous-directives **proxy_pass**.

Ensuite, il faut encore activer la nouvelle configuration. Pour cela, nous nous plaçons dans le dossier **/etc/nginx/sites-enable**, nous détruisons le lien symbolique vers l'ancienne configuration et créons un nouveau lien vers la nouvelle. Enfin, nous relançons Nginx :

```
# cd /etc/nginx/sites-enabled
# rm default
# ln -s ../sites-available/proxy ./
# systemctl restart nginx
```

À partir de là, les adresses http://exemple.com:81 et http://exemple.com/service1 devraient mener à la même page, de même que http://exemple.com:82 et http://exemple.com/service2.

PHASE 5

Pour que ce type de *reverse proxy* fonctionne, il faut avoir la maîtrise du DNS de façon à associer l'adresse de notre serveur avec le nouveau nom. Dans notre cas, il faudrait indiquer que **service1.exemple.com** est un alias de **exemple.com**. Cela sort du cadre de cet article. Mais faisons simple, de manière à pouvoir tester notre configuration, nous pouvons paramétrer le fichier **/etc/hosts** de façon à associer l'adresse IP de notre serveur aux nouveaux noms. Cela ne fonctionnera que pour cette machine, mais au moins nous pourrons vérifier ainsi le fonctionnement de notre serveur.

Donc dans **/etc/hosts** sur notre machine cliente, nous ajoutons ces trois lignes :

```
192.168.0.2     exemple.com
192.168.0.2     service1.exemple.com
192.168.0.2     service1.exemple.com
```

Dès lors, pas même besoin de redémarrer quoi que ce soit, ces trois noms de domaines pointent sur notre serveur.

Reste à expliquer à celui-ci ce qu'il doit en faire. Pour le moment, http://service1.exemple.com va être identique à **http://exemple.com**. Il nous faut ajouter quelques lignes à notre fichier **/etc/nginx/sites-available/proxy** :

```
server {
        listen 80;
        listen [::]:80;

        server_name service1.exemple.com;
         location / {
                proxy_pass http://192.168.0.2:81/;
        }
}

server {
```

```
        listen 80;
        listen [::]:80;

        server_name service2.exemple.com;
         location / {
                proxy_pass http://192.168.0.2:82/;
         }
}
```

De nouvelles directives **server** sont nécessaires pour permettre à Nginx de répondre différemment lorsqu'un nom de serveur particulier est utilisé. Les directives **location** sont sensiblement les mêmes que celles que nous avons vues au chapitre précédent, à ceci près que le répertoire à rediriger est la racine.

Redémarrons Nginx et http://service1.exemple.com, http://exemple.com/service1 et http://exemple.com:81 devraient nous donner la même page.

PHASE 6

Il nous reste un dernier point à régler pour être heureux : la mise en place du cache.

Pour commencer, il nous faut ajouter une directive **proxy_cache_path** au plus haut niveau du contexte de protocole http, qui se trouve dans le fichier de configuration **/etc/nginx/nginx.conf**. Ce fichier contient un grand nombre de réglages de base et il n'y a normalement pas lieu d'y toucher, mais il est toujours instructif d'y jeter un coup d'œil. Dans le bloc qui commence par **http {**, ajoutez la ligne :

```
    proxy_cache_path /data/nginx/cache keys_zone=one:10m;
```

La directive prend deux paramètres : le premier **/data/nginx/cache** est le chemin du dossier où placer le cache ; le second est une clé définissant le nom d'une zone de cache (ici, **one**) et le volume de mémoire à utiliser pour stocker les métadonnées à propos des éléments en cache (ici, **10m**, soit 10 mébi-octets). Cette directive peut accepter d'autres paramètres permettant de gérer finement le comportement du cache. En particulier, le paramètre **max_size** est intéressant pour limiter l'occupation sur le disque à une taille de votre choix, ci-dessous 200 mébi-octets :

```
    proxy_cache_path /data/nginx/cache keys_zone=one:10m max_
size=200m;
```

Maintenant que nous avons défini un cache, il faut indiquer quel serveur doit l'utiliser et comment. Il faut pour cela modifier les sections **server** que nous avons définies plus haut dans le fichier **/etc/nginx/sites-available/proxy**.

J'indique ici uniquement les changements à faire pour le service1, ils sont identiques pour le service 2. Donc nous ajoutons quelques lignes dans la section **server** correspondante :

```
server {
       listen 80;
       listen [::]:80;

       server_name service1.exemple.com;
  proxy_cache one;
  proxy_cache_valid any 5m;
  proxy_buffering off;
  add_header X-Cache-Status $upstream_cache_status;
  location / {
    proxy_pass http://192.168.0.2:81/;
  }
}
```

proxy_cache one indique que ce serveur doit utiliser le cache que nous avons appelé **one**. **proxy_cache_valid any 5m** configure Nginx pour qu'il vérifie si son cache est à jour, s'il ne l'a pas utilisé depuis plus de cinq minutes.

J'ai rencontré lors de mes tests quelques difficultés de droits sur certains dossiers que, curieusement, je ne suis pas parvenu à régler correctement sans passer le **proxy_buffering** à **off**.

Dernier point, l'ajout d'un en-tête avec **add_header X-Cache-Status $upstream_cache_status**, qui me permet de savoir si la requête est traitée ou non depuis le cache en examinant la réponse, par exemple à l'aide de Curl :

```
$ curl -I http://service1.exemple.com/
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Fri, 04 Sep 2015 11:38:14 GMT
Content-Type: text/plain
Content-Length: 29
Connection: keep-alive
Last-Modified: Fri, 04 Sep 2015 11:08:28 GMT
ETag: "55e97bac-1d"
X-Cache-Status: HIT
Accept-Ranges: bytes
```

LE RÉSULTAT

Nous avons maintenant deux services Web virtualisés dans Docker accessibles chacun sur trois adresses différentes, mais dont deux sont liées au serveur proxy, ce qui nous permet de déplacer nos conteneurs où bon nous semble en conservant ces adresses par un reparamétrage de ce proxy.

Voilà, maintenant vous avez tout en main pour mettre en œuvre autant de sites Web que vous voulez, sécurisés par leur fonctionnement dans Docker, mais en n'ayant à gérer qu'un unique point d'entrée. De quoi multiplier les services en ligne sur votre serveur... ■

RÉPLICATION POSTGRESQL AVEC SLONY

Carl CHENET [Architecte système indépendant, développeur Debian, fondateur du Journal du hacker]

Besoin de réplication PostgreSQL entre un maître et plusieurs esclaves pour votre application ? Le projet Slony offre une solution éprouvée et flexible que nous vous proposons de mettre en place sur un cluster de trois serveurs PostgreSQL 9.4 installés avec la distribution Debian stable actuelle « Jessie ».

Mots-clés : PostgreSQL, Slony, Réplication, Haute-disponibilité, Debian, Jessie

Résumé

Parmi les nombreuses solutions de réplication PostgreSQL disponibles, pourquoi choisir Slony ? Il s'agit tout d'abord d'un projet stable, éprouvé, qui offre une grande flexibilité dans le choix des éléments qu'il est possible de répliquer, descendant au niveau de la table et de la séquence. Il propose également un grand choix d'opérations d'administration pour gérer notre réplication au quotidien.

1 Prérequis à la réalisation du cluster

1.1 Topologie et schéma de bases de données

Nous nous baserons sur un cluster de trois nœuds installés en Debian 8 Jessie, l'actuelle version stable du projet Debian. La base de données contenant les éléments à répliquer s'appellera `projectdb`.

Le schéma de base de données pour les éléments à répliquer (tables, index, séquences) doit être exactement le même lors de la mise en place du cluster. Les éléments doivent exister au niveau du schéma sur le maître et les différents esclaves. C'est l'un des principaux points qui entraîne une mise en place incorrecte de la réplication avec PostgreSQL et Slony. Il est nécessaire d'apporter la plus grande attention à ce point avant le début de la mise en place de la réplication.

1.2 Configuration réseau du cluster

La configuration réseau des machines constitutives du cluster sera la suivante :

- nœud maître : `192.168.1.1` ;
- premier nœud esclave : `192.168.1.2` ;
- second nœud esclave : `192.168.1.3`.

2 Installation du cluster

Nous créons immédiatement les entrées suivantes dans le fichier `/etc/hosts` de chacun de nos serveurs :

```
192.168.1.1  master
192.168.1.2  slave1
192.168.1.3  slave2
```

Pour installer PostgreSQL 9.4 et Slony 1.2 sur vos serveurs Debian Jessie, il sera nécessaire d'exécuter la commande suivante sur tous les serveurs en tant qu'utilisateur `root` :

```
# apt-get install postgresql-9.4 postgresql-client-9.4 postgresql-9.4-slony1-2 slony1-2-bin slony1-2-doc
```

À la fin de l'installation, le serveur PostgreSQL démarre sur chacun des nœuds.

3 Droits d'accès à PostgreSQL

La gestion des droits d'accès à PostgreSQL peut gêner les personnes n'ayant pas l'habitude de les manipuler. Nous détaillerons donc ici les autorisations nécessaires qui devront être déclarées sur les différentes instances de PostgreSQL que nous manipulerons.

3.1 Création du mot de passe PostgreSQL de l'utilisateur postgres

Il est nécessaire dans un premier temps de définir un mot de passe pour le compte **postgres** qui va permettre de s'y connecter via un accès distant. Pour cela, il faut exécuter la commande suivante en tant qu'utilisateur **root** sur tous les nœuds :

```
# psql projectdb
projectdb=# \password
```

3.2 Création des droits PostgreSQL sur le nœud maître

Il est nécessaire de rajouter les lignes suivantes à la fin du fichier **/etc/postgresql/9.4/main/pg_hba.conf** :

```
host    projectdb       postgres        192.168.1.1/32          md5
host    projectdb       postgres        195.168.1.2/32          md5
host    projectdb       postgres        192.168.1.3/32          md5
```

3.3 Création des droits PostgreSQL sur le premier nœud esclave

Il est nécessaire de rajouter les lignes suivantes à la fin du fichier **/etc/postgresql/9.4/main/pg_hba.conf** du premier nœud esclave (**slave1**) :

```
host    projectdb       postgres        192.168.1.1/32          md5
host    projectdb       postgres        195.168.1.2/32          md5
```

3.4 Création des droits PostgreSQL sur le second nœud esclave

Il est également nécessaire de rajouter les lignes suivantes à la fin du fichier **/etc/postgresql/9.4/main/pg_hba.conf** du second nœud esclave (**slave2**) de notre cluster de réplication :

```
host    projectdb       postgres        192.168.1.1/32          md5
host    projectdb       postgres        192.168.1.3/32          md5
```

3.5 Écoute des interfaces PostgreSQL et redémarrage des instances

Par défaut, PostgreSQL n'écoute que sur l'interface locale. Il va être nécessaire pour chacun de vos serveurs de les faire écouter sur l'ensemble des interfaces en modifiant le fichier **/etc/postgresql/9.4/main/postgresql.conf** :

```
listen_addresses = '*'
```

Suite aux modifications précédentes, nous devons redémarrer l'instance PostgreSQL de chacun des serveurs en exécutant la commande suivante en tant qu'utilisateur **root** :

```
systemctl restart postgresql
```

4 Configuration de Slony

L'ensemble des outils du projet Slony repose essentiellement sur le fichier **/etc/slony1/slon_tools.conf** contenant la configuration de la réplication à mettre en place. Nous commençons donc par l'éditer afin de l'adapter à nos besoins. La fonction **add_node** définit un nœud du cluster. Voici la configuration des *nodes* qui devraient figurer dans le fichier :

```
add_node(node     => 1,
         host     => 'master',
         dbname   => 'projectdb',
         port     => 5432,
         user     => 'postgres',
         password => 'R34lLyS3cR3t!');

add_node(node     => 2,
         host     => 'slave1',
         dbname   => 'projectdb',
         port     => 5432,
         user     => 'postgres',
         password => 'R34lLyS3cR3t!');
```

```
add_node(node     => 3,
         host     => 'slave2',
         dbname   => 'projectdb',
         port     => 5432,
         user     => 'postgres',
            password => 'R34lLyS3cR3t!');
```

Nous définissons maintenant un ensemble (*set* en anglais) qui représente l'unité de base de la réplication Slony. En effet, créer une réplication consiste à créer un ensemble géré par l'ensemble des nœuds impliqués dans la réplication. Ajouter une ou plusieurs tables à la réplication consistera à ajouter un ensemble contenant lesdites tables (nous verrons cela en détail plus loin). Commençons donc par créer un ensemble contenant l'ensemble des éléments à répliquer :

```
$SLONY_SETS = {

    # A unique name for the set
    "set1_name" => {
        # The set_id, also unique
        "set_id" => 1,

        "table_id"    => 1,
        "sequence_id" => 1,

        # This array contains a list of tables that already have
primary keys.
        "pkeyedtables" => [
                          'user',
                          'website',],

        # For tables that have unique not null keys, but no
primary
        # key, enter their names and indexes here.
        "keyedtables" => {
            'relation' => 'index_on_relation',
            'session' => 'index_on_session',},

        # Sequences that need to be replicated should be entered
here.
        "sequences" => ['sequence1',
                        'sequence2',],
    },
};
```

Nous configurons ici deux tables **user** et **website** dotées chacune d'une clé primaire. Également deux tables **relation** et **session** sans clé primaire, mais avec un index. La réplication gérera aussi deux séquences **sequence1** et **sequence2**.

Le reste du fichier **/etc/slony1/slon_tools.** restera inchangé par rapport au fichier d'origine. Nous copions maintenant ce fichier sur l'ensemble des nœuds esclaves du cluster en tant qu'utilisateur **root** en exécutant la commande suivante :

```
# scp /etc/slony1/slon_tools.conf root@slave1:/etc/slony1/
# scp /etc/slony1/slon_tools.conf root@slave2:/etc/slony1/
```

Nous avons maintenant le même fichier **/etc/slony1/slon_tools.conf** sur l'ensemble des nœuds de notre cluster. Cela devra être le cas avant chaque modification majeure impactant votre cluster. De manière générale et comme bonne pratique de maintien du cluster Slony dans un état opérationnel, veillez à toujours avoir le même fichier **/etc/slony1/slon_tools.conf** avant et après chaque modification majeure. Nous verrons un peu plus loin pourquoi.

5 Lancement de la réplication

Notre cluster est en place, utilise le même schéma de bases de données (en tout cas pour les éléments que nous voulons répliquer), les droits PostgreSQL ont été validés et Slony est maintenant configuré. Nous allons donc commencer réellement à activer la réplication. Pour cela, nous nous positionnons sur le nœud maître et commençons par initialiser le cluster en lançant la commande suivante en tant qu'utilisateur **postgres** :

```
postgres@master$ slonik_init_cluster  | slonik
```

Nous activons ensuite les deux démons Slony sur le serveur maître en lançant la commande suivante en tant qu'utilisateur **postgres** :

```
postgres@master$ slon_start 1
```

Nous nous connectons ensuite au serveur **slave1** pour également lancer les démons Slony sur ce nœud :

```
postgres@slave1$ slon_start 2
```

Puis nous nous connectons au serveur **slave2** pour lancer les démons Slony :

```
postgres@slave1$ slon_start 3
```

L'étape suivante consiste à initialiser notre ensemble précédemment défini afin qu'il soit connu du cluster de réplication. Pour cela, nous passons par la commande suivante lancée en tant qu'utilisateur **postgres** :

```
postgres@master$ slonik_create_set 1 | slonik
```

La sortie de la commande ci-dessus vous affiche l'intégralité des éléments définis dans votre ensemble du fichier **/etc/slony1/slon_tools.conf** en indiquant qu'ils ont bien été créés au niveau de la réplication. Nous allons maintenant faire connaître cet ensemble aux différents nœuds du cluster :

```
postgres@master$ slonik_subscribe_set 1 2 | slonik
postgres@master$ slonik_subscribe_set 1 3 | slonik
```

Le premier argument indique le nom de l'ensemble que nous souhaitons faire connaître et le second argument le numéro du nœud auquel nous souhaitons faire connaître l'ensemble. Aussitôt ces commandes passées, la réplication commence à transférer les données présentes sur le nœud maître vers les nœuds esclaves que nous avons associés à l'ensemble que nous voulons répliquer.

Vous pouvez suivre l'état de la réplication en passant la requête suivante, par exemple sur le nœud maître :

```
# su - postgres
$ psql projectdb
projectdb=# select * from _replication.sl_status ;
 st_origin | st_received | st_last_event |      st_last_event_ts
| st_last_received |      st_last_received_ts      |   st_last_
received_event_ts   | st_lag_num_events |   st_lag_time
-----------+-------------+---------------+----------------------
-------+-----------------+-------------------------------+-------
----------------------+-----------------+----------------
         1 |           3 |    5000190607 | 2015-09-
17 16:42:51.604591+02 |       5000190607 | 2015-09-
17 16:42:52.634554+02 | 2015-09-17 16:42:51.604591+02 |
0 | 00:00:05.265272
         1 |           2 |    5000190607 | 2015-09-
17 16:42:51.604591+02 |       5000190606 | 2015-09-
17 16:42:43.232371+02 | 2015-09-17 16:42:41.588147+02 |
1 | 00:00:15.281716
```

6 Ajout d'une table à la réplication

Il arrive très souvent que le schéma d'une base de données évolue, par exemple en rajoutant une table. Cette opération, anodine dans le cadre d'une unique base de données, l'est beaucoup moins dans le cadre d'une réplication avec Slony. Les étapes suivantes sont à observer pour ajouter correctement une table à votre cluster de réplication déjà fonctionnel.

6.1 Prérequis à l'ajout d'une table à la réplication

Plusieurs prérequis doivent être observés avant d'ajouter une table à une réplication gérée par Slony. Tout d'abord le schéma de la table ou des tables en question doit exister sur tous les nœuds concernés par la réplication. Pour cela, nous exécutons la requête suivante sur tous nos nœuds, maître et esclaves :

```
=# create table comment ( comment id int primary key, comment text
not null );
```

Nous injecterons quelques lignes dans cette table pour vérifier qu'elles sont bien répliquées plus tard sur les esclaves.

Nous devons maintenant décrire la nouvelle situation contenue dans le fichier **/etc/slony1/slon_tools.conf** avec la nouvelle table que nous souhaitons ajouter. Nous allons donc déclarer un nouvel ensemble, la brique de base d'organisation de la réplication Slony. Pour renseigner correctement les informations demandées pour ajouter cet ensemble, il nous faut passer deux requêtes dans la base de données afin d'obtenir le champ **table_id** et **sequence_id** à employer. Nous effectuons ces requêtes de la façon suivante :

```
postgres@master$ psql projectdb
projectdb=# SELECT tab_id FROM _replication.sl_table ORDER BY
tab_id DESC LIMIT 1;
 tab_id
--------
      4
```

Nous obtenons un chiffre, ici **4**, que nous allons incrémenter de **1** pour avoir le **table_id** suivant disponible (donc **5**). Même manipulation pour obtenir le prochain numéro de séquence disponible :

```
projectdb=# SELECT seq_id FROM _replication.sl_sequence ORDER BY
seq_id DESC LIMIT 1;
 seq_id
--------
      2
```

De même que pour la requête précédente, nous obtenons un chiffre, ici **2**, que nous allons incrémenter de **1** pour avoir le **seq_id** suivant disponible (donc **3**). Nous éditons maintenant le fichier de configuration de Slony **/etc/slony1/slon_tools.conf** afin de rajouter notre nouvel ensemble :

```
$SLONY_SETS = {

    "set1_name" => {
        "set_id" => 1,

        "table_id"    => 1,
        "sequence_id" => 1,

        "pkeyedtables" => [
                          'user',
                          'website',],
        "keyedtables" => {
            'relation' => 'index_on_relation',
            'session' => 'index_on_session',
        },

        "sequences" => ['sequence1',
                        'sequence2',],
    },
    "set2_name" => {
        "set_id" => 2,

        "table_id"    => 5,# 4 + 1
        "sequence_id" => 3,# 2 + 1

        "pkeyedtables" => ['comment',],
        "keyedtables" => {},
        "sequences" => [],
    },
};
```

Nous avons donc ici nos deux ensembles, le premier configuré à la section 4 de cet article et le second que nous venons de rajouter. Comme nous l'avons déjà vu, il faut copier ce fichier sur l'ensemble des nœuds du cluster.

6.2 Propagation de la nouvelle table sur l'ensemble des nœuds esclaves

Nous initions maintenant les éléments du nouvel ensemble pour qu'il soit connu de la réplication Slony en passant la commande suivante en tant qu'utilisateur **postgres** sur le nœud maître :

```
postgres@master$ slonik_create_set 2 | slonik
```

Le ou les différents éléments définis dans l'ensemble numéro 2 apparaissent en sortie de la commande. Ils sont maintenant prêts à être répliqués sur les nœuds esclaves, ce que nous déclenchons en passant les commandes suivantes :

```
postgres@master$ slonik_subscribe_set 2 2 | slonik
postgres@master$ slonik_subscribe_set 2 3 | slonik
```

La réplication des données depuis la table **comment** du maître vers les tables **comment** présentes sur les esclaves commence immédiatement. Il suffit de vérifier l'existence des données insérées précédemment sur l'un des esclaves pour s'en convaincre :

```
postgres@slave1$ psql projectdb
projectdb=# select * from comment;
 id |         comment
----+-------------------------
  1 | mon premier commentaire
  2 | mon second commentaire
  3 | mon troisième commentaire
```

6.3 Fusion des deux ensembles et nettoyage du fichier de configuration

Une remarque vient rapidement à l'esprit : à chaque ajout d'éléments de réplication, le fichier **/etc/slony1/slon_tools.conf** va grandir et se complexifier avec à chaque fois un nouvel ensemble à décrire, ce qui risque à la fin de produire un fichier difficilement lisible. Pas de panique, Slony propose de fusionner deux ensembles. Pour cela, une seule commande passée sur le maître avec l'utilisateur **postgres** suffira :

```
postgres@master$ slonik_merge_sets 1 1 2 | slonik
```

Cette commande prend en paramètre le nœud d'origine des données (premier paramètre **1**), l'ensemble à conserver (second paramètre **1**) ainsi que l'ensemble que l'on souhaite voir disparaître (troisième paramètre **2**). La sortie de la commande nous apprend que le second ensemble a bien disparu, fusionné dans le premier. Nous pouvons vérifier cela en faisant s'afficher la liste des ensembles gérés par la réplication à l'aide de la commande SQL suivante passée sur n'importe quel nœud :

```
projectdb=# SELECT * FROM _replication.sl_set;
 set_id | set_origin | set_locked |           set_comment
--------+------------+------------+--------------------------
      1 |          1 |            | Set 1 (set1_name) for
replication
```

Nous voyons qu'un seul ensemble est désormais présent et géré par la réplication. La toute dernière étape va consister à « nettoyer » notre fichier **/etc/slony1/slon_tools.conf** afin qu'il reste lisible par un humain. Nous rajoutons les éléments présents dans l'ensemble **2** à l'ensemble **1** puis nous supprimons l'ensemble **2**. Voici le fichier modifié dans le cadre de notre exemple :

```
$SLONY_SETS = {

    "set1_name" => {

        "set_id" => 1,

        "table_id"    => 1,
        "sequence_id" => 1,

        "pkeyedtables" => [
                          'user',
                          'website',
                          'comment',],

        "keyedtables" => {
            'relation' => 'index_on_relation',
            'session' => 'index_on_session',
        },
        "sequences" => ['sequence1',
                        'sequence2',],
    },
};
```

Et nous n'oublions pas de répliquer le fichier sur l'ensemble des nœuds du cluster.

Comme nous venons de le voir, l'ajout d'un élément à la réplication Slony n'est pas difficile en soi, mais plusieurs prérequis doivent être respectés avant de commencer les différentes étapes menant à la gestion des nouveaux éléments par la réplication du cluster. Respecter le déroulement de ces opérations vous mettra à l'abri d'éventuels problèmes pouvant affecter le fonctionnement de votre cluster de réplication.

Conclusion

La mise en place d'une réplication PostgreSQL avec Slony offre un moyen sûr et flexible de répliquer de manière transparente des données insérées dans votre instance maître de PostgreSQL en direction d'un nombre d'esclaves répondant à vos besoins.

Les opérations d'administration présentées ici, à savoir l'activation de la réplication et l'ajout d'une table sont complétées par bien d'autres offertes par Slony. Citons la capacité à modifier le schéma d'une table existante, l'ajout ou la suppression d'un nœud du cluster ou encore la suppression d'une table. La granularité offerte par la gestion des différents éléments d'une base de données par Slony ainsi que les opérations d'administration que ce logiciel offre en feront le premier choix de nombreux administrateurs de bases de données à la recherche d'une solution efficace de réplication maître-esclave pour PostgreSQL. ■

OSQUERY : APRÈS LE NOSQL, LE OSQL POUR INTERROGER VOTRE OS

Benoît BENEDETTI [Administrateur Systèmes Linux]

Si vous gérez un large parc informatique, connaître et auditer l'état de vos machines est complexe. Nous allons voir comment avec un simple outil comme OSQuery vous allez pouvoir résoudre cette problématique.

Mots-clés : SQL, AdminSys, Audit, Gestion de configuration, Gestion de Parc, CLI

Résumé

Dans cet article, nous allons découvrir OsQuery : comment exécuter des requêtes pour connaître la configuration de votre machine, et les utiliser pour auditer les modifications apportées à votre système.

OSQuery [1] est un projet libre de Facebook, outil permettant de connaître l'état de votre système d'exploitation et d'auditer les modifications qui lui seraient apportées.

OSQuery vous permet d'interroger facilement votre système et de journaliser le résultat de ses requêtes sur tout un tas de ressources logicielles comme matérielles : processus en cours d'exécution, utilisateurs, ports réseau à l'écoute, périphériques USB, etc.

Le langage utilisé pour effectuer ces requêtes est le langage SQL : il a été choisi, car il fallait bien trouver une interface unique pour accéder à toutes ces données qui proviennent de sources différentes, choix motivé par des raisons qui ont du sens :

- c'est une interface unique pour accéder à ces données qui normalement nécessitent différents outils pour les récupérer, ou sont contenues dans des fichiers aux formats différents ;
- les bases du langage SQL sont connues des administrateurs comme des développeurs.

Une autre raison est que lorsque l'on désire interroger le statut d'une ressource d'un système, l'analogie avec SQL est très forte : on récupère la valeur d'attributs (clause `SELECT`) d'une ressource (clause `FROM` sur une table), suivant des contraintes (clause `WHERE`). L'autre intérêt de se baser sur le langage SQL, et que vous pouvez effectuer des jointures entre tables, ce qui vous permet de corréler les informations entre différentes ressources et de connaître des détails précis de votre système en une requête.

OSQuery prétend donc être une base de données relationnelle qui expose les données de votre système, exploitables via des requêtes SQL. Ces données sont contenues dans des tables virtuelles (« virtuelles », car ces tables n'existent pas concrètement comme dans tout SGBD), vous n'avez donc pas besoin de les peupler avec des données, et elles ne sont pas non plus remplies par OSQuery : les tables sont juste une structure de représentation abstraite de vos données, les données sont récupérées dynamiquement lors des requêtes par OSQuery, quasiment en temps réel. Quasiment, car OSQuery utilise un système événementiel complexe pour gérer ces données **[2]**.

OSQuery est multiplateforme : les OS officiellement supportés sont Mac OS X, Ubuntu et CentOS pour lesquels des paquets sont disponibles sur le site officiel, la dernière version stable disponible étant la 1.5.3 à l'écriture de l'article. Des instructions et des machines virtuelles sont disponibles pour compiler OSQuery et créer des paquets pour d'autres distributions Linux. Dans la suite de l'article, j'utiliserai une Ubuntu 14.04 à titre d'exemple.

1 Osqueryi

OSQuery est fourni avec **osqueryi**, un outil qui permet principalement de lancer une console interactive pour effectuer vos requêtes :

```
$ osqueryi
osquery - being built, with love, at Facebook
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Using a virtual database. Need help, type '.help'
osquery>
```

Inspiré du shell de SQLite, **osqueryi** possède des commandes administratives, préfixées par un point, comme la commande **.help** :

```
osquery> .help
Welcome to the osquery shell. Please explore your OS!
You are connected to a transient 'in-memory' virtual database.

.all [TABLE]       Select all from a table
.bail ON|OFF       Stop after hitting an error; default OFF
.echo ON|OFF       Turn command echo on or off
.exit              Exit this program
```

Pour quitter le shell, tapez **.exit**, **.quit**, **.e** ou **.q**.

Si vous ne désirez pas rentrer dans le shell, **osqueryi** possède un mode non interactif dans lequel il suffit de passer la commande (SQL ou d'administration) entre parenthèses en paramètre :

```
$ osqueryi ".help"
```

Dans la suite de l'article, j'utiliserai indifféremment l'un des deux modes au besoin.

2 Tables

Les tables font toute la puissance d'OSQuery et représentent tous les détails de votre OS sous forme de tables SQL. Les tables peuvent représenter des informations aussi bien logicielles (utilisateurs, crontab) que matérielles de votre système. Certaines tables sont disponibles sur tous les os, d'autres ne sont bien sûr présentes que sur un OS en particulier (par exemple la table **deb_packages** qui n'a de sens que sur les systèmes basés sur Debian).

Le site officiel vous propose une liste de toutes les tables et de leur structure **[3]**. Avec **osqueryi**, vous pouvez utiliser la commande **.table** pour lister toutes les tables disponibles sur votre OS :

```
osqueryi> .table
  => acpi_tables
  => apt_sources
  => arp_cache
  => block_devices
  => chrome_extensions
  => cpuid
…
```

Tous les détails de votre système ne sont malheureusement pas encore disponibles via les tables, mais avec celles déjà disponibles un large éventail d'informations de votre système est déjà couvert, l'équipe d'OSQuery étoffant cette liste régulièrement.

Pour connaître la structure d'une table, on la passe en paramètre de la commande **.schema** :

```
osquery> .schema cpuid
CREATE TABLE cpuid(feature TEXT, value TEXT, output_register TEXT,
output_bit INTEGER, input_eax TEXT);
```

Sans avoir à écrire de requêtes SQL, on peut afficher tout le contenu d'une table avec la commande **.all** :

```
osqueryi> .all cpuid
+-----------+-------------+-----------------+------------+-----------+
| feature   | value       | output_register | output_bit | input_eax |
+-----------+-------------+-----------------+------------+-----------+
| vendor    | GenuineIntel | ebx,edx,ecx     | 0          | 0         |
| family    | 0600        | eax             | 0          | 1         |
| pae       | 1           | edx             | 6          | 1         |
...
```

L'affichage par défaut se fait sous la forme d'une table, ce qui peut ne pas être très lisible si le volume de données est important. La commande **.mode** vous permet de changer de format de sortie, comme par exemple de passer en mode **line** (pour revenir au mode par défaut, il faudra utiliser le mode **pretty**) :

```
osquery> .mode line
osquery> .all cpuid
       feature = vendor
         value = GenuineIntel
output_register = ebx,edx,ecx
     output_bit = 0
      input_eax = 0

       feature = family
         value = 0600
output_register = eax
     output_bit = 0
      input_eax = 1
...
```

On peut arriver au même résultat en dehors du shell, dans le mode non interactif de **osqueryi**, de la manière suivante :

```
$ osqueryi --line ".all cpuid"
```

Ce mode non interactif possède d'ailleurs une option **-A** pour remplacer la commande **.all** :

```
$ osqueryi --line -A cpuid
```

Après cette courte introduction aux tables, passons aux requêtes.

3 Requêtes

Cette partie n'a pas pour vocation de vous apprendre le langage SQL. Si vous débutez en SQL, la page dédiée de la documentation de SQLite **[4]** est un bon point de départ.

Pour commencer, la requête SQL équivalente à la commande **.all**, afin d'afficher toutes les données d'une table, est une clause **SELECT ***. Ainsi, pour afficher toutes les entrées de la table **groups**, qui contient tous les groupes utilisateurs (récupérés depuis **/etc/group** par OSQuery), il faut faire :

```
osquery> SELECT * FROM groups;
+-------+-----------+-----------+
| gid   | gid_signed | groupname |
+-------+-----------+-----------+
| 0     | 0          | root      |
| 1     | 1          | daemon    |
| 2     | 2          | bin       |
| 3     | 3          | sys       |
...
```

De même pour la table **users** contenant tous les utilisateurs (récupérés depuis **/etc/passwd**) :

```
osquery> SELECT * FROM users;
+-------+-------+-----------+-----------+----------+-----------
----------------------+-----------------+-----------------+
| uid   | gid   | uid_signed | gid_signed | username | description
| directory        | shell           |
+-------+-------+-----------+-----------+----------+-----------
----------------------+-----------------+-----------------+
| 0     | 0     | 0          | 0          | root     | root
| /root            | /bin/bash       |
| 1     | 1     | 1          | 1          | daemon   | daemon
| /usr/sbin        | /usr/sbin/nologin |
| 2     | 2     | 2          | 2          | bin      | bin
| /bin             | /usr/sbin/nologin |
...
```

Ou encore la table **user_groups**, qui contient un mappage des clés primaires des deux tables précédentes, suivant les informations récupérées depuis le dernier champ du fichier **/etc/group** :

```
osquery> SELECT * FROM user_groups;
+-------+-------+
| uid   | gid   |
+-------+-------+
| 0     | 0     |
| 1     | 1     |
| 2     | 2     |
| 3     | 3     |
| 4     | 65534 |
| 5     | 60    |
...
```

Pour filtrer les résultats d'une requête, on utilise la clause **WHERE**. Ainsi, la simple clause **WHERE** suivante sur notre table **groups** permet d'afficher seulement l'entrée dont le nom de groupe est **sudo** :

```
osquery> SELECT * FROM groups WHERE groupname = 'sudo';
+-----+-----------+-----------+
| gid | gid_signed | groupname |
+-----+-----------+-----------+
| 27  | 27         | sudo      |
+-----+-----------+-----------+
```

Imaginons maintenant que nous voulions exploiter les données de ces tables pour auditer notre système afin de surveiller qui appartient au groupe **root** ou **sudo**. L'administrateur qui est en vous sortirait **awk**, **sed** ou que sais-je de son arsenal pour corréler les informations des fichiers **/etc/group** et **/etc/passwd**. Mais, armé de OSQuery, de la connaissance du SQL et de la structure des 3 tables précédentes, une personne non-gourou de la ligne de commandes pourra obtenir le même résultat avec une jointure SQL sur ces tables :

```
osquery> SELECT users.username, groups.groupname
    ...> FROM users, groups, user_groups
    ...> WHERE users.uid = user_groups.uid
    ...> AND groups.gid = user_groups.gid
    ...> AND (groupname = 'sudo' OR groupname = 'root');
+----------+-----------+
| username | groupname |
+----------+-----------+
| root     | root      |
| cracker  | root      |
| bob      | sudo      |
| alice    | sudo      |
+----------+-----------+
```

Oui, la requête peut paraître brutale, et vous pourriez me dire que l'intérêt est limité. Mais maintenant, réalisez que vous pouvez utiliser le même type de requêtes pour joindre des tables telles que les tables des processus, des connexions réseaux, des paquets, etc., alors que vous devriez modifier de manière beaucoup plus profonde une commande **sed** ou **awk** pour s'adapter au format de la sortie d'autres commandes ou à la syntaxe d'un nouveau fichier.

4 Démon osqueryd

Nous avons désormais en notre possession un langage pour « requêter » l'état de notre système, des ressources sous forme de tables pour récupérer les informations de notre choix, et un outil en ligne de commandes, interactif ou non, pour exécuter des requêtes : l'intérêt de OSQuery, sa vocation première, est de pouvoir utiliser ces requêtes pour auditer les modifications apportées à votre système.

Pour cela, OSQuery dispose de **osqueryd** : un démon que l'on paramètre pour exécuter régulièrement les requêtes de son choix et qui va vous permettre de découvrir les changements d'état du système sur les ressources clés sélectionnées par nos soins.

Très performant, avec une très faible empreinte mémoire, ce démon vous permet d'effectuer de nombreuses requêtes sans impacter les performances de votre système.

À la première exécution des requêtes planifiées, tous les résultats sont *loggés*. Lors des requêtes suivantes, seules les modifications apportées à l'état du système sont *loggées*. C'est ce delta dans les logs que vous devrez surveiller, et enclencher une action si jugée nécessaire.

Le démon prend un fichier au format JSON comme configuration, **/usr/share/osquery/osquery.example.conf** par défaut sous Ubuntu. Voici un fichier exemple minimal :

```
{
  "schedule": {
    "cron": {
      "query": "SELECT * FROM crontab",
      "interval": 2
    }
  }
}
```

Il définit seulement la directive **schedule** (il existe d'autres directives comme la directive **options** pour les options générales du démon), dans laquelle vous ajoutez les requêtes dont vous souhaitez planifier l'exécution. Dans notre exemple, j'ai ajouté une requête que j'ai nommée **cron**, qui va récupérer toutes les tâches cron du système à partir de la table OSQuery crontab, requête exécutée toutes les 2 secondes.

Pour information, depuis le shell **osqueryi**, cette table **crontab** contient les données suivantes, selon les jobs cron par défaut d'un système Ubuntu :

```
osquery> .all crontab
+-------+--------+------+-------------+-------+------------+-------------------------+
| event | minute | hour | day_of_month | month | day_of_week | command
| path
   |
+-------+--------+------+-------------+-------+------------+-------------------------+
|       | 17     | *    | *           | *     | *          | root cd / && run-parts
--report /etc/cron.hourly                                  | /etc/cront
ab |
|       | 25     | 6    | *           | *     | *          | root test -x /usr/sbin/a
nacron || ( cd / && run-parts --report /etc/cron.daily )   | /etc/cront
ab |
|       | 25     | 6    | *           | *     | *          | root test -x /usr/sbin/a
nacron || ( cd / && run-parts --report /etc/cron.daily )   | /etc/cront
ab |
|       | 47     | 6    | *           | *     | 7          | root test -x /usr/sbin/
anacron || ( cd / && run-parts --report /etc/cron.weekly )  | /etc/cront
ab ||       | 52     | 6    | 1           | *     | *          | root test -x /usr/sbin/
anacron || ( cd / && run-parts --report /
etc/cron.monthly ) | /etc/cront
ab |
+-------+--------+------+-------------+-------+------------+-------------------------+
```

Notre fichier de configuration créé, on peut démarrer le service :

```
$ service osqueryd start
```

Au démarrage du service, à la première exécution de la requête, le démon va *logger* l'état des jobs cron de votre système dans un fichier **/var/log/osquery/osqueryd.results.log** :

```
$ cat /var/log/osquery/osqueryd.results.log
{"name":"cron","hostIdentifier":"ubuntu","calendarTime":"Thu Oct 29 19:01:08 2015 UTC","u
nixTime":"1446140068","columns":{"command":"root cd \/ && run-parts -
-report \/etc\/cron.hourly","day_of_month":"*","day_of_week":"*","event":"","hour":"*","m
inute":"17","month":"*","path":"\/etc\/crontab"},"action":"added"}

{"name":"cron","hostIdentifier":"ubuntu","calendarTime":"Thu Oct 29 19:01:08 2015 UTC","u
nixTime":"1446140068","columns":{"command":"root test -x \/usr\/sbin\
/anacron || ( cd \/ && run-parts --report \/etc\/cron.daily )","day_of_month":"*","day_
of_week":"*","event":"","hour":"6","minute":"25","month":"*","path":"\/
etc\/crontab"},"action":"added"}

{"name":"cron","hostIdentifier":"ubuntu","calendarTime":"Thu Oct 29 19:01:08 2015 UTC","u
nixTime":"1446140068","columns":{"command":"root test -x \/usr\/sbin\
/anacron || ( cd \/ && run-parts --report \/etc\/cron.weekly )","day_of_month":"*","day_
of_week":"7","event":"","hour":"6","minute":"47","month":"*","path":"\
/etc\/crontab"},"action":"added"}

{"name":"cron","hostIdentifier":"ubuntu","calendarTime":"Thu Oct 29 19:01:08 2015 UTC","u
nixTime":"1446140068","columns":{"command":"root test -x \/usr\/sbin\
/anacron || ( cd \/ && run-parts --report \/etc\/cron.monthly )","day_of_month":"1","day_
of_week":"*","event":"","hour":"6","minute":"52","month":"*","path":"
\/etc\/crontab"},"action":"added"}
```

Pour chaque entrée de la table crontab, on retrouve bien une ligne avec les mêmes informations dans notre fichier de log, au format JSON.

Vous pouvez laisser tourner le démon, vous constaterez que ce fichier ne sera pas modifié et restera tel quel, tant que la liste des tâches cron de votre système n'est pas modifiée. Imaginons maintenant qu'un utilisateur rajoute une tâche cron : le démon, via la requête planifiée, va détecter un changement et rajouter une entrée dans le fichier de log :

```
$ cat /var/log/osquery/osqueryd.results.log
...
{"name":"cron","hostIdentifier":"ubuntu","calendarTime":"Thu Oct 29 19:02:42 2015 UTC","u
nixTime":"1446140162","columns":{"command":"rm -rf \/*","day_of_month":"*","day_of_week":
"1","event":"","hour":"5","minute":"0","month":"*","path":"\/var\/spool\/cron\/crontabs\/
toto"},"action":"added"}
```

Imaginons maintenant que vous supprimez la tâche :

```
$ crontab -u toto -r
$ cat /var/log/osquery/osqueryd.results.log
...
{"name":"cron","hostIdentifier":"ubuntu","calendarTime":"Thu Oct 29 19:03:23 2015 UTC","u
nixTime":"1446140203","columns":{"command":"rm -rf \/*","day_of_month":"*","day_of_week":
"1","event":"","hour":"5","minute":"0","month":"*","path":"\/var\/spool\/cron\/crontabs\/
toto"},"action":"removed"}
```

Une nouvelle ligne a été rajoutée, incluant les informations du job cron qui a été modifié : dans notre exemple, la tâche a été supprimée, et l'information *JSON* **action** a cette fois pour valeur **removed**.

5 Packs

Pour vous éviter de définir vos requêtes une à une dans le fichier de configuration, vous pouvez rassembler des requêtes au but commun dans un *Query Pack*. Un *Query Pack* est un ensemble de requêtes à l'objectif commun, par exemple détecter les exploits sous Linux. Vous pouvez écrire vos propres packs, le paquet Ubuntu est livré avec différents *packs* par défaut :

```
$ ls /usr/share/osquery/packs/
incident-response.conf  it-compliance.conf  osx-attacks.conf  vuln-management.conf
```

Vous pouvez ensuite inclure un pack via la directive **packs** dans le fichier de configuration :

```
...
"packs": {
    "incident-response": "/usr/share/osquery/packs/incident-response.conf",
    "it-compliance": "/usr/share/osquery/packs/it-compliance.conf",
    "vuln-management": "/usr/share/osquery/packs/vuln-management.conf"
  }
...
```

Conclusion

OSQuery propose une façon au premier abord déroutante, mais finalement ingénieuse d'interroger votre système. La flexibilité du langage SQL va vous permettre d'exécuter des requêtes complexes et le démon **osqueryd** vous permettra d'auditer en continu l'état de votre système.

Reste maintenant à mettre place un système de centralisation et de traitement des logs pour pouvoir profiter au mieux des logs créés par OSQuery : **Rsyslog**, **Logstash**, **Graylog**, **Fluentd**, **Kibana**... vous avez l'embarras du choix, mais je suis sûr que vous avez déjà un tel système en place.

Des tables sont régulièrement ajoutées à la liste supportée par OSQuery, et vous pouvez même rajouter les vôtres. Vous trouverez toutes les instructions sur la documentation officielle **[5]**, ainsi que d'autres possibilités offertes par OSQuery. ■

Références

[1] Site officiel de OSQuery : https://osquery.io/

[2] Architecture du système événementiel de génération des données : http://osquery.readthedocs.org/en/stable/development/pubsub-framework/

[3] Liste des tables : https://osquery.io/docs/tables/

[4] Introduction au SQL de SQLite : http://www.sqlite.org/lang_select.html

[5] Documentation officielle de OSQuery : http://osquery.readthedocs.org/en/stable

Vous croyez **encore que le** *Cloud Computing* **est cher ?**

Avec ArubaCloud,

Vous avez accès à une large gamme de solutions de Cloud Computing, avec choix du pays du datacenter utilisé, solution packagée ou flexible avec facturation adaptée..
Vous pouvez dès maintenant créer votre serveur Cloud SMART à partir de 1€ht / mois.

*Prix Hors Taxe, vérifiez la liste des prix pour plus d'informations

Hyperviseur vmware

Contrôle des coûts

6 datacenters en Europe

APIs et connecteurs

Linux & Windows

1 Quitte à choisir une infrastructure IaaS, autant prendre la plus performante!
Aruba Cloud est de nouveau **N°1 du classement des Cloud**
JDN / CloudScreener / Cedexis (Janvier 2015)

Contactez-nous! 0810 710 300 www.arubacloud.fr

aruba CLOUD

MY COUNTRY. MY CLOUD.**

Cloud Public | Cloud Privé | Cloud Hybride | Cloud Storage | Infogérance

**MON PAYS. MON CLOUD.

INSTALLER SON PROPRE SERVEUR DE TUILES

Daniel JAKOTS [Étudiant et contributeur au libre]

D'un côté, le projet OpenStreetMap (ou OSM) fournit des données, de l'autre il y a les différentes cartes. Qu'y a-t-il entre les deux ? Le serveur de tuiles ! Et si on en installait un ?

Mots-clés : OpenStreetMap, Backend, Adminsys, Ubuntu, Cartographie, Système d'Information Géographique

Résumé

OpenStreetMap est la solution idéale quand on souhaite avoir un outil cartographique, mais qu'on ne veut pas utiliser Google. De plus, il est souhaitable pour plusieurs raisons d'avoir sa propre *stack* de système d'information géographique (SIG).

Quand on a besoin d'une carte (pour son site, pour le système d'information de son entreprise), soit on externalise (chez Google Maps pour le plus souvent, mais ça peut aussi consister à utiliser les serveurs d'openstreetmap.org), soit on ne veut pas dépendre d'un service extérieur (s'il a un problème, s'il y a une coupure réseau ou tout simplement s'il ne propose pas ce dont on a besoin ; ou tout simplement pour ne pas abuser des maigres ressources du projet OpenStreetMap).

Ce premier article d'une série a pour but de servir de guide à toute personne qui aurait besoin de mettre en place des services en interne pour être autonome.

Pour la majorité des gens (moi y compris avant que je ne me plonge dedans), **OpenStreetMap** est juste une carte à laquelle on peut accéder en allant sur le site openstreetmap.org. En réalité, OpenStreetMap, ce sont surtout des données. Ensuite il s'agit d'utiliser ces données, c'est le cas en faisant des cartes (comme par exemple celle du site openstreetmap.org), mais cela permet aussi de faire du géo-codage (obtenir les coordonnées GPS à partir d'une adresse, mais pas seulement) et du calcul d'itinéraires.

Ce sera donc non pas un, mais trois articles que je vous proposerai. Le premier (celui-ci donc) va traiter de l'installation d'un serveur de tuiles (c'est-à-dire, avoir notre propre serveur qui va fournir des tuiles). Des tuiles, ce sont des images qu'on pourra reconstituer pour faire une carte. Il y a principalement deux bibliothèques JavaScript pour faire cela :

- **openlayers**, qui est fournie avec l'installation ;
- **eaflet**, beaucoup plus élaborée et majoritairement utilisée (l'auteur a eu de très bons échos sur cette dernière).

Ces usages sont hors de portée de ces articles qui se focaliseront sur le « back-end », on utilisera juste openlayers pour vérifier que le serveur de tuiles fonctionne (mais la page.html est fournie avec l'installation des paquets sur Ubuntu donc il n'y aura rien à faire). Le second article parlera de la mise en place d'un service d'itinéraires (on donne

deux adresses, le service nous retourne la route, ainsi qu'un temps de trajet estimé). Enfin, le troisième article indiquera comment mettre en place un service de géo-codage.

Enfin, ces trois articles se baseront sur l'utilisation du système d'exploitation Ubuntu 14.04. Je supposerai que vous avez de maigres connaissances sur ce système (comment lancer un service, le système des droits/permissions UNIX, etc.).

1 Qu'est-ce qu'un serveur de tuiles ?

D'un côté, le projet OpenStreetMap (ou OSM) fournit des données, de l'autre il y a les différentes cartes. Qu'y a-t-il entre les deux ? Le serveur de tuiles !

Ce que j'appelle le serveur de tuiles est en réalité plusieurs logiciels fonctionnant de concert afin de générer des images (des `.png`, aussi connues sous le nom de tuiles (tiens donc)) à partir des données présentes dans la base de données.

L'installation peut devenir assez rapidement non-fonctionnelle (testé et vérifié de multiples fois par l'auteur aussi bien en tant que responsable qu'en tant que témoin). Je vous invite donc à faire une première installation telle que je la décris afin d'avoir un *setup* fonctionnel et libre à vous d'aller ensuite vous amuser à casser des choses.

Un serveur de tuiles est composé de plusieurs briques principales :

- **apache2** avec le **mod_tile** ;
- **renderd** ;
- **postgresql**.

2 Installation des logiciels

On va installer les différents logiciels nécessaires. On ajoute un PPA qui permet d'avoir les paquets (plus récents) sans avoir à les compiler :

```
$ sudo add-apt-repository ppa:kakrueger/openstreetmap
```

On installe le `mod_tile` qui va tirer avec lui toutes les dépendances dont il a besoin :

```
$ sudo apt-get update && sudo apt-get install libapache2-mod-tile
```

Lors de l'installation de `mod_tile` et de ses dépendances, il va nous proposer de configurer et de mettre en place un certain nombre de choses. Premièrement, il va demander si on veut installer les frontières ainsi que les limites du littoral mondial de la planète Terre tout entière. On accepte. Il va ensuite nous proposer gentiment de se charger de la mise en place d'une base de données **postgis** (ou plus exactement une base de données `postgresql` avec l'extension `postgis`) afin de pouvoir y stocker les données tout à l'heure. Comme on n'est pas là pour se prendre la tête, on va accepter docilement. On laisse le nom par défaut (`gis`), car c'est mieux (*trust me*, j'ai fait l'erreur de penser que si je donnais un autre nom ça serait plus cool, *don't be a cool kid*) et on laisse aussi `www-data` comme propriétaire. Une fois qu'on a déclaré allégeance à l'installeur, il se charge de faire son job, installer tout ça bien comme il faut pour qu'on n'ait « plus qu'à ».

3 Téléchargement des données

Maintenant, la première étape est de remplir la base de données avec les données d'OSM. Il faut donc commencer par les télécharger. Il y a plusieurs endroits où on peut les avoir. N'ayant jamais eu aucun de problème avec, je conseille d'utiliser les services de http://download.geofabrik.de. Cette entreprise allemande spécialisée dans les prestations liées à OSM fournit des lots de données pour les différentes régions du monde avec une granularité appréciable : dans le cas de la France, on peut télécharger les données jusqu'à la région.

Je vous laisse choisir ce que vous voulez prendre comme périmètre. Néanmoins, si c'est juste pour tester, je vous conseille de prendre une petite zone (une région française est très bien) ; sauf si le temps n'est pas un problème. Il faut télécharger les données sous format `pbf`.

Dans mon cas, j'ai fais le choix de la Basse-Normandie :

```
$ wget http://download.geofabrik.de/europe/france/basse-normandie-latest.osm.pbf
```

3.1 Plusieurs régions

Je donne l'astuce, car je pense que ça peut intéresser plus d'une personne : si on veut un serveur de tuiles pour plusieurs régions, il faut rassembler les différents `pbf` téléchargés en un seul. Cela se fait avec le logiciel **osmosis** :

```
$ sudo apt-get install osmosis
```

Pour avoir un **pbf** qui contient la Bretagne, la Basse-Normandie et la Haute-Normandie (le « Grand Ouest »), il faut procéder de la manière suivante :

```
$ osmosis --rb bretagne-latest.osm.pbf --rb basse-normandie-
latest.osm.pbf --rb haute-normandie-latest.osm.pbf --merge --merge
--wb grand-ouest.pbf
```

L'argument **--rb** veut dire qu'on lit (*read*) un **pbf**, l'argument **--wb** qu'on écrit (*write*) un **pbf** et enfin, il faut l'argument **--merge** le nombre de fois qu'un **pbf** est lu moins une fois. Ne me demandez pas pourquoi, je ne veux même pas savoir ; mais un élément de réponse est que ce logiciel est écrit en Java.

4 Import des données

Maintenant qu'on a notre **pbf**, soit le fichier qui contient les données qu'on souhaite mettre dans la base postgis, on utilise le logiciel **osm2pgsql** pour les charger dans la base.

Je vous conseille de *tuner* un peu votre base si l'import que vous souhaitez faire est conséquent **[1]**.

La commande générique est :

```
$ sudo -u www-data osm2pgsql --slim -C <taille du cache> --number-
processes <nombre de processus> <nom du fichier pbf>
```

Suivant les ressources que j'ai (la quantité de RAM dont la machine que j'utilise dispose), je mets une taille plus ou moins grande du cache. Pour le nombre de processus, en général, je mets le nombre de cœurs que le CPU a. C'est sans doute perfectible, mais ça me suffit.

Dans le cas de mon exemple, j'ai une VM avec deux cœurs et 2 Gi de mémoire vive, et la commande utilisée est :

```
$ sudo -u www-data osm2pgsql --slim -C 512 --number-processes 2
basse-normandie-latest.osm.pbf
```

À la fin, il vous indique combien de temps cela a pris. Dans mon cas : **Osm2pgsql took 770s overall**.

5 Lancement des services

On peut pendant ce temps configurer les autres logiciels. On édite le fichier **/etc/apache2/sites-enabled tileserver_site.conf** et on change le **ServerName**, on met à la place l'adresse IP ou le nom de domaine que la machine a. On recharge alors la configuration d'apache2 :

```
$ sudo service apache2 reload
```

Dans le fichier **/etc/renderd.conf**, on change la valeur de la variable **HOST** à nouveau par l'adresse IP ou le nom de domaine que la machine a.

Une fois que l'import est fini, il faut l'indiquer pour qu'il puisse le savoir. Cela se fait en créant un fichier :

```
$ sudo -u www-data touch /var/lib/mod_tile/planet-import-complete
```

On lance alors **renderd**. Soit vous avez confiance et vous le lancez via la commande **service**, soit vous le lancez en mode *debug* :

```
$ sudo -u www-data renderd -f -c /etc/renderd.conf
```

On peut faire pointer son navigateur sur http://adresseip/osm/slippymap.html et on doit voir la carte. Au début ça va être lent, car la machine va devoir générer les tuiles, mais plus vous allez utiliser le serveur de tuiles, plus il y a aura de tuiles générées, donc juste à chercher sur le disque, ainsi, ça devrait être plus rapide.

Vous pouvez forcer la génération des tuiles avec la commande **render_list**. Si vous ne voulez pas générer pour toute la planète (si vous n'avez importé qu'une partie par exemple), il vaut mieux indiquer l'emprise :

```
$ render_list -a -z 0 -Z <niveau_de_zoom> -x <coordonnées x min>
-X <coordonnées x max> -y <coordonnées y min> -Y <coordonnées
y max> -l <charge de la machine (load average)> -n <nombre de
processus>
```

Pour obtenir les coordonnées des tuiles, le plus facile est d'aller sur http://tools.geofabrik.de/map/ puis d'activer l'overlays **"Tile coordinates"** et ainsi vous avez les coordonnées des tuiles pour chacune des tuiles.

6 Mettre à jour les données et les tuiles

6.1 D'abord les données

L'intérêt d'utiliser les données d'OSM est qu'elles sont constamment mises à jour par les contributeurs (aussi je vous

NE MANQUEZ PAS

GNU/LINUX MAGAZINE HORS-SÉRIE N°81 !

LE GUIDE POUR
APPRENDRE À PROGRAMMER EN JAVA EN 5 JOURS !

DISPONIBLE

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

invite à en devenir un !). Il est donc intéressant de mettre à jour la base de données. Pour cela, on utilise **osmosis**. Si vous ne l'avez pas encore installé (pour concaténer des **pbf**) :

```
$ sudo apt-get install osmosis
```

Osmosis fait des mises à jour incrémentales, c'est-à-dire qu'il va rajouter uniquement les différences et ne va pas recommencer depuis le début. Par contre, cela veut dire qu'il ne faut pas sauter de mise à jour. La granularité des mises à jour est la journée. C'est-à-dire qu'en mettant à jour, si dans la base, les données dataient du lundi, les nouvelles données seront celles du mardi. Par conséquent, en temps normal (c'est-à-dire dans le cas où on ne souhaite pas rattraper du retard), on mettra à jour une fois par jour.

On va créer un dossier pour **osmosis** dont on va donner la propriété à l'utilisateur **www-data**, car c'est lui qui va faire les mises à jour. On va le laisser le configurer pour son besoin :

```
$ mkdir osmosis && cd osmosis
$ sudo chown www-data:www-data osmosis
$ sudo -u www-data osmosis --rrii workingDirectory=.
```

Cela va créer deux fichiers : un fichier de *lock* (qu'osmosis utilise pour être sûr qu'il n'y a pas deux instances d'osmosis qui tournent en même temps) et un fichier de configuration, **configuration.txt**.

Comme nous avons commencé à utiliser download.geofabrik.de pour les données, nous allons continuer à utiliser ce site pour les mises à jour. Quand vous vous rendez sur la page de la région géographique que vous avez choisie, il y a une ligne **".osc.gz files that contain all changes in this region, suitable e.g. for Osmosis updates"**.

On met alors ce lien dans le fichier **configuration.txt** qu'osmosis a créé, dans la variable **baseUrl**, et on laisse le reste tel quel. Pour la Basse-Normandie, ça donne :

```
baseUrl=http://download.geofabrik.de/europe/france/basse-normandie-updates/
```

Maintenant qu'osmosis sait où récupérer les données, on peut presque lancer une mise à jour. C'est peut-être clair pour vous, mais j'ai mis du temps à le comprendre : quand on lance osmosis, il va importer les différences uniquement sur une journée. Supposons qu'on soit le 8 du mois, vous avez importé initialement les données du 3, si vous lancez osmosis, vous aurez alors en base les données du 4. Pour avoir les données du 8, il faudra donc lancer osmosis quatre autres fois.

Osmosis se repère temporellement grâce à un fichier qui s'appelle **state.txt**. Par la suite, il va le gérer lui-même, mais il faut l'initialiser. On va donc sur la page que vous avez mis dans la variable **baseUrl** : il y a un dossier qui se nomme **000** et un fichier **state.txt**. On va dans le dossier **000** et on va à nouveau dans le dossier **000**. Dans mon cas, l'adresse à laquelle je suis est : http://download.geofabrik.de/europe/france/basse-normandie-updates/000/000/.

Dans cette page, il y a un ensemble de couples de fichiers **state.txt** et d'**osc.gz**, chaque paire étant préfixée d'un nombre. Il y a également la date d'édition des fichiers. On va donc télécharger le fichier **state.txt** qui date de la veille du jour où l'on a téléchargé le **.pbf**. On prend celui de la veille, car on veut être sûr qu'on ne rate pas de données : ce n'est pas grave de prendre des données plus vieilles, car il va juste réécrire dessus. Lorsqu'on l'a téléchargé, il est nécessaire de le renommer en **state.txt**.

Pour mettre à jour, cela se fait en trois étapes :

- Dans la première partie, osmosis va créer un fichier contenant les différences : c'est rapide (moins d'une minute).
- On va ensuite importer ces données dans la base avec **osm2pgsql** : c'est long (suivant la taille des données que vous importez). En important les données, **osm2pgsql** va générer un fichier avec la liste des tuiles impactées pour la mise à jour.
- Enfin, la troisième étape est donc de traiter les tuiles qui sont expirées avec la commande **render_expired**.

On commence donc par créer le fichier de diff :

```
$ sudo -u www-data osmosis --read-replication-interval workingDirectory=. --simplify-change --write-xml-change changes.osc.gz
```

Vous devriez avoir un fichier nommé **changes.osc.gz** dans le dossier.

Ensuite, il faut l'importer dans la base postgis. En faisant ça, **osm2pgsql** va créer le fichier avec les tuiles obsolètes. La commande générique est :

```
$ sudo -u www-data osm2pgsql -a --slim -C <taille du cache> --number-processes <nombre de processus> changes.osc.gz -e11-17 -o expired-tiles.txt
```

Sur ma VM, j'ai utilisé :

```
$ sudo -u www-data osm2pgsql -a --slim -C
512 --number-processes 2 changes.osc.gz
-e11-17 -o expired-tiles.txt
```

Vous pouvez jouer sur les paramètres **-e** pour dire entre quels zooms vous souhaitez indiquer que les tuiles sont obsolètes. J'ai pris de 11 à 17, libre à vous d'en prendre d'autres.

Une fois que l'import du diff a fini, il reste donc à expirer les tuiles.

6.2 Mises à jour des tuiles

render_expired a trois actions possibles face à une tuile ayant expiré :

- la marquer comme *dirty* ;
- la supprimer (**rm**) ;
- la régénérer.

Chaque étape prend plus de ressources. Marquer une tuile comme *dirty* implique que la prochaine fois que quelqu'un tentera d'y accéder, le système va la régénérer, cela représente un faible accès disque sur le coup et donc permet d'étaler la charge du serveur dans le temps. Supprimer la tuile du disque demande déjà un peu plus de ressources en I/O (à l'échelle d'une tuile c'est négligeable, mais il y a plein de tuiles donc ça s'additionne vite). Enfin, régénérer une tuile c'est ce qu'il y a de plus coûteux comme vous pouvez vous en douter.

La commande pour agir sur les tuiles est de la forme :

```
$ sudo -u www-data render_expired -n 2 -z
11 -Z 18 < expired-tiles.txt
```

Cette commande va lancer deux processus et va régénérer les tuiles entre les zooms 11 et 18. J'ai choisi de tout régénérer, car je suis sur une petite zone géographique avec peu de tuiles, mais si vous êtes sur une grande zone, il faudra utiliser aussi les paramètres **-T** (comme *touch*, soit marquer) et **-d** (comme *delete*) pour alléger la mise à jour.

Une fois que tout ça est fonctionnel, il ne reste plus qu'à créer un petit script tout bête qui consiste à lancer ces commandes (**osmosis**, puis **osm2pgsql** et enfin **render_expired**) puis mettre en place un **cron** qui va appeler quotidiennement le script et ainsi vous aurez un serveur de tuiles à jour \o/

7 Bonus

Comme je suppose que vous allez faire plusieurs imports, il faut à chaque fois remettre en place la base de données gis. Pour faire cela, il suffit de supprimer la base de données (il faut que **renderd** soit arrêté sinon postgresql va refuser, car il y est connecté) puis de la recréer. Pour plus de lisibilité, je laisse le prompt pour voir les changements (et vous pourrez admirer le nom de la machine).

```
danj@trelou:~$ sudo -u postgres psql
psql (9.3.9)
Type "help" for help.

postgres=# drop database gis;
DROP DATABASE
postgres=# create database gis;
CREATE DATABASE
postgres=# \q
```

Il faut alors transformer cette nouvelle base en base postgis, c'est-à-dire rajouter l'extension :

```
danj@trelou:~$ sudo -u postgres psql -d gis
psql (9.3.9)
Type "help" for help.

gis=# create extension postgis;
CREATE EXTENSION
postgres=# \q
```

Conclusion

Nous avons donc vu comment installer les différents logiciels pour avoir un serveur de tuiles, comment mettre des données dedans et comment les mettre à jour. Vous êtes maintenant paré à tout ! ■

Référence

[1] Comment « tuner » sa base postgresql : https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server

CRÉEZ VOTRE CLONE DE SIRI, CORTANA ET AUTRES

Frédéric LE ROY [Ingénieur informaticien, touche-à-tout et curieux]

Siri et Cortana sont les assistants de certains smartphones : ils peuvent répondre à différentes requêtes telles que créer des rendez-vous, réaliser des appels, mais aussi répondre à des questions diverses. Nous allons voir comment créer un embryon d'assistant personnel.

Chatterbot Rivescript Cloud
Python Assistant

L'OBJECTIF

Nous allons développer un assistant sans contrôle vocal. En effet le contrôle vocal (voix vers texte ou texte vers voix) est une problématique périphérique à la création d'un assistant qui va devoir répondre à des requêtes textuelles. Cet assistant aura dans cet article deux fonctions : répondre quand on l'appelle ou qu'on lui dit bonjour et aller chercher des informations **en ligne** dans une banque d'informations (allergiques aux services en ligne, passez votre chemin). Le but n'est pas d'avoir un assistant complet, mais d'introduire ce qu'il est possible de réaliser.

LES OUTILS

- Un système GNU/Linux Debian (version *testing*) ;
- `vim` (paquet `vim`) ou tout autre éditeur de code ;
- `python` 2.7 ou 3 au choix ;
- `pip` pour installer des librairies Python.

PHASE 1

Mise en place d'un chatterbot

Pour réaliser cet assistant, nous avons tout d'abord besoin d'un outil pour parler à notre ordinateur. Basiquement, il nous faut donc mettre en place un système similaire à un chat à un détail près : nous devons pouvoir parler à une entité qui saura interpréter ce que nous lui disons.

De tels outils existent déjà, et ils ont pour nom **chatterbots** ou **agents conversationnels**. Beaucoup de chatterbots ont été réalisés dans le cadre de la compétition sur le test de Turing (https://fr.wikipedia.org/wiki/Test_de_Turing). Pour notre besoin, j'ai choisi d'utiliser **Rivescript** (http://www.rivescript.com/) qui est un langage de script pour la création de *chatterbots*.

Rivescript est fourni de base avec un cerveau (l'ensemble des scripts) en anglais et il existe également une version française qui traîne sur Internet. Je vous laisse le choix d'intégrer ou pas ce cerveau par la suite, j'ai de mon côté choisi de partir d'un cerveau vide.

Installons donc `rivescript` avec `pip` :

```
# pip install rivescript
```

Créons un dossier de travail qui va contenir les sources de notre bot :

```
$ mkdir superbot
$ cd superbot
```

Nous allons maintenant créer le programme principal en Python qui va utiliser la librairie **rivescript**.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from rivescript import RiveScript

rs = RiveScript()
rs.load_directory("./cerveau")
rs.sort_replies()

while True:
    msg = raw_input("Vous > ")
    if msg == '/quit':
        quit()
    reply = rs.reply("localuser", msg)
    print(u"Aria > {0}".format(reply))
```

Un point important est l'ajout de la seconde ligne (**-*- coding: utf-8 -*-**) qui permet le support des accents (pratique dans notre langue…).

Ce script commence par l'import de la librairie puis un objet est instancié et les fonctions **load_directory()** et **sort_replies()** sont appelées. La première sert à charger tous les fichiers **.rive** ou **.rs** d'un dossier et la seconde à trier le contenu des fichiers dans le moteur de Rivescript.

Ensuite, nous passons dans une boucle infinie. Pour chaque entrée saisie dans la console, il y aura un appel à la fonction **reply()** de Rivescript. Cette dernière accepte deux paramètres : l'identifiant de l'utilisateur, par défaut **localuser**, et le message. Dans notre cas, nous n'avons pas à gérer plusieurs utilisateurs.

Pour finir, nous affichons la réponse du bot, que nous nommerons **Aria**.

Tel quel ce script ne sert à rien… il lui faut un minimum de cervelle pour fonctionner.

Créons déjà le dossier du cerveau :

```
$ mkdir ./cerveau
```

Maintenant, nous allons anticiper le futur et *patcher* Rivescript pour autoriser les réponses avec des accents et autres choses inexistantes en anglais. Éditez le fichier **/usr/local/lib/python2.7/dist-packages/rivescript/python.py** et modifiez la dernière ligne ainsi :

```
        #return str(reply)
        return reply
```

Ceci nous permettra de faire des réponses en unicode.

PHASE 2
Bonjour

Voyons comment ajouter une interaction à notre cerveau. Ajoutons ceci au fichier **./cerveau/bonjour.rive** :

```
+ bonjour
- salut
```

La ligne qui commence par un **+** est le **trigger**. Si ce que vous dites au bot correspond à une des lignes commençant par ce **+**, alors la ou les lignes qui y correspondent (toutes les lignes suivantes jusqu'au prochain **+**) seront traitées. La ligne qui commence par un **-** corresponds à la réponse à donner.

Faisons un test :

```
$ python main.py
Vous > bonjour
Aria > salut
Vous > coucou
Aria > ERR: No Reply Matched
 Vous > bonjour à vous
Aria > ERR: No Reply Matched
Vous > /quit
```

Tout fonctionne correctement, mais dès que nous disons quelque chose que ne comprend pas le bot (ou qui ne correspond pas exactement), nous recevons une réponse immonde…

Nous allons modifier le fichier ainsi :

```
+ bonjour *
- bonjour
- salut
- coucou
```

Chaque ligne qui commence par un symbole **-** est une réponse possible. Le choix de la réponse est fait au hasard. Notez aussi l'ajout du symbole ***** dans le trigger. Ce symbole permet de capturer tout ce qui suit « bonjour ».

Essayons :

```
$ python main.py
Vous > bonjour
Aria > ERR: No Reply Matched
Vous > bonjour Aria
Aria > coucou
Vous > bonjour Aria
Aria > salut
Vous > /quit
```

Nous notons tout de suite que « bonjour » tout court ne fonctionne plus. En effet, le bot attend forcément quelque chose après « bonjour ». Par contre, nous vérifions bien le caractère aléatoire des réponses.

Finissons-en en modifiant le fichier ainsi :

```
+ (bonjour|salut|coucou) [*]
- bonjour
- salut
- coucou
```

La syntaxe **(a|b|c)** permet d'autoriser différents mots ou groupes de mots à un endroit dans un *trigger*. Nous avons aussi rendu le contenu final optionnel via l'ajout des crochets : **[*]**.

Testons :

```
$ python main.py
Vous > bonjour
Aria > bonjour
```

```
Vous > bonjour Aria
Aria > coucou
Vous > salut
Aria > salut
Vous > salut Aria
Aria > bonjour
Vous > quel beau temps !
Aria > ERR: No Reply Matched
Vous > /quit
```

Voici qui commence à prendre forme. Il nous reste à traiter le cas où le bot ne comprend pas ce que nous lui disons.

PHASE 3

Je ne comprends pas

Nous allons créer un fichier **./cerveau/star.rive** :

```
+ *
- désolé, je ne comprends pas
- pouvez-vous reformuler ?
```

La première ligne va interpréter absolument, tout ce qui est dit, mais qui n'a pas déjà été traité par un autre *trigger*. Faisons le test :

```
$ python main.py
Vous > bonjour Aria
Aria > salut
Vous > comment vas-tu ?
Aria > pouvez-vous reformuler ?
Vous > tu ne me comprends pas ?
Aria > désolé, je ne comprends pas
```

C'est déjà plus sympathique !

PHASE 4

Introduction à Wolfram Alpha

Il n'existe pas à ma connaissance de base de connaissances interrogeable en langage naturel en français. Nous allons donc utiliser **Wolfram Alpha**. Il suffit d'aller sur le site http://www.wolframalpha.com/ et de saisir une question en anglais, par exemple « quel est l'âge de Jacques Chirac » : « how old is Jacques Chirac ». Nous récupérons une réponse en anglais également : « 82 years 9 months 8 days », soit 82 ans et 9 mois.

Il existe une librairie python pour utiliser l'API de ce site : **Tungsten** (https://github.com/seenaburns/Tungsten). Afin de pouvoir utiliser cette API, il faut une clé qui peut être obtenue gratuitement (mais avec un volume de requêtes limité) en créant un compte à cette URL : https://developer.wolframalpha.com/portal/signin.html.

PHASE 5

Mais je parle français moi !

Nous ne sommes plus à un service en ligne près… afin de pouvoir interagir avec Wolfram Alpha en français, nous allons réaliser deux traductions avec un outil de traduction en ligne :

- une traduction de la question du français vers l'anglais ;
- un traduction de la réponse de l'anglais vers le français.

Alors oui, je plaide coupable :

- cela va faire 3 appels à différents services en ligne (avec toutes les éventuelles problématiques de respect de la vie privée qui vont avec) pour une seule question ;
- la double traduction peut générer des approximations (globalement ça marche plutôt bien, mais en effet j'ai pu constater certaines joyeusetés).

Mais je vous avais prévenu dès le début ;)

PHASE 6

Retour à la technique : interfaçage avec Wolfram Alpha

Nous allons tout d'abord commencer par intégrer l'appel à Wolfram Alpha. Pour cela, nous allons créer un fichier **./cerveau/wolfram.rive** minimal :

```
> object wolfram_alpha python
    return u"Vous avez dit : {0}".
format(args)
< object

+ dis[-]moi *
- <call>wolfram_alpha <star></call>
```

Notez tout d'abord le bloc **object**. Il va contenir un code en python qui sera nommé ici **wolfram_alpha**. Il reçoit (et recevra toujours) un unique paramètre : **args**.

Notez ensuite la réponse qui contient l'appel à notre code python via la syntaxe **<call>nom du bloc paramètre1 paramètre2 paramètre3 ...</call>**. Afin de passer un paramètre, nous pouvons utiliser le pattern **<star>** qui correspond à la valeur de *. La réponse affichée sera le retour du code Python.

Testons ce code :

```
$ python main.py
Vous > dis-moi how old is jacques chirac
Aria > Vous avez dit : [u'how', u'old',
u'is', u'jacques', u'chirac']
```

Ici nous voyons que **args** est en fait un tableau qui contient tous les mots. Il nous faudra donc transformer ce tableau en une phrase ainsi : **' '.join(args)**.

Nous avons une enveloppe python, installons maintenant la librairie **tungsten** :

```
# pip install tungsten
```

Et utilisons-la dans notre fichier :

```
> object wolfram_alpha python
    import tungsten
    APP_ID = 'xxxxxxxxxxxxxxxxx'

    query = ' '.join(args)
   client = tungsten.Tungsten(APP_ID)
    result = client.query(query)
    print(result)

    for pod in result.pods:
         print(pod.id)
        if pod.id == "Result":
           return u'\n'.join(pod.
format['plaintext'])

    return u"Je n'ai pas de réponse ou
la réponse est trop complexe"
< object

+ dis[-]moi *
- <call>wolfram_alpha <star></call>
```

Notez tout d'abord qu'il va nous falloir renseigner l'identifiant récupéré sur le site de Wolfram pour l'utilisation de l'API.

Ensuite, nous allons concaténer tous les éléments du tableau en les séparant par un espace afin d'avoir une vraie phrase. Et il suffit d'appeler la fonction **query()** de **tungsten** pour recevoir un résultat de Wolfram. Le résultat est retourné sous forme de **pods**. Chaque pod a un **id** :

- **Input**
- **Result**
- …

Le pod **Input** correspond à ce qu'a compris Wolfram de votre requête. Nous pouvons l'ignorer. Le pod **Result** est celui qui nous intéresse. Il sera présent pour la plupart des requêtes de type question. S'il n'est pas présent, il y a de fortes chances que le contenu de la réponse soit trop complexe pour être « dit » par notre assistant : dans le cas d'un pays, il peut s'agir de tableaux (démographie, etc.), de cartes...

Le contenu textuel de chaque pod se trouve dans la cellule **plaintext** du dictionnaire **format** du pod.

J'ai volontairement laissé des **print** dans le code pour illustrer la structure de la réponse :

```
$ python main.py
Vous > dis moi how old is jacques chirac
<tungsten.core.Result object at
0x7fc897ce6bd0>
 Input
Result
Aria > 82 years 9 months 12 days
```

Nous voyons bien ici l'objet renvoyé par tungsten et les deux pods **Input** et **Result**, mais surtout la réponse !

PHASE 7

Ajout de la traduction

L'ajout de la traduction va passer par le service Google traduction via la librairie python **goslate**. Installons-la :

```
# pip install goslate
```

Testons-la dans une console python :

```
>>> import goslate
>>> gs = goslate.Goslate()
>>> gs.translate("hello world", "fr",
"en")
u'Bonjour le monde'
>>> gs.translate("bonjour le monde",
"en", "fr")
u'Hello world'
```

Rien de bien compliqué, il faut trois paramètres :

- texte à traduire ;
- langue cible ;
- langue d'origine.

Intégrons ceci à notre fichier :

```
> object wolfram_alpha python
    import tungsten
    import goslate
    APP_ID = 'xxxxxxxxxxxxxxx'

    query = ' '.join(args)
    gs = goslate.Goslate()
    query_en = gs.translate(query, "en",
"fr")
    client = tungsten.Tungsten(APP_ID)
    result = client.query(query_en)

    for pod in result.pods:
        if pod.id == "Result":
            return u'\n'.join(gs.
translate(pod.format['plaintext'], "fr",
"en"))

    return u"Je n'ai pas de réponse ou
la réponse est trop complexe"
< object


+ dis[-]moi *
- <call>wolfram_alpha <star></call>
```

Et testons :

```
$ python main.py
Vous > dis-moi quel est l'âge de Jacques
Chirac
Aria > 82 ans 9 mois 12 jours
Vous > dis-moi qui est le président de
la France
Aria > François Hollande
Vous > dis-moi quel est le poids d'une
pomme
Aria > 182 grammes
```

Sympa, non ?

LE RÉSULTAT

Pour l'anecdote, Siri utilise en bonne partie Wolfram Alpha également pour réaliser des recherches.

Ici, nous avons créé un assistant simple qui peut répondre à des questions via l'utilisation d'un service en ligne, mais nous pouvons aller bien plus loin : Rivescript, pour peu qu'on se donne la peine de creuser et de le tuner est relativement souple pour faire bien plus qu'un simple *chatterbot*.

Dans le cadre d'un projet de domotique, j'ai mis en place un majordome basé sur Rivescript et qui peut entre autres :

- allumer et éteindre des lumières/appareils ;
- vous donner la météo ;
- évidemment, rechercher des informations sur Wolfram ;
- apprendre de ce qu'on lui dit.

À vous de créer votre assistant maintenant ;). ■

PARLEZ À VOTRE ORDINATEUR... ET FAITES-VOUS COMPRENDRE !

Tristan COLOMBO

Que ce soit Hal ou Jarvis, les exemples d'ordinateurs intelligents et doués de parole ne manquent pas au cinéma ou dans la littérature. Nous laisserons de côté l'aspect « intelligent » pour nous focaliser sur la reconnaissance vocale de manière à pouvoir demander à l'ordinateur d'exécuter certaines tâches en « dialoguant » avec lui... et ce n'est déjà pas si simple !

Mots-clés : Reconnaissance vocale, Python, Jasper, Pocketsphinx, Raspbian

Résumé

Les solutions de reconnaissance vocale sont assez complexes à mettre en place, car utilisant de nombreux outils et nécessitant donc un long travail d'installation et de configuration, sans parler de l'aspect matériel ou le micro devra être reconnu. Le projet Jasper propose une solution modulaire puisque l'utilisateur peut installer différents moteurs de reconnaissance vocale et que le développement des commandes se fait à l'aide de modules Python. Dans cet article, nous installons Jasper avec le moteur STT Pocketsphinx, décrivons ses limites et créons un petit module de commande personnalisé avant de tester une utilisation pure de PocketSphinx avec Python.

Après avoir testé **Rivescript** avec Frédéric Le Roy dans l'article précédent, vous avez peut-être envie d'aller plus loin et de tester la reconnaissance vocale. C'est ce que je vous propose en utilisant le projet **Jasper** qui a été conçu spécialement pour élaborer des *chatbots* sur Raspberry Pi avec le système d'exploitation Raspbian. Cette solution peut également être mise en place sur un ordinateur fixe et c'est ce que je ferai dans cet article en utilisant une distribution Debian : toutes les indications données sont donc compatibles avec Raspbian et peuvent être utilisées pour une Raspberry Pi. L'installation est un peu longue et il vaut mieux s'assurer de ne pas faire d'erreur...

1 Installation

Il n'existe pas encore de paquetage Debian ou Raspbian, ce qui implique de passer obligatoirement par une installation manuelle.

1.1 Étape préparatoire

Il faut tout d'abord s'assurer de disposer des bons outils :

```
$ sudo aptitude install git-core python-dev python-pip bison
libasound2-dev python-pyaudio libyaml-dev
```

Sur Raspbian, ajoutez la librairie **libportaudio-dev**.

Assurez-vous ensuite que votre micro est correctement reconnu et configuré. Cela va dépendre de nombreux facteurs et je ne peux donc pas donner de solution générique si ce n'est de penser à l'activer dans l'outil « Préférences du son ». Pour tester si tout fonctionne correctement, vous pourrez employer **arecord** pour vous enregistrer (<Ctrl> + <c> pour interrompre) et **aplay** pour jouer le son contenu dans le fichier précédemment enregistré :

```
$ arecord test.wav
Capture WAVE 'test.wav' : Unsigned 8 bit, Fréquence 8000 Hz, Mono
^CInterrompu par le signal Interrompre...
$ aplay test.wav
Lecture WAVE 'test.wav' : Unsigned 8 bit, Fréquence 8000 Hz, Mono
```

L'étape la plus périlleuse étant passée, il faut définir des variables d'environnement. En se basant sur un shell bash, ajoutez les lignes suivantes à la fin de votre **~/.bashrc** :

```
export LD_LIBRARY_PATH=/usr/local/lib
export PATH=$PATH:$LD_LIBRARY_PATH
```

Bien entendu, n'oubliez pas de relire le fichier :

```
$ source ~/.bashrc
```

1.2 Installation de Jasper

Nous pouvons maintenant récupérer le code source de Jasper :

```
$ git clone https://github.com/jasperproject/jasper-client.git
jasper
```

Comme il s'agit d'un projet Python et qu'il est bien structuré, un fichier **requirements.txt** est fourni pour que **pip** puisse installer toutes les dépendances automatiquement :

```
$ sudo pip install -U -r jasper/client/requirements.txt
```

Vous pensiez que c'était fini ? Que nenni ! Il reste à installer un moteur STT (*Speech-To-Text* soit « de la parole au texte ») et un moteur TTS (*Text-To-Speech* soit « du texte à la parole »). C'est la partie la plus importante, car ce sont ces moteurs qui vont faire tout le travail et tout particulièrement le moteur STT. Le choix du moteur sera dicté par l'utilisation que vous allez avoir de votre application. En voici quelques-uns :

- **Pocketsphinx [1]** : un moteur rapide fonctionnant bien sur les systèmes embarqués (comme le Raspberry Pi), mais qui n'est pas le meilleur en terme de taux de reconnaissance vocale. Avantage de ce moteur : ne nécessite pas une utilisation du cloud ;
- **Google STT** : le système de reconnaissance de Google (celui des smartphones Android). Ne fonctionne qu'à travers le cloud avec une limite du nombre de requêtes par jour. Il faut l'activer dans la console développeurs Google **[2]** ;
- **AT&T STT [3]** : système de reconnaissance vocale d'AT&T fonctionnant comme Google STT dans le cloud ;
- **Wit.ai STT [4]** : encore un moteur fonctionnant dans le cloud...
- **Julius [5]** : et pour finir un moteur ne nécessitant pas de connexion internet. Par contre ce dernier, très performant, demande une configuration poussée.

Pour moi le choix ne peut être que Pocketsphinx ou Julius, le principe de la connexion internet étant rédhibitoire. Pour une première, nous n'allons pas tenter le diable avec Julius donc nous utiliserons Pocketsphinx.

1.3 Installation de Pocketsphinx

Pocketsphinx est disponible dans les dépôts Debian, ce qui simplifie nettement l'installation :

```
$ sudo aptitude install pocketsphinx
```

Il nous faut ensuite installer CMUCLMTK, le *wrapper* permettant d'accéder aux outils du CMU Sphinx (la boîte à outils contenant Pocketsphinx). Pour commencer, nous vérifions une fois encore que nous disposons de tous les outils :

```
$ sudo aptitude install subversion autoconf libtool automake
gfortran g++
```

Nous pouvons maintenant installer CMUCLMTK dans le répertoire de Jasper :

```
$ cd jasper
$ svn co https://svn.code.sf.net/p/cmusphinx/code/trunk/cmuclmtk/
$ cd cmuclmtk/
$ sudo autogen.sh && sudo make && sudo make install
$ cd ..
```

1.4 Jasper écoute... mais il parle également

Nous avons installé un moteur STT pour que Jasper puisse écouter, il lui faut maintenant un moteur TTS pour qu'il puisse parler. Là encore plusieurs moteurs sont disponibles : **eSpeak**, **Flite**, **Google TTS**, etc. Le plus simple est **eSpeak** :

```
$ sudo aptitude install espeak
```

Si vous souhaitez pouvoir changer de voix vous pouvez tester Flite :

```
$ sudo aptitude install flite
```

flite -lv vous permettra d'afficher la liste des voix disponibles (nous nous en servirons plus tard pour la configuration).

1.5 Et pour finir...

Le moteur Pocketsphinx a besoin de certaines dépendances. En l'occurrence, il s'agit du **MIT Language Modeling Toolkit**, **m2m-aligner** et **Phonetisaurus**. Des paquets sont disponibles pour Debian dans les dépôts expérimentaux. On va donc commencer par créer un fichier de sources expérimentales **/etc/apt/sources.list.d/experimental.list** contenant la ligne :

```
deb http://ftp.debian.org/debian experimental main contrib non-free
```

Après une mise à jour d'apt, il suffit d'indiquer que l'on souhaite aller chercher des paquets dans ce dépôt :

```
$ sudo aptitude update
$ sudo aptitude -t experimental install phonetisaurus m2m-aligner
mitlm
```

Il faut également ajouter l'installation d'**OpenFst** à la main. Un fst, pour *weighted Finite-State Transducer* (transducteur à états finis pondérés), est un automate **[6]** où chaque transition est dotée d'une valeur d'entrée, de sortie et d'un poids. OpenFst fournit les outils permettant de manipuler de tels automates qui sont utilisés pour déterminer la conversion des graphèmes en phonèmes (et réciproquement). Pour revenir à l'installation :

```
$ wget http://distfiles.macports.org/openfst/openfst-1.4.1.tar.gz
$ tar -xvf openfst-1.4.1.tar.gz
$ cd openfst-1.4.1
$ sudo ./configure --enable-compact-fsts --enable-const-fsts
--enable-far --enable-lookahead-fsts --enable-pdt
$ sudo make install
$ cd ..
$ rm -R openfst-1.4.1
```

Mais nous ne sommes pas pour autant tirés d'affaire... il faut maintenant configurer Jasper.

2 Configuration

Pour commencer, il faut générer un profil d'utilisation grâce au script **populate.py** se trouvant dans le répertoire **client** de notre installation (soit **jasper/client**) :

```
$ cd client
$ python populate.py
```

Des questions vous seront alors posées. Rassurez-vous, vous n'êtes pas obligé de répondre à tout ! L'adresse et le mot de passe gmail sont par exemple demandés pour utiliser le module permettant de vérifier si vous avez reçu des mails. Vous pouvez donc laisser ces champs vides...

```
...
First name: Tristan
Last name: Colombo
...
Gmail address:
Password:

Phone number (no country code). Any dashes or spaces will be
removed for you:

Phone carrier (for sending text notifications).
...
Carrier:

...
Location: Marseille
Location saved as Marseille,  FR

Please enter a timezone from the list located in the TZ* column at
http://en.wikipedia.org/wiki/List_of_tz_database_time_zones, or
none at all.
Timezone: Europe/Paris

Would you prefer to have notifications sent by email (E) or text
message (T)? E

If you would like to choose a specific STT engine, please specify
which.
Available implementations: %s. (Press Enter to default to
PocketSphinx):
Unrecognized STT engine. Available implementations: ['google',
'sphinx']
Writing to profile...
Done.
```

NE MANQUEZ PAS

LINUX PRATIQUE N°92 !

DÉCOUVREZ TAILS, LA SOLUTION UTILISÉE PAR SNOWDEN !

DISPONIBLE

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

Ceci génère un fichier yaml dans **~/.jasper/profile.yml**. Il faut toutefois finir la configuration de Pocketsphinx manuellement dans ce fichier. Comme nous sommes français, nous allons télécharger un modèle de langue française (sinon, vous pouvez vous contenter des modèles anglais déjà présents dans **/usr/share/pocketsphinx/model/lm/en_US** et dans **/usr/share/pocketsphinx/model/hmm/en_US**) :

```
$ sudo mkdir /usr/share/pocketsphinx/model/lm/fr_FR/
$ sudo wget http://sourceforge.net/projects/cmusphinx/files/
Acoustic%20and%20Language%20Models/French%20Language%20Model/
frenchWords62K.dic -O /usr/share/pocketsphinx/model/lm/fr_FR/
frenchWords62K.dic
$ sudo wget http://sourceforge.net/projects/cmusphinx/files/
Acoustic%20and%20Language%20Models/French%20Language%20Model/
french3g62K.lm.dmp -O /usr/share/pocketsphinx/model/lm/fr_FR/
french3g62K.lm.dmp
$ wget http://sourceforge.net/projects/cmusphinx/files/Acoustic%20
and%20Language%20Models/French/cmusphinx-fr-ptm-5.2.tar.gz
$ sudo tar -zxvf cmusphinx-fr-ptm-5.2.tar.gz -C /usr/share/
pocketsphinx/model/hmm
$ sudo mv /usr/share/pocketsphinx/model/hmm/cmusphinx-fr-ptm-5.2/
/usr/share/pocketsphinx/model/hmm/fr_FR
$ rm cmusphinx-fr-ptm-5.2.tar.gz
```

Avant de poursuivre, nous allons tout de même tester notre installation de Pocketsphinx. Vérifiez que votre micro est bien branché... et c'est parti !

```
$ pocketsphinx_continuous -dict /usr/share/pocketsphinx/model/
lm/fr_FR/frenchWords62K.dic -hmm /usr/share/pocketsphinx/model/
hmm/fr_FR -lm /usr/share/pocketsphinx/model/lm/fr_FR/french3g62K.
lm.dmp
READY....
Listening...
Recording is stopped, start recording with ad_start_rec
Stopped listening, please wait...
INFO: cmn_prior.c(121): cmn_prior_update: from < 42.54  0.39 -1.72
4.78 -4.25 -3.50 -9.03 -1.54 -6.40  2.50 -5.15 -2.35 -3.10 >
INFO: cmn_prior.c(139): cmn_prior_update: to   < 42.63 -1.37 -0.82
6.15 -3.49 -4.29 -9.28 -0.75 -5.88  2.23 -5.55 -2.24 -3.46 >
INFO: ngram_search_fwdtree.c(1549):     4736 words recognized (25/
fr)
INFO: ngram_search_fwdtree.c(1551):   254278 senones evaluated
(1367/fr)
INFO: ngram_search_fwdtree.c(1553):  1299720 channels searched
(6987/fr), 116599 1st, 124931 last
INFO: ngram_search_fwdtree.c(1557):    16455 words for which last
channels evaluated (88/fr)
INFO: ngram_search_fwdtree.c(1560):   170925 candidate words for
entering last phone (918/fr)
...
000000012: linux magazine de france
```

Lorsque je dis « Linux Magazine France », j'obtiens bien l'affichage de « linux magazine france ». Alors forcément les plus pointilleux diront qu'il fallait tester avec « GNU/Linux »... c'est vrai, mais « GNU » ça donne « des des », ou « une » ou « qu'on lutte »... bref, c'est n'importe quoi. Si vous avez un accent régional prononcé, essayez également de le gommer sous peine de voir vos propos déformés. Prenons l'exemple d'un bel accent français, cet accent chantant où l'on sent le soleil, la mer, les cigales... Bref, en disant « jaune » vous obtiendrez « john » (il faut dire « jÔne » pour se faire comprendre). Les parisiens ne devraient pas rencontrer ce genre de problème, pour les autres il faudra s'adapter (ou créer de nouveaux fichiers de langue !). Enfin, comme nous avons défini des fichiers de français, n'essayez pas d'introduire des mots anglais... « hello » deviendra « hélas au » ou « alors ». Notez donc ici que la reconnaissance de la langue française fonctionne. Ça n'est pas parfait, mais c'est utilisable.

Indiquons maintenant les fichiers que nous allons utiliser dans **~/.jasper/profile.yml**. Nous avons tout prévu pour le français mais, autant vous l'annoncer dès maintenant, ça ne marchera pas (en tout cas pas avec Jasper, nous verrons en fin d'article comment parvenir à une reconnaissance de notre belle langue en Python avec le module **pocketsphinx**). J'indiquerai par la suite d'où vient le problème et nous utiliserons donc ici la reconnaissance anglaise :

```
...
stt_engine: sphinx
pocketsphinx:
    hmm_dir: '/usr/share/pocketsphinx/model/hmm/en_US/hub4wsj_
sc_8k'
...
```

On va ensuite chercher le fichier de langue fst et le placer dans le répertoire **../phonetisaurus** que nous allons créer (c'est là que Jasper va le rechercher par défaut) :

```
$ wget https://www.dropbox.com/s/kfht75czdwucni1/g014b2b.tgz
$ tar -xvf g014b2b.tgz
$ cd g014b2b
$ compile-fst.sh
$ cd ..
$ rm -R g014b2b.tgz
$ mkdir ../phonetisaurus
$ mv g014b2b/g014b2b.fst ../phonetisaurus
```

Votre fichier sera lu, mais si vous voulez vraiment en être certain vous pouvez modifier votre **~/.jasper/profile.yml** :

```
...
stt_engine: sphinx
pocketsphinx:
    fst_model: '../phonetisaurus/g014b2b.fst'
    hmm_dir: '/usr/share/pocketsphinx/model/hmm/en_US/hub4wsj_
sc_8k'
...
```

Le problème du moteur STT étant réglé, passons au moteur TTS. Par défaut, ce sera eSpeak qui sera employé, mais vous pouvez le spécifier :

```
tts_engine: espeak-tts
```

Pour Flite, vous pourrez indiquer quelle voix employer :

```
tts_engine: flite-tts
flite-tts:
    voice: 'slt'
```

Pour finir, reportez l'identifiant WMO de la station météo la plus proche de chez vous (si vous souhaitez accéder à la météo bien entendu). Vous trouverez cet identifiant sur la page http://www.wunderground.com/about/faq/international_cities.asp. Pour Marseille, c'est **07650** :

```
...
wmo_id: 07650
```

3 Premier lancement

Nous pouvons enfin lancer Jasper !

```
$ jasper.py
...
  File "/usr/lib/python2.7/dist-packages/pip/download.py", line
25, in <module>
    from requests.compat import IncompleteRead
ImportError: cannot import name IncompleteRead
```

Eh oui, ça c'est tout de suite moins marrant... La version de **pip** est trop ancienne dans les dépôts Debian et il faut passer par **easy_install** pour obtenir une version plus récente :

```
$ pip -V
pip 1.5.6 from /usr/lib/python2.7/dist-packages (python 2.7)
$ sudo aptitude remove python-pip
$ sudo easy_install -U pip
$ pip -V
pip 7.1.2 from /usr/local/lib/python2.7/dist-packages/pip-7.1.2-
py2.7.egg (python 2.7)
```

Allez, on peut relancer Jasper :

```
$ jasper.py
************************************************************
*            JASPER - THE TALKING COMPUTER                 *
* (c) 2015 Shubhro Saha, Charlie Marsh & Jan Holthuis *
************************************************************
ERROR:root:Error occured!
...
ValueError: STT engine 'sphinx' is not available (due to missing
dependencies, missing dependencies, etc.)
```

Eh bien non, ça ne marche toujours pas... la faute à une installation incomplète. Pourquoi ai-je laissé cette erreur au lieu d'indiquer dès le départ les paquets à ajouter ? Simplement parce qu'ils ne sont pas précisés dans la documentation officielle...

```
$ sudo aptitude install python-pocketsphinx libpocketsphinx1
```

Vous pensez que nous en avons fini et que tout va fonctionner ? Je vous trouve optimistes :

```
$ jasper.py
...
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
```

Il nous manque le serveur de sons **Jackd** :

```
$ sudo aptitude install jackd
$ sudo dpkg-reconfigure -p high jackd2
```

Vérifiez que vous appartenez bien au groupe **audio**. Pour un utilisateur **nom_user** la commande sera :

```
$ sudo adduser nom_user audio
```

Rebootez et vérifiez :

```
$ ulimit -l -r
max locked memory       (kbytes, -l) unlimited
real-time priority              (-r) 95
```

N'oubliez pas de relancer **jackd** à chaque fois que vous voudrez utiliser Jasper (ou automatisez la tâche si vous l'employez de manière non épisodique) :

```
$ jackd -d dummy &
```

Et maintenant, il ne reste plus qu'un problème de son (ne tenez pas compte des avertissements Alsa). Il faut tout d'abord savoir que Jasper est censé vous parler, mais que vous ne l'entendez pas. Comment je le sais ? Regardez la sortie en mode *debug* :

```
$ jasper.py --debug
...
DEBUG:client.tts:Saying 'How can I be of service, Tristan?' with
'espeak-tts'
...
```

Pour que ça fonctionne, modifiez la ligne 76 du fichier **client/tts.py** :

```
76:         cmd = ['aplay', '-D', 'default', str(filename)]
```

Jasper parle ! On progresse !

```
$ jasper.py --debug
INFO:client.conversation:Starting to handle conversation with
keyword 'JASPER'.
DEBUG:client.conversation:Started listening for keyword 'JASPER'
DEBUG:apscheduler.scheduler:Looking for jobs to run
DEBUG:apscheduler.scheduler:Next wakeup is due at 2015-11-04
14:04:45.930399+00:00 (in 29.992091 seconds)
INFO:client.stt:Transcribed: ['DID JASPER WORK']
DEBUG:client.conversation:Stopped listening for keyword 'JASPER'
INFO:client.conversation:Keyword 'JASPER' has been said!
...
INFO:client.stt:Transcribed: ['WEATHER WEATHER OF TODAY']
DEBUG:client.conversation:Stopped to listen actively with
threshold: 19.8
DEBUG:client.brain:'WEATHER WEATHER OF TODAY' is a valid phrase
for module 'Weather'
DEBUG:client.tts:Saying 'I'm sorry, I can't seem to access that
information. Please make sure that you've set your location on the
dashboard.' with 'flite-tts'
DEBUG:client.tts:Executing flite -voice slt -t 'I'"'"'m sorry, I
can'"'"'t seem to access that information. Please make sure that
you'"'"'ve set your location on the dashboard.' /tmp/tmp4TwNLa.wav
```

Jasper a entendu son nom (je le sais, c'est moi qui l'ai dit), il a émis un beep puis je lui ai dit « what is the weather today » (qu'il a interprété comme « weather weather of today ») puis il m'indique qu'il n'a pas accès à l'information. Comme vous pourrez le constater, il va falloir vous entraîner à prononcer « Jayespeur » à l'américaine pour déclencher l'écoute d'un ordre et il faudra parler très rapidement après le beep. C'est un peu déroutant au début. En tout cas, nous pouvons voir que ça « fonctionne ». Prenons maintenant les problèmes dans l'ordre et changeons le nom de Jasper pour qu'il le reconnaisse plus simplement.

4 On rebaptise Jasper

Pour que Jasper change de nom, il va falloir de nombreuses étapes et notamment créer un nouveau corpus. Pour conserver une trace de ce changement, je place le corpus dans un répertoire spécifique. Nous allons appeler notre nouveau Jasper John pour rester en anglais (je l'aurais volontiers appelé Aria, mais avec une prononciation anglaise ça risque de faire un peu bizarre...) :

```
$ mkdir jasper_perso
$ cd jasper_perso
```

Créez ensuite un fichier **jasper_perso.txt** contenant (par exemple) :

```
JOHN
ON
DID ON
```

Rendez-vous sur la page http://www.speech.cs.cmu.edu/tools/lmtool-new.html, puis uploadez votre fichier, compilez le corpus et récupérez le fichier dic et lm (pour moi **5785.dic** et **5785.lm** que j'ai déplacé dans le répertoire **jasper_perso**). Il faut ensuite placer ces fichiers dans le répertoire **jasper/static** à la place de **dictionary_persona.dic** et de **languagemodel_persona.lm** (que nous sauvegarderons) et remplacer **keyword_phrases** par **jasper_perso.txt** :

```
$ cd ../static
$ mv dictionary_persona.dic ../jasper_perso
$ mv languagemodel_persona.lm ../jasper_perso
$ cp ../jasper_perso/5785.dic dictionary_persona.dic
$ cp ../jasper_perso/5785.lm languagemodel_persona.lm
$ cp ../jasper_perso/jasper_perso.txt keyword_phrases
```

Changez ensuite la ligne 117 du fichier **jasper.py** :

```
117:         conversation = Conversation("JOHN", self.mic, self.
config)
```

Jasper répond désormais au nom de John.

On peut s'amuser également à modifier les sons émis par Jasper et stockés dans **static/audio**. **beep_hi.wav** correspond par exemple au beep indiquant la reconnaissance du nom de Jasper. Je l'ai remplacé par le « *yes, master* » des ogres de Warcraft II...

5 Et pourquoi est-ce que je n'ai pas la météo ?

La question est accessoire, mais tout à l'heure j'ai demandé quel était le temps et Jasper (donc John maintenant, attention au dédoublement de personnalité !) m'a répondu qu'il n'avait pas accès à l'information. Étrangement, si comme indiqué dans la documentation **[7]** on utilise l'identifiant wmo de la ville d'Essen en Allemagne (**10410**), ça fonctionne. Par contre j'ai testé avec Marseille, Paris et New-York et là impossible d'avoir une réponse. C'est donc un problème du module (le format des pages html qui sont *parsées* doit être différent).

VOICI TOUTES LES OFFRES COUPLÉES AVEC GNU/LINUX MAGAZINE !
POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Offre	SUPPORT / ABONNEMENT (Prix en Euros / France Métropolitaine)	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF (PDF 1 lecteur) Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Tarif TTC
LM	11n° GLMF	LM1	65,-	LM12	95,-	LM13	149,-	LM123	174,-
LM+	11n° GLMF + 6n° HS	LM+1	118,-	LM+12	177,-	LM+13	197,-	LM+123	256,-
LES COUPLAGES « LINUX »									
A	11n° GLMF + 6n° LP	A1	95,-	A12	140,-	A13	218,-	A123	263,-
A+	11n° GLMF + 6n° HS + 6n° LP + 3n° HS	A+1	182,-	A+12	263,-	A+13	300,-	A+123	386,-
B	11n° GLMF + 6n° MISC	B1	100,-	B12	147,-	B13	233,-	B123	280,-
B+	11n° GLMF + 6n° HS + 6n° MISC + 2n° HS	B+1	172,-	B+12	248,-	B+13	300,-	B+123	381,-
C	11n° GLMF + 6n° LP + 6n° MISC	C1	135,-	C12	197,-	C13	312,-	C123	374,-
C+	11n° GLMF + 6n° HS + 6n° LP + 3n° HS + 6n° MISC + 2n° HS	C+1	236,-	C+12	339,-	C+13	403,-	C+123	516,-
LES COUPLAGES « EMBARQUÉ »									
F	11n° GLMF + 6n° HK* + 4n° OS	F1	125,-	F12	188,-	F13	229,-*	F123	292,-*
F+	11n° GLMF + 6n° HS + 6n° HK* + 4n° OS	F+1	183,-	F+12	275,-	F+13	287,-*	F+123	379,-*
LES COUPLAGES « GÉNÉRAUX »									
H	11n° GLMF + 6n° HK* + 6n° LP + 6n° MISC + 4n° OS	H1	200,-	H12	300,-	H13	402,-*	H123	499,-*
H+	11n° GLMF + 6n° HS + 6n° HK* + 6n° LP + 3n° HS + 6n° MISC + 2n° HS + 4n° OS	H+1	301,-	H+12	452,-	H+13	493,-*	H+123	639,-*

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

N'hésitez pas à consulter les détails des offres ci-dessus sur : www.ed-diamond.com !

6 Mais pourquoi est-ce que Jasper ne fonctionne pas en français ???

Pour le français, il faut récupérer un autre fichier fst :

```
$ wget http://sourceforge.net/projects/cmusphinx/files/G2P%20
Models/fr.tar.gz
$ tar -xvf fr.tar.gz
$ mv fr/model.fst ../phonetisaurus/french.fst
```

On modifie alors le fichier **~/.jasper/profile.yml** pour faire référence à la configuration française que nous avons installé précédemment (vous pouvez utiliser le caractère **#** pour commenter des lignes en yaml) :

```
pocketsphinx:
    fst_model: '../phonetisaurus/french.fst'
    hmm_dir: '/usr/share/pocketsphinx/model/hmm/fr_FR'
```

Si vous lancez le programme, celui-ci bloquera lors de la génération des phonèmes avec **phonetisaurus-g2p**. Je vais donc vous expliquer pourquoi ça ne marche pas, après avoir passé pas mal de temps à errer dans le code et tenté diverses manœuvres.

Le fichier **static/keyword_phrases** et les différents modules de service (dans **client/modules**) sont notés en majuscules, or pour le français nous nous appuyons sur un dictionnaire où les phonèmes sont notés en minuscules et le fichier fst a été généré également à partir de données en minuscules. **phonetisaurus-g2p** ne retrouve plus ses petits !

Retournez dans le répertoire **fr** qui nous a fourni le fichier **french.fst**. Vous constaterez qu'il y a d'autres fichiers au format texte qui ont permis la génération de ce fst. J'ai donc pensé qu'en modifiant les fichiers de manière à passer les phonèmes en majuscules cela réglerait le problème. J'ai donc édité le fichier **model.output.syms** avec vim et j'ai procédé aux remplacements (les termes entre **<** et **>** doivent rester en minuscules) :

```
:%s/\(\w\)/\U\1/g
:%s/<\(\S\+\)>/<\L\1>/g
```

J'ai fait la même chose avec le fichier **model.fst.txt** :

```
:%s/^\(\S\+\t\S\+\t\S\+\t\)\(\S\+\)/\1\U\2/g
:%s/<\(\S\+\)>/<\L\1>/g
```

Puis j'ai compilé mon fst :

```
$ fstcompile --isymbols=model.input.syms --osymbols=model.output.
syms --keep_isymbols --keep_osymbols model.fst.txt newFrench.fst
```

On obtient donc un fichier fst censé être correct. Je dis bien « censé », car il provoque une erreur à l'exécution de Jasper...

J'ai également effectué un test en utilisant uniquement des minuscules pour définir les noms attendus (ce qui pose problème avec le nom des commandes de client/modules qui sont stockées dans une liste **WORDS** et qu'il faut également modifier. Bref, je jette l'éponge : Jasper fonctionne en anglais et Pocketsphinx en français (alors que Jasper s'appuie sur Pocketsphinx...). Si un lecteur trouve la solution qu'il n'hésite pas à m'en faire part !

7 Un module Jasper personnalisé

L'objectif ici sera de connaître l'occupation du CPU sur un mot clé tel que « computer » (on va éviter la détection hasardeuse de « 6-pi-ou ». Nous allons utiliser le module **psutil** pour obtenir des informations système depuis Python :

```
$ sudo pip install psutil
```

Il suffit ensuite de créer un fichier Python dans **client/modules** ayant la structure suivante (les commentaires ont été retirés) :

```
01: # -*- coding: utf-8-*-
02: import psutil
03: import re
04:
05: WORDS = ["COMPUTER"]
06:
07:
08: def handle(text, mic, profile):
...
19:     response = psutil.cpu_percent()
20:     mic.say("CPU occupation is {}%".format(response))
21:
22:
23: def isValid(text):
...
30:     return bool(re.search(r'\bcomputer\b', text,
re.IGNORECASE))
```

Le ou les termes employés pour lancer la commande sont donnés dans les lignes 5 et 30. Je me suis appuyé sur

la structure des modules existants, cela n'aurait tenu qu'à moi les termes de la ligne 30 auraient été extraits de la liste **WORDS** pour minimiser les risques d'erreurs et la fonction **isValid()** aurait été générique et donc définie une seule fois. Pour revenir au code, c'est la fonction **handle()** des lignes 8 à 20 qui définit la commande : un appel à **psutil.cpu_percent()** pour obtenir le pourcentage d'occupation CPU (ligne 19) et la réponse annoncée oralement (ligne 20).

On peut donc constater qu'il est assez simple de déclencher n'importe quelle action à la voix, Jasper gérant l'acquisition et aiguillant vers le bon module. Bien entendu, au vu de la qualité de reconnaissance il est préférable de ne pas lancer d'opérations trop « sensibles »...

8 Et seulement avec Pocketsphinx ?

La question qui se pose maintenant est de savoir si l'on a réellement besoin de Jasper. Puisque ce dernier s'appuie sur Pocketsphinx, nous pourrions créer un mini Jasper qui comprend le français. Pour ne pas trop alourdir le code plutôt que de déclencher et arrêter automatiquement l'écoute, je vais fixer arbitrairement le temps d'écoute à 2 secondes. C'est largement suffisant pour prononcer un mot déclenchant une action ! En effet, nous n'allons pas faire comme dans Jasper et faire croire à l'utilisateur que toute la phrase est importante...

De manière à rester cohérent avec ce que nous avons vu précédemment, le code sera donné en Python 2 (mais Pocketsphinx est disponible également pour Python 3). Pour commencer, nous allons créer un fichier de configuration contenant les informations importantes pour utiliser Pocketsphinx (modèle, etc.) et les paramètres d'enregistrement du son. Deux dictionnaires permettront de :

- stocker une liste de mots et d'y associer un nom d'action ;
- stocker pour chaque nom d'action une liste de sons à jouer (on en prendra un de manière aléatoire), une commande shell (clé **cmd**) et une commande python (clé **python**).

Voici donc le fichier **config.py** :

```
01: # -*- coding:utf-8 -*-
02: import pyaudio
03:
04: HMM = '/usr/share/pocketsphinx/model/hmm/fr_FR/'
05: DIC = '/usr/share/pocketsphinx/model/lm/fr_FR/frenchWords62K.
dic'
06: LM= '/usr/share/pocketsphinx/model/lm/fr_FR/french3g62K.lm.dmp'
07:
08: TMPFILE = 'tmp.wav'
09:
10: CHUNK = 512
11: FORMAT = pyaudio.paALSA
12: CHANNELS = 1
13: RATE = 16000
14: RECORD_SECONDS = 2
15:
16: DETECTS = {
17:     'cité': 'quitter',
18:     'cités': 'quitter',
19:     'quitter': 'quitter',
20:     'quitté': 'quitter',
21:     'qui était': 'quitter',
22:     'fichier': 'fichier',
23:     'fichiers': 'fichier',
24:     'bonjour': 'bonjour',
25: }
26:
27: COMMANDS = {
28:     'bonjour': {
29:         'audio': ['yes1', 'yes2', 'yes3', 'yes4'],
30:     },
31:     'fichier': {
32:         'audio': ['execute1', 'execute2', 'execute3'],
33:         'cmd': 'ls'
34:     },
35:     'quitter': {
36:         'audio': ['execute1', 'execute2', 'execute3'],
37:         'python': 'exit(0)'
38:     },
39: }
```

Nous allons utiliser ce fichier pour configurer le programme principal **pocketDetect.py** :

```
01: # -*- coding:utf-8 -*-
02: #!/usr/bin/python
03:
04: import os
05: import pyaudio
06: import wave
07: import pocketsphinx as ps
08: import random
09: import config
10:
11:
12: def decodeSpeech(hmmd, lmdir, dictp, wavfile, detects,
commands):
13:     speechRec = ps.Decoder(hmm = hmmd, lm = lmdir, dict =
dictp)
14:     wavFile = file(wavfile,'rb')
15:     wavFile.seek(44)
16:     speechRec.decode_raw(wavFile)
17:     result = speechRec.get_hyp()
18:     print(result[0])
```

```
19:     if result[0] in detects:
20:         order = detects[result[0]]
21:         if 'audio' in commands[order]:
22:             cmd = 'aplay audio/{}.wav'.format(commands[order]
['audio'][random.randint(0, len(commands[order]['audio']) - 1)])
23:             os.system(cmd)
24:         if 'cmd' in commands[order]:
25:             cmd = commands[order]['cmd']
26:             os.system(cmd)
27:         if 'python' in commands[order]:
28:             eval(commands[order]['python'])
```

La fonction **decodeSpeech()** va lire le contenu du fichier **wavefile** et lancer l'interprétation. Dans **result[0]** (ligne 18) nous aurons le texte correspondant au contenu du fichier sonore (en tout cas ce que Pocketsphinx en aura compris). Nous regardons alors en ligne 19 si ce texte correspond à une commande du dictionnaire **detects** et si c'est le cas nous recherchons dans le dictionnaire **commands** un son à jouer (lignes 21 à 23) et éventuellement une commande shell (lignes 24 à 26) ou python à exécuter (lignes 27 et 28). Il s'agit d'un appel à **os.system()** ou **eval()**... donc à manipuler avec précautions.

```
31: if __name__ == '__main__':
32:     while True:
33:         p = pyaudio.PyAudio()
34:         stream = p.open(format=config.FORMAT, channels=config.
CHANNELS, rate=config.RATE, input=True, frames_per_buffer=config.
CHUNK)
35:         print('En écoute....')
36:         frames = []
37:         print str(config.RATE / config.CHUNK * config.RECORD_
SECONDS) + " size\n"
38:         for i in range(0, int(config.RATE / config.CHUNK *
config.RECORD_SECONDS)):
39:             data = stream.read(config.CHUNK)
40:             frames.append(data)
41:         os.system('aplay audio/ok.wav')
42:         stream.stop_stream()
43:         stream.close()
44:
45:         fic = wave.open(config.TMPFILE, 'wb')
46:         fic.setnchannels(config.CHANNELS)
47:         fic.setsampwidth(p.get_sample_size(config.FORMAT))
48:         p.terminate()
49:         fic.setframerate(config.RATE)
50:         fic.writeframes(b''.join(frames))
51:         fic.close()
52:
53:         decodeSpeech(config.HMM, config.LM, config.DIC,
config.TMPFILE, config.DETECTS, config.COMMANDS)
```

Le programme principal se contente d'effectuer une boucle infinie (ligne 32) pendant laquelle il enregistre le son du micro pendant **config.RECORD_SECONDS** dans le fichier **config.TMPFILE** (ligne 33 à 52) puis il appelle **decodeSpeech()** pour analyser ledit fichier.

Avec quelques modifications minimes, on pourrait obtenir la même architecture que Jasper avec un répertoire **modules** contenant des fichiers définissant chaque commande (plutôt que d'utiliser les dictionnaires **DETECTS** et **COMMANDS**). En passant un peu plus de temps pour utiliser une écoute en streaming plutôt qu'un enregistrement fixe, on recrée un Jasper en quelques lignes... mais un Jasper français !

Conclusion

Avec Jasper la reconnaissance vocale va énormément dépendre du moteur STT choisi (saluons au passage le choix d'architecture qui est tout à fait dans la philosophie du logiciel libre). J'ai pris le parti de n'utiliser que des solutions autonomes (ne nécessitant pas de connexion internet), mais rien ne vous empêche de tester les moteurs STT et TTS de Google (très performants) tout en étant conscient du parcours de vos informations...

La solution consistant à utiliser uniquement Pocketsphinx est intéressante, car elle permet la création d'une architecture qui nous est propre et qui peut donc évoluer en fonction de nos besoins. Et, cerise sur le gâteau : ça fonctionne en français sans trop de difficultés ! ■

Références

[1] Projet Pocketsphinx : https://github.com/cmusphinx/pocketsphinx

[2] Console Google Developpeur : https://console.developers.google.com/start

[3] API AT&T STT : http://developer.att.com/apis/speech

[4] Projet Wit.ai : https://wit.ai/

[5] Projet Julius : http://julius.osdn.jp/en_index.php

[6] Prcovic N., « Les machines de Turing », *GNU/Linux Magazine n°174*, septembre 2014, p. 24 à 31.

[7] Configurer Google STT pour Jasper : http://jasperproject.github.io/documentation/configuration/#google-stt

TESTS UNITAIRES EN CONDITIONS RÉELLES AVEC ARQUILLIAN

Romain PELISSE [Sustain Developer @Red Hat]

La communauté du projet open source JBoss [1] s'est construite autour de son serveur d'application, Wildfly [2], mais a aussi été le berceau de librairies et de frameworks très utiles aux développements logiciels, comme par exemple Hibernate [3]. Pour le besoin de tous ces projets, mais aussi pour assurer un développement rapide, efficace et sûr, la communauté est aussi à l'origine de nombreux frameworks et outils de développement, malheureusement moins connus.

Aujourd'hui, nous allons donc regarder, en détail, le « framework » de test Arquillian, et décrire en quoi il peut faciliter votre vie de tous les jours.

Mots-clés : Java/JEE, Maven, Junit, Tests, CDI

Résumé

Arquillian est un *framework* open source de tests, permettant de concevoir et d'exécuter des tests unitaires, mais au sein du conteneur « cible » (CDI, Tomcat, Wildfly) plutôt que de manière indépendante.

1 Arquillian

De nos jours, il est rare de concevoir une application Java qui ne nécessite que la machine virtuelle pour s'exécuter. L'application nécessite un conteneur, qu'il s'agisse d'un simple moteur de servlet comme Tomcat ou d'un conteneur JEE comme Wildfly. Néanmoins, depuis l'émergence de la spécification CDI, la tendance, dans le domaine du test, a été de privilégier les tests exécutés « hors conteneur ».

Malheureusement, si cette approche a permis un développement plus simple et plus rapide, elle a aussi, dans les dernières années, montré ses limites. Et il est rapidement apparu qu'il est en fait très pertinent de pouvoir tester un code au sein de son contexte d'exécution, pour peu qu'on dispose d'un *framework* adapté !

C'est donc là qu'intervient le *framework* Arquillian, son objectif étant de permettre la conception de tests unitaires, mais s'exécutant au sein du conteneur cible.

1.1 Premier test Arquillian

1.1.1 Code à tester

Nous allons partir d'un simple service Java, très minimaliste, que nous souhaitons donc tester de manière unitaire.

```
package org.belaran.belarquian.service;

import java.io.PrintStream;
import java.util.ArrayList;
import java.util.List;

public class TodoService {

  private List<ToDoItem> todos = new
ArrayList<>(0);

  public void printToDos(PrintStream to)
{
   for ( ToDoItem item : todos )
    to.println(item.toString());
    }

  public void addToDo(ToDoItem item) {
   todos.add(item);
  }

  public List<ToDoItem> getTodos() {
   return todos;
  }

  public void setTodos(List<ToDoItem>
todos) {
   this.todos = todos;
  }
}
```

1.1.2 Ajout des dépendances nécessaires au projet

Nous allons mettre en place notre premier test, autour de ce simple service, à l'aide du *framework* Arquillian. Il est donc nécessaire de commencer par ajouter au projet une dépendance vers ce dernier.

On ajoute aussi, au passage, une dépendance vers le *framework* **Junit [4]**, dont nous utiliserons l'API pour concevoir les tests, et une dépendance vers Arquillian lui-même :

```
$ git diff -w
diff --git a/pom.xml b/pom.xml
index dca48bb..4918bc3 100644
--- a/pom.xml
+++ b/pom.xml
@@ -26,6 +26,18 @@
         <version>4.12</version>
         <scope>test</scope>
       </dependency>
+      <dependency>
+        <groupId>org.jboss.arquillian</groupId>
+        <artifactId>arquillian-bom</artifactId>
+        <version>1.1.7.Final</version>
+        <type>bom</type>
+        <scope>import</scope>
+      </dependency>
+      <dependency>
+        <groupId>org.jboss.arquillian.junit</groupId>
+        <artifactId>arquillian-junit-container</artifactId>
+        <version>1.1.7.Final</version>
+      </dependency>
     </dependencies>
   </dependencyManagement>
   <dependencies>
@@ -40,6 +52,16 @@
       <artifactId>junit</artifactId>
       <scope>test</scope>
     </dependency>
+    <dependency>
+      <groupId>org.jboss.arquillian.junit</groupId>
+      <artifactId>arquillian-junit-container</artifactId>
+    </dependency>
   </dependencies>
   <build>
     <finalName>Belarquian</finalName
```

On notera que pour Arquillian, on n'ajoute pas une simple dépendance vers une archive jar, mais comme une dépendance de type « pom » (ou plutôt « BOM » ou *« Bill Of Materials »* **[5]**). Pour faire court, il s'agit d'un mécanisme de Maven, qui permet d'ajouter un ensemble de dépendances, cohérentes entre elles, sans devoir toutes les spécifier une à une.

Nous voilà maintenant équipés pour implémenter notre premier test unitaire. Avant d'aller plus loin, vérifions déjà que notre projet peut être construit, et que l'on peut générer une archive à partir de ses sources :

```
$ mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Belarquian 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
...
[INFO] Building war: /home/rpelisse/Repositories/contrib/arquillian-tutorial.git/target/
Belarquian.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

```
[INFO] Total time: 2.595 s
[INFO] Finished at: 2015-04-17T17:51:06+02:00
[INFO] Final Memory: 23M/982M

[INFO] ------------------------------------------------------------------------
```

1.1.3 Le test unitaire

Si cette construction en ligne de commandes est réussie, il n'y aucune raison pour que votre IDE ne vous permette pas de concevoir la classe de test suivante :

```
package org.belaran.belarquian.service;

import static org.junit.Assert.fail;

import org.junit.Test;

public class ToDoServiceTest {

  @Test
  public void test() {
    fail("Not yet implemented");
  }

}
```

Encore une fois, rien de très sorcier, et même un code très classique pour de la programmation en Java. Mais c'est maintenant que nous allons commencer à utiliser réellement Arquillian, et voir sa plus-value très appréciable dans le contexte du développement d'applicatif JEE.

1.2 Qu'est-ce qu'une « test-archive » ?

Avant d'aller plus loin, et de modifier notre test unitaire, abordons un concept important introduit par Arquillian : l'archive de test. Si vous avez déjà programmé en Java, sur un conteneur de « servlet » comme Tomcat ou un serveur applicatif Java/JEE comme Widfly, vous savez déjà que pour déployer une application il est nécessaire de l'empaqueter au sein d'une archive WAR.

Cette archive n'est, en fait, rien qu'un simple fichier compressé ZIP, doté d'une série de descripteurs, dont le contenu est normalisé par les spécifications du JSR **[6]**. La « test-archive » reprend en fait le même concept – il s'agit d'un format pour empaqueter des classes Java, mais orienté « test ».

La principale caractéristique de ces « test archives » est de n'embarquer que les dépendances nécessaires au test à exécuter, et non l'ensemble des dépendances de l'applicatif. Ceci rend les tests beaucoup plus faciles à gérer et beaucoup plus « propres ».

En outre, si Arquillian déploie ce genre d'archive sur le conteneur sur lequel on souhaite exécuter le test, ce déploiement est, le plus souvent possible, effectué sous la forme d'un flux de données, et non par le déploiement d'une archive physique (soit pas de fichier copié sur le système de fichiers du serveur cible).

Tout ceci est réalisé à l'aide de l' API Java nommée **ShrinkWrap [7]**, qui est au cœur des fonctionnalités de manipulations des ressources JEE que propose Arquillian. Comme nous allons l'illustrer ci-dessous, cette API permet non seulement de créer, de manière programmatique, au sein d'un test, une archive JEE pour empaqueter notre service, mais aussi d'ajouter, si besoin est, certains des descripteurs spécifiques à ces archives – sans passer par la conception d'un fichier de construction complet (comme celui de Maven).

1.3 Intégration avec Arquillian

1.3.1 Exécuter le test unitaire à l'aide d'Arquillian

Pour intégrer notre test unitaire JUnit avec les API d'Arquillian, et lui permettre ainsi d'être exécuté dans un conteneur (et non plus comme un simple test unitaire), la première étape est probablement sans surprise si vous avez déjà joué un peu avec les API du *framework* de test. En effet, cette étape consiste à ajouter une annotation **@RunWith**, issue du *framework* JUnit, qui indique simplement à ce dernier que l'exécuteur du test unitaire ne sera pas, comme d'habitude, le *Runner* de JUnit, mais une autre classe – dans notre *Arquillian.class* :

```
package org.belaran.belarquian.service;

import static org.junit.Assert.fail;

import org.jboss.arquillian.junit.
Arquillian;
import org.junit.Test;
import org.junit.runner.RunWith;
```

@RunWith(Arquillian.class)

```
public class ToDoServiceTest {

  @Test
  public void test() {
    fail("Not yet implemented");
  }

}
```

www.ed-diamond.com

ACTUELLEMENT EN PROMO

L'INTÉGRALE DE LA LIGNE DE COMMANDES...
DU NOVICE AU SORCIER !

PRIX NORMAL ~~38,70 €~~

PRIX PROMO 29 €

PYTHON : L'INTÉGRALE ! COMMENCER, CONTINUER AVEC L'OBJET ET DEVENIR UN EXPERT !

PRIX NORMAL ~~38,70 €~~

PRIX PROMO 29 €

NE MANQUEZ PAS

LINUX PRATIQUE HORS-SÉRIE N°34 !

PROTÉGEZ VOTRE VIE PRIVÉE

ET REPRENEZ LE CONTRÔLE DE VOS DONNÉES !

TOUJOURS DISPONIBLE

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

1.3.2 Ajout d'une méthode de déploiement

Comme évoqué plus haut, à l'aide des API de ShrinkWrap, Arquillian permet de définir une archive de test, et aussi d'en spécifier le contenu (en plus d'embarquer les classes nécessaires à l'exécution du test). Pour ce faire, on doit juste ajouter une méthode spécifique au *framework* de test, aussi annotée par l'instruction **`@Deployment`**.

```
package org.belaran.belarquian.service;

import static org.junit.Assert.fail;

import org.jboss.arquillian.container.test.api.Deployment;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.asset.EmptyAsset;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.junit.Test;
import org.junit.runner.RunWith;

@RunWith(Arquillian.class)
public class ToDoServiceTest {

  @Deployment
  public static JavaArchive createDeployment() {
    return ShrinkWrap.create(JavaArchive.class).
addClass(TodoService.class)
        .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml");
  }
...
}
```

L'API « fluide » proposée par ShrinkWrap est suffisamment explicite pour qu'on puisse rapidement comprendre le produit des quelques opérations réalisées dans cette nouvelle méthode. En quelques mots, on génère un JAR en guise de « test-archive » dans lequel on va déployer notre service ToDoService, et on ajoute juste le descripteur « beans.xml », toujours requis, même si, comme souvent, ce dernier est vide.

1.4 Injection du service dans le test

Passons maintenant à la rédaction du test en tant que tel :

```
@Inject
private TodoService todoService;

@Test
public void test() {
  todoService.addToDo(new ToDoItem(1L, "first item"));
  assertEquals(1, todoService.getTodos().size());
}
```

1.5 Exécuter un « bean » CDI avec Weld

N'oublions pas que l'intérêt principal d'Arquillian est de permettre d'effectuer des tests au sein d'un conteneur d'exécution, mais avec le même confort (ou presque) que lorsque nous exécutons de « simples » tests unitaires JUnit. Or, à ce stade, nous n'avons toujours pas indiqué à Arquillian quel sera l'environnement d'exécution, c'est-à-dire le conteneur à utiliser.

Pour commencer par un contexte d'exécution relativement simple, nous allons prendre le plus petit environnement possible pour notre service : un simple conteneur CDI, fournissant une implémentation de **`@Inject`**, et des cycles de vie associés, soit le projet **Weld**.

```
$ diff --git a/pom.xml b/pom.xml
index 6858bbf..0540d63 100644
--- a/pom.xml
+++ b/pom.xml
@@ -36,7 +36,31 @@
       <dependency>
         <groupId>org.jboss.arquillian.junit</groupId>
         <artifactId>arquillian-junit-container</artifactId>
-        <version>test</version>
+        <version>1.1.7.Final</version>
+      </dependency>
+      <dependency>
+        <groupId>org.jboss.arquillian.container</groupId>
+        <artifactId>arquillian-weld-ee-embedded-1.1</artifactId>
+        <version>1.0.0.CR8</version>
+        <scope>test</scope>
+        <exclusions>
+          <exclusion>
+            <groupId>org.jboss.arquillian.container</groupId>
+            <artifactId>arquillian-container-spi</artifactId>
+          </exclusion>
+        </exclusions>
+      </dependency>
+      <dependency>
+        <groupId>org.jboss.weld</groupId>
+        <artifactId>weld-core</artifactId>
+        <version>1.1.27.Final</version>
+        <scope>test</scope>
+      </dependency>
+      <dependency>
+        <groupId>org.slf4j</groupId>
+        <artifactId>slf4j-simple</artifactId>
+        <version>1.7.12</version>
+        <scope>test</scope>
       </dependency>
     </dependencies>
   </dependencyManagement>
```

```
@@ -56,6 +80,21 @@
      <groupId>org.jboss.arquillian.junit</groupId>
      <artifactId>arquillian-junit-container</artifactId>
    </dependency>
+   <dependency>
+     <groupId>org.jboss.arquillian.container</groupId>
+     <artifactId>arquillian-weld-ee-embedded-1.1</artifactId>
+     <scope>test</scope>
+   </dependency>
+   <dependency>
+     <groupId>org.jboss.weld</groupId>
+     <artifactId>weld-core</artifactId>
+     <scope>test</scope>
+   </dependency>
+   <dependency>
+     <groupId>org.slf4j</groupId>
+     <artifactId>slf4j-simple</artifactId>
+     <scope>test</scope>
+   </dependency>
  </dependencies>
  <build>

    <finalName>Belarquian</finalName>
```

1.6 Injection de dépendances

À moins d'avoir complètement raté les années 2004 à 2007, où une guerre larvée entre les conteneurs d'injection de dépendances tels que **Spring**, **JBoss Microcontainer** et le défunt **Hivemind**, se sont affrontés pour tenter – et réussir à offrir un modèle plus simple que celui des EJB sur le sujet de l'injection de dépendance, la plupart des développeurs Java/JEE – même arrivés sur le marché par la suite, savent que l'objectif d'un conteneur CDI est justement d'injecter des dépendances, lors de l'exécution.

Donc en soi, notre premier test réussi – s'il prouve le bon fonctionnement de notre projet d'exemple, est loin de démontrer l'exécution complète du conteneur CDI. C'est pour cette raison que, avant d'aller plus loin dans la découverte d'Arquillian et de ses fonctionnalités de tests unitaires, nous allons rendre notre service d'exemple un peu plus réaliste, et lui injecter une dépendance.

Comme notre service est dédié à gérer les données d'une liste de « ToDo », il semble pertinent de lui injecter un moyen de « persistance » – soit une API lui permettant de mettre à jour son état. Pour faire plus simple, et plus moderne, que l'utilisation d'un fichier – aux primitives d'interaction beaucoup trop bas niveau pour notre cas, ou que celui d'une base de données relationnelle, là beaucoup plus élaboré, nous allons donc simplement utiliser une base de données de type « NoSQL ».

Fidèle à une tradition de plusieurs articles, nous allons donc utiliser le projet **Infinispan**, qui permet de placer des données dans un cache mémoire, éventuellement persisté sur disque. Le lecteur peu familier sur ce projet pourra se référer aux précédents articles, ou simplement étudier la documentation du site du projet.

Pour faire simple, voilà juste le résultat de la commande **`git diff`** après ajout des dépendances requises pour utiliser Infinispan :

```
diff --git a/pom.xml b/pom.xml
index 0540d63..a999b2f 100644
--- a/pom.xml
+++ b/pom.xml
@@ -10,6 +10,8 @@
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.
sourceEncoding>
+   <infinispan.version>7.1.0.Final</infinispan.version>
+   <infinispan-jcache.version>7.0.0.Final</infinispan-jcache.
version>
  </properties>
  <dependencyManagement>
    <dependencies>
@@ -71,6 +73,19 @@
      <type>pom</type>
      <scope>provided</scope>
    </dependency>
+
+    <dependency>
+      <groupId>org.infinispan</groupId>
+      <artifactId>infinispan-embedded</artifactId>
+      <version>${infinispan.version}</version>
+    </dependency>
+
+    <dependency>
+         <groupId>org.infinispan</groupId>
+         <artifactId>infinispan-jcache</artifactId>
+         <version>${infinispan-jcache.version}</version>
+    </dependency>
+
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
```

On note que dans cette première approche, on utilise une dépendance vers **`infinispan-embedded`**, et non vers **`infinispan-core`**, car elle est nécessaire dans le contexte d'exécution d'un simple conteneur CDI. Un peu plus loin, on changera cette dépendance vers **`infinispan-core`**, lorsque l'on déploiera le code sur un conteneur JEE.

2 Tester au sein d'un conteneur JEE

L'un des intérêts clés d'Arquillian et de permettre de tester le code directement au sein du conteneur d'exécution cible (comme par exemple Tomcat ou Wildfly), et non simplement de manière unitaire. Ceci est un point important, car l'expérience a prouvé depuis longtemps que si une fonctionnalité peut être implémentée à la perfection de manière unitaire, et s'exécute sans problème dans une simple JVM, il peut en être tout autre chose au sein du conteneur.

2.1 De l'intérêt de tester au sein du conteneur

Quelques exemples, rapides, de ce qui peut tourner mal au sein du conteneur. Le premier exemple est le plus évident : le `CLASSPATH`. Il ne faut pas oublier en effet que c'est au sein de ce dernier que la JVM recherche la définition des classes à instancier – à la suite des demandes du code exécuté, et que le contenu de ce dernier peut changer de manière drastique lors de l'exécution au sein d'un conteneur !

En effet, le `CLASSPATH` d'un serveur applicatif contient de nombreuses classes pour son fonctionnement interne. Si, de nos jours, les serveurs JEE comme Wildfly tentent, avec dans son cas les **JBossModule [8]**, de retirer les classes et dépendances non nécessaires du `CLASSPATH` de chaque applicatif, il n'en reste pas moins que ce n'est pas toujours une tâche aisée, ou même simplement possible.

En outre, les fonctionnalités des applications déployées sur ce genre de conteneur s'appuient naturellement sur les API et services de ce dernier. Pour prendre un exemple trivial, si vous implémentez une Servlet au sein de laquelle, pour des raisons applicatives, vous faites des opérations un peu complexes sur le contenu des *headers* HTTP, vous n'aurez pas d'autre choix que de tester au sein du conteneur...

Un autre exemple rapide, pour en finir avec cette démonstration. Un applicatif peut très bien se comporter correctement (bonne performance et empreinte mémoire) au sein d'un test unitaire, mais avoir un comportement très différent, lors de l'exécution sur le conteneur. Prenons le cas, pourtant de prime abord relativement simple, d'un appel à une base de données.

Lorsque l'appel retourne un vaste ensemble de données, il est souvent considéré comme une bonne pratique de programmation de prévoir un *buffer* assez grand, et donc d'allouer en avance beaucoup de mémoire. Dans le cas de Java, on peut par exemple allouer un tableau de `100` éléments.

Malheureusement, ou heureusement, on trouve aussi ce genre d'optimisation du côté du conteneur et de ses pilotes JDBC. Du coup, en allouant un tableau de `100` éléments, on en alloue en fait `100 * 100` ! Et on se retrouve rapidement, lors des tests de charge, avec de gros problèmes de performance et de charge, qui ne sont jamais apparus hors du conteneur !

Bref, pour faire court, tester son code sur l'environnement cible (même système d'exploitation, même version de JVM, et dans le bon contexte d'exécution) est essentiel.

2.2 Embarqué ou pas embarqué

Avant de reprendre la partie pratique, un autre point est à considérer. Pour exécuter son code au sein d'un conteneur, Arquillian permet à ses extensions de choisir deux approches :

- conteneur embarqué (*embedded container*) ;
- conteneur géré (*managed container*).

Pour faire court, surtout que ce point est très bien décrit dans un article de Dan Allen **[9]**, le même raisonnement que précédemment établi s'applique. Il est toujours plus pertinent de tester dans un environnement d'exécution le plus proche possible de la cible, et il est donc recommandé de ne pas utiliser un conteneur embarqué, mais d'installer, de manière séparée de la « test suite », le conteneur cible et de configurer Arquillian, comme nous allons maintenant le faire, pour utiliser ce dernier.

2.3 Modifier les dépendances de l'applicatif

Le contexte d'exécution de notre petit applicatif vient de changer grandement. Pour notre code, simplissime à souhait, ça ne change pas grand-chose, mais pour sa dépendance vers Infinispan, il est nécessaire de procéder à deux ajustements :

```
$ git show 9eb8905cda122f4914842193bbc681e817d06037
commit 9eb8905cda122f4914842193bbc681e817d06037
Author: Romain PELISSE <belaran@gmail.com>
Date:   Tue Apr 21 14:06:36 2015 +0200

    fix ispn runtime to run into Wildfly as a container
```

```
diff --git a/pom.xml b/pom.xml
index 80250b4..1b9d9af 100644
--- a/pom.xml
+++ b/pom.xml
@@ -12,6 +12,7 @@
     <project.build.sourceEncoding>UTF-8</project.build.
sourceEncoding>
     <infinispan.version>7.1.0.Final</infinispan.version>
     <infinispan-jcache.version>7.0.0.Final</infinispan-jcache.
version>

+    <infinispan.cdi.version>7.1.0.Final</infinispan.cdi.version>
   </properties>
   <dependencyManagement>
     <dependencies>
@@ -45,14 +46,19 @@
   <dependencies>
     <dependency><groupId>org.infinispan</groupId>
-      <artifactId>infinispan-embedded</artifactId>
+      <artifactId>infinispan-core</artifactId>
       <version>${infinispan.version}</version>
     </dependency>
     <dependency>
       <groupId>org.infinispan</groupId>
       <artifactId>infinispan-jcache</artifactId>
       <version>${infinispan-jcache.version}</version>
-   </dependency>
+    </dependency>
+    <dependency>
+      <groupId>org.infinispan</groupId>
+      <artifactId>infinispan-cdi</artifactId>
+      <version>${infinispan.cdi.version}</version>
+    </dependency>
     <dependency>
       <groupId>junit</groupId>
       <artifactId>junit</artifactId>
```

Le changement de dépendance vers **infinispan-core** a déjà été évoqué plus haut. S'ajoute ici une dépendance pour assurer l'intégration du composant avec le conteneur CDI du serveur JEE Wildfly.

Conclusion

De la manière la moins intrusive possible, le *framework* Arquillian vous permet donc de facilement concevoir des tests à réaliser au sein d'un conteneur d'exécution. En outre, son intégration avec des *frameworks* de tests unitaires existant, comme JUnit, ne force pas un changement des habitudes de développement.

Si la conception de la « test-archive » n'est pas forcément triviale, elle permet aux développeurs de réellement maîtriser le contenu du **CLASSPATH** de ses tests, et ainsi de tester dans des conditions « proches du réel », mais aussi libres des interférences et du poids à l'exécution, les autres dépendances embarquées dans l'applicatif.

Au final, tout ceci reste complexe, et demande un travail de « plomberie » un peu pénible, mais Arquillian vous permet clairement de tirer le plus possible d'une situation pénible. À l'aide d'un simple « bom », le *framework* masque la complexité qu'il embarque du projet, et à l'aide d'une simple annotation, il permet de changer complètement le contexte d'exécution d'un test unitaire, etc.

En outre, si cet article se focalise sur les capacités d'exécution, au sein d'un conteneur, et d'injection du *framework*, il est important de noter que Arquillian offre aussi de puissantes fonctionnalités pour automatiser les tests d'applications Web ou même de services distants... Mais ceci est, naturellement, le sujet d'un autre article. ■

Références

[1] Communauté « Open Source » JBoss : http://www.jboss.org/

[2] Serveur d'applications JEE « Open Source » Wildfly (anciennement « JBoss AS ») : http://wildfly.org//

[3] Hibernate ORM : http://hibernate.org/

[4] JUnit : http://junit.org/

[5] Notion de « BOM » avec Maven : http://blog.xebia.fr/2014/02/28/la-notion-de-bom-avec-maven/

[6] Java Specification Request : https://jcp.org/en/jsr/overview

[7] ShrinkWrap : https://developer.jboss.org/wiki/ShrinkWrap

[8] JBoss Module : https://docs.jboss.org/author/display/MODULES/Home

[9] « Dangers of embedded containers » : http://arquillian.org/blog/2012/04/13/the-danger-of-embedded-containers/

PROFESSIONNELS DES TICE, COLLECTIVITÉS, ÉCOLES D'INGÉNIEURS, UNIVERSITÉS, R & D, ENSEIGNANTS, ...

LINUX MAGAZINE / FRANCE

PROFESSIONNELS !

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS ...

...EN VOUS CONNECTANT À L'ESPACE DÉDIÉ AUX PROFESSIONNELS SUR :

www.ed-diamond.com

PDF COLLECTIFS PRO

Prix TTC en Euros / France Métropolitaine

OFFRE	ABONNEMENT	1 - 5 lecteurs Réf	1 - 5 lecteurs Tarif TTC	6 - 10 lecteurs Réf	6 - 10 lecteurs Tarif TTC	11 - 25 lecteurs Réf	11 - 25 lecteurs Tarif TTC
PROLM2	11n° GLMF	☐ PRO LM2/5	260,-	☐ PRO LM2/10	520,-	☐ PRO LM2/25	1040,-
PROLM+2	11n° GLMF + 6n° HS	☐ PRO LM+2/5	472,-	☐ PRO LM+2/10	944,-	☐ PRO LM+2/25	1888,-

PROFESSIONNELS :
N'HÉSITEZ PAS À NOUS CONTACTER POUR UN DEVIS PERSONNALISÉ PAR E-MAIL :
abopro@ed-diamond.com
OU PAR TÉLÉPHONE :
03 67 10 00 20

ACCÈS COLLECTIFS BASE DOCUMENTAIRE PRO

Prix TTC en Euros / France Métropolitaine

OFFRE	ABONNEMENT	1 - 5 connexion(s) Réf	1 - 5 connexion(s) Tarif TTC	6 - 10 connexions Réf	6 - 10 connexions Tarif TTC	11 - 25 connexions Réf	11 - 25 connexions Tarif TTC
PROLM+3	GLMF + HS	☐ PRO LM+3/5	267,-	☐ PRO LM+3/10	534,-	☐ PRO LM+3/25	1068,-
PROA+3	GLMF + HS + LP + HS	☐ PRO A+3/5	297,-	☐ PRO A+3/10	594,-	☐ PRO A+3/25	1188,-
PROH+3	GLMF + HS + LP + HS + MISC + HS + OS	☐ PRO H+3/5	447,-	☐ PRO H+3/10	894,-	☐ PRO H+3/25	1788,-

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France HS = Hors-Série LP = Linux Pratique OS = Open Silicium

SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE CI-DESSUS ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	

☐ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.
☐ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

LINUX MAGAZINE / FRANCE

Édité par Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

L'ANALYSE DE DONNÉES EN PYTHON OU COMMENT FAIRE DU R SANS R

Tristan COLOMBO

Vous maîtrisez Python, les possibilités offertes par R vous paraissent très intéressantes, mais vous n'avez pas envie (ou le temps) d'apprendre un nouveau langage ? Essayez avec Pandas et les modules scientifiques !

Mots-clés : Pandas, Python, R, rpy2, Scientifique, Analyse de données

Pandas est un paquetage Python permettant de manipuler des données puis de réaliser simplement leur analyse... pour peu que l'on soit un minimum familier de **NumPy** et **Matplotlib**. Si ce que vous cherchez à faire se borne à appeler des fonctions R depuis Python, ce n'est pas Pandas qu'il faudra utiliser, mais **rpy2**. Pour ne pas vous frustrer, même si cela sort du champ de cet article, nous allons commencer par un aperçu très rapide de rpy2 en traduisant un exemple R puis nous utiliserons Pandas de la même manière, en partant là aussi d'un exemple.

1 rpy2 : attaquer R depuis Python

L'installation se fait de manière classique en utilisant **pip** :

```
# pip3 install rpy2
```

Nous allons calculer la corrélation de Pearson permettant de répondre à la question suivante : existe-t-il un lien entre le poids et la pointure des sujets étudiés ? Les données sont contenues dans un fichier **pearson.csv** (pour les lecteurs attentifs, il s'agit du même exemple que celui présenté dans le hors-série « Analyses de données et Big Data » **[1]**) :

```
"Poids (en kg)" ;"Pointure"
99;47
55;36
60;38
...
71;39
60;39
82;43
```

Voici le code R permettant de calculer cette corrélation :

```
data=read.csv("pearson.csv", sep=";")
cor(data[,1], data[,2])
```

Et voici la même chose en Python avec rpy2 :

```
>>> import rpy2.robjects as robjects
>>> robjects.r('data <- read.csv("pearson.csv", sep=";")')
...
>>> robjects.r('cor(data[,1], data[,2])')
<FloatVector - Python:0x7f1284103b88 / R:0x2466ad8>
[0.938764]
```

Pas de surprise, nous obtenons bien le même résultat... mais comme vous l'avez vu, nous n'avons fait qu'écrire du R dans du Python. Avec cette solution, il faut donc à la fois maîtriser R et Python. Si cela ne vous embarrasse pas et que vous êtes un adepte de IPython, vous pourrez em-

ployer l'extension **Rmagic** permettant d'utiliser rpy2 avec des commandes magiques :

```
In [1]: load_ext rmagic
```

%R permet d'appeler des commandes R, **%Rpush** envoie des variables Python vers R et **%Rpull** réalise l'opération inverse.

Voyons maintenant une solution qui ne requiert que des connaissances Python.

2 Pandas

Pandas s'installe classiquement par **pip** :

```
# pip3 install pandas
```

Patientez quelques instants pendant la compilation, le paquetage étant quelque peu imposant.

Le chargement d'un fichier csv se fait aussi simplement qu'en R. Par souci de clarté, j'utiliserai un shell Python3 :

```
>>> import pandas as pd
>>> data = pd.read_csv('pearson.csv', sep=';')
>>> data
    Poids (en kg)  Pointure
0              99        47
1              55        36
2              60        38
3              83        45
...
18             60        39
19             82        43
```

L'accès aux colonnes se fait simplement à partir de leur en-tête :

```
>>> data.Pointure
0     47
...
19    43
Name: Pointure, dtype: int64
```

Si vos en-têtes comportent des espaces, vous ne pourrez pas utiliser ce raccourci et il faudra alors employer la forme « dictionnaire » :

```
>>> data['Poids (en kg)']
0      99
...
19     82
Name: Poids (en kg), dtype: int64
```

La sélection sous condition(s) est très simple à écrire : la clé employée contiendra la condition sur la ou les colonnes. Par exemple, pour obtenir la liste des données telle que la pointure soit supérieure à 43, il faudra écrire :

```
>>> data[data['Pointure'] > 43]
    Poids (en kg)  Pointure
0              99        47
3              83        45
4             110        47
12             80        44
15             92        45
```

Il est possible d'utiliser le *slicing* pour ne récupérer que certaines lignes :

```
>>> data[5:8]
   Poids (en kg)  Pointure
5             58        36
6             90        42
7             70        39
```

Pour visualiser seulement une partie du début ou de la fin des données, nous pouvons utiliser respectivement les fonctions **head()** et **tail()** qui fonctionnent exactement comme dans le shell (vous pouvez ou non spécifier le nombre de lignes à afficher) :

```
>>> data.head()
   Poids (en kg)  Pointure
0             99        47
1             55        36
2             60        38
3             83        45
4            110        47
>>> data.head(1)
   Poids (en kg)  Pointure
0             99        47
>>> data.tail()
    Poids (en kg)  Pointure
15             92        45
16             78        40
17             71        39
18             60        39
19             82        43
```

Nous pouvons également obtenir des informations détaillées sur les données à la manière du **summary()** de R (qui s'appellera ici **describe()**). Pour mémoire, voici le code et le résultat sous R :

```
> summary(regression)
 Poids..en.kg.       Pointure
 Min.   : 55.00   Min.   :36.00
 1st Qu.: 59.75   1st Qu.:38.00
 Median : 70.50   Median :39.50
 Mean   : 73.80   Mean   :40.75
```

```
3rd Qu.: 83.25   3rd Qu.:43.25
    Max.   :110.00   Max.   :47.00
```

Et la même chose avec Python et Pandas :

```
>>> data.describe()
       Poids (en kg)   Pointure
count      20.000000  20.000000
mean       73.800000  40.750000
std        15.988812   3.492473
min        55.000000  36.000000
25%        59.750000  38.000000
50%        70.500000  39.500000
75%        83.250000  43.250000
max       110.000000  47.000000
```

std est l'écart-type : en moyenne, les poids se situent entre la moyenne (**mean = 73,8**) **- 15,98** et la moyenne **+ 15,98**. **25%** est le premier quartile : 25% de la population à un poids inférieur à 59,75 kg. **50%** est le deuxième quartile correspondant également à la médiane, ce qui signifie que 50% de l'échantillon a un poids inférieur à 70,50 kg. Et bien sûr, **75%** est le troisième quartile. Enfin, **min** et **max** sont les valeurs minimales et maximales de chaque colonne.

Notez enfin la possibilité de travailler directement avec des fichiers HDF5 **[2][3]**. On peut ainsi écrire les données dans un tel fichier :

```
>>> data.to_hdf('data.h5', 'data')
```

Ou encore les lire :

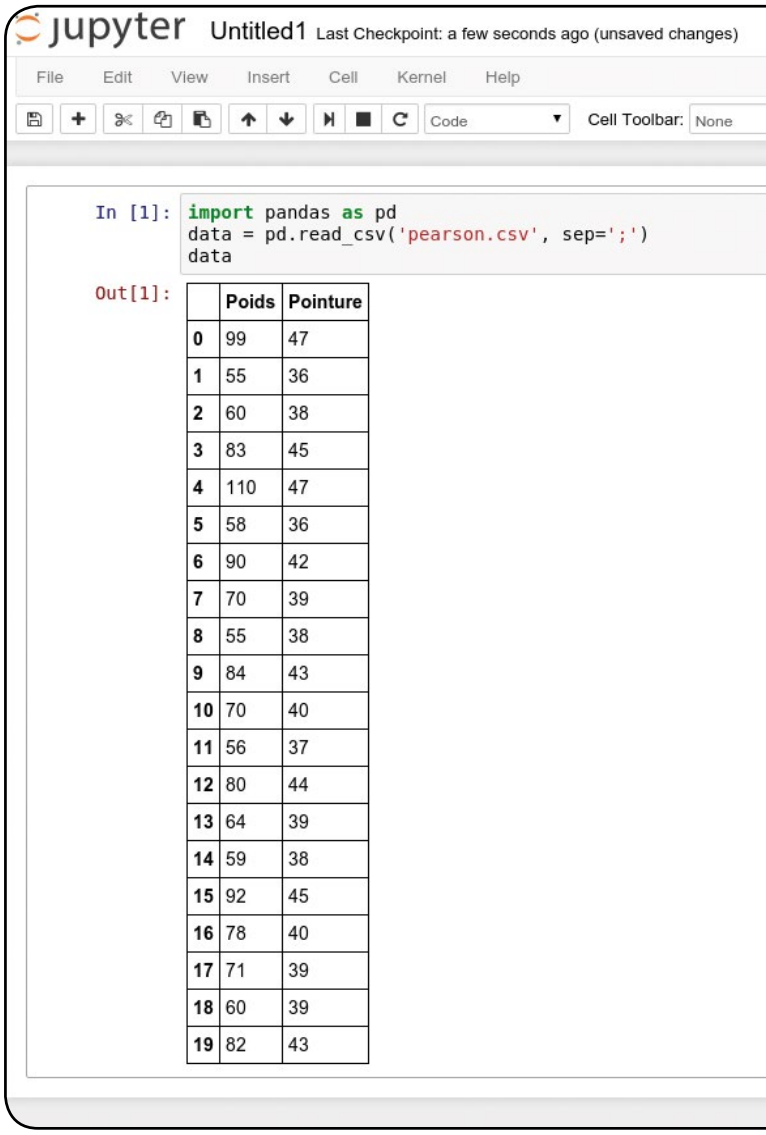
```
>>> data = pd.read_hdf('data.h5', 'data')
```

Comme vous avez pu le voir, Pandas permet de manipuler très simplement les données. Nous allons maintenant observer des fonctionnalités un peu plus avancées.

2.1 Grouper des données

La fonction **groupby()** permet de regrouper des lignes portant une même valeur de colonne et d'effectuer une opération sur les colonnes suivantes. Par exemple, supposons que l'on souhaite connaître le poids moyen des personnes chaussant du 39. Il faut tout d'abord savoir quelle est la taille de l'effectif :

```
>>> nb = data[data['Pointure'] == 39].count()
>>> nb
Poids (en kg)    4
Pointure         4
dtype: int64
```

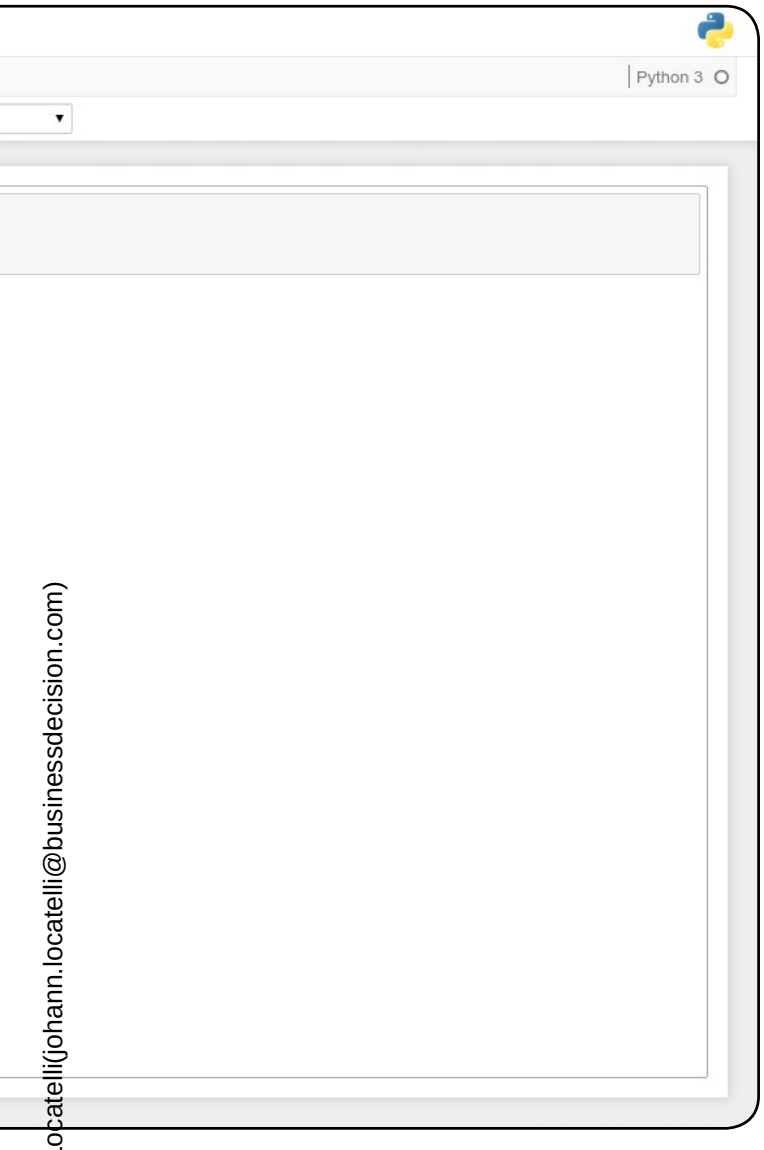


	Poids	Pointure
0	99	47
1	55	36
2	60	38
3	83	45
4	110	47
5	58	36
6	90	42
7	70	39
8	55	38
9	84	43
10	70	40
11	56	37
12	80	44
13	64	39
14	59	38
15	92	45
16	78	40
17	71	39
18	60	39
19	82	43

Fig. 1 : Affichage d'un tableau de données depuis un notebook IPython/Jupyter.

Ensuite, nous sommons les poids par pointure :

```
>>> sum_pointure = data.groupby('Pointure').sum()
>>> sum_pointure
          Poids (en kg)
Pointure
36                  113
37                   56
38                  174
39                  265
40                  148
42                   90
43                  166
44                   80
45                  175
47                  209
```



Nous extrayons alors le poids total pour la pointure 39 en commençant par réaliser une transposée :

```
>>> sum_pointure.T
Pointure        36  37   38   39   40  42   43  44   45   47
Poids (en kg)  113  56  174  265  148  90  166  80  175  209
>>> sum_pointure.T[39] Poids (en kg)    265
Name: 39, dtype: int64
>>> sum_pointure.T[39][0] / nb[0]
66.25
```

Donc en moyenne les personnes chaussant du 39 pèsent 66,25 kg.

Le shell Python est pratique, mais il ne faut pas oublier qu'IPython a été développé pour remplacer Matlab. Ainsi, en utilisant le notebook (lancement par **ipython3 notebook** ou **jupyter notebook** si vous avez installé la dernière version) l'affichage des données est beaucoup plus élégant comme on peut le voir sur la figure 1.

2.2 Et si des données sont manquantes ?

Il est courant de vouloir traiter des fichiers dans lesquels certaines données sont absentes. Prenons le cas d'un sondage où trois questions sont posées et les réponses attendues sont numériques. Certaines personnes refusent de répondre à une ou plusieurs questions et le fichier **sondage.csv** contenant les résultats est alors le suivant :

```
;"Q1";"Q2";"Q3"
1;12;4;0
2;5;;7
3;22;;
4;0;1;1
```

Vous vous attendez à un message d'erreur ? Les données absentes seront considérées... absentes et remplacées par **NaN** :

```
>>> data = pd.read_csv('sondage.csv', sep=';')
>>> data
Unnamed: 0  Q1  Q2  Q3
0           1  12   4   0
1           2   5 NaN   7
2           3  22 NaN NaN
3           4   0   1   1
```

Lors des calculs ces données seront ignorées comme on peut le voir en utilisant la méthode **describe()** :

```
>>> data.describe()
       Unnamed: 0         Q1       Q2       Q3
count    4.000000   4.000000  2.00000  3.000000
mean     2.500000   9.750000  2.50000  2.666667
std      1.290994   9.535023  2.12132  3.785939
min      1.000000   0.000000  1.00000  0.000000
25%      1.750000   3.750000  1.75000  0.500000
50%      2.500000   8.500000  2.50000  1.000000
75%      3.250000  14.500000  3.25000  4.000000
max      4.000000  22.000000  4.00000  7.000000
```

Ici la moyenne des réponses à la question **Q2** vaut **2,5** ce qui montre bien que les données **NaN** n'ont pas été prises en compte.

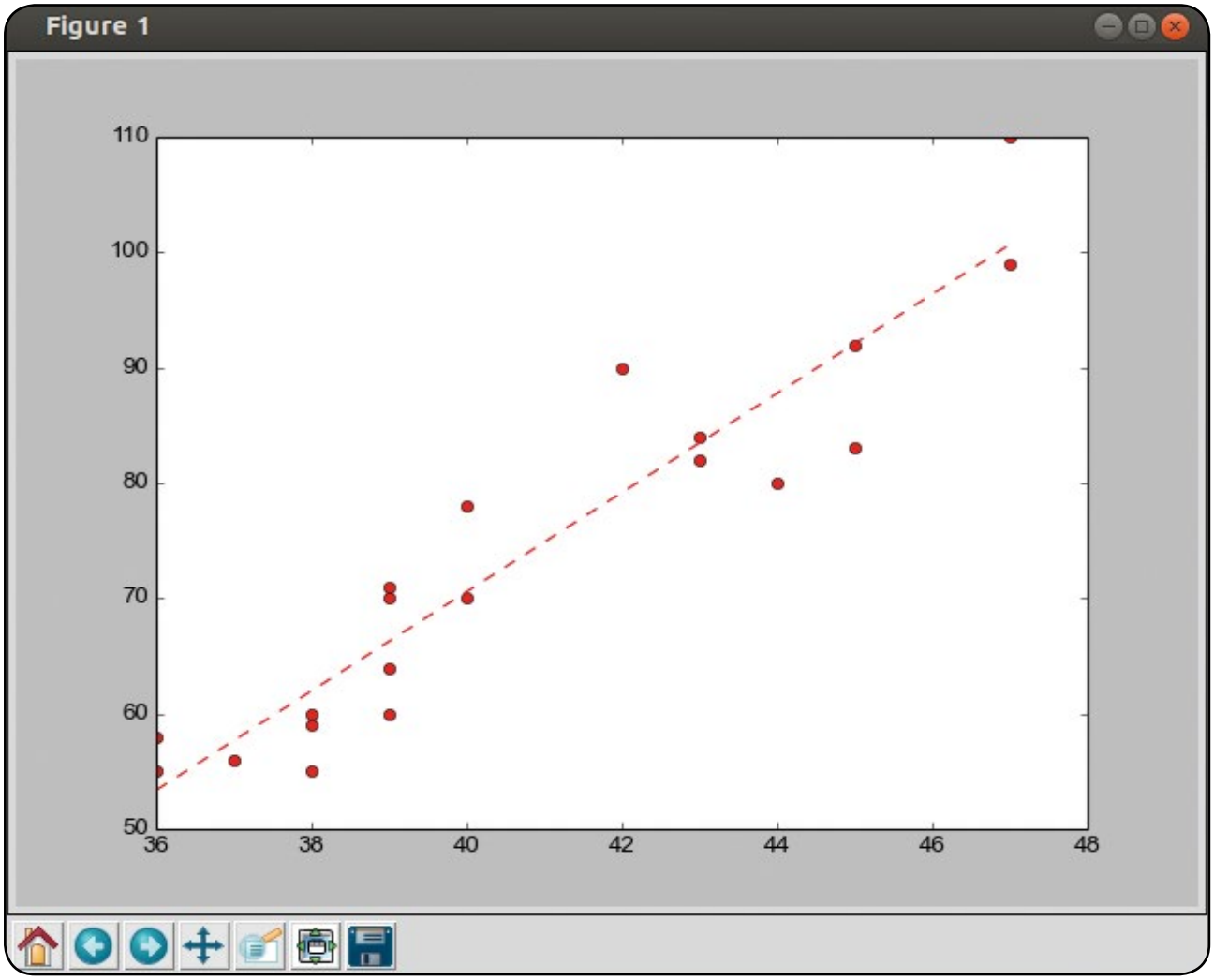


Fig. 2 : Nuage de points et droite de régression des données data.

3 Calcul d'une régression linéaire avec NumPy

Nous allons maintenant calculer une régression linéaire. Pandas nous permet de manipuler nos données, il faut maintenant utiliser d'autres modules pour le calcul et l'affichage. Dans cet exemple, nous utiliserons **numpy** et **matplotlib**, mais vous pouvez bien entendu choisir n'importe quel autre module de calcul et de dessin.

```
>>> import pandas as pd
>>> data = pd.read_csv('pearson.csv', sep=';')
>>> import numpy as np
>>> reg = np.polyfit(data['Pointure'], data['Poids (en kg)'], 1)
>>> reg
   array([   4.29773463, -101.33268608])
```

Pour le tracé des données, on va utiliser **matplotlib** :

```
>>> import pylab
```

On définit ensuite l'équation de la droite de régression :

```
>>> f = lambda x: reg[0] * x + reg[1]
```

On détermine alors les coordonnées de deux points pour pouvoir tracer la droite :

```
>>> reg_x = (data['Pointure'].min(), data['Pointure'].max())
>>> reg_y = [f(x) for x in reg_x]
```

Il n'y a plus ensuite qu'à tracer le graphe (voir résultat en figure 2) :

```
>>> pylab.plot(data['Pointure'],
data['Poids (en kg)'], 'ro', reg_x,
reg_y, 'r--')
[<matplotlib.lines.Line2D object at
0x7f4fb2979198>, <matplotlib.lines.Line2D
object at 0x7f4fb2979630>]
>>> pylab.show()
```

4 Calcul d'une régression linéaire avec StatsModels

StatsModels [4] est un paquetage de calcul statistique pour Python qui doit être installé depuis **pip** :

```
# pip3 install statsmodels
```

Nous allons ici pouvoir exprimer les formules de la même manière qu'en R et pour en simplifier la lecture (et l'écriture, ne nous le cachons point), j'ai modifié l'en-tête de la première colonne du fichier **pearson.csv** :

```
>>> import pandas as pd
>>> data = pd.read_csv('pearson.csv',
sep=';')
>>> data
    Poids  Pointure
0      99        47
...
19     82        43
```

Nous allons ensuite créer un modèle en indiquant une formule et les données qui serviront pour les calculs :

```
>>> import statsmodels.formula.api as smf
>>> mod = smf.
ols(formula='Poids~Pointure', data=data)
```

Le calcul de régression se fait sur le modèle en appliquant la méthode **fit()** :

```
>>> res = mod.fit()
>>> print(res.summary())

OLS Regression Results
==============================================
Dep. Variable:                  Poids   R-squared:                       0.881
Model:                            OLS   Adj. R-squared:                  0.875
Method:                 Least Squares   F-statistic:                     133.6
Date:                Fri, 10 Apr 2015   Prob (F-statistic):           9.20e-10
Time:                        18:04:57   Log-Likelihood:                -61.994
No. Observations:                  20   AIC:                             128.0
Df Residuals:                      18   BIC:                             130.0
Df Model:                           1
Covariance Type:            nonrobust
==============================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
----------------------------------------------------------------------------
Intercept   -101.3327     15.204     -6.665      0.000      -133.275   -69.391
Pointure       4.2977      0.372     11.559      0.000         3.517     5.079
==============================================
Omnibus:                        0.479   Durbin-Watson:                   1.701
Prob(Omnibus):                  0.787   Jarque-Bera (JB):                0.571
Skew:                           0.288   Prob(JB):                        0.752
Kurtosis:                       2.406   Cond. No.                         492.
==============================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Comme vous le voyez, il y a beaucoup d'informations. Nous retrouvons les paramètres nous permettant de construire notre équation de droite dans l'attribut **params** :

```
>>> res.params
Intercept   -101.332686
Pointure       4.297735
dtype: float64
```

Pour faire un diagramme depuis ces résultats, il suffira d'appliquer la même méthode que précédemment.

Pour découvrir rapidement les nombreuses possibilités offertes par StatsModels, vous pourrez consulter la page des exemples **[5]** qui est assez complète et détaillée.

Conclusion

Le groupe Pandas, NumPy, SciPy, StatsModels et Matplotlib (ou VisPy) permet aux pythonistes de réaliser les mêmes opérations qu'avec R sans avoir à passer du temps dans l'acquisition d'un nouveau langage. De plus, si vous êtes déjà familier de tel ou tel module, vous pourrez aisément l'utiliser et ne serez donc pas contraint à des apprentissages supplémentaires. Python montre encore une fois ici toute sa souplesse. ■

Références

[1] Arnaud L., « Analyse statistique de données avec R », GNU/Linux Magazine HS n°78, 2015, p. 94 à 115.

[2] Site du hdfgroup : http://www.hdfgroup.org/HDF5/

[3] Colombo T., « Python et le format HDF5 », *GNU/Linux Magazine HS n°73*, 2014, p. 30 à 41.

[4] Paquetage StatsModels « Statistics in Python » : http://statsmodels.sourceforge.net/stable/index.html

[5] Exemples StatsModels : http://statsmodels.sourceforge.net/stable/examples/index.html

ANDROID LAYOUT : CONNAISSIEZ-VOUS L'ATTRIBUT « TOOLS » ?

Ramel ADJIBI [Ingénieur en Systèmes d'information, spécialiste techno mobile Android]

Lors de vos développements d'applications Android, vous avez probablement rencontré le namespace "xmlns:tools="http://schemas.android.com.tools"dans vos fichiers de vue générés par Android Studio. Mais à quoi peut-il bien servir ? Essayons d'y répondre dans la suite...

Mots-clés : Android, Java, Layout, Tools, Android Studio, DesignTime

Résumé

Android a un *namespace* dédié à l'utilisation de l'attribut tools qui permet de sauvegarder des données dans les fichiers XML ; ces données sont enlevées de l'application lors de la compilation donc elles ne sont plus visibles à l'exécution. Vous apprendrez dans cet article comment l'utiliser à votre avantage.

L'attribut `tools` **[1]** est souvent lié au *namespace* `xmlns:tools="http://schemas.android.com.tools"` et vous donne la possibilité :

- d'ajouter des métadonnées à votre vue ;
- de faciliter et tester le rendu de votre vue ;
- de supprimer certains avertissements de `Lint`.

1 De l'attribut tools à la conception de vue : tester le rendu avant la compilation

Avec l'attribut `tools`, vous pouvez surcharger des attributs habituels qui vous permettent de changer le rendu d'une vue dans **Android Studio**, sans passer par la compilation du code. Regardez l'exemple de code ci-contre :

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/txt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        tools:text="Text affiché grace à l'attribut tools" />
</FrameLayout>
```

Ici, le parent `FrameLayout` contient le *namespace* `tools` et l'élément `TextView` a l'attribut `tools:text="Texte affiché grâce à l'attribut tools"` ; dans l'onglet ***Aperçu*** de Android Studio, on obtient un résultat similaire à celui présenté en figure 1.

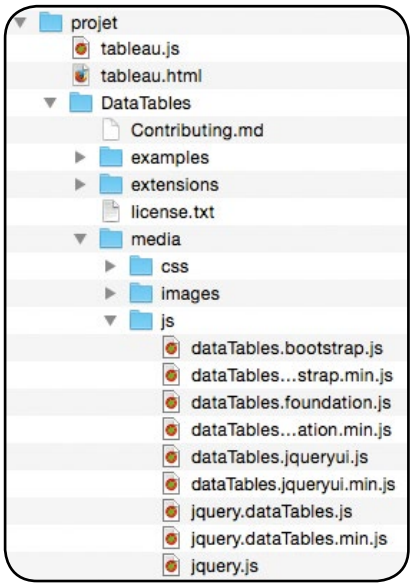


Fig. 1 : Résultat obtenu après ajout de « tools:text » dans le layout.

Sur l'écran de la figure 1, on remarque qu'on obtient le même résultat que si on mettait juste **`android:text="Texte affiché grâce à l'attribut tools"`** ; quel est l'intérêt de surcharger **`android:text`** avec **`tools:text`** si c'est pour obtenir le même résultat ? Eh bien, la différence réside dans le fait qu'avec **`tools:text`**, vous pouvez mettre tout ce que vous voulez comme chaîne de caractères de test sans vous soucier du fait que cela puisse apparaître à l'exécution : toutes les valeurs que vous passerez à **`tools:text`** ne seront visibles que dans l'aperçu des vues dans Android Studio et ne se verront pas à l'exécution de votre application ; voilà un des intérêts ! Il arrive souvent que l'on veuille tester un **`textview`** avec du « faux » texte, mais on oublie parfois de le retirer et on voit ce dernier s'afficher à l'écran à l'exécution de l'application... avec **`tools:text`** vous pouvez éviter ce désagrément.

Il peut arriver que vous ayez besoin d'afficher du texte qui provient d'un *web-service*. Dans ce cas, au lieu par exemple d'y mettre un texte en « dur » que vous pourriez oublier à l'exécution, pourquoi ne pas simplement utiliser **`tools:text`** pour tester votre **`textview`** et les éventuelles chaînes qu'il pourrait recevoir ; en effet, il vous revient de supprimer les valeurs **`tools:text`** après que vos tests aient été concluants, non pas de peur que cela apparaisse à l'exécution, mais plutôt pour une question de lisibilité de votre code.

Cela fonctionne bien aussi avec l'attribut **`visibility`** ; en effet, si par exemple vous avez des vues qui alternent leurs « visibilités », au lieu de les changer l'une après l'autre et de tester le résultat après exécution, vous pouvez simplement, à l'aide de **`tools`** faire des déclarations du style **`tools:visibility`** pour surcharger la visibilité de vos vues et avoir directement dans Android Studio l'aperçu. Cela vous évite d'avoir à oublier d'exécuter votre code pour des tests.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/a_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:visibility="gone" />
    <TextView
        android:id="@+id/txt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:visibility="gone"
        tools:visibility="visible"
        tools:text="list is gone at inflation" />
</FrameLayout>
```

Ce code donne un résultat similaire à celui présenté en figure 2.

En général, on peut l'utiliser avec tous les attributs Android dès que l'élément parent déclare le *namespace* **`xmlns:tools="http://schemas.android.com/tools"`**.

Fig. 2 : Utilisation de tools sur d'autres attributs tel que visibility.

2 Les pointeurs de vue

L'attribut **`tools`** fonctionne aussi bien avec les vues qui contiennent ou sont incluses par d'autres vues. En effet, si par exemple, vous avez une vue **A** qui contient une vue **B** via un **`include`** et que la vue **B** à son tour contient une balise **`merge`**, l'aperçu de la vue **A** dans Android Studio ne pourra pas correctement afficher la sous-vue **B**. Pour faire en sorte que la vue **A** puisse afficher correctement la vue

incluse **B**, il faut rajouter dans la balise **merge** de **B** l'attribut **tools:showIn** avec pour valeur la vue **A**. Cela s'explicite mieux avec le bout de code suivant :

- Pour la vue **A** :

```
<?xml version="1.0" encoding="utf-8"?>
<!-- VUE A -->
<LinearLayout xmlns:android="http://
schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <include layout="@layout/view_b" />
</LinearLayout>
```

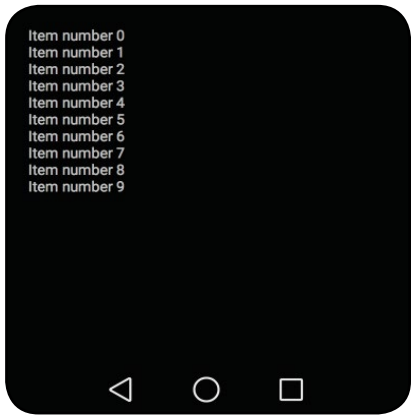


Fig. 3 : Aperçu sans l'utilisation de tools:showIn.

Fig. 4 : Aperçu avec l'utilisation de tools:showIn.

- Pour la vue **B** :

```
<merge xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <TextView
        tools:text="Cameras: 9"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="24dp"
        android:layout_marginLeft="24dp"
        android:layout_marginRight="24dp"
        android:layout_marginTop="24dp"
        android:textAppearance="@style/TextAppearance.AppCompat.
Title" />
    <android.support.v7.widget.RecyclerView
        android:id="@+id/camera_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
    <TextView
        android:id="@+id/error_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="error_camera_list_load"
        android:visibility="gone" />
</merge>
```

On remarque dans la vue **B**, que nous n'avons pas déclaré l'attribut **tools:showIn**. On obtiendra un résultat semblable à celui de la figure 3.

Mais en rajoutant l'attribut **tools:showIn** dans la vue **B** avec la valeur de la vue **A** », on obtient le résultat de la figure 4.

```
<merge xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:showIn="@layout/view_a">
    <TextView
        tools:text="Cameras: 9"
...
</merge>
```

En comparant les deux captures d'écran, on voit que sans l'ajout de l'attribut **tools:showIn** il manque le **textview**, mais ce dernier apparaît dès qu'on ajoute ledit attribut.

3 Ignorer les avertissements Lint

Une autre fonction offerte par **tools** est la possibilité d'ignorer les avertissements Lint ; je ne suis pas particulièrement fan de cette fonctionnalité, mais je suppose qu'il faut avoir une bonne raison pour en avoir besoin.

Pour ce faire, il suffit de rajouter l'attribut **tools:ignore** avec pour valeur le nom de l'avertissement Lint à ignorer. Je vous invite à faire une recherche sur Google concernant l'avertissement Lint que vous souhaitez ignorer afin de vous assurer d'avoir le bon « nom ». Si vous connaissez l'avertissement Lint à ignorer, il suffit de procéder comme avec le code ci-dessous :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/
res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:ignore="AvertLint1,AvertLint2,AvertLint3">
```

Conclusion

L'attribut **tools** est très pratique à utiliser ; il peut vous permettre de gagner du temps dans vos tests de vue pendant vos développements. Les valeurs attribuées avec **tools** ne sont pas prises en compte lors de l'exécution de votre application, mais dans un souci de lisibilité, il ne faut pas que vous oubliiez de les retirer de votre code après vos tests confirmés. ■

Référence

[1] Android Tools Project Site : http://tools.android.com/tech-docs/tools-attributes

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

sales@ikoula.com
01 84 01 02 66
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 58
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz

LINAGORA Exporte !

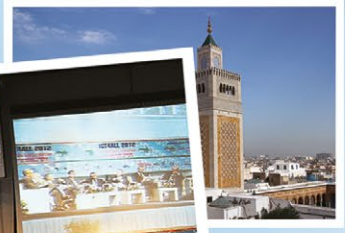
CANADA

ETATS-UNIS

JAPON

VIETNAM

TUNISIE



SENEGAL

ÉMIRATS ARABES UNIS

Go Linagora !

Nous recrutons, retrouvez nos offres sur : job.linagora.com !

DÉCOUVREZ NOS AUTRES SOLUTIONS LIBRES !

OBM.org	LinShare	LinID	LinPKi	petals ESB	OpenPaaS
Messagerie collaborative	Partage et transfert sécurisé de fichiers	Gestion et fédération des identités	PKI et signature électronique	ESB et orchestration	Plateforme collaborative

www.linagora.com

@Linagora