N°180 mars 2015

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

ALGO / REGEXP

Comprenez enfin comment utiliser les expressions régulières ! p.24

ANDROID / TV

Découvrez Android TV et le TV Input Framework p.66

DJANGO / REST

Mettez en place des Webservices REST sous Python p.52

STOCKAGE / BIG DATA

Vous avez de plus en plus de données à conserver ?

UTILISEZ CEPH, LE SYSTÈME DE FICHIERS DISTRIBUÉ HAUTE PERFORMANCE

p.30

CODE / JAVA EE

Développez beaucoup plus rapidement avec JBoss Forge p.72

SHINKEN / ACTU

Parcourez les nouveautés de Shinken v2 p.06

SYSADMIN / SUPERVISION

Supervisez simplement vos processus avec Monit p.40

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

Quelle interopérabilité entre mes différents fournisseurs Cloud

Avec Aruba Cloud,

vous avez l'assurance de ne pas être prisonnier d'un fournisseur. Nos services sont intégrés au **driver DeltaCloud** et compatibles **S3**. De plus, vous pouvez utiliser des formats standards d'images de machines virtuelles, **avec VHD et VMDK**, ainsi que des modèles personnalisés provenant éventuellement d'autres sources.

3 hyperviseurs

6 datacenters en Europe

APIs et connecteurs

70+ templates

Contrôle des coûts

Nous avons choisi Aruba Cloud car nous bénéficions d'un haut niveau de performance, à des coûts contrôlés et surtout car ils sont à dimension humaine, comme nous. Xavier Dufour - Directeur R&D - ITMP

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

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

MY COUNTRY. MY CLOUD.*

*MON PAYS, MON CLOUD.

GNU/Linux Magazine France
est édité par Les Éditions Diamond

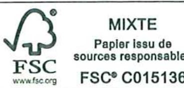
B.P. 20142 – 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 – **Fax :** 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – boutique.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Réalisation graphique : Kathrin Scali & Carine Greppat
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 €



La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans GNU/Linux Magazine France est interdite sans accord écrit de la société Les éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à GNU/Linux Magazine France, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire. Toutes les marques citées dans ce numéro sont déposées par leur propriétaire respectif. Tous les logos représentés dans le magazine sont la propriété de leur ayant droit respectif.

SUIVEZ-NOUS SUR :

https://www.facebook.com/editionsdiamond

@gnulinuxmag

LES ABONNEMENTS ET LES ANCIENS NUMÉROS SONT DISPONIBLES !

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

ÉDITORIAL

C'est fou ce que l'on peut faire croire aux gens ! Peu après l'attentat contre Charlie Hebdo, des voisins, à la retraite depuis déjà quelques temps, me dirent : « Quand même, c'est dangereux ces réseaux sociaux et internet ! ». Et après un court silence : « Et d'abord... c'est quoi les réseaux sociaux ?! ». À ma mine déconfite, ils ont vite compris qu'il y avait un problème. Et ce problème est fondamental : on fait peur aux gens, on leur assène des vérités qu'ils prennent pour argent comptant et à aucun moment on ne tente de les amener à réfléchir par eux-mêmes, à se construire une pensée. Mais qui est ce « on » ? Ce « on » ce sont d'abord les politiques de tous bords qui y vont de leurs petites déclarations pour apeurer le bon peuple : « (...) 90 % des individus qui basculent dans des groupes ou des activités terroristes le font par le biais d'internet. » [1]. Le chiffre pourra difficilement être remis en cause puisque « 100% des gagnants au Loto ont tenté leur chance... ». Par contre, le sous-entendu et le raccourci qui s'ensuit sont bien dangereux : internet = le Mal, donc on va surveiller à outrance le réseau. Il suffit ensuite d'ajouter quelques petits reportages bien sentis de confrères journalistes en recherche de « scoop » qui initieront un travail « d'investigation » sur internet et les fameux réseaux sociaux pour créer un sentiment de peur. C'est sur cet état d'esprit que les politiques s'appuient ensuite pour nous faire accepter bien des choses inacceptables quant à la violation de notre vie privée (déjà largement entamée de manière fortuite par des applications sur smartphone, bien souvent des jeux, qui ont évidemment besoin de connaître votre localisation, votre carnet d'adresses, d'avoir accès à vos SMS, etc.). Nous en arrivons ainsi à des collectes automatisées de données dont les volumes deviennent de plus en plus impressionnants et qui échappent totalement à notre contrôle.

D'un point de vue technique, il faut être capable de stocker ces données et c'est le rôle de solutions telles que Ceph qui vous est présenté dans le magazine de ce mois-ci. Bien sûr il ne faut pas voir que le côté obscur de la for^W^W^Wsituation car des domaines tels que la recherche produisent également énormément de données à stocker et à analyser et il ne faut pas rejeter en bloc tout ce qui touche aux « big data » (oui, de nos jours les services de marketing trouvent des termes pour vendre un tas de choses...). Comparer des séquences génétiques est essentiel pour la compréhension de la vie, découvrir des thérapies. Pister les individus, les classer en fonction de leurs recherches sur le net, de leurs centres d'intérêts est un objectif nettement moins louable. D'ailleurs, vous utilisez un système Linux et pour peu que vous ayez installé Tor, vous êtes catalogués en tant qu'extrémiste (potentiel) par la NSA [2] !

Nous nous sommes tous mobilisés pour la liberté de la presse, mais cette liberté c'est aussi la liberté d'expression qui est à la base d'internet, lieu d'échanges par excellence. Nous, professionnels, parents, utilisateurs réguliers de ce fabuleux outil, nous avons tous notre rôle à jouer en prenant le temps d'expliquer, de démystifier les principes qui régissent internet. Tout peut être dangereux suivant l'utilisation qui en est faite. Un couteau de cuisine dans la main d'un Paul Bocuse permettra de concevoir de somptueux plats alors que dans celles d'un terroriste ce pourra être une arme redoutable. Pour autant faut-il surveiller toute personne utilisant un couteau ? Un couteau, ça peut faire des dégâts, mais on apprend alors à l'utiliser et l'on met en garde nos enfants pour qu'ils en aient une utilisation adéquate... il en est de même pour les réseaux sociaux et internet !

Tristan Colombo

[1] B. Cazeneuve, séance de l'Assemblée nationale du 14 janvier 2015 : http://www.assemblee-nationale.fr/14/cri/2014-2015/20150108.asp

[2] Analyse du code source de XKeyscore par J. Appelbaum et al. : http://daserste.ndr.de/panorama/aktuell/nsa230_page-1.html

AGENDA

Devoxx 2015

Date : du 8 au 10 avril 2015

Lieu : Paris - France

Site officiel : http://www.devoxx.fr/

Pour sa quatrième édition, Devoxx propose trois jours de conférences et d'ateliers pratiques. Au moment où vous lirez ces lignes le programme devrait être en ligne. Les principaux thèmes abordés, tirés du nombre de soumissions, seront les suivants :

- technologies Web : frameworks permettant la création d'applications mobiles (natives iOS ou Android, ou alors application Web) ;
- agilité, méthodologie et tests ;
- Java : tout ce qui tourne autour de Java avec le JDK, le tuning de JVM, la modularité, etc ;
- architecture, performance et sécurité ;
- Cloud, Big Data et no SQL ;
- applications embarquées avec le développement sur Arduino, Raspberry Pi, etc.

Mini DebConf

debian

Date : les 11 et 12 avril 2015

Lieu : Lyon - France

Site officiel : https://france.debian.net/events/minidebconf2015/

Les mini DebConfs sont des réunions locales organisées par les membres du projet Debian. La structuration est la même que pour une DebConf (discussions, ateliers pratiques et coding parties)... mais en plus petit.

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE

N° 180

actualités

06 **Shinken v2, vers plus de flexibilité**

Débuté voilà plus de 5 ans, le projet Shinken continue son chemin dans la supervision des grands environnements. Regardons ce qui a changé depuis notre dernier article dans ces colonnes.

humeur

12 **Internet & élections : tout n'est pas permis**

À voir certaines campagnes électorales, notamment américaines, on peut en venir à rêver d'avoir le même genre de bataille en France. Mais les règles ne sont pas les mêmes dans l'hexagone et les élections départementales nous donnent l'occasion de revenir sur ces aspects.survenue entre le lundi 15 septembre 2014 et le jeudi 18 septembre 2014.

repères

16 **Une histoire de l'informatique #6 - La naissance des nains surpuissants**

Les ordinateurs pris en main par de grands industriels, principalement aux États-unis, il reste aux pionniers quelques chantiers au niveau matériel : tester et valider l'utilisation des transistors, trouver de meilleurs systèmes pour stocker des données en mémoire vive, le plus rapidement possible et, surtout inventer tous les concepts pour obtenir des performances de calcul maximales dans des superordinateurs surpuissants, concepts qui animent aujourd'hui nos objets du quotidien.

algo / IA

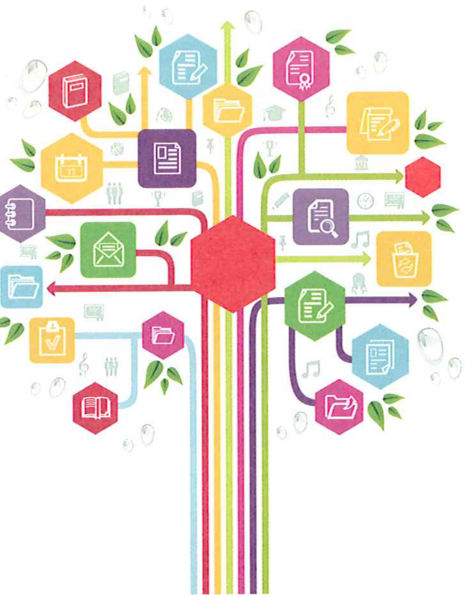
24 **Les expressions régulières : la théorie et les normes**

Tout le monde a déjà entendu parler des expressions régulières mais la population mondiale se décompose en deux groupes : ceux qui connaissent les expressions régulières et les utilisent avec bonheur chaque jour... et les autres. Cet article s'adresse plus particulièrement au second groupe (mais, pour les membres du premier groupe, quelques rappels ne font jamais de mal).

sysadmin / netadmin

30 **Mise en œuvre de Ceph**

Dans cet article nous allons voir comment Ceph, le système de stockage objet à haute disponibilité et hautes performances, fonctionne et s'il est possible de le soumettre, sans danger, à quelques petites expériences. Nous verrons également qu'une expérimentation à grande échelle est menée au CERN.



40 **Supervision distribuée avec Monit (et Puppet)**

Depuis le début de l'ère internet, le nombre de systèmes et d'applicatifs déployés par les entreprises ne cesse de croître de manière exponentielle, et l'arrivée du cloud et du Big Data n'a fait qu'accélérer la tendance. En outre, là où il y a vingt ans un administrateur système gérait une douzaine de machines, le même administrateur en gère aujourd'hui parfois plusieurs centaines. Or, il devient de plus en plus compliqué de proposer une supervision adaptée qui puisse supporter une telle mise à l'échelle.

python

52 **REST in Peace : Présentation de Django REST Framework**

Les plus vieux d'entre vous, ou les plus malchanceux, savent que dans les temps immémoriaux du Web, il existait une horrible façon de mettre en place des Webservices, le protocole SOAP. Heureusement pour nous, REST est arrivé.

android

66 **Android TV et TV Input Framework**

Fraîchement disponible depuis Android 5.0 (Lollipop), Android TV est le nouvel élan de Google pour s'imposer dans le monde déjà très fermé des Set-Top-Box et autres consoles de jeux. Comme à l'accoutumée, découvrons dès à présent les entrailles de la bête.

code

72 **JBoss Forge2, Java EE facile, très facile**

Le framework JBoss Forge offre de puissantes possibilités dans le domaine de la création d'applications Java EE et son extensibilité est un atout pour chaque développeur Java.

abonnements

59/60 : abonnements multi-supports

65 : offres spéciales professionnelles

DjangoCong 2015

Date : du 8 au 10 mai 2015

Lieu : Clermont-Ferrand - France

Site officiel : http://rencontres.django-fr.org

Pour la cinquième fois, la rencontre française des utilisateurs de Django revient. Une fois n'est pas coutume, cette année elle aura lieu dans les terres du milieu, i.e. à Clermont-Ferrand (Auvergne) les 8, 9 et 10 Mai. Contrairement aux années précédentes, et profitant d'un jour férié, la cuvée 2015 se tiendra sur trois jours :

- le vendredi 8 sera l'occasion de mettre en place des ateliers d'initiation à Django pour celles et ceux qui voudraient découvrir le framework ;
- le samedi sera — comme traditionnellement — le jour des conférences et des barcamps ;
- enfin, le dimanche sera l'occasion de vous faire une surprise : sprint sur des projets ou ballade touristique, vous verrez sur place !

Appel à participation DjangoCon Europe 2015

Date : du 31 mai au 5 juin 2015

Deadline : 18 mars 2015

Lieu : Cardiff - Irlande (en anglais)

Site officiel : http://2015.djangocon.eu

Une annonce un peu tardive concernant l'appel à participation de DjangoCon Europe, la conférence européenne sur le framework Web Python nommé Django, qui aura lieu cette année en Irlande. En étant un peu rapide vous pouvez encore proposer une présentation ou un workshop... ■

SHINKEN V2, VERS PLUS DE FLEXIBILITÉ

par **Jean Gabès** *[Leader du projet Shinken, CTO Shinken Solutions]*

Débuté voilà plus de 5 ans, le projet Shinken continue son chemin dans la supervision des grands environnements. Regardons ce qui a changé depuis notre dernier article dans ces colonnes.

Si l'origine de Shinken est fortement liée à l'histoire de Nagios, son évolution n'a pas été limitée par la stagnation de son aîné et de ses dérivés. Dans un précédent numéro **[1]** nous avions traité de la version 1.2 de Shinken, mais depuis nous avons eu droit à une nouvelle version majeure en avril 2014. S'orientant vers toujours plus de flexibilité, les possibilités d'architectures se sont encore étoffées.

1 Une installation simplifiée

Si la version de shinken n'est pas assez à jour dans votre distribution, ou si vous désirez avoir la toute dernière version, vous pouvez passer par la commande `pip` de python pour l'installer :

```
$root@debian: adduser shinken
$root@debian: apt-get install python-pycurl python-setuptools python-pip
$root@debian: pip install shinken
```

Les chemins d'installations sont relativement classiques :

- */etc/shinken* : la configuration
- */usr/bin/shinken** : scripts d'entrée des daemons et la commande CLI shinken
- */var/lib/shinken* : données, sondes et modules
- */var/log/shinken* : les logs

Vous pouvez alors démarrer vos daemons Shinken :

```
$root@debian: /etc/init.d/shinken start
```

Premier gros changement par rapport aux versions 1.x, les communications entre les daemons se font par HTTP(s) au lieu d'un format python auparavant (via la librairie **Pyro**). Vous pouvez ainsi facilement vérifier que vos daemons sont vivants en les requêtant, comme ici pour l'élément principal, l'*Arbiter* :

```
$root@debian: curl http://localhost:7770/
OK
```

Les plus curieux pourront lister l'API disponible avec l'aide de l'URL /api-full disponible sur chaque daemon :

```
$root@debian: curl http://localhost:7770/api-full | json_pp
```

2 Un repository de modules et de packs

Depuis la version 2.0, seul le framework shinken (daemons et une configuration d'exemple) est installé. Les modules et les packs de configurations sont toujours disponibles, mais ont désormais chacun leur propre place. Ils sont disponibles sur le site http://shinken.io. La nouvelle commande CLI `shinken` permet de les lister et les installer facilement.

Pour appeler cette dernière, il est préférable de se placer avec le compte shinken :

```
$root@debian: su - shinken
```

Lors de sa première utilisation, une phase d'initialisation est nécessaire afin qu'elle génère son fichier de configuration `~/.shinken.ini` :

```
$shinken@debian: shinken --init
```

C'est au sein de ce fichier *ini* que vous pouvez configurer un serveur proxy si besoin. Lister les différentes commandes disponibles se fait avec :

```
$shinken@debian: shinken -l
Available commands:
desc:
        desc : List this object type properties
doc:
        doc-compile : Compile the doc before enabling it online
        doc-serve : Publish the online doc on this server
shinkenio:
        install : Grab and install a package from shinken.io
        inventory : List locally installed packages
        publish : Publish a package on shinken.io. Valid api key required
        search : Search a package on shinken.io by looking at its keywords
        update : Grab and update a package from shinken.io. Only
the code and doc, NOT the configuration part! Do not update an not
installed package.
```

Ensuite nous pouvons par exemple regarder quels packs de configuration sont disponibles pour superviser un linux :

```
$shinken@debian: shinken search linux
linux-snmp [pack,linux,snmp] : Linux checks based on SNMP
linux-ssh  [pack,linux,ssh] : Linux checks based on SSH without
any script on distant server
pack-glances [pack,system,linux,glances] : Standard check
through checkglances.py and glances server
```

Ayant une préférence personnelle pour la supervision sans agent nous allons partir sur la supervision par ssh (je peux également avoir été influencé par le fait que je suis également l'auteur de ces sondes ssh :)). L'installation du pack et des sondes de supervision est très simple :

```
$shinken@debian: shinken install linux-ssh
OK
```

Dans le cas où votre serveur shinken ne peut avoir accès à internet, vous pouvez télécharger les archives des packs et des modules sur l'organisation github du projet **[2]** et lancer l'installation de votre archive en rajoutant le paramètre `--local`.

Les sondes du pack `linux-ssh` nécessitent la librairie `python-paramiko` que nous installons également :

```
$shinken@debian: su -
$root@debian: apt-get install python-paramiko
```

Ces sondes ont juste besoin d'une clé ssh pour se connecter à un compte non root, ainsi que des outils `sysstat`. Si vous n'avez pas de clé déjà générée et distribuée, c'est relativement simple à faire :

```
$shinken@debian: ssh-keygen
$shinken@debian: ssh-copy-id -i ~/.ssh/id_rsa shinken@localhost
```

Vous pouvez essayer de lancer la sonde manuellement en tant qu'utilisateur *shinken* afin de vérifier que tout est bon :

```
$shinken@debian: /var/lib/shinken/libexec/check_load_average_by_
ssh.py -H localhost -i ~/.ssh/id_rsa
Ok: load average is good 0.00,0.00,0.00 |  load1=0.00;1.00;2.00;;
load5=0.00;1.00;2.00;; load15=0.00;1.00;2.00;;
```

Configurons notre premier hôte avec les sondes proposées par ce pack de configuration. Et autant commencer par notre propre serveur de supervision, en rajoutant simplement le template `linux-ssh` à sa définition. La définition de notre hôte est située dans le fichier `/etc/shinken/hosts/localhost.cfg` :

```
define host{
    use                     linux-ssh,generic-host
    host_name               localhost
    address                 localhost
    contact_groups          admins
}
```

Un reload de shinken plus tard, notre hôte est supervisé avec ses nouveaux services :

```
$root@debian: /etc/init.d/shinken reload
```

3 Une documentation en ligne et en local

La documentation du projet a été revue et est désormais hébergée sur http://shinken.readthedocs.org/. En plus de cette version en ligne, Shinken est installé avec sa propre documentation que vous pouvez lire sur votre réseau local en lançant simplement la commande :

```
$shinken@debian: shinken doc-serve
```

Ceci vous présentera la documentation de Shinken sur le port **8080**. Si vous installez de nouveaux modules ou packs de configurations, ces derniers peuvent avoir une documentation intégrée et vous pourrez recompiler la documentation afin d'incorporer ces nouvelles parties de documentation à la vôtre :

```
$root@debian:   apt-get install python-sphinx
$shinken@debian: shinken doc-compile
```

La documentation de shinken étant relativement grande, cette opération est susceptible de prendre quelques minutes.

4 De nouvelles possibilités d'architecture

Les dernières versions de Shinken ont vu l'arrivée de nouvelles possibilités dans un domaine déjà très fourni : les architectures distribuées. La principale est la possibilité de définir plusieurs niveaux de daemons *broker*. Pour rappel, ces derniers ont pour responsabilité de récupérer les informations de supervision des autres daemons, et de les présenter/exporter. Ce sont, par exemple, sur eux que nous activons le module d'interface graphique **webui** afin de voir les états des éléments supervisés.

Dans les anciennes versions de Shinken, l'accès aux informations d'états de vos hôtes et services étaient limité à un seul daemon *Broker*. Ceci interdisait d'avoir deux interfaces de supervision à des niveaux différents : une distante présentant les informations d'un seul datacenter et une globale ayant les informations de tous les datacenters.

Cette limitation n'est plus de mise depuis la version 2 de Shinken. Afin d'activer cette possibilité, il suffit de rajouter le paramètre **broker_complete_links** à ses objets *realms* (datacenters) dans votre configuration. Par exemple :

```
define realm {
    realm_name  Central
    realm_members   DC1,DC2
    broker_complete_links  1
}
define realm {
    realm_name  DC1
    broker_complete_links  1
}
define realm {
    realm_name  DC2
    broker_complete_links  1
}
```

Vous obtiendrez une architecture similaire à la figure 1. Les interfaces Web utilisant les broker des datacenters distants n'auront que les informations de leur propre datacenter, mais ceux du Central auront accès à toutes les informations.

5 Une meilleure visibilité des performances internes

5.1 Utilité des données de performances internes

Shinken est notamment reconnu pour sa rapidité, mais chaque daemon a ses limites. Afin de voir facilement si votre installation a besoin de nœuds supplémentaires il est intéressant de commencer à regarder les performances internes de vos daemons. Ces derniers sont capables d'exporter leurs données de métrologie internes (longueur des files d'attente, temps d'appels des différentes API, ...) à l'extérieur.

Pour cela deux solutions s'offrent à vous :

- les envoyer vers **statsd** ;
- les envoyer vers le service kernel.shinken.io qui est gratuit mais a une durée de conservation limitée par défaut ;

Nous allons voir la mise en place des deux méthodes.

5.2 Export vers statsd

statsd est un daemon tournant sur **nodejs** qui permet de récupérer des données numériques et d'en calculer diverses statistiques (min, max, moyenne, 99 percentile, ...).

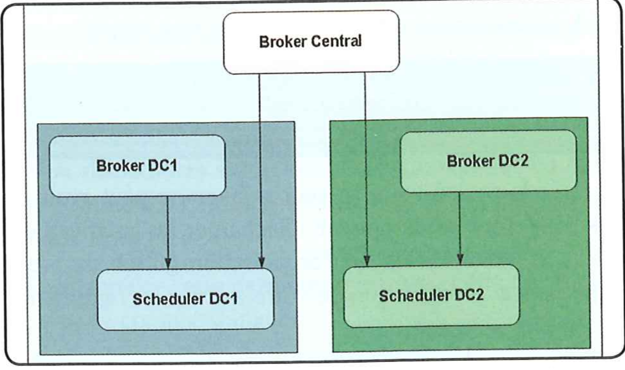


Fig. 1 : Exemple d'architecture distribuée avec données présentes en local sur les datacenters et en centralisé.

Pour chaque métrique les statistiques sont calculées toutes les 10 secondes et sont ensuite exportées vers un outil de métrologie, comme par exemple vers **graphite**.

La mise en place de **statsd** et de graphite est laissée en exercice au lecteur. Il est cependant à noter que dans le cas de graphite il est fortement recommandé de suivre à la lettre les différentes documentations, l'outil n'étant pas réputé facile à mettre en place...

Une fois votre daemon **statsd** en place, la configuration dans Shinken est très simple. Tout est centralisé dans le fichier de configuration principal **/etc/shinken/shinken.cfg** :

```
statsd_host=mon-serveur-statsd
statsd_port=8125
statsd_prefix=shinken
statsd_enabled=1
```

Vous n'avez qu'à redémarrer shinken pour voir apparaître dans quelques secondes les données les plus intimes de shinken dans votre interface graphite.

```
$root@debian: /etc/init.d/shinken restart
```

5.3 Export vers kernel.shinken.io

Si vous n'avez pas de daemon **statsd** sous la main ou si la mise en place de graphite a vidé votre boîte d'antidépresseurs, vous pouvez toujours utiliser le service kernel.shinken.io. Au moment de l'écriture de l'article les données y sont conservées 7 jours, mais ceci pourrait évoluer suivant le succès du service, et la charge des daemons graphite qui stockent ces données.

Point important : ne seront envoyées que les données de performances de vos daemons ainsi que le nom de ces derniers. Aucune ip/adresse, nom d'hôte ou quelconque mot de passe ne seront envoyés à ce service. Les données seront similaires à ce qui est envoyé à **statsd**.

L'utilisation de ce service nécessite uniquement un compte sur le site de partage des packs et des modules http://shinken.io. Une fois enregistré, vous trouverez sur la page de votre compte http://shinken.io/~ vos données de connexions (**api_key** et **secret**).

La configuration dans Shinken est similaire à celle du cas **statsd**, à savoir la modification du fichier **/etc/shinken/shinken.cfg** :

```
api_key=ABCDEFGHIJ
secret=KLMOPQRST
```

Si une connexion directe entre vos daemons shinken et le monde extérieur n'est pas autorisé, il est possible de définir un proxy :

```
http_proxy=http://monproxy:3128
```

Là encore un simple redémarrage de shinken suffira à faire envoyer les données.

```
$root@debian: /etc/init.d/shinken restart
```

Vous pouvez voir sur la figure 2 un exemple de performance interne d'un daemon.

6 Les règles métiers et les clusters

Les règles métiers (*bp_rule*) font parties de Shinken depuis longtemps. Si elles sont très pratiques pour présenter une vision métier de son système informatique, elles avaient comme limitation de reposer sur des éléments nommés du SI. Ceci est trop restrictif dans un monde qui devient de plus en plus élastique, avec des nœuds de cluster qui se rajoutent régulièrement.

C'est la raison pour laquelle depuis la version 2 de Shinken il est possible d'utiliser des expressions pour lister les éléments des règles métiers. Ces liens peuvent se faire suivant les tags, les groupes ou des regexp basées sur le nom des éléments. Si vous souhaitez par exemple créer un indicateur unique qui représente l'état de tous vos services Http des hôtes ayant le tag **http** il vous suffira de déclarer :

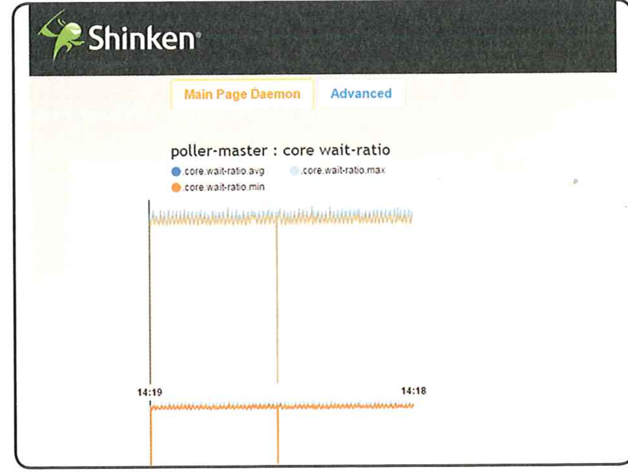


Fig. 2 : Exemple de donnée interne de vos daemons shinken sur kernel.shinken.io.

```
define host{
    host_name          HttpCluster
    use                generic-host
    check_command      bp_rule!t:http,Http
}
```

7 Vision sur le passé

Une des nouvelles fonctionnalités de la version 2.2 est la présence des commandes de *snapshots*. Elles servent à répondre à une problématique que la plupart des administrateurs ont déjà rencontré dans leur vie : il y a eu un soucis de charge anormale la nuit précédente et a causé beaucoup de soucis en cascade. Si on peut facilement être prévenu de cette charge nous n'avons gère d'informations sur quels processus ou traitements l'ont provoqués.

Les *snapshots* permettent de prendre une photo de votre serveur au moment où le problème est détecté et de la conserver pour analyse future. Concrètement, l'utilisateur peut définir n'importe quelle commande pour obtenir les informations qu'il souhaite, comme par exemple la liste des processus ou bien celle des requêtes dans une base de données. Libre à lui de choisir ce qu'il souhaite.

Ces commandes de récupération pourront retourner le texte qu'elles souhaitent. Leur définition se fait comme n'importe quelle commande. Imaginons ici que nous souhaitons avoir la liste des processus lors d'une charge anormale se produisant la nuit. Nous allons tout d'abord définir la commande de récupération :

```
define command {
  command_name    dump_processes
  command_line    /usr/bin/ssh -H $HOSTADDRESS$ "ps ax -o 
user,vsz,rss,pcpu,command"
}
```

Une fois la commande définie nous pouvons la relier à notre service de *Load Average* qui mesure la charge de nos serveurs. Il est défini dans le fichier `/etc/shinken/packs/linux-ssh/services/load_average.cfg` installé précédemment avec le pack `linux-ssh` :

```
define service{
   service_description    Load Average
   use                    linux-ssh-service
   register               0
   host_name              linux-ssh
   check_command          check_ssh_linux_load_average
   snapshot_command     dump_processes
   snapshot_enabled     1
   snapshot_criteria    c
   snapshot_period      night
   snapshot_interval    15
}
```

Ici durant la nuit si le service *Load Average* est en état Critical (paramètre `snapshot_criteria`) alors la commande `dump_processes` sera lancée, mais pas plus d'une fois toutes les 15 minutes (paramètre `snapshot_interval`).

Une fois la commande lancée et sa sortie récupérée il nous faut un moyen de sauvegarder cette dernière. Un module est actuellement dédié à cette tâche. Il permet d'exporter les données vers une base **mongodb**. Pour l'installer rien de plus simple avec la commande CLI :

```
$shinken@debian: shinken install snapshot-mongodb
OK snapshot-mongodb
```

Le module installé, vous pouvez modifier sa configuration pour le faire pointer vers votre instance mongodb, dans le fichier `/etc/shinken/modules/snapshot-mongodb.cfg` :

```
define module {
    module_name       snapshot-mongodb
    module_type       snapshot_mongodb
    uri               mongodb://localhost/?safe=false
    database          shinken
}
```

Il faut ensuite déclarer ce module dans le daemon `broker` afin que ce dernier exporte les données provenant de toute votre infrastructure, dans le fichier `/etc/shinken/brokers/broker-master.cfg` :

```
define broker {
    broker_name       broker-master
    [...]
    modules  webui, snapshot-mongodb
}
```

Libre à vous désormais d'inclure ces nouvelles données dans vos interfaces de visualisation.

Conclusion

Les dernières versions de Shinken ont apporté des améliorations pour bâtir des architectures toujours plus flexibles et avoir des informations toujours plus précises. Les prochaines évolutions porteront sur un support amélioré des architectures élastiques (cloud, docker, …) et le rajout à chaud d'éléments à superviser. ■

Références

[1] Collectif, « La supervision avec Shinken », GNU/Linux Magazine hors série n°62, septembre/octobre 2012.

[2] Repository des packs et modules shinken : https://github.com/shinken-monitoring

Les Salons Solutions ELECTRONIQUES

EXPOSITION - CONFERENCES - ATELIERS

10ème édition

Machine to Machine & Objets connectés

Le salon des professionnels des Objets Communicants

Systèmes Temps Réel & Embarqués
Affichage et Visualisation - Conception et test de systèmes électroniques

1er & 2 avril 2015
CNIT PARIS LA DEFENSE

- Opérateurs telecoms ou MVNO
- Fournisseurs de solutions réseaux
- Constructeurs ou distributeurs
- SSII et cabinets de consulting
- Intégrateurs de solutions
- Internet des Objets

- Cartes, composants et modules
- OS temps réel et embarqués
- Environnements de développement
- Outils de test et de validation
- Conception & test de systèmes électroniques
- PC Industriels / Ecrans

www.salons-solutions-electroniques.com

En parallèle

Partenaire officiel

INTERNET & ÉLECTIONS : TOUT N'EST PAS PERMIS

par *Tris Acatrinei [Consultante pour FAIR-Security]*

À voir certaines campagnes électorales, notamment américaines, on peut en venir à rêver d'avoir le même genre de bataille en France. Mais les règles ne sont pas les mêmes dans l'hexagone et les élections départementales nous donnent l'occasion de revenir sur ces aspects.

En mars 2015, nous allons élire des conseillers départementaux – auparavant appelés conseillers généraux – et la bataille électorale a déjà commencé. Mais que peut faire un candidat sur le Web ?

1 Les élections départementales pour les Nuls

Comme vous le savez certainement, le Parlement a récemment voté une controversée réforme territoriale, découpant la France en 13 régions mais instaurant également de nouvelles règles concernant la gestion au niveau local. Avant d'entrer dans les règles applicables en matière de scrutin, arrêtons-nous quelques instants sur cette entité qu'est le conseiller départemental.

Un conseiller départemental siège au sein d'un conseil départemental, anciennement appelé conseil général. Le département, en tant qu'entité administrative, peut décider d'une politique concernant les aides sociales, le développement du tissu économique urbain et rural – plus simplement : comment inciter les commerces et les entreprises à s'implanter sur un territoire – l'aménagement du territoire notamment pour les personnes à mobilité réduite mais également tout ce qui touche aux transports, l'enfance et la gestion des collèges et le développement culturel comme la création des bibliothèques.

Pour le scrutin, les territoires sont découpés en cantons. Par exemple, Saint-Denis (93) est découpé en deux cantons et la taille des cantons est fixée par le Code Général des Collectivités Territoriales, en fonction du nombre d'habitants.

Enfin, pour les départementales, l'élection s'organise en binôme, un homme et une femme, élus au scrutin majoritaire – celui qui a le plus de voix gagne – pour une durée de 6 ans, exception faite de Paris, qui est à la fois une ville et un département. Normalement, il n'y a qu'un seul tour mais s'il n'y a pas au moins 25% des électeurs inscrits et une majorité absolue des suffrages exprimés (donc plus de 50%), un second tour aura lieu.

Maintenant que vous savez tout (ou presque) sur les conseillers départementaux, passons au nerf de la guerre : la chasse aux voix.

2 Une histoire de temps

Il existe trois marqueurs temporels dans la propagande électorale :

- Six mois avant l'élection ;
- Vingt jours avant le scrutin ;
- 48 heures avant le scrutin.

Concrètement, pour l'élection départementale de 2015, il n'était pas possible de s'y prendre six mois à l'avance, en raison de la réforme territoriale. Mais ces trois marqueurs temporels sont le principe de base. Autre « détail » : lorsque l'on parle de collectivité locale, cela peut indifféremment désigner une ville, un village, une intercommunalité, un département, un canton, une région.

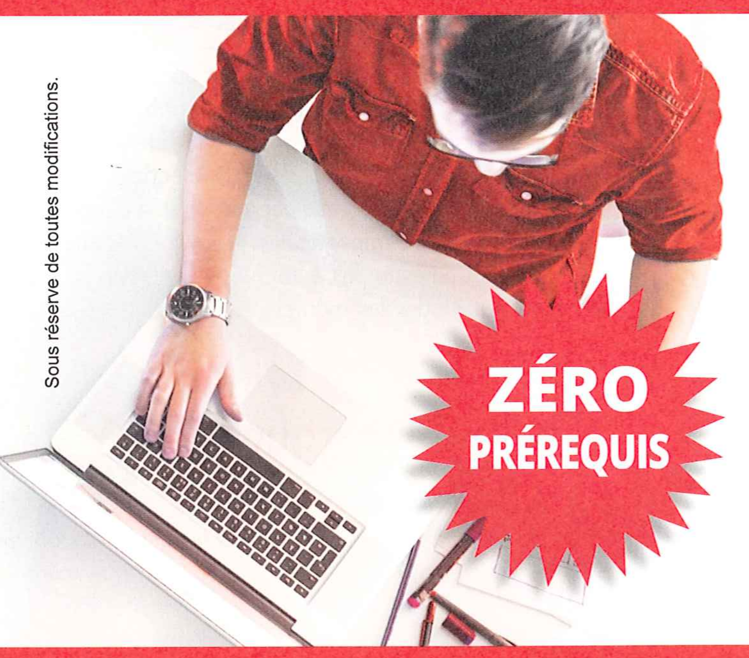
Au sein d'une collectivité locale, un candidat ne pourra pas utiliser les supports de communication institutionnels pour mettre en valeur un éventuel bilan à partir de six mois avant l'élection. Concrètement, il ne devra pas publier le résultat des actions qu'il a pu mener dans le bulletin local, il ne pourra pas utiliser le matériel et les locaux mis à sa disposition dans le cadre de son mandat pour préparer la campagne. Cette interdiction découle du principe d'égalité entre les candidats. Si un candidat sortant se représente et utilise les moyens d'une collectivité pour préparer sa réélection, il est dans une position beaucoup plus favorable qu'une personne se présentant pour la première fois ou n'ayant pas de mandat. Cette obligation de neutralité est contenue dans l'article L52-1 du Code Électoral et s'il est établi qu'un candidat a utilisé les moyens de la collectivité dont il avait un mandat pour promouvoir sa candidature, l'élection peut être annulée. Cette interdiction vaut également pour les sites Web et les comptes de réseaux sociaux. Si la page Facebook d'une collectivité locale est utilisée pour faire l'éloge des actions d'un candidat sortant, cela a le même impact que si les habitants de la collectivité le recevaient dans leurs boîtes aux lettres. *Quid* des moyens de communications que l'on pourrait dire privé ? Un candidat sortant ne pourra pas non plus utiliser l'adresse mail obtenue au titre de son mandat, surtout si l'adresse est nomducandidat@nomdelacollectivite.fr. De la même manière, il est interdit d'utiliser la ligne de téléphone portable pour envoyer ou recevoir des SMS permettant de faire la propagande électorale du candidat sortant. Il n'existe pas (encore) de jurisprudence spécifique à la communication électronique mais dans la mesure où le Tribunal Administratif et le Conseil d'État jugent ce type d'affaire de façon très concrète, il y a fort à parier que ce type de litiges surviendra dans les années à venir.

Pour autant, toute communication institutionnelle de la part d'un candidat sortant n'est pas à exclure. Il peut tout à fait faire la promotion de sa collectivité si le but recherché n'est pas de le favoriser. S'il s'agit de mettre en avant des arguments touristiques, économiques, commerciaux de la localité, c'est tout à fait acceptable.

Les hypothèses que l'on vient d'évoquer partent du postulat que le candidat sortant d'une collectivité va se

À NE PAS MANQUER !

HORS-SÉRIE N°77



J'APPRENDS LA PROGRAMMATION ORIENTÉE OBJET EN 7 JOURS !

COMPATIBLE LINUX / MAC OS X / WINDOWS

DISPONIBLE DÈS LE 13 MARS CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR : www.ed-diamond.com

représenter et qu'il a fait clairement connaître ses intentions. Mais en réalité, la candidature d'une personne n'existe que lorsqu'il a déclaré à la préfecture un mandataire financier, chargé de recueillir et de gérer les fonds permettant le financement d'une élection. La déclaration en préfecture doit intervenir au plus tard vingt jours avant le scrutin. Donc, si l'élu ne se déclare pas publiquement, que le mandataire financier n'est pas enregistré, la candidature n'existe pas et officiellement, la communication reste institutionnelle.

Cela ne veut pas dire pour autant qu'un élu qui se déclarerait candidat sortant au dernier moment, qui aurait détourné les moyens de communication institutionnelle à son profit, ne risque pas de voir son élection annulée. En effet, le Conseil d'État passe par un faisceau d'indices pour déterminer si une campagne électorale a été menée de façon loyale et n'hésite pas à annuler des élections, comme en mars 2008 pour les municipales de Briançon.

3 À vos marques. Prêts ? Partez !

Vingt jours avant le scrutin, tous les candidats doivent être déclarés en préfecture et la propagande, que l'on pourrait qualifier d'agressive peut débuter. Pour l'avoir vécu, j'aurai tendance à dire que tous les coups sont permis ou presque : tracts, meetings, page Facebook, compte Twitter, site Web, etc. Tous les moyens sont exploitables.

Le nerf de la guerre reste le financement d'une campagne. Une page Facebook ou un compte Twitter ne coûtent rien à la création mais il faut quelqu'un pour l'animer et si le candidat souhaite avoir un site Web dédié à la communication, les coûts relatifs à la création et à l'hébergement doivent impérativement figurer dans les comptes.

Dans le cadre de la communication numérique, sont autorisés les frais suivants :

- Achat d'un ou plusieurs noms de domaine ;
- Achat ou location d'un hébergement en France de préférence ou dans un État au sein de l'Union Européenne ;
- Achat d'une ou plusieurs prestations relatives à la création et à la gestion d'un site Web ;
- Achat d'une ou plusieurs prestations relatives à l'animation des comptes sur les réseaux sociaux.

Sont formellement interdits :

- L'achat de mots-clefs dans Google Adwords ;
- L'achat de publicité sur Facebook ;
- L'achat de tweets et de *trendic topic* sur Twitter ;
- Toutes les formes de publicité électronique.

Quid des prestations suivantes :

- L'achat de fans sur Facebook ou de *followers* sur Twitter ?
- L'achat de fichiers de mailing et de SMS ?
- De la mise en place d'un système de recueillement de dons par voie électronique ?

Dans la communication politique et institutionnelle, il n'y a, pour le moment, pas d'illustration formelle permettant de dire sans aucune ambiguïté que l'achat de fans et de *followers* est interdit. Mais si on se réfère à l'article L52-1 du Code Électoral, on comprend bien que cette pratique est illicite. Le sens général de cet article est de dire que le candidat peut payer pour être visible mais ne doit certainement pas être déloyal ni se transformer en un produit de consommation. Par ailleurs, l'achat de fans sur Facebook se voit très facilement. Sur toutes les pages Facebook, les statistiques d'activités sont librement accessibles. Si un candidat passe de 24 000 « likes » à 50 000 en quelques heures, sans qu'aucune actualité ne vienne le justifier, cela paraîtra suspect, d'autant qu'un grand nombre d'applications Web permettent d'obtenir des statistiques très fines sur les fans d'une page Facebook, notamment la localisation. Si soudainement, un candidat de la Mayenne passe de 15 000 fans à 75 000 fans sur une période de temps très réduite et que 45% des fans sont localisés en Inde, on peut raisonnablement supposer qu'il y a eu achat de fans. Cela peut paraître tentant mais ce moyen est parfaitement déloyal et là encore, le Tribunal Administratif sera tout à fait en droit d'annuler une élection. L'analyse est similaire pour les *followers* sur Twitter.

Concernant l'achat de fichiers de mailing et de SMS, le candidat dispose d'une très grande liberté. Il peut utiliser les fichiers qu'il a lui-même constitué, définis pour fichiers internes par la CNIL et il peut également acheter auprès d'une société privée des fichiers commerciaux. Mais il devra veiller à ce que le consentement des personnes soit respecté, faire figurer toutes les informations de désabonnement sur les mailings envoyés et ne pas faire de sélection sur des critères prohibés (origines ethniques, orientation sexuelle, opinion religieuse, politique et/ou syndicale). Le vendeur du fichier devra, au préalable, avoir informé les personnes que leurs coordonnées sont susceptibles d'être utilisées à des fins de communication politique et le candidat devra s'assurer que le consentement a bien été recueilli. Le candidat devra également donner l'origine du fichier aux personnes, la possibilité de se désabonner et indiquer que le fichier a été obtenu par un prestataire extérieur.

Enfin, 48h avant le scrutin, la campagne s'arrête : pas de messages, pas de tweets, pas de publication sur Facebook et toute la difficulté pour un candidat va être de discipliner ses militants pour éviter toute forme de propagande préjudiciable pendant ces 48h. En effet, si propagande il y a pendant ces 48h, en cas de contestation, cela peut être de nature à faire annuler une élection.

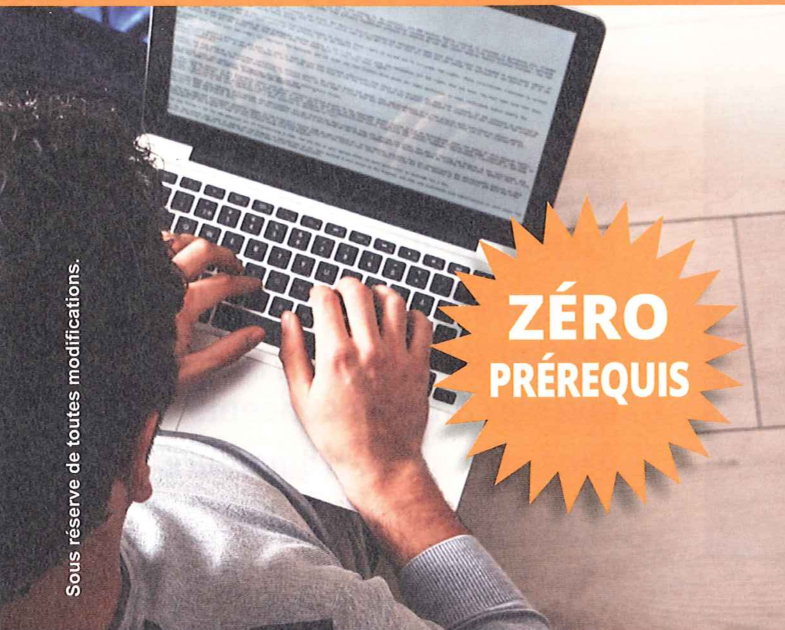
4 Et l'argent dans tout ça ?

La question des dons est juridiquement et techniquement intéressante. Dans le cadre d'une élection, le candidat doit désigner un mandataire financier, qui doit ouvrir un compte en banque distinct et peut commencer à recueillir des dons pour financer la campagne. L'État rembourse les frais de campagnes selon un barème défini par l'article L52-11 mais ce remboursement intervient après l'élection et présentation des comptes. Un candidat peut-il donc recourir au financement participatif à travers des plate-formes dédiées comme Ulule, KissKissBankers ou Pot Commun ? La Commission Nationale des Comptes de Campagnes a émis un avis défavorable sur le sujet car cela contrevient aux articles L52-5 alinéa 2 et L52-6 alinéa 2 : il ne doit pas y avoir d'intermédiaire financier entre celui qui donne et celui qui reçoit. Or, dans le cas du financement participatif, il y a un intermédiaire financier puisque les dons sont d'abord envoyés sur la plate-forme de financement pour ensuite être rétrocédés à la personne demandant le financement et c'est la même chose pour les comptes PayPal. Celui qui fait un don à un politique doit le faire directement sur le compte de l'association du politique.

On le voit, les règles s'appliquant aux élections ont pris le virage de numérique et la commission nationale de comptes de campagnes a travaillé sérieusement sur cette question. Mais cela n'avait pas empêché quelques cafouillages lors des municipales et j'avoue attendre avec une certaine gourmandise les prochaines irrégularités des candidats car en matière de numérique, les politiques français sont majoritairement incultes. ■

À NE PAS MANQUER !

HORS-SÉRIE N°32



Sous réserve de toutes modifications.

APPRENDRE LE SHELL EN 7 JOURS !

COMPATIBLE LINUX / MAC OS X / WINDOWS

DISPONIBLE DÈS LE 20 MARS CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR : www.ed-diamond.com

UNE HISTOIRE DE L'INFORMATIQUE #6 - LA NAISSANCE DES NAINS SURPUISSANTS

par **Pierre-Alexandre Voye** *[Caméliste heureux]*

Les ordinateurs pris en main par de grands industriels, principalement aux États-unis, il reste aux pionniers quelques chantiers au niveau matériel : tester et valider l'utilisation des transistors, trouver de meilleurs systèmes pour stocker des données en mémoire vive, le plus rapidement possible et, surtout inventer tous les concepts pour obtenir des performances de calcul maximales dans des superordinateurs surpuissants, concepts qui animent aujourd'hui nos objets du quotidien.

En 1955, alors que le Whirlwind prenait la place d'un immeuble et consommait autant d'électricité qu'une petite ville, le transistor était un composant nouveau, certes fascinant, mais dont on ne connaissait pas la résistance dans le temps et, de toute façon, encore instable. Le MIT, qui venait de terminer le Whirlwind, a décidé de tester cette innovation en l'utilisant pour l'électronique autour de la mémoire en tore de ferrite. Constatant que l'ensemble fonctionnait très bien et consommait considérablement moins que les tubes, la construction d'un ordinateur a été décidé.

1 Le TX-0, premier ordinateur à transistor

Le TX-0 a été conçu en 1955 au Lincoln Laboratories afin de tester l'intérêt de créer des circuits logiques avec des transistors. Les transistors n'avaient alors pas dix ans et le modèle à jonction venait d'être découvert en 1951. Celui-ci était encore très instable, lent et sensible à la température (les premiers modèles baignaient dans de la graisse de silicone), ce qui rendait son utilisation assez délicate. Il faut donc bien voir qu'à l'esprit de la plupart des électroniciens de l'époque, le transistor était perçu comme un gadget sans avenir, qui ne servait pas à grand chose, à part peut-être pour un poste radio. Construire un ordinateur utilisant plusieurs milliers de ces gadgets était donc un défi un peu farfelu.

En analysant son architecture, on peut considérer le TX-0 comme l'ancêtre de nos machines actuelles : il a de toute façon largement influencé la série des PDP (le PDP-1 étant une version de production du TX-2) dont l'importance dans l'histoire de l'informatique est connue (Unix fut développé sur PDP-7), et peut-être plus encore. Néanmoins, il n'utilisait pas encore le système à interruptions que l'on connaît maintenant sur les machines modernes, bien que le système offert permettait une certaine flexibilité. De même, il ne disposait pas de pile processeur, ce qui ne posait pas de problème majeur, car la mémoire était accessible peu ou prou en un seul cycle processeur. Cette machine reprenait pour beaucoup les idées développées pour le Whirlwind.

Le TX-0 était basé sur une architecture 18 bits assez ingénieuse : la mémoire pouvait atteindre au maximum 65 535 mots (soit un registre d'index tenant sur 16 bits), chaque mot stocké en mémoire ou dans un registre ayant 18 bits de longueur. Une instruction mesurant aussi 18 bits de long, on pouvait utiliser deux bits pour coder quatre instructions et 16 bits pour encoder un index mémoire.

Vous vous demandez évidemment comment faire tenir un assembleur exploitable en 4 instructions ? Par une organisation très astucieuse dont les 4 instructions de base étaient :

`sto [adresse]`	Affecte le contenu d'`[adresse]` avec le contenu du registre `AC`, sans réinitialiser `AC`
`add [adresse]`	Ajoute la valeur contenue dans `[adresse]` à la valeur d'`AC`, le résultat est stocké dans `AC`
`trn [adresse]`	Branchement vers `[adresse]+1` si `AC` est négatif, continu à l'adresse courante sinon
`opr nbr`	Exécute une des instructions spéciales selon la valeur de `nbr`

En gros, on a deux instructions de *Load/Store* où le *load* est une opération d'addition : il suffit de vider le registre pour qu'il se transforme en *load* classique. Il reste une place pour l'instruction de branchement conditionnel et les autres opérations ne nécessitant pas d'adresse en paramètre, ou utilisant le `MAR`, le registre d'index du TX-0. Parmi les registres importants on peut signaler :

`MBR`	*Memory Buffer Register*	18 bits
`AC`	*Accumulator*	18 bits
`MAR`	*Memory Address Register*	16 bits
`PC`	*Program Counter*	16 bits
`IR`	*Instruction Register*	2 bits
`LR`	*Live Register*	18 bits
`TBR`	*Toggle Switch Buffer Register*	18 toggle switches
`TAC`	*Toggle Switch Accumulator*	18 toggle switches

Le TX-0 et le TX-2 consommaient environ 2KW, à comparer aux 3MW du Whirwind, il ouvrit ainsi la porte aux ordinateurs « minis ». Le TX-0 évoluera vite vers le TX-2, plus puissant, sur lequel sera développé un logiciel révolutionnaire dont on reparlera, Sketchpad.

2 Un génie : Seymour Cray

Si l'on doit retenir quelques grands noms de la naissance de l'industrie des ordinateurs, Seymour Cray (voir figure 1) doit nécessairement y figurer. Pendant 30 ans, c'est lui qui a conçu les ordinateurs les plus puissants de son époque. Inventeur de nombreux concepts essentiels, qui équipent toujours nos machines, son empreinte a été déterminante. Né en 1925 dans le Wisconsin, son génie se manifeste dès l'âge de 10 ans, lorsqu'il construit en mécano un appareil pour convertir des cartes perforées en morse.

Ayant travaillé à briser les codes japonais pendant la guerre, ses études d'ingénieur terminées, il rejoint en 1951 la société ERA justement spécialisée dans cette activité. C'est là, qu'en coopération avec Remington Rand, la société qui a racheté l'entreprise de Mauchly et Eckert, il va faire ses armes sur la conception de l'Univac 1103. La division informatique scientifique étant mise de côté, il rejoint la société Control Data en 1958 et va y diriger la conception de sa première machine, le CDC 1604.

2.1 Le CDC 1604

Le CDC 1604, sorti en 1960, est une machine à transistors de 24 bits, doté de 32 768 mots de 48 bits (soit 192 kio), permettant de stocker deux instructions de 24 bits par mot, comme l'avait proposé Von Neumann dans son architecture. Il était extrêmement modique pour son époque (à partir de $110 000). Les 62 instructions étaient basées sur un format 6-3-15, dans lequel 6 bits étaient dévolus à l'instruction, 3 à la désignation du registre ou la spécialisation de l'instruction et 15 à l'adresse mémoire. La plupart des instructions sont dédiées aux calculs sur les entiers et flottants, et s'exécutent en un ou deux cycles.

La mémoire était divisée en deux banques, l'une pour les adresses paires et l'autre impaires, ce qui permettait de les faire travailler en parallèle.

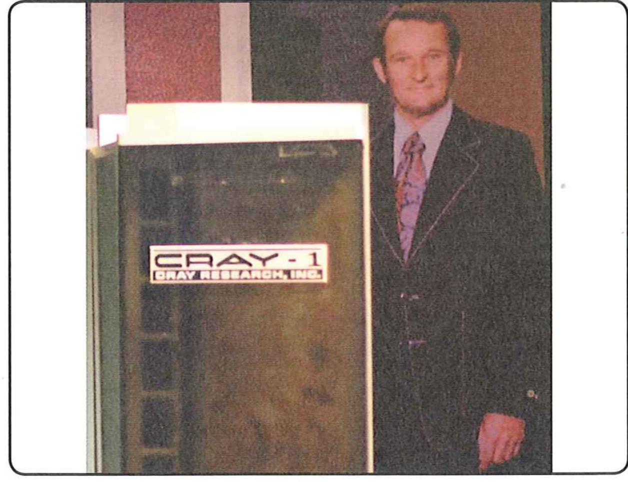


Fig 1 : Seymour Cray - "Anyone can build a fast CPU. The trick is to build a fast system."

Bien qu'à l'époque les compilateurs étaient encore balbutiants, il était offert la possibilité d'écrire des programmes en Jovial, un langage construit à partir d'Algol. Avec nos yeux d'aujourd'hui, on peut le voir comme un langage ressemblant peu ou prou au QBasic (que ceux qui ont connus ce langage lèvent la main).

Le CDC 1604 est le prélude des architectures géniales de Cray : toute l'électronique de contrôle (gestion des entrées/sorties) est construite pour interférer le moins possible avec le processeur, qui ne s'occupe tant que possible que de la mémoire. Cette machine est en effet conçue pour faire du calcul haute performance et non de l'acquisition de données en temps réel (bien que ce soit possible).

En outre, d'un point de vue électronique, cette machine, une des premières à transistors à l'époque, est construite par empilement très organisé de cartes électroniques de quelques centimètres carrés, avec une intégration aussi poussée que possible (pour l'époque). Chaque carte représente une unité (porte logique à 4 bits, additionneur, etc...) qui sont combinés les uns avec les autres. Cette architecture modulaire basée sur des composants standardisés réduit la panne, facilite la réparation et permet la conception d'une machine rassemblant énormément de transistors. Cette machine, ainsi que le TX-0, inspirera fortement les PDP-5 et PDP-8 dont l'importance sera grande pour la suite.

2.2 La série CDC 3000

Reprenant le 1604, le CDC 3300 (donc de la série CDC 3000) était une machine 24-bit, utilisant le complément à 1, doté de deux registres `A` et `Q`. `A` était le registre principal, et `Q` le registre quotient, principalement utilisé pour les multiplications et divisions. On disposait aussi de 3 registres d'index `R1`, `R2`, `R3` pour la manipulation de tableaux.

Le CDC 3300 était la machine la plus puissante de son époque, atteignant 1 MIPS (voir encadré) en 1963. En fait, à partir de cette date, plus personne n'arrivera à faire de machines plus puissantes que celles de Seymour Cray et ce, pendant 30 ans.

Qu'est-ce qu'un MIPS ?

MIPS est l'acronyme de Million Instruction Per Second. Soit un million d'instructions par seconde. Si le processeur est capable d'exécuter une instruction par seconde, une cadence horloge de 1 Mhz suffira, sinon celui-ci devra être plus rapide. Sur un 8088, certaines instructions mettaient 80 cycles à s'exécuter, malgré 4,77 Mhz, on n'atteignait pas forcément 1 MIPS et certainement pas en calcul flottant.

2.3 La série CDC 6000

La série 6600, vendue à partir de 1965 propose une amélioration architecturale vraiment innovante **[1]** : les Peripheral Processor Units (ou PPU). La vitesse n'était pas en reste : grâce à des transistors de très bonne qualité, la machine tournait à 10 Mhz, soit 10 fois plus rapidement que la plupart des machines de cette époque. Persévérant son concept de spécialiser le processeur pour le calcul pur, il a découplé radicalement l'électronique d'interfaçage avec les périphériques externes et le processeur de calculs.

Voyons plus en détail comment fonctionnait cet ordinateur vraiment original (vous pouvez voir un CDC 6600 en figure 2).

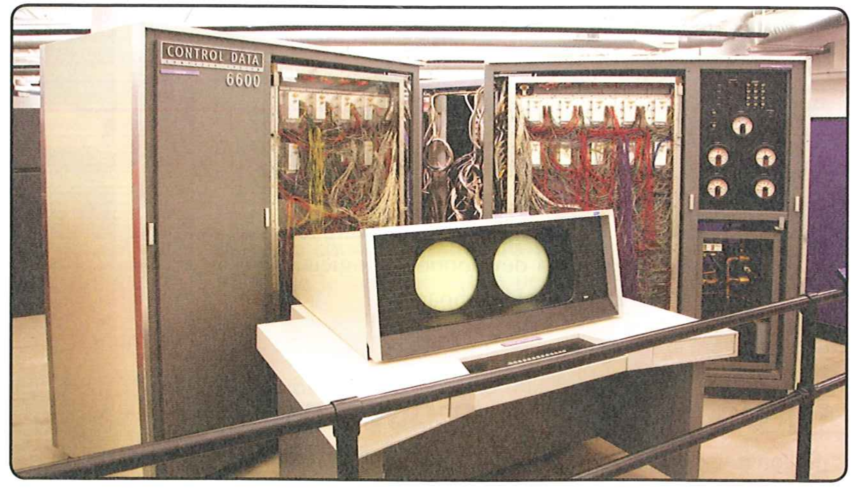


Fig 2 : Un CDC 6600 avec son bureau de commande et ses deux oscilloscopes faisant office d'écrans.

2.3.1 Le PPU

Le 6600 par exemple, était doté de 10 PPU, ayant chacun accès à la mémoire. Le processeur avait lui aussi accès à la mémoire et n'était doté que d'instructions de calculs et de branchements. Chaque PPU était un processeur logique, doté d'un jeu d'instructions pour gérer les accès périphériques et déverser les informations en mémoire centrale. Chaque PPU pouvait communiquer avec les autres. Il pouvait aussi communiquer avec le CPU, soit en remplissant la mémoire, nécessitant une coopération avec le CPU pour éviter les conflits, soit en utilisant un mécanisme de "swapping" forçant le CPU à changer de contexte d'exécution et se brancher sur une autre adresse d'exécution du code, permettant à l'OS de changer le processus à l'exécution.

En terme matériel, les dix PPU n'étaient que virtuels : en réalité seul un PPU était soudé, cadencé à 10Mhz, mais il gérait 10 jeux de registres qu'il manipulait à tour de rôle, donnant l'impression de disposer de 10 PPU. Cette approche permettait d'éviter de gâcher les onéreux et spéciaux transistors de la machine et permettait de conserver une efficacité même si les temps d'attente des entrées/sorties pouvaient être très long - pensons aux disques-durs et périphériques à bande de l'époque dont la bande passante dépassait rarement les 300 kio/s.

Cela dit, les optimisations étaient possibles, car chaque PPU possédait sa propre mémoire de 4096 mots de 12 bits pour les besoins de buffering.

2.3.2 Le CPU

Le CPU était une unité d'opération manipulant des mots de 60 bits. Il était doté du premier cache d'instructions (8 valeurs) de l'histoire ! Il comprenait 8 registres d'opérandes, 8 registres d'index et 8 d'incrémentations. De plus, il disposait d'une mémoire centrale en tore de ferrite de 128 000 mots de 60 bits, soit 940 kio.

Le processeur central est aussi un bijou d'architecture, divisé en 10 unités indépendantes, chacune dévolue à un groupe d'instructions. Celles-ci étaient donc capable de s'exécuter en parallèle. Les 10 unités étaient les suivantes :

- Addition flottante
- Multiplication flottante
- Division flottante
- Addition entière
- Incrémentation
- Opération booléenne
- Décalages
- Branchements

Cela signifiait que si le programmeur était malin, il pouvait faire s'exécuter en même temps des instructions sans dépendances entre elles. John Carmack a utilisé cette technique sur x86 pour Quake 4 en mélangeant, instruction par instruction, du calcul sur entiers et flottants.

Ce jeu d'instructions réduit, fût la base des jeux d'instructions RISC (Reduced Instruction Set Computer) ultérieurs, ce qui était complètement nouveau à l'époque car les jeux d'instructions proposaient couramment 100 à 200 instructions : la programmation étant réalisée en assembleur, il fallait simplifier la vie du programmeur en lui proposant un langage machine « haut niveau ».

2.3.3 La programmation d'un CDC

De par son architecture atypique, la programmation des CDC 6000 ressemblait beaucoup à de la programmation avec instructions MMX, SSE et autres 3Dnow actuels, en l'espèce une seule instruction pour travailler sur de multiples données.

En 1963, lors de la sortie du 6600, trois fois plus puissant que le meilleur modèle IBM, un échange célèbre, par presse interposée, eut lieu entre Thomas Watson, PDG de Big Blue et Seymour Cray :

« La semaine dernière, Control Data a annoncé son 6600. J'ai appris qu'il a été développé dans un laboratoire comprenant seulement 34 personnes, y compris son concepteur. Parmi eux, 14 sont ingénieurs et 4 sont programmeurs. Contrastant entre cet effort modeste et nos vastes équipes de R&D, je ne parviens pas à comprendre comment nous avons pu perdre le leadership en laissant une autre entreprise construire la machine la plus puissante au monde avec une aussi petite équipe. »

La réponse de Seymour Cray est restée fameuse :

« Il semble que M. Watson ait répondu à sa propre question. »

2.4 Le 7600

Le CDC 7600 **[2]** est la dernière machine fonctionnelle conçue par Cray à Control Data. La machine est mise sur le marché en 1969. La société connaissant des difficultés lors de la conception du 8600, Cray ira créer sa société Cray Research pour créer le Cray-1. C'est aussi le dernier super-ordinateur à transistors de Seymour Cray. De toute façon, le 8600 ne marchera jamais vraiment bien, arrivant aux limites de complexité permise par une machine à transistors.

Le 7600 a inauguré des techniques hardware toujours présentes dans les microprocesseurs actuels : le processeur est pipeliné, concept découpant l'exécution d'une instruction en morceaux, tous les étages du pipeline travaillant à chaque cycle, `1000` instructions prennent `1000 + n` cycles, `n` étant le nombre d'étages du pipeline. Cette innovation, difficile à mettre en place, a multiplié les performances par 3. Tous les processeurs un tant soit peu orientés puissance de calcul utilisent cette technologie aujourd'hui.

Continuant sur la lancée du 6600, le 7600 propose un cache de 40 instructions et ajoute un cache de données de 60 000 mots de 60 bits. De même, lorsqu'une instruction est chargée dans le cache stockant le code, le CPU va remplir le cache de données

avec la valeur se trouvant à l'adresse à laquelle fait référence l'instruction. Les processeurs actuels utilisent ce genre de techniques : le processeur ne travaille qu'avec le cache L1 et échange des données avec le cache L2 qui lit ou met à jour la mémoire si besoin. Le nombre d'instructions reste limité, surtout par rapport aux machines IBM de l'époque qui proposaient facilement 300 instructions micro-codées. Bien évidemment, toutes les instructions du CDC sont câblées, pour plus d'efficacité.

Le 7600 a commencé à poser un certain nombre de problèmes électriques : la complexité de la machine, sa sensibilité à la panne d'un de ses composants, et les difficultés de refroidissement vont rendre l'architecture impossible à développer plus avant. La figure 3 montre un module utilisé dans la construction des CDC 6600 et 7600.

Fig 4 : Le Cray-1, dessiné pour s'accorder avec la déco du bureau très seventies, était fourni avec un banc en cuir chauffant

2.5 Le Cray-1

Quatre ans après son départ de Control Data, Cray annonce au monde l'avènement de ce qui sera son plus grand succès, le Cray-1 (voir figure 4), en 1976. Cray consent enfin à adopter la technologie des circuits intégrés, qui ont longtemps été beaucoup plus lents que les meilleurs transistors

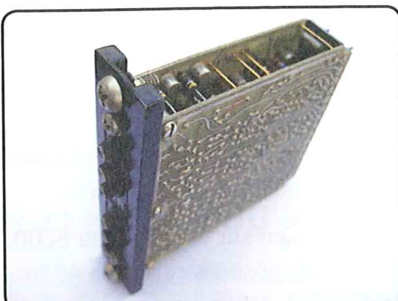


Fig 3 : Les CDC 6600 et 7600 étaient construits avec un assemblage de modules enfichables tels que celui-ci. On atteignait une très grande densité pour l'époque.

bipolaires qu'il utilisait pour les CDC. Le Cray-1 est un compromis entre une architecture vectorielle et scalaire, offrant une unité de calcul pour chacune des approches. On appelle « vectorielle » une architecture dans laquelle on peut demander en une instruction à la machine d'additionner 10 000 nombres stockés dans deux tableaux pour mettre le résultat dans un troisième. Le Cray-1 pouvait travailler sur des tableaux de 64 valeurs à la fois. Les GPU actuels sont une synthèse de ces deux architectures.

Cray se fait construire par Motorola et Fairchild des circuits intégrés sur mesure, construits selon la technique Emitter Coupled Logic (ECL), offrant une vitesse de commutation très grande, même sur circuit intégré (comparé aux TTL ou MOS de l'époque, très lents), mais beaucoup plus consommateurs en ressources. De cette façon, il est possible d'atteindre 80 Mhz sur l'ensemble du circuit. Le circuit de refroidissement de la machine a fait l'objet d'une attention particulière vu sa propension à atteindre des températures qui laisseraient pantois un gamer d'aujourd'hui.

Le Cray-1 est une machine 64 bits à une époque où l'habitude de faire des machines avec un nombre de bits différents d'une puissance de deux commence (il était temps) à disparaître. Doté de 1 M mots, soit 8 Mio de mémoire vive, répartie sur 16 banques pour être accessible en parallèle, cette machine est dotée d'un cache de 64 instructions et 64 valeurs, synchronisé avec la mémoire centrale.

Le concept de PPU évolue : 6 canaux externes accédant à la mémoire tous les 4 cycles sont mis à disposition du programmeur, ce qui est plus simple à programmer. Un SuperNova S/200 était utilisé pour contrôler la machine (qui envoyait les processus à exécuter sur la machine) exploitée par un Cray Operating System. Les modèles suivants tourneront sous un Unix adapté.

Cette machine dont la puissance est équivalente à une machine de bureau de 1996 pesait cinq tonnes et consommait 115 KW de puissance, plus autant pour le refroidissement.

Le Cray-1 fut un réel succès commercial : vendu environ 8 millions $ l'unité, il s'en est vendu environ une centaine. C'est son taux de disponibilité, 98 %, qui a fait la différence, à une époque où la fiabilité des machines n'était pas du tout ce qu'elle est aujourd'hui.

2.6 Rattrapé par les microprocesseurs

Jean Louis Gassée raconte sa discussion avec Seymour Cray, peu avant sa mort dans un accident de voiture en 1996, où Cray avoua « Ça va, j'ai compris, les microprocesseurs ont gagné !». C'est peu dire que Cray a résisté jusqu'au bout à l'utilisation de microprocesseurs pour construire des super-ordinateurs, les concevant « à la main » avec des transistors puis des circuits intégrés.

Depuis 15 ans, où le nombre d'instructions par cycle d'un microprocesseur grand public atteint 1 voire 2 (on était environ à 1/80 en 1980, à part sur le 6502...), il est quasi très difficile, et de toute façon trop couteux, de gagner quoi que ce soit sur une architecture « custom ».

3 Linpack

Qu'est-ce qu'un MFLOPS ?

MFLOPS est l'acronyme de Million FLoating OPeration Per Second. Soit un million d'instructions flottantes par seconde. Cette notion est plus floue et difficile à mesurer, car pendant longtemps, toutes les instructions n'avaient pas la même durée d'exécution : une multiplication est généralement beaucoup plus lente qu'une addition. Linpack a mis tout le monde d'accord en mesurant des MFLOPS en situation réelle.

Le Linpack est à l'origine une bibliothèque de fonctions écrites dans les années 1970 pour l'algèbre linéaire en Fortran. L'exécution de certains de ses algorithmes a depuis lors servi de standard pour comparer la puissance des ordinateurs entre eux. Certaines des machines présentées précédemment étant encore branchées à la fin des années 1970, ce qui permet de pouvoir les comparer en situation réelle. Nous utilisons les résultats de **[3-6]** afin de comparer avec les machines que nous connaissons.

Les résultats sont à prendre avec les pincettes d'usage, car la qualité du compilateur et du code assembleur écrit à la main – ou pas – ont une grande influence, à tel enseigne que les résultats varient entre machines similaires (ayant elles-mêmes leurs options). On a pris les meilleurs résultats obtenus pour chaque machine, lorsqu'il s'en présentait plusieurs. Les chiffres pour les machines anciennes sont reconstitués.

Machine	Année d'introduction	MFLOPS
Zuse Z1	1938	0,0000005
ENIAC	1946	0,0000182
EDVAC	1952	0,0000217
Whirlwind I	1950	0,000071
Whirlwind II	1953	0,00015
Atari ST (µprocesseur 68 000)	1985	0,0051
SAGE/AN FSQ7	1958	0,008
TX-2	1956	0,01
IBM PC (coprocesseur 8087)	1981	0,01
IBM 370/158	1972	0,187
Intel i386 DX 40 Mhz	1987	0,3324
CDC 6600	1964	0,477
Intel 486 DX-2 66mhz	1992	0,56
IBM 3033 (superdordinateur IBM)	1977	1,77
IBM 370/195	1973	2,31
CDC 7600	1969	4,64
Intel Pentium 60 Mhz	1993	5,3
Cray 1	1976	14
Apple Power Mac 5500/250 (PowerPC 603e 250 Mhz)	1996	14
Cray 1S	1979	27
Intel Pentium 166 Mhz	1996	28,37
Qualcomm APQ8064T Snapdragon 600 1,7 Ghz QuadCore (ARM)	2013	31,38 (un seul coeur)
Cray X-MP	1982	53
IBM RS/6000-390 (66.5 MHz)	1994	53
Cray 2	1985	62
AMD K6-II 350 Mhz	1998	64
Intel Pentium III 450 Mhz	1999	65

On constate un schéma intéressant dans ces statistiques : plus le temps avance, plus les microprocesseurs rattrape rapidement les super-ordinateurs : il a fallu 28 ans à Intel pour dépasser un CDC 6600, mais seulement 17 ans pour dépasser un Cray 1S. Le Cray 2 a quant à lui été dépassé par un processeur grand public en moins de 13 ans. Les super-ordinateurs actuels ne sont maintenant que des clusters de version haut de gamme de processeurs grand public.

4 Les interruptions

Le mécanisme d'interruptions programmables est disponible sur la quasi totalité des microprocesseurs actuels. Néanmoins, bien qu'apparu assez tôt, il n'a pas atteint tout de suite sa forme actuelle.

C'est le DYSEAC, en 1954, qui fût le premier ordinateur portable (dans un semi-remorque...) au monde à proposer un mécanisme d'interruption : il possédait deux Program Counter (registre d'index de code), et la détection d'une I/O impliquait que le programme change de Program Counter. Chaque instruction possédait un bit pour indiquer lequel elle utilisait. La notion de vecteur d'interruption, autrement dit, l'affectation d'une adresse de code à exécuter lorsque survient un événement, a été inventée par le TX-0 en 1958.

5 Complément à 1 ou 2 ?

Le format binaire permet de représenter les nombres entiers positifs sur un intervalle, mais comment représenter les nombres négatifs ?

Nombre de machines des années 50 à 70 utilisaient soit un complément à 1 soit un complément à 2. Dans un complément à 1, les nombres négatifs sont représentés en réalisant un `NOT` sur la valeur binaire, ce qui implique que `0` ait deux représentations (`1111` et `0000` en 4 bits). Dans un complément à 2, en plus de négativer, on ajoute `1` à la valeur, ce qui permet d'éviter d'avoir deux représentations pour `0`.

Le débat a longtemps fait rage entre les experts dans les années 1950 et 1960 entre le complément à 1 ou à 2, voire l'approche à magnitude (un bit de signe et pis c'est tout). Les arguments étaient très solides pour chaque approche et très tranchés. Le complément à 2 l'a finalement emporté, car il est plus simple à implémenter au niveau matériel.

Conclusion

La puissance de nos ordinateurs actuels n'est pas uniquement imputable aux milliards de transistors que l'on est capable de graver sur moins d'un centimètre carré, mais est aussi l'héritage de décennies de recherches et développements, et en grande partie celles de Seymour Cray qui a conçu nombre de techniques encore utilisées dans la plupart des microprocesseurs actuels. Il est aussi à l'origine de l'avènement du RISC, jeu d'instructions réduit, qui allait s'imposer dans les années 1990. ■

Références

[1] J. Thornton, « Design of a computer - The control data 6600 », 1970 : http://ed-thelen.org/comp-hist/DesignOfAComputer_CDC6600.pdf

[2] Le CDC 7600 : http://research.microsoft.com/en-us/um/people/gbell/craytalk/sld050.htm

[3] C. Moler, « The LINPACK Benchmark » : http://blogs.mathworks.com/cleve/2013/06/24/the-linpack-benchmark/

[4] J. Dongarra, « Performance of Various Computers Using Standard Linear Equations Software », University of Manchester, 2014 : http://www.netlib.org/benchmark/performance.pdf

[5] N. Juffa, « Performance comparison Intel 386DX, Intel RapidCAD, C&T 38600DX, Cyrix 486DLC », University of Karlsruhe, 1993 : http://www.textfiles.com/programming/cyrix.pf

[6] Définition d'un superordinateur : https://fr.wikipedia.org/wiki/Superordinateur

EQUIPE DE SUPER-HÉROS LINUX/KERNEL

RECHERCHE NOUVEAU COÉQUIPIER !

Tu es un(e) passionné(e) de Linux et de développement Kernel ?
Le logiciel embarqué ne te fait pas peur ?
Ton passeport est à jour ?

ALORS CE JOB EST FAIT POUR TOI !
REJOINS BAYLIBRE... ET TRAVAILLE POUR LES PLUS PRESTIGIEUSES ENTREPRISES HIGH-TECH CALIFORNIENNES !

CV et lettre de motivation à adresser à :
jobs@baylibre.com

?
FUTUR SUPER-HÉROS
Mega Linux Hacker

©vcomk

www.baylibre.com

LES EXPRESSIONS RÉGULIÈRES : LA THÉORIE ET LES NORMES

par Tristan Colombo

Tout le monde a déjà entendu parler des expressions régulières mais la population mondiale se décompose en deux groupes : ceux qui connaissent les expressions régulières et les utilisent avec bonheur chaque jour... et les autres. Cet article s'adresse plus particulièrement au second groupe (mais, pour les membres du premier groupe, quelques rappels ne font jamais de mal).

Les expressions régulières sont des outils très pratiques permettant de détecter, supprimer ou modifier des éléments précis à l'intérieur d'une chaîne de caractères. Une expression régulière n'est en fait qu'une notation utilisée pour décrire certains langages. Nous allons donc étudier ici une partie de la théorie des langages, déjà abordée pour construire un langage de programmation dans GNU/Linux Magazine n°175 **[1]**.

Pour commencer, nous verrons quel formalisme a été utilisé pour définir ces éléments puis, nous nous pencherons sur l'utilisation des expressions régulières en informatique : les normes, les différences en fonction des langages et enfin la mise en pratique sur un cas concret.

1 Expressions régulières

Les **expressions régulières**, encore appelées parfois **expressions rationnelles** ou **regex** (abréviation de *regular expression* en anglais), ont été introduites par Kleene en 1956 **[2]** pour définir un langage. Pour pouvoir comprendre leurs définitions, nous aurons besoin de quelques rappels de vocabulaire :

- Un **alphabet** est un ensemble fini de caractères (n'importe quel type de caractère : des lettres, des chiffres, etc.). On le note Σ. Par exemple, l'ensemble Σ_1 `= {a, b, c, 1, 2, 3, _}` est un alphabet.
- Une **chaîne** (ou **mot**) `m` sur Σ est une concaténation de caractères de Σ. Par exemple, `ab_12` est une chaîne sur l'alphabet Σ_1.
- La **chaîne vide** (ou **mot vide**) est notée `ε`.

Sachant cela, nous pouvons définir une expression régulière à partir d'un cas de base et de quatre règles d'induction permettant de construire des expressions régulières plus grandes à partir d'expressions régulières plus petites.

1.1 Cas de base

Le cas de base est défini par deux règles :

1. `ε` (la chaîne vide) est une expression régulière ;
2. Si `a` appartient à l'alphabet Σ, alors `a` est une expression régulière.

Ces règles ne nous permettent pas de définir d'expressions régulières très complexes... Pour l'instant, si nous avons un alphabet `{a, b, c}` nous ne pouvons définir que quatre expressions régulières : `ε`, `a`, `b`, et `c`. Nous avons donc besoin de règles supplémentaires pour définir des expressions plus complexes par récurrence.

1.2 Règles d'induction

Les règles d'induction sont au nombre de quatre. Pour les définir, supposons que `R` et `S` sont des expressions régulières. Nous avons alors :

1. La **concaténation** : `RS` est une expression régulière. Par exemple, si `R = a` et `S = b`, alors `ab` est une expression régulière. Si `R` et `S` sont déjà des expressions plus complexes engendrant des ensembles tels que `{a, ab}` et `{cc, bc}`, alors elles permettront d'obtenir `{acc, abc, abcc, abbc}`.
2. L'**union** : `R|S` est une expression régulière. Si `R = a` et `S = b` alors `R|S` permet d'obtenir `{a, b}`.
3. Le **parenthésage** : on peut ajouter des paires de parenthèses autour d'une expression régulière sans modifier sa signification. Par exemple `R`, `(R)` et `((R))` représentent la même expression régulière `R`.
4. L'**étoile de Kleene** (ou *Kleene star*) : `R*` est l'ensemble obtenu par concaténation des chaînes de `R` (chaîne pouvant éventuellement être `ε`, même si la chaîne vide ne fait pas partie de `R`). Ainsi, pour `R = {a, b}`, `R*` permet d'obtenir `{ε, a, b, ab, aa, bb, aab, ...}`.

À ces règles, nous pouvons adjoindre des priorités sur les opérateurs (qui sont tous associatifs à gauche) pour simplifier les écritures : étoile de Kleene > concaténation > union.

Il est ainsi trop lourd d'écrire `(((a)*(b))|(c))` car cette expression se simplifie en `a*b|c`.

1.3 Théorèmes algébriques

Il existe ensuite toute une suite de théorèmes algébriques « classiques » ou « intuitifs » applicables aux expressions régulières. En voici un rapide rappel :

- Associativité de l'union : `R|(S|T) = (R|S)|T`
- Commutativité de l'union : `R|S = S|R`
- Associativité de la concaténation : `R(ST) = (RS)T`
- Distributivité de la concaténation par rapport à l'union : `R(S|T) = RS|RT`.
- `ε` est l'élément neutre pour la concaténation : `εR = Rε = R`
- Idempotence de l'étoile de Kleene : `R** = R*`

Ces bases théoriques étant posées, nous pouvons maintenant nous pencher sur l'aspect pratique.

2 Deux types de standards

L'application des expressions régulières en informatique a donné naissance à deux normes : la norme POSIX et la norme PCRE. On s'est inspiré des expressions régulières et on les a adaptées à un usage informatique (jusqu'à ce qu'elles ne soient plus strictement des expressions régulières telles que définies mathématiquement).

2.1 Expressions régulières POSIX

POSIX signifie *Portable Operating System Interface for uniX* (interface portable de système d'exploitation pour unix). C'est une collection de normes qui indique comment doivent se comporter certaines commandes d'un système unix. Les commandes `grep` et `egrep` suivent par exemple la norme définissant les expressions régulières POSIX.

2.1.1 POSIX BRE

La norme la plus simple définissant les expressions régulières est la norme POSIX BRE **[3]** pour *Basic Regular Expression*. Le tableau ci-après résume les méta-caractères autorisés par cette norme et leur signification. Pour illustrer le comportement de ces méta-caractères, nous utiliserons la commande `grep` que nous appliquerons sur le fichier `test_regex.txt` suivant :

```
GNU/Linux Magazine GNU
Linux Magazine$ et ^GNU/Linux
LinuxLinuxLinux Maag
PNU/Minux Tinux Linux
Piiingouin!gouin!
```

Méta-caractère	Description

`^`	En début d'expression régulière signifie « début de ligne » (pas d'autre caractère avant). Sinon il s'agit du caractère `^`.
	`$ grep "^GNU" test_regex.txt`
	`GNU/Linux Magazine GNU`
	`$ grep "et ^GNU" test_regex.txt`
	`Linux Magazine$ et ^GNU/Linux`

`$`	En fin d'expression régulière signifie « fin de ligne » (pas d'autre caractère après). Sinon il s'agit du caractère `$`.
	`$ grep "Magazine$" test_regex.txt`
	`$ grep "Magazine$ et" test_regex.txt`
	`Linux Magazine$ et ^GNU/Linux`

.	N'importe quel caractère excepté le retour à la ligne (à condition d'être utilisé à l'extérieur des crochets, sinon il s'agit du caractère point). $ grep ".NU" test_regex.txt GNU/Linux Magazine GNU Linux Magazine$ et ^GNU/Linux PNU/Minux Tinux Linux Pour rechercher le caractère ., il faudra préfixer celui-ci par un antislash.
*	Recherche la présence de l'élément précédent répété entre 0 et n fois. $ grep "i*ng" test_regex.txt Piiingouin!gouin!
[ ]	Représente un caractère pris dans l'ensemble défini entre accolades. Par exemple [abc] signifie le caractère a ou b ou c. Il est possible de définir des intervalles à l'aide du caractère - : [a-d] est équivalent à [abcd].
[^]	Représente un caractère qui n'est pas présent dans l'ensemble défini entre accolades. Il s'agit en fait d'un [ ] inversé. $ grep "Ma[^g]" test_regex.txt LinuxLinuxLinux Maag
\(\)	Défini une sous-expression ou un bloc sur lequel d'autres traitements pourront être appliqués. Par exemple \(Linux\)* pour rechercher plusieurs occurrences de Linux (entre 0 et n répétitions).
\{m\}	Recherche la présence de l'élément précédent répété exactement m fois. $ grep "\(Linux\)\{3\}" test_regex.txt LinuxLinuxLinux Maag Note : fonctionne également avec 2 occurrences puisque LinuxLinux est inclus dans LinuxLinuxLinux.
\{m,n\}	Recherche la présence de l'élément précédent répété entre m et n fois. $ grep "\(Linux\)\{2,4\}" test_regex.txt LinuxLinuxLinux Maag
\{m,\}	Recherche la présence de l'élément précédent répété au moins m fois. $ grep "\(Linux\)\{3,\}" test_regex.txt LinuxLinuxLinux Maag
\n	Ici n est un entier compris entre 1 et 9. Ce méta-caractère permet de faire référence à un bloc de l'expression régulière déjà découvert (on parlera de **motif** ou **pattern**). Avec un exemple ce sera plus simple : $grep "\(.inux\)\1\1" test_regex.txt LinuxLinuxLinux Maag \(.inux\) peut correspondre à Linux, Minux, ou Tinux mais comme on concatène deux fois à la suite la chaîne qui a été découverte, l'expression régulière ne peut correspondre qu'à LinuxLinuxLinux. Utiliser ce méta-caractère fait que l'expression régulière *informatique* ne suit plus la définition mathématique.

En plus de ces méta-caractères, des intervalles ont été prédéfinis et portent le nom de **classes**. Il ne s'agit que de raccourcis pouvant être utilisés dans [ ] comme vous pouvez le voir dans le tableau suivant où je n'indique que les classes les plus courantes :

Classe	Équivalence	Description
[:upper:]	A-Z	Lettres majuscules
[:lower:]	a-z	Lettres minuscules
[:alpha:]	A-Za-z	Lettres
[:digit:]	0-9	Chiffres
[:space:]	\t\n\r\f\v + espace	Caractères d'espacement

Voici un exemple d'utilisation des classes :

```
$ grep "^\([[:upper:]]NU\)" test_regex.txt
GNU/Linux Magazine GNU
PNU/Minux Tinux Linux
```


2.1.2 POSIX ERE

La norme POSIX BRE a été étendue en une norme plus moderne, plus simple à écrire et comportant plus de méta-caractères. Cette norme se nomme POSIX ERE pour *Extended Regular Expressions*. La commande awk utilise par exemple cette norme où l'expression régulière devra être encadrée par des slashes indiquant son début et sa fin.

Pour définir la norme POSIX ERE on va partir de la norme POSIX BRE dont l'écriture va être allégée dans les cas suivants :

- les accolades n'ont plus besoin d'être « protégées » par un antislash ;
- idem pour les parenthèses ;
- conséquence des deux points précédents, pour rechercher un caractère (,), {, ou } il faudra le préfixer par un antislash ;
- le méta-caractère ^ ne désigne plus que le début d'une ligne, où qu'il soit placé. Pour détecter le caractère ^ il faudra le préfixer d'un antislash ;
- même modification pour le méta-caractère $ qui ne désigne plus que la fin de ligne.

De nouveaux méta-caractères vont également être ajoutés (les exemples utiliseront la commande awk appliquée au fichier test_regex.txt défini précédemment) :

<table>
<tr><th>Méta-caractère</th><th>Description</th></tr>
<tr><td>?</td><td>Répétition du bloc précédent 0 ou 1 fois.
$ awk '/Maa?g/ { print $0 }' test_regex.txt
GNU/Linux Magazine GNU
Linux Magazine$ et ^GNU/Linux
LinuxLinuxLinux Maag</td></tr>
<tr><td>+</td><td>Répétition du bloc précédent au moins une fois (entre 1 et n fois).
$ awk '/(gouin!)+/ { print $0 }' test_regex.txt
Piiingouin!gouin!</td></tr>
<tr><td>|</td><td>Mise en place de l'union qui n'était pas disponible dans la norme POSIX BRE.
$ awk '/^(G|P)NU/ { print $0 }' test_regex.txt
GNU/Linux Magazine GNU
PNU/Minux Tinux Linux</td></tr>
</table>

Les classes restent les mêmes que dans la norme POSIX BRE. Voici un exemple d'utilisation avec la classe [:upper:] pour les caractères majuscules (recherche en début de ligne d'un G ou d'un P suivi de deux caractères majuscules) :

```
$ awk '/^(G|P)[[:upper:]]{2}/ { print $0 }' test_regex.txt
GNU/Linux Magazine GNU
PNU/Minux Tinux Linux
```

2.2 Expressions régulières PCRE

Les expressions régulières PCRE, pour *Perl Compatible Regular Expressions*, adoptent une syntaxe plus flexible et efficace que celles définies par les normes POSIX. Comme son nom l'indique, PCRE a été tout d'abord introduit dans Perl puis adopté par de nombreux langages ou bibliothèques.

On y trouve des abréviations pour désigner les classes. Par exemple :

- \d correspond à [[:digit:]] c'est-à-dire [0-9] ;
- \w correspond à tout caractère autorisé dans un identifiant (un « mot » d'où w pour *word*) c'est-à-dire [A-Za-z0-9_]. \W correspond à l'inverse, c'est-à-dire tout caractère n'appartenant pas à \w.
- \s correspond à un caractère d'espacement soit [[:space:]] mais sans \v c'est-à-dire [\n\r\t\f]. \S est l'inverse : tout caractère qui n'est pas un caractère d'espacement au sens de \s.

On peut nommer les sous-motifs (motifs utilisés à l'intérieur d'une expression régulière). Ainsi, en POSIX ERE on pouvait écrire : /([A-Z][a-z]+) ([A-Z][a-z]+) <=> \2 \1/ (ce qui correspond à l'inversion de deux mots commençant par une majuscule comme dans Richard Stallman <=> Stallman Richard). Avec PCRE, l'expression régulière pourra être beaucoup plus lisible grâce aux noms attribués aux sous-motifs \1 et \2 : /(?<prenom>[A-Z][a-z]+) (?<nom>[A-Z][a-z]+) <=> (?P<nom>) (?P<prenom>)/.

On peut également ajouter des commentaires directement dans l'expression régulière avec la syntaxe (?# commentaire_ici).

Des options permettent de modifier l'analyse d'une même expression. On peut trouver, entre autres :

- i : insensible à la casse ;
- m : multi-lignes (ie $ détecte \n et ^ détecte le début de chaque ligne) ;
- s : le méta-caractère . détecte également le retour à la ligne \n ;
- x : mode verbeux, tout ce qui suit un caractère # est considéré comme commentaire.

Voici ce que cela peut donner en Python en activant les options m, i et x :

```
$ python3
>>> import re
>>> fic = open('test_regex.txt', 'r')
>>> lines = fic.read()
>>> re.findall('\wnu$ # Recherche [lettre_id]nu en fin de ligne',
lines, re.M | re.I | re.X)
['GNU']
```

En Perl, PHP, Ruby et Javascript les expressions régulières peuvent être utilisées nativement. En Java il faudra utiliser le paquet java.util.regex et en C il faudra utiliser la bibliothèque regex (POSIX) ou encore PCRE **[4]** ou slre **[5]** pour un support de PCRE allégé.

3 Détecter une adresse mail bien formée

Vous savez maintenant tout ce qu'il y a à savoir sur les expressions régulières. Le plus compliqué est de créer le motif qui correspondra à l'élément que l'on recherche ou que l'on souhaite extraire d'une chaîne de caractères. Pour illustrer cela nous allons vérifier qu'une adresse mail a été correctement écrite. Cela ne semble pas si complexe et pourtant... Nous allons construire plusieurs expressions régulières (norme PCRE) qui nous permettront d'être de plus en plus strict et nous discuterons par la suite de la pertinence de cette rigueur.

3.1 Détecter seulement le @

Nous voulons une suite de n'importe quels caractères qui ne soient pas des caractères d'espacement suivis d'un @, lui-même suivi à nouveau d'une suite de n'importe quels caractères (sauf espacements). Ceci peut s'écrire de plusieurs façons. Nous pouvons tout d'abord lister les caractères autorisés et indiquer avec le méta-caractère + qu'il nous faut au moins un caractère dans l'ensemble défini :

```
/^[A-Za-z0-9_-]+@[A-Za-z0-9_-]$/
```

Mais est-on vraiment sûr que seuls les caractères [A-Za-z0-9_-] soient autorisés ? Cette expression peut être simplifiée en utilisant \S (\w ne contient pas le caractère -) :

```
/^\S+@\S+$/
```

3.2 Ajout du nom de domaine

Un nom de domaine (top-level) est composé d'un point suivi de deux à six lettres (par exemple .fr, .com, etc). Nous pouvons ajouter cette contrainte :

```
/^\S+@\S+\.[A-Za-z]{2,6}$/
```

3.3 Restrictions sur les caractères

Il est temps maintenant de faire les choses de manière plus exacte. La RFC 3696 **[6]** nous indique qu'une adresse mail est de la forme local_part@domain_part où local_part est composée de lettres, chiffres ou de l'un des caractères suivant : !, #, $, %, &, ', *, +, -, /, =, ?, ^, _, `, ., {, |, }, ~. De plus le point ne peut pas apparaître en début ou en fin de chaîne et ne peut pas être suivi d'un autre point (ou plus). Là ça se complique un peu. Si nous traduisons textuellement ce que nous venons d'écrire sous forme d'expression régulière (uniquement pour local-part), nous obtenons :

```
/[A-Za-z0-9!#$%&'*+-/=?^`{|}~](\.{0,1}[A-Za-z0-9!#$%&'*+-/=?^`{|}~]+)*/
```

La première partie [A-Za-z0-9!#$%&'*+-/=?^`{|}~] correspond à l'ensemble des caractères autorisés exception faite du point puisque l'adresse ne peut pas commencer par un point. Dans la seconde partie nous indiquons qu'il peut y avoir éventuellement un point (\.{0,1}) suivi d'au moins un caractère autorisé ([A-Za-z0-9!#$%&'*+-/=?^`{|}~]+), le tout pouvant être répété entre 0 et n fois.

Nous pouvons simplifier cette écriture qui n'utilisait jusque là que la norme POSIX ERE :

```
/[\w!#$%&'*+-/=?^`{|}~](\.{0,1}[\w!#$%&'*+-/=?^`{|}~]+)*/
```

Pour domain_part, la RFC 3696 indique que le TLD (*top-level domain*) est composé de 2 à 6 lettres... c'est exactement ce que nous avons fait dans la section précédente. Par contre il faut ajouter le fait que l'on ne peut pas commencer domain_part par un point ni avoir plusieurs points consécutifs (les caractères autorisés sont les lettres, les chiffres, le caractère - et le point) :

```
/[A-Za-z0-9-]+(\.{0,1}[A-Za-z0-9-]+)*\.[A-Za-z]{2,6}/
```

L'expression régulière complète contenant local_part@domain_part est donc particulièrement longue et difficile à lire (et à maintenir ou corriger par la même occasion) :

```
/^[\w!#$%&'*+-/=?^`{|}~](\.{0,1}[\w!#$%&'*+-/=?^`{|}~]+)*@
[A-Za-z0-9-]+(\.{0,1}[A-Za-z0-9-]+)*\.[A-Za-z]{2,6}$/
```

L'utilisation de l'option x activant le mode verbeux s'avère très utile ici :

```
/^ # local_part
[\w!#$%&'*+-/=?^`{|}~] # Premier caractère (pas de point)
(\.{0,1}[\w!#$%&'*+-/=?^`{|}~]+)* # Caractères suivants (pas de séries de points)
@ # domain_part
[A-Za-z0-9-]+ # Premier(s) caractère(s) (pas de point)
(\.{0,1}[A-Za-z0-9-]+)* # Caractères suivants (pas de séries de points)
\.[A-Za-z]{2,6} # TLD sur 2 ou 6 lettres
$/x
```

Là où ça devient vraiment amusant, c'est que la RFC 3696 précise que local_part ne peut pas faire plus de 64 caractères et domain_part ne doit pas dépasser 255 caractères... Bon courage !

3.4 De l'utilité d'une telle vérification

Cette expression régulière n'est pas simple à construire, ce qui implique une forte probabilité qu'une erreur ait pu s'y glisser. De plus, même si une adresse fournie par un utilisateur est valide au sens de la RFC 3696, rien ne dit qu'elle existera réellement. #!/?@toto.toto est valide... mais si vous envoyez un mail à cette adresse je ne suis pas persuadé que l'on vous réponde.

Dans ce cas (et dans d'autres) il n'est pas forcément pertinent de vouloir à tout prix écrire une expression régulière parfaite, capable de valider toute adresse mail... car vous n'y arriverez pas ! Et il est préférable d'accepter de mauvaises adresses que de refuser de bonnes adresses. De toute façon vous vous apercevrez bien au moment de l'envoi que quelque chose cloche.

La solution médiane consistera alors à réaliser une vérification simple qui sera validée par la suite. Dans le cas des adresses mails on peut se contenter de l'expression régulière suivante :

```
/^[\w.!#$%&'*+-/=?^`{|}~]+@[A-Za-z0-9.-]+$/
```

Nous sommes ici dans le cas d'une vérification. Bien entendu, dans un cas d'extraction vous n'aurez pas d'autre choix que de cibler au plus près l'expression régulière décrivant les éléments que vous souhaitez récupérer de manière à limiter les faux positifs.

Conclusion

En fonction de la commande ou du langage que vous utilisez il faudra employer la norme POSIX BRE, POSIX ERE ou PCRE. Il est donc important de connaître l'existence et les différences existant entre ces trois normes pour être efficace. Quand vous aurez le choix, vous utiliserez vraisemblablement la norme PCRE qui est bien plus pratique.

Lorsque l'on écrit une expression régulière, c'est comme lorsque l'on écrit un programme : en fonction de la manière dont on va écrire l'expression régulière, la consommation en ressources sera plus ou moins grande. Même avec les expressions régulières il faut optimiser le « code » !

Il faut noter l'existence de la bibliothèque re2 **[7]** (pyre2 **[8]** pour Python) développée par Google qui accepte la norme POSIX BRE ou une syntaxe s'approchant de PCRE **[9]** mais qui est bien plus rapide (recherche en temps linéaire de la taille de la chaîne en entrée).

Un avertissement pour finir : les expressions régulières représentent un formidable outil qu'il faudra manipuler avec moult précautions sous peine de se retrouver écrasé par sa complexité... ■

Références

[1] T. Colombo, « Créez votre langage de programmation ! », GNU/Linux Magazine n° 175, octobre 2014, p. 24 à 36

[2] S. Kleene, « Representation of events in nerve nets and finite automata », dans Automata Studies, Princeton University Press, 1956, p. 3 à 41

[3] IEEE Std 1003.1, «Chapter 9 - Regular Expressions », 2004

[4] Bibliothèque PCRE : http://pcre.org/

[5] Bibliothèque SLRE : https://github.com/cesanta/slre

[6] RFC 3686, « Restrictions on email addresses » : http://tools.ietf.org/html/rfc3696#page-5

[7] Bibliothèque re2 : https://code.google.com/p/re2/

[8] Module pyre2 : https://github.com/facebook/pyre2

[9] Syntaxe re2 : https://code.google.com/p/re2/wiki/Syntax

MISE EN ŒUVRE DE CEPH

par **Olivier Delhomme** *[Développeur de logiciels libres pour mon loisir]*

Dans cet article nous allons voir comment Ceph, le système de stockage objet à haute disponibilité et hautes performances, fonctionne et s'il est possible de le soumettre, sans danger, à quelques petites expériences. Nous verrons également qu'une expérimentation à grande échelle est menée au CERN.

Lorsque l'on confie ses données à un système de fichiers ou un système de stockage, réparti ou non, il faut avoir une certaine confiance dans ce système (les sauvegardes aident un peu). Pour avoir un peu confiance, il faut voir comment le système fonctionne en temps normal mais aussi par gros temps (en mode dégradé par exemple). De même je trouve qu'une bonne documentation et des commandes ou outils qui permettent de triturer le système augmentent la confiance que l'on peut avoir dans un système de stockage. Voyons donc quel degré de confiance nous pouvons accorder à Ceph.

1 Architecture

1.1 Haute disponibilité

Ceph est un système de stockage objet à haute disponibilité et hautes performances dont l'installation a déjà été expliquée dans GNU/Linux Magazine n°179 **[1]**. Pour le moment seuls les serveurs **mon** et **osd** (voir encadré) peuvent être plusieurs dans un cluster. Il est même très vivement recommandé d'avoir au moins trois serveurs **mon** (plus ne peut pas nuire) et le maximum possible de serveurs physiques exécutant un serveur **osd** par périphérique de stockage (les plus nombreux possible).

Rappel

osd, **mon** et **mds** sont des abréviations pour, respectivement, Object Server Daemon (c'est lui qui va enregistrer les données), **mon**itor server (c'est lui qui va gérer le dispatch) et Meta Data Server (c'est lui qui gère les méta données des fichiers tels dates d'accès, de création, taille...). Pour plus d'informations reportez vous à mon article précédent **[1]**.

Le serveur **mds** qui est utile à Ceph n'est, à l'heure où j'écris ces lignes, pas encore en version stable pour le support de multiples instances (d'après le site Web officiel du projet). Il n'est donc pas redondant (si le serveur **mds** est cassé l'accès au système de fichiers **CephFS** est rendu impossible). Nous n'utiliserons donc pas ce système de fichiers en production (en tout cas, pas pour le moment).

Pour voir les configurations des différents serveurs on utilisera les commandes **ceph osd dump**, **ceph mon dump** et **ceph mds dump**. Ce qui donnera une sortie similaire à :

```
$ ceph mon dump
dumped monmap epoch 1
epoch 1
fsid a5bf6b38-02f4-43fb-bffc-9c932e98888e
last_changed 0.000000
created 0.000000
0: 192.168.122.77:6789/0 mon.ceph-mon
```

Note

En tapant la commande **ceph osd dump** on se rend compte que les serveurs **osd** sont numérotés à partir de 0. Dans une configuration de production on veillera à ce que le premier **osd** soit nommé 0 afin d'éviter toute confusion possible lors des interventions (ce qui m'est trop souvent arrivé).

Avec Ceph, la haute disponibilité n'est pas un vain mot. Par exemple, pour accéder à un objet Ceph ne recherche pas sa position dans un index, il la calcule. En effet, si un index était utilisé il serait un point d'accès systématique (pour chaque objet) et donc un goulot d'étranglement et un ensemble de données très critiques (en cas de perte de l'index plus aucun objet ne pourrait être localisé !). La méthode de calcul de la position d'un objet se nomme **CRUSH** : elle est déterministe et pondérée ce qui permet une définition hiérarchique des serveurs de stockage **osd**. Cela permet de définir comment les serveurs **osd** peuvent tomber en panne et donc induit le placement des données pour permettre cela. Par exemple on pourra indiquer **host** pour que la panne d'un serveur physique n'entraîne aucune perte de données même s'il fait fonctionner plusieurs serveurs logiciels **osd**. On peut aussi indiquer **datacenter** par exemple ! Les valeurs possibles sont :

```
type 0 osd
type 1 host
type 2 chassis
type 3 rack
type 4 row
type 5 pdu
type 6 pod
type 7 room
type 8 datacenter
type 9 region
type 10 root
```

Avec les commandes **ceph osd crush add-bucket** pour créer les emplacements (par exemple : des racks, des salles, des datacenters ou des régions...) et **ceph osd crush move** pour déplacer les éléments dans les bons emplacements. Nos racks étant dotés de 3 PDU (Power Distribution Unit – ensemble de prises électriques, type multiprise) indépendants électriquement on pourra définir des emplacements et ajouter les serveurs comme suit par exemple :

```
$ ceph osd crush add-bucket rack1-pdu1 rack
$ ceph osd crush add-bucket rack1-pdu2 rack
$ ceph osd crush add-bucket rack1-pdu3 rack
$ ceph osd crush add-bucket rack2-pdu1 rack
$ ceph osd crush add-bucket rack2-pdu2 rack
$ ceph osd crush add-bucket rack2-pdu3 rack
$ ceph osd crush move ceph-osd0 rack=rack1-pdu1
$ ceph osd crush move ceph-osd1 rack=rack1-pdu1
$ ceph osd crush move ceph-osd2 rack=rack1-pdu1
$ ceph osd crush move ceph-osd3 rack=rack1-pdu2
...
$ ceph osd crush move ceph-osd10 rack=rack2-pdu1
...
$ ceph osd crush move ceph-osd15 rack=rack2-pdu3
$ ceph osd crush move rack1-pdu1 root=default
$ ceph osd crush move rack1-pdu2 root=default
...
$ ceph osd crush move rack2-pdu2 root=default
```

Pour voir la totalité de la carte du cluster ceph il faut exécuter les deux commandes qui suivent. La première récupère la carte sous forme binaire, la seconde la transforme en fichier texte (**/tmp/crush.txt**) lisible par l'informaticien moyen :

```
$ ceph osd getcrushmap -o /tmp/crush.map
$ crushtool -d /tmp/crush.map -o /tmp/crush.txt
```

De manière plus rapide et assez lisible, l'arborescence du cluster est directement visible avec la commande **ceph osd tree** qui pourra produire un résultat similaire à celui-ci (tiré de l'exemple précédent) :

```
ceph osd tree
# id    weight  type name       up/down reweight
-1      8.12    root default
-18     2.26    rack rack1-pdu1
-2      0.45    host ceph-osd0
0       0.45            osd.0       up      1
-3      0.45    host ceph-osd1
1       0.45            osd.1       up      1
-4      0.45    host ceph-osd2
2       0.45            osd.2       up      1
...
-22     1.81    rack rack2-pdu2
-14     0.45    host ceph-osd12
12      0.45            osd.12      up      1
-15     0.45    host ceph-osd13
13      0.45            osd.13      up      1
-16     0.91    host ceph-osd14
14      0.91            osd.14      up      1
```

Par la suite on pourra créer une règle nommée **rack-pdu** via la commande **ceph osd crush rule create-simple rack-pdu default rack** :

```
$ ceph osd crush rule dump rack-pdu
{ "rule_id": 2,
  "rule_name": "rack-pdu",
  "ruleset": 2,
  "type": 1,
  "min_size": 1,
  "max_size": 10,
  "steps": [
        { "op": "take",
          "item": -1,
          "item_name": "default"},
        { "op": "chooseleaf_firstn",
          "num": 0,
          "type": "rack"},
        { "op": "emit"}
  ]
}
```

Finalement il est possible d'associer cette règle à un ou plusieurs pool (ici pour le pool **periph**) : **ceph osd pool set periph crush_ruleset 2.** Une fois associée, cette règle prendra soin de distribuer les données au niveau « rack » de sorte que la perte inopinée de l'ensemble des serveurs **osd** se trouvant dans un « rack » ne pose aucun problème d'un point de vue données. Dans notre cas il s'agit d'un tiers de rack physique (un PDU complet).

On prendra soin de bien définir les bonnes règles et de les associer rapidement aux différents pools avant qu'ils ne contiennent beaucoup de données car cela peut conduire à de très grands mouvements de données dans le cluster (cela peut rendre le cluster indisponible pour la production normale !).

1.2 Les commandes

Il y a de nombreuses commandes pour dialoguer avec un cluster Ceph mais en temps normal nous en utiliserons principalement trois. La commande **ceph** que nous avons déjà vue, la commande **rbd** qui permet, entre autres, la gestion de périphériques et la commande **rados** plutôt portée sur la gestion des « pools ». Notons enfin qu'une librairie, la **librados**, fourni l'API aux différents langages de programmation (C, C++, Python...) pour accéder au cluster (voir la figure 1 pour un schéma complet de l'architecture).

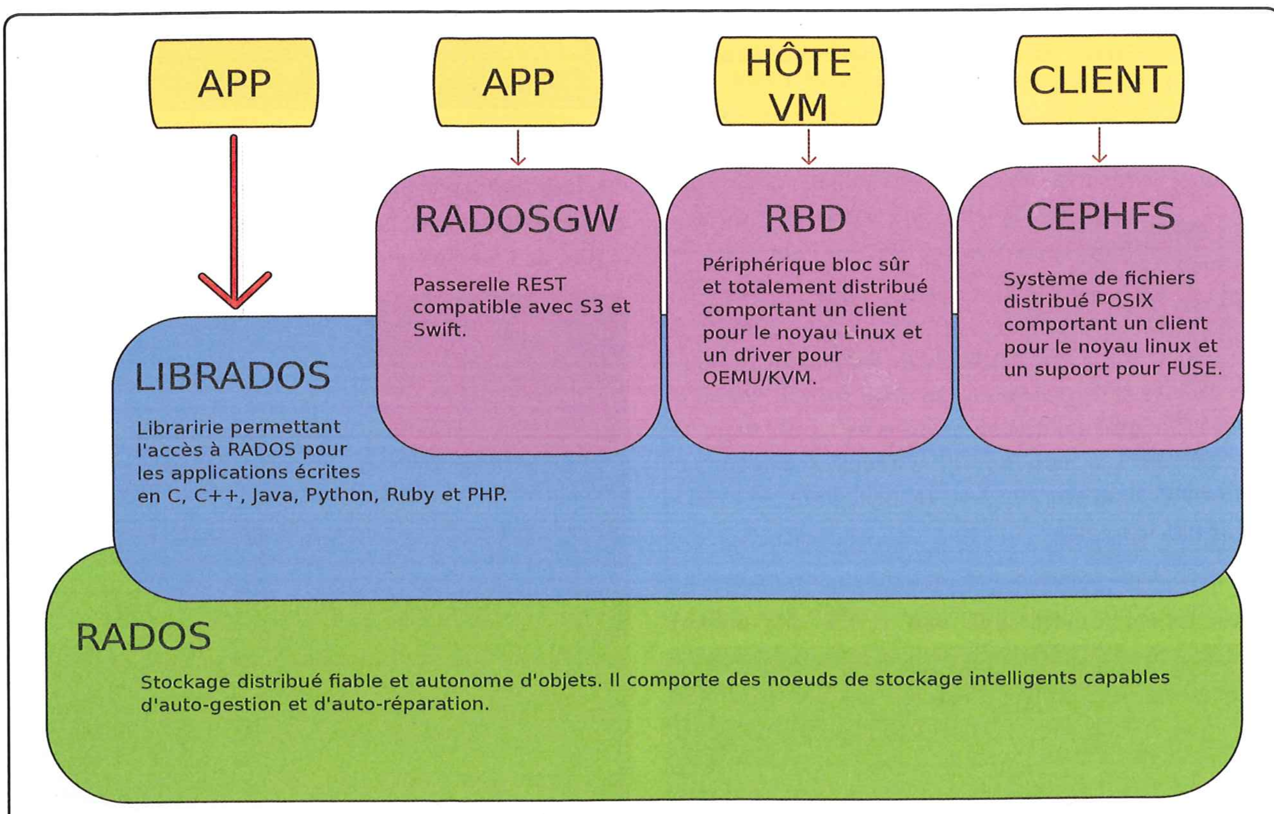


Fig. 1 : Pile de l'architecture (Inspirée d'une image d'Inktank Storage).

Certaines des options de ces commandes sont redondantes mais donnent les résultats sous une forme différente (par exemple **ceph osd lspools** et **rados lspools**).

La cerise sur le gâteau est que l'ensemble des commandes peuvent effectuer leur sortie selon plusieurs formats (option **--format=**). Le format par défaut est **plain** (celui qui est utilisé dans cet article), mais il est possible d'utiliser **json**, **xml**, **json-pretty** ou **xml-pretty**. Ces deux dernières sont moins économes en octets mais plus lisibles par le commun des mortels.

1.2.1 Ceph

Il y a trois façon d'utiliser la commande **ceph** :

- en ligne de commande comme nous l'avons vu jusqu'à maintenant ;
- en mode interactif : invoquer **ceph** puis entrer vos commandes directement sans répéter la chaîne « ceph » à chaque fois (par exemple **osd lspools**) ;
- en observation des changements de l'état du cluster (à la manière d'un **tail -f**) : **ceph -w**.

La commande **ceph --help** donne l'aide complète (et là on se dit « mais pourquoi ai-je pris la pilule bleue ? »). On peut obtenir l'aide d'une commande spécifique en utilisant **--help** après celle-ci (par exemple **ceph mon --help**).

1.2.2 Rados et rbd

Pour les commandes **rados** et **rbd** il faudra indiquer le pool sur lequel portera la commande avec l'option **-p** (typiquement : **rados -p metadata ls**) ou directement comme on le ferait avec un dossier (par exemple : **rbd create –size=1024 periph/harddrive**).

rados permet de gérer les *pools* et les objets qu'ils contiennent tandis que **rbd** permet de gérer les images des machines virtuelles (ou de CD, DVD...) et les périphériques blocs pour une machine hôte.

1.3 Partitionnement logique (les pools)

Les *pools* permettent le partitionnement logique de l'espace de stockage du cluster. Il est possible d'indiquer quelques paramètres qui seront spécifiques à chaque pool :

- le nombre de réplicats pour les objets et le nombre minimal de réplicats acceptables pour continuer de fonctionner en mode dégradé (par défaut c'est respectivement 2 et 1) ;
- les propriétaires et les droits d'accès ;
- le nombre de groupes de placement (par défaut c'est 8) ;
- la règle CRUSH à utiliser (par défaut la règle 0) ;

Il est fortement recommandé de changer ces valeurs par défaut qui ne sont pas faites pour une utilisation normale. La formule pour calculer le nombre de groupes de placement idéal est la puissance de 2 immédiatement supérieure à **(100 * nombre d'osd) / nb de réplicats** (ce qui donne 128 pour notre cas particulier avec 2 réplicats et 2 **osd**). Voyons maintenant quelques commandes sur les pools.

Note

La valeur par défaut pour la taille (le nombre de réplicats) a changé depuis la version firefly 0.80.x elle est maintenant de 3 car c'est la valeur constatée le plus couramment chez les utilisateurs de Ceph.

1.3.1 Lister les pools

Pour voir les pools déjà existants utilisez la commande **ceph osd lspools** ou **rados lspools** qui vous donnera un résultat similaire (au moins au début) :

```
$ ceph osd lspools
0 data,1 metadata

$ rados lspools
data
metadata
```

1.3.2 Créer un pool

Pour ajouter des *pools* à ceux qui sont créés par défaut il convient d'utiliser la sous-commande **mkpool** comme suit :

```
$ rados mkpool test1
successfully created pool test1
```

Si on veut spécifier le nombre de groupes de placement souhaités on peut utiliser une autre commande qui permet également la création d'un *pool* :

```
$ ceph osd pool create test2 100
pool 'test2' created
```

1.3.3 Détruire un pool

Pour détruire un *pool* il faut lui donner deux fois le nom à la commande et indiquer l'option **--yes-i-really-really-mean-it**. Ce sont des garde-fous pour éviter de détruire accidentellement un *pool* (il n'existe pas de commande ou d'option pour reconstruire ou récupérer un pool détruit) :

```
$ ceph osd pool delete test3 test3 --yes-i-really-really-mean-it
pool 'test3' deleted
```

1.3.4 Lister les objets d'un pool

Chacun des *pools* est destiné à stocker des objets. Pour voir quels objets sont contenus dans un *pool* il est possible d'utiliser la sous-commande **ls** (raccourci pour « list ») :

```
$ rados -p metadata ls
1.00000000.inode
100.00000000
100.00000000.inode
1.00000000
2.00000000
200.00000000
...
```

1.3.5 Instantanés d'un pool

Les instantanés d'un *pool* permettent de garder l'état de ce pool au moment précis où l'instantané a été pris, un peu comme une photographie. Pour créer un instantané (ou *snapshot* en anglais) nommé **instantane1** du *pool* **test1**, on utilisera la commande **mksnap** :

```
$ ceph osd pool mksnap test1 instantane1
created pool test1 snap instantane1
```

Nous pouvons maintenant lister les instantanés disponibles grâce à la sous-commande **lssnap** de **rados** (cette sous-commande ne semble pas disponible avec la commande **ceph osd pool**) :

```
$ rados -p test1 lssnap
1   instantane1 2014.01.20 22:02:02
1 snaps
```

La destruction d'un instantané est similaire à sa création (d'ailleurs il est possible d'utiliser la sous-commande **rmsnap** avec la commande **ceph osd pool**) :

```
$ rados -p test1 rmsnap instantane1
removed pool test1 snap instantane1
```

Note

La commande **rbd** propose également des sous-commandes pour créer, détruire et gérer des instantanés. Il s'agit ici d'instantanés réalisés sur les images des machines virtuelles (c'est à dire sur des objets d'un pool).

2 Dans les faits, ça se comporte comment ?

Tout d'abord, je dois dire que je n'ai pas porté une attention particulière à l'ordre dans lequel j'éteignais mes machines virtuelles lors de mes tests. Je n'ai jamais eu de problème lorsque je ré-allumais mes machines : le cluster était le plus souvent dans un état **active+clean**. Lorsque ce n'était pas le cas, le système a retrouvé un état **active+clean** assez rapidement.

Les tests de performance n'auraient aucun sens s'agissant de machines virtuelles stockées dans un PC de bureau sur lequel il n'y a qu'un seul disque dur. Cela d'autant plus que les volumes sont assez faibles et sont certainement très rapidement en cache (cette machine de bureau est plutôt bien fournie en mémoire). Je vous propose donc quelques essais sur les aspects comportement et fiabilité.

2.1 Ajouter des serveurs « monitor » au cluster Ceph

Pour faire de la haute disponibilité il faut au moins 3 serveurs **monitor** dans le cluster Ceph. En effet, il faut que puisse se dégager une majorité entre eux et cela est impossible avec 2 moniteurs seulement. Ici nous n'avons qu'un seul et il faut donc en ajouter deux. Il faut préparer deux autres machines virtuelles (comme nous l'avons déjà fait dans les paragraphes précédents). Mes machines seront :

```
192.168.122.240 ceph-mon1
192.168.122.66  ceph-mon2
```


2.1.1 Méthode automatique

Pour le moment, utiliser **ceph-deploy** est une mauvaise idée : le cluster devient injoignable car le quorum ne peut-être atteint avec deux moniteurs et il semble que **ceph-deploy** n'arrive pas à terminer son ajout !

Si comme moi vous avez fait cette erreur et que plus rien ne répond vous pouvez vous en sortir en réinjectant la bonne carte (le **monmap**) directement dans votre premier moniteur (ici sur **ceph-mon**) :

```
# /etc/init.d/ceph stop
# ceph-mon -i ceph-mon --extract-monmap /tmp/monmap
# monmaptool --generate --set-initial-members --rm ceph-mon1 \\
  --fsid a5bf6b38-02f4-43fb-bffc-9c932e98888e /tmp/monmap
# ceph-mon --inject-monmap /tmp/monmap
# /etc/init.d/ceph start
```

Dans mon cas, avant de récupérer l'accès à mon cluster ceph il faudra que je redémarre les serveurs **ceph-osd1**, **ceph-osd2** et **ceph-mds**.

2.1.2 Méthode manuelle

Il faut donc ajouter les deux moniteurs à la main. J'exécute sur **ceph-mon1** (faire de même avec **ceph-mon2**) les commandes suivantes :

```
# mkdir -p /var/lib/ceph/mon/ceph-ceph-mon1/tmp
# cd /var/lib/ceph/mon/ceph-ceph-mon1/tmp
# ceph auth get mon. -o mon.keyring
# ceph mon getmap -o monmap
# ceph-mon -i ceph-mon1 --mkfs --monmap monmap --keyring mon.keyring
# ceph mon add ceph-mon1 192.168.122.240:6789
```

Lorsque l'on tape cette dernière commande la sortie nous indique qu'il est parti chasser le moniteur... et plus rien ne fonctionne. Sur le premier moniteur **ceph-mon** j'ai récupéré la carte du cluster et je lui ai ajouté le troisième moniteur. J'ai ensuite indiqué aux deux nouveaux moniteurs leur adresse publique respective :

```
# /etc/init.d/ceph stop
# ceph-mon -i ceph-mon --extract /tmp/monmap
#  monmaptool --generate --set-initial-members --add ceph-mon2
192.168.122.66:6789 \\
   --fsid a5bf6b38-02f4-43fb-bffc-9c932e98888e /tmp/monmap
# /etc/init.d/ceph start
# ssh ceph-mon1 ceph-mon -i ceph-mon1 --public-addr
192.168.122.240:6789
# ssh ceph-mon2 ceph-mon -i ceph-mon2 --public-addr
192.168.122.66:6789
```

Et voilà... la commande **ceph mon dump** exécutée sur chacun des moniteurs achèvera de vous convaincre que tous les moniteurs se connaissent.

Maintenant, pour les clients nous pouvons indiquer les différents moniteurs dans le fichier de configuration **/etc/ceph/ceph.conf** (les moniteurs, eux, utilisent en interne la carte **monmap**) :

```
[global]
fsid = a5bf6b38-02f4-43fb-bffc-9c932e98888e
mon_initial_members = ceph-mon
mon host = ceph-mon,ceph-mon1,ceph-mon2
mon addr = 192.168.122.77:6789,192.168.122.240:6789,192.168.122.66:6789
auth_supported = cephx
osd_journal_size = 1024
filestore_xattr_use_omap = true
```

Note

N'oubliez pas de mettre à jour le fichier **ceph.conf** du dossier **/home/dup/demo-ceph** de la machine d'administration.

2.1.3 Finalement

Au vu de la difficulté que j'ai eue pour ajouter un moniteur je recommande très chaudement d'installer directement trois moniteurs à la création du cluster. Au passage il faut noter d'une part l'existence de nombreux outils pour éditer et triturer le cluster et d'autre part l'excellente documentation qui contient des paragraphes entiers pour la résolution de problèmes. Cela est rassurant lorsqu'on envisage un éventuel passage en production.

2.2 Arrêter brutalement l'un des deux osd

Pour les besoins du test, j'exécute le script suivant dans le répertoire exporté par le zfs (**/testpool**) : **watch --interval=1 "ls -ls; date >>date.txt"** puis je force l'extinction de la machine virtuelle **ceph-osd2**. Après environ 40 secondes le statut du cluster devient :

```
# ceph status
cluster a5bf6b38-02f4-43fb-bffc-9c932e98888e
 health HEALTH_WARN 252 pgs degraded; 247 pgs stuck unclean;
 recovery 329/658 objects degraded (50.000%); 1/2 in osds are down
 monmap e6: 3 mons at {ceph-mon=192.168.122.77:6789/0,
                      ceph-mon1=192.168.122.240:6789/0,
                      ceph-mon2=192.168.122.66:6789/0}, election
epoch 4,
                      quorum 0,1,2 ceph-mon,ceph-mon1,ceph-mon2
 mdsmap e156: 1/1/1 up {0=ceph-mds=up:active}
 osdmap e102: 2 osds: 1 up, 2 in
  pgmap v23634: 252 pgs, 6 pools, 1145 MB data, 329 objects
        2293 MB used, 27636 MB / 29929 MB avail
        329/658 objects degraded (50.000%)
             252 active+degraded
client io 21190 B/s wr, 14 op/s
```

Puis je redémarre la machine virtuelle. Lorsque celle ci est démarrée je regarde le contenu de mon fichier **date.txt** et il contient bien toutes les secondes. Le processus en cours n'a pas été affecté et le fichier est cohérent. Une phase de reconstruction a eue lieu (elle n'a pas affecté le processus non plus) et l'état du cluster est revenu à la normale :

```
# ceph status
cluster a5bf6b38-02f4-43fb-bffc-9c932e98888e
 health HEALTH_OK
 monmap e6: 3 mons at {ceph-mon=192.168.122.77:6789/0,
                      ceph-mon1=192.168.122.240:6789/0,
                      ceph-mon2=192.168.122.66:6789/0}, election
epoch 4,
                      quorum 0,1,2 ceph-mon,ceph-mon1,ceph-mon2
 mdsmap e156: 1/1/1 up {0=ceph-mds=up:active}
 osdmap e104: 2 osds: 2 up, 2 in
  pgmap v23646: 252 pgs, 6 pools, 1145 MB data, 329 objects
        2308 MB used, 27621 MB / 29929 MB avail
             252 active+clean
client io 3386 B/s wr, 2 op/s
```

S'agit-il d'un coup de chance ? Le fichier est si petit qu'il est, tout entier, dans la mémoire cache ? Pour éviter cela, je vais copier une image iso de la distribution 2013 de **texlive** (2,3 Gi) pendant la copie je coupe la machine virtuelle **ceph-osd2** : La copie se déroule sans accroche, même si les performances sont visiblement dégradées (en comparaison des performances lorsque les deux **osd** sont en fonctionnement). Redémarrons la machine **ceph-osd2**. La récupération se lance automatiquement et se déroule sans accroche jusqu'à son terme (alors même que la copie est toujours en cours). En revanche, les accès sont très lents et la copie indique parfois **-stalled-** ce qui n'est pas forcément bon signe. Toutefois, cette procédure s'effectue sur des machines virtuelles qui sont toutes stockées sur un même et unique disque dur dans une même machine physique et, vu que les auteurs de ceph indiquent que la procédure de récupération est gourmande en ressource, on pouvait s'attendre à ce résultat.

En tous les cas les sommes de contrôle **sha1** du fichier source et du fichier copié sont identiques !

2.3 Un osd tombe, vite montons un nouvel osd

Dans la vraie vie les serveurs qui s'arrêtent ont souvent des problèmes qu'il est parfois difficile de résoudre en trois minutes. Une fois le serveur **ceph-osd2** arrêté et que l'on est en mode dégradé, est-il facile de configurer et faire rentrer un nouveau serveur **osd** dans le cluster ?

Il faut préparer une nouvelle machine virtuelle et suivre la procédure du paragraphe « 2.6.5 Les serveurs de données (**osd**) » du précédent article **[1]**. Le cluster intègre le nouvel **osd** et commence la récupération de données. Lorsque celle-ci est terminée, ceph indique qu'il contient 3 **osd** et que 2 sont *up* et 2 sont *in*.

Ce n'est pas plus difficile d'ajouter un serveur **osd** quand le cluster est en mode dégradé qu'en temps normal !

2.4 Changement du disque dur dans l'osd en panne

Finalement, c'était le disque dur où sont stockées les données qui était cassé... On le remplace avec un nouveau disque dur tout neuf... (on simule cela en détachant, dans la machine virtuelle, l'image disque et en en ajoutant une nouvelle).

Redémarrons la machine. Rien ne se passe car il n'y a ni donnée, ni journal. Au point où nous en sommes il reste à réinstaller complètement le serveur. Avant cela prenons la précaution de supprimer l'ancien moniteur (**ceph-osd2** qui était nommé **osd.1** en interne) :

```
$ ceph osd crush remove osd.1
$ ceph auth del osd.1
$ ceph osd rm 1
```

On peut maintenant réinstaller le serveur **osd**. J'utilise l'option **--overwrite-conf** car le fichier **/etc/ceph/ceph.conf** est toujours là et sans cette option **ceph-deploy** refuse de l'écraser :

```
$ ceph-deploy disk zap ceph-osd2:sdb
$ ceph-deploy --overwrite-conf osd prepare ceph-osd2:sdb
$ ceph-deploy osd activate ceph-osd2:sdb
```

Nous voilà avec 3 serveurs **osd** et 3 serveurs **mon** (notez que j'ai arrêté le serveur **mds** pour libérer un peu de mémoire sur ma machine) :


```
$ ceph status
cluster a5bf6b38-02f4-43fb-bffc-9c932e98888e
 health HEALTH_WARN mds ceph-mds is laggy
 monmap e6: 3 mons at {ceph-mon=192.168.122.77:6789/0,
                      ceph-mon1=192.168.122.240:6789/0,
                      ceph-mon2=192.168.122.66:6789/0}, election
epoch 4,
                      quorum 0,1,2 ceph-mon,ceph-mon1,ceph-mon2
 mdsmap e8397: 1/1/1 up {0=ceph-mds=up:active(laggy or crashed)}
 osdmap e392: 3 osds: 3 up, 3 in
  pgmap v26450: 436 pgs, 6 pools, 8196 MB data, 2122 objects
        7319 MB used, 37575 MB / 44894 MB avail
             436 active+clean
```

Le serveur **ceph-osd2** est maintenant nommé **osd.3** en interne :

```
$ ceph osd tree
# id    weight  type name       up/down reweight
-1      0.02998 root default
-2      0.009995                host ceph-osd1
0       0.009995                        osd.0   up      1
-3      0.009995                host ceph-osd2
3       0.009995                        osd.3   up      1
-4      0.009995                host ceph-osd3
2       0.009995                        osd.2   up      1
```

Note

La perte du disque où sont stockés les journaux (qui est généralement un SSD pour des questions de performances) n'entraîne pas nécessairement la perte complète des **osd** correspondants. En effet, grâce à une procédure **[2]** de reconstruction des partitions du disque, (à partir de l'UUID des journaux de chacun des **osd**) il est possible de repartir comme si de rien était !

2.5 Arrêter deux osd

Normalement l'arrêt de deux serveurs **osd** sur trois devrait me rendre un certain nombre d'objets inutilisables étant donné que mes *pools* sont configurés avec une taille de 2 et une taille minimale de 1 :

```
$ ceph osd pool get images size
size: 2
$ ceph osd pool get images min_size
min_size: 1
```

L'arrêt brutal non prévu de deux des trois serveur **osd** laisse le cluster Ceph dans un état dégradé :

```
$ ceph health
HEALTH_WARN 291 pgs degraded;
145 pgs stale; 145 pgs stuck stale; 291 pgs stuck unclean;
2 requests are blocked > 32 sec;
recovery 3081/6686 objects degraded (46.081%); mds ceph-mds is laggy
```

On voit que 46% des objets ne sont plus accessibles. Comme mes objets sont probablement en cache dans mes machines virtuelles cette coupure ne semble pas avoir d'incidence sur les périphériques et notamment le système de fichiers ZFS : **/testpool** est toujours accessible ! Mais si on exécute la commande **dd if=texlive2013.iso of=morceau bs=1M count=200** qui produira un fichier **morceau** avec les 200 premiers méga octets du fichier **texlive2013.iso** rien ne se passe (même au bout d'une minute la commande est toujours en suspend et aucune erreur n'est apparue...).

Démarrons les machines virtuelles qui étaient arrêtées. Dès que les deux serveurs **osd** sont démarrés et accessibles, Ceph entame une procédure de récupération des données et arrive rapidement à un état stable. La commande qui était « stoppée » a démarré toute seule et s'est correctement terminée :

```
# ls -lsh /testpool
total 2,6G
190M -rw-r--r-- 1 root root 200M juin  17 11:50 morceau
2,4G -rw-r--r-- 1 root root 2,4G juin  12 14:39 texlive2013.iso
```

2.6 Arrêt subit d'un serveur moniteur

Maintenant que trois serveurs moniteur existent, que se passe t'il si l'un de ces serveurs s'arrête subitement ? L'existence du fait que l'un des serveurs moniteur est HS se note sur la deuxième ligne : **1 mons down** et au niveau du quorum qui n'est plus constitué que de deux serveurs moniteur (**ceph-mon1** et **ceph-mon2**) :

```
$ ceph status
cluster a5bf6b38-02f4-43fb-bffc-9c932e98888e
 health HEALTH_WARN mds ceph-mds is laggy; 1 mons down,
                  quorum 0,2 ceph-mon1,ceph-mon2
 monmap e6: 3 mons at {ceph-mon=192.168.122.77:6789/0,
                  ceph-mon1=192.168.122.240:6789/0,
                  ceph-mon2=192.168.122.66:6789/0},
                  election epoch 6,
                  quorum 0,2 ceph-mon1,ceph-mon2
 mdsmap e16666: 1/1/1 up {0=ceph-mds=up:active(laggy or crashed)}
 osdmap e415: 3 osds: 3 up, 3 in
  pgmap v27896: 436 pgs, 6 pools, 4599 MB data, 1198 objects
        7727 MB used, 37167 MB / 44894 MB avail
             436 active+clean
client io 11480 B/s rd, 34736 B/s wr, 21 op/s
```

Sinon, toutes les opérations sont faisables sans aucun problème particulier :

```
# rbd -p periph --size 1024 create dd4
# rbd map periph/dd4
# zpool add testpool spare /dev/rbd/periph/dd4
# zpool status -v
  pool: testpool
 state: ONLINE
 scrub: none requested
config:
        NAME              STATE     READ WRITE CKSUM
        testpool          ONLINE       0     0     0
          rbd/periph/dd1  ONLINE       0     0     0
          rbd/periph/dd2  ONLINE       0     0     0
          rbd/periph/dd3  ONLINE       0     0     0
        spares
          rbd/periph/dd4  AVAIL

    errors: No known data errors
```

La copie d'un fichier ou l'installation d'une machine virtuelle se terminent sans aucun souci. Le retour du serveur moniteur est immédiat et, là encore, cela ne pose aucun problème.

2.7 Maintenance programmée

Lorsque l'on doit intervenir sur un serveur **osd** pour réaliser une maintenance, comme par exemple l'ajout de barrette mémoire, il est possible de prévenir le cluster afin d'éviter que les données ne soient automatiquement redistribuées :

```
$ ceph osd set noout
set noout
```

Dans le même ordre d'idée on peut demander à mettre en pause le **scrub** et le **deep-scrub** :

```
$ceph osd set noscrub
set noscrub
$ ceph osd set nodeep-scrub
set nodeep-scrub
```

On peut éteindre l'**osd** pour effectuer la maintenance programmée. À l'issue de la maintenance, lors de la mise sous tension du serveur, les scripts d'initialisation redémarrent automatiquement le service **ceph** :

```
$ ssh ceph-osd2 sudo /etc/init.d/ceph stop
=== osd.3 ===
Stopping Ceph osd.3 on ceph-osd2...kill 3148...kill 3148...done

$  ceph status
cluster a5bf6b38-02f4-43fb-bffc-9c932e98888e
 health HEALTH_WARN 276 pgs degraded; 276 pgs stuck unclean;
 recovery 3741/8218 objects degraded (45.522%); mds ceph-mds is
laggy;
 1/3 in osds are down; noout flag(s) set
 monmap e6: 3 mons at {ceph-mon=192.168.122.77:6789/0,
                       ceph-mon1=192.168.122.240:6789/0,
                       ceph-mon2=192.168.122.66:6789/0},
                       election epoch 4,
                       quorum 0,1,2 ceph-mon,ceph-mon1,ceph-mon2
mdsmap e16511: 1/1/1 up {0=ceph-mds=up:active(laggy or crashed)}
osdmap e396: 3 osds: 2 up, 3 in
       flags noout
 pgmap v27615: 436 pgs, 6 pools, 15962 MB data, 4109 objects
       7310 MB used, 37584 MB / 44894 MB avail
       3741/8218 objects degraded (45.522%)
            160 active+clean
            276 active+degraded
```

Une fois la maintenance terminée, le serveur redémarré et le service **ceph** en fonctionnement il ne faut pas oublier d'enlever les drapeau **noout**, **noscrub** et **nodeep-scrub** :

```
$ ceph osd unset noout
unset noout
$ ceph osd unset noscrub
unset noscrub
$ ceph osd unset nodeep-scrub
unset nodeep-scrub
```

Et voilà, c'est tout ! Évidemment, il faut veiller à ce que ce mode de maintenance soit le plus court possible et si possible ne pas éteindre le mauvais serveur **osd** !

2.8 Mise à jour du cluster (passage à firefly)

D'après la note de version il convient de migrer en premier les serveurs moniteurs puis les serveurs **osd** et enfin seulement le serveur **mds**. Sur chacun des serveurs moniteurs on change la version de ceph de « *emperor* » à « *firefly* » :

```
$ ssh ceph-mon sudo sed -i s/emperor/firefly/ /etc/apt/sources.
list.d/ceph.list
```

Sur chacune des machines on réalise alors la mise à jour :

```
$ ssh ceph-mon sudo aptitude update
$ ssh ceph-mon sudo aptitude safe-upgrade
```

On réalise la même chose sur chacun des serveurs **osd**, sur le serveur **mds** puis sur les clients (ici **ceph-admin** et **debian-virt2**). Ensuite la documentation indique que si l'on utilise Ceph avec une version plus ancienne du noyaux Linux que 3.9 il convient de réaliser la commande suivante (debian wheezy est en version 3.2) :

```
$ ceph osd crush tunables legacy
```

Par acquit de conscience sur la machine **debian-virt2** j'ai arrêté le service **zfs-fuse**, redémarré **rbdmap** et relancé **zfs-fuse**.

Note

La mise à jour de la version « firefly » vers la version « giant » s'effectue exactement de la même façon et selon le même protocole.

3 Une expérimentation à grande échelle au CERN

Le CERN, l'organisation européenne pour la recherche nucléaire, possède les équipements scientifiques les plus imposants de la planète. De par leur gigantisme (aussi gros que des cathédrales) mais aussi de par les masses de données qu'ils produisent. En effet, quelques chiffres **[3]** en disent long : le LHC produit 1 péta octet de données par seconde d'expérience (40 millions d'images à 100 Méga pixels) qui une fois filtré génère 30 pétaoctets de données par an. Les données sont traitées au rythme d'un pétaoctet par jour.... Songez que les détecteurs du LHC ne sont pas les seules expériences du CERN !

Les systèmes de fichiers CASTOR et EOS recueillent les données des expériences et ils supportent un flot de données d'environ 10 gigaoctets par secondes. CASTOR fait environ 89 pétaoctets et EOS environ 20 pétaoctets. Un « petit » système de fichiers AFS stocke l'ensemble des répertoires personnels des 30 000 personnes liées au CERN. Il fait environ 300 téraoctets et contient plus de 2 milliards de fichiers.

Un « petit » cluster de test **[4]** a permis de réaliser la validation de Ceph comme solution de stockage pour le CERN. Il s'agit de 47 serveurs physiques de stockage comportant chacun 2 processeurs Xeon E2650 (8 cœurs à 2 Ghz soit 16 cœurs et 32 threads par serveurs), 64 Go de mémoire vive, une interface Ethernet à 10 Gbits connectée (sur deux utilisables), 24 disques de 3 téraoctets pour les données et 3 disques de 2 téraoctets en miroirs pour le système. Chacun de ces serveurs physiques fait fonctionner 24 serveurs logiciels **osd** (un par disque). Le cluster comporte également 5 serveurs physiques moniteur dont la particularité est d'utiliser des disques dur SSD pour les serveurs **mon**. Un petit calcul rapide (24 disques de 3 téraoctets sur 47 serveurs) donne un volume utilisable de 3 384 téraoctets soit 3,3 pétaoctets !!

En effet, Le CERN est en train d'opérer un changement majeur dans son architecture informatique en passant à un système d'information virtualisé et redondant. Pour la redondance, un nouveau *datacenter* a été construit à Budapest. La plateforme choisie pour la virtualisation est OpenStack (elle est déjà en production). Ceph (le cluster de test est maintenant passé en production) réalise une partie du stockage pour cette plate-forme et comporte déjà plus de 500 volumes OpenStack Cinder, 15 millions d'objets, et environ 1000 images OpenStack Glance. Le tout occupe un espace de 60 téraoctets (180 sur les disques puisque la configuration du cluster stocke 3 fois les données) et supporte en temps normal entre 3000 et 5000 opérations par seconde.

Ce cluster est déployé via **Puppet** et les scripts d'**Enovance [5]** légèrement modifiés **[6]**. L'ajout d'un serveur est automatique (**mon** ou **osd**). Les disques des serveurs **osd** sont détectés, préparés, initialisés et insérés dans le cluster. Le fichier **ceph.conf** est autogénéré, seule l'opération de démarrage du cluster (**service ceph start**) reste manuelle afin de laisser la possibilité de tout vérifier avant.

Conclusion

Ceph réalise l'unification, la convergence et la virtualisation du stockage (et aussi le stockage de la virtualisation ;-)). C'est un changement de paradigme. Cet article et le précédent **[1]** ne sont qu'une toute petite introduction à Ceph et ce que nous avons vu est déjà impressionnant. Les quelques tests basiques qui ont été réalisés montrent sa robustesse et sa fiabilité. Le degré de confiance dans ce système de stockage me semble tout à fait compatible avec un usage en production (ce qu'est en train de faire le CERN). Maintenant c'est à vous d'aller plus loin dans la découverte du système et peut-être d'écrire un programme utilisant la **librados** ou l'API REST pour accéder directement au cluster Ceph fraîchement mis en œuvre via les interfaces S3 ou Swift ? ■

Je remercie ma femme et mes enfants pour leur patience infinie, Fanny pour sa relecture de la version alpha de l'article, Dan Van Der Ster et Mélissa Gaillard pour leur autorisation de publier un paragraphe sur le CERN (et leur relecture bienveillante de ce paragraphe).

Références et liens intéressants

[1] O. Delhomme, « Présentation et installation du système de stockage réparti Ceph », GNU/Linux Magazine n°179, février 2015, p. 32 à 38

[2] Récupération des données après la perte du journal : http://www.sebastien-han.fr/blog/2014/11/27/ceph-recover-osds-after-ssd-journal-failure/

[3] l'informatique au CERN : http://home.web.cern.ch/about/computing

[4] Une présentation de Ceph au CERN : http://fr.slideshare.net/Inktank_Ceph/scaling-ceph-at-cern

[5] Les scripts d'Enovance : https://github.com/enovance/puppet-ceph/

[6] Les modifications du CERN : https://github.com/cernceph/puppet-ceph/

SUPERVISION DISTRIBUÉE AVEC MONIT (ET PUPPET)

par **Romain Pelisse** [Cloud Architect @ Red Hat Gmbh]
Relecture par Adrien Saurat et Aurélie Garnier

Depuis le début de l'ère internet, le nombre de systèmes et d'applicatifs déployés par les entreprises ne cesse de croître de manière exponentielle, et l'arrivée du cloud et du Big Data n'a fait qu'accélérer la tendance. En outre, là où il y a vingt ans un administrateur système gérait une douzaine de machines, le même administrateur en gère aujourd'hui parfois plusieurs centaines. Or, il devient de plus en plus compliqué de proposer une supervision adaptée qui puisse supporter une telle mise à l'échelle.

Évitons les longs discours et prenons un exemple concret. Un serveur applicatif héberge un serveur Apache HTTPd en frontal d'une paire d'instances du serveur applicatif JEE, Wildfly. Le serveur Apache reçoit les requêtes HTTP à leur intention et effectue une simple balance de charge, entre les deux instances, à l'aide d'un module dédié, `mod_cluster`. Maintenant, voyons comment superviser ce système.

1 Introduction

1.1 Petit exemple concret

La première chose à superviser est bien évidemment la disponibilité du système, donc on va souhaiter naturellement effectuer une commande `ping` à intervalles réguliers, pour vérifier si le système est toujours accessible. Il nous faut ensuite superviser la disponibilité de Apache - à minima, avec une requête régulière vers une page de statut. En outre, il serait appréciable de suivre le processus associé à Apache, et, s'il vient à « disparaître », de le redémarrer immédiatement.

Sur la partie Wildfly, on doit non seulement surveiller la disponibilité du service, mais aussi quelques indicateurs clés. Partons du principe que les deux instances hébergent des applicatifs Web ; il faut donc vérifier leurs disponibilités, à travers Apache HTTPd (sur le port 80) mais aussi sur les ports spécifiques de chaque instance (8080 et 8180), pour pouvoir déterminer si, quand un problème survient, ce sont les instances Wildfly ou Apache HTTPd en lui-même qui en sont à l'origine.

On ne peut néanmoins pas s'arrêter à ces seules métriques. En effet, les instances Wildfly s'exécutent chacunes sur une machine virtuelle Java et il est donc essentiel de bien surveiller certains indicateurs clés spécifiques à cette dernière. Les premiers qui viennent à l'esprit sont bien évidemment la consommation mémoire et la durée d'exécution du « ramasse-miettes » (*garbage collection*).

Mais il est aussi important de surveiller le nombre de sous-processus (*threads*) et le taux d'utilisation des connexions aux bases de données (*connection pool*). Enfin, si les développeurs des applicatifs ont bien fait leur travail, on devrait trouver des URL dédiées à la supervision et peut-être même des données et opérations exposées par JMX (Java Management Extensions) **[1]**.

En outre, à toutes ces métriques finalement très « fonctionnelles », il faut ajouter de nombreuses métriques techniques (utilisation du CPU, surveillance des systèmes de fichiers, etc).

1.2 Trop de métriques tue les métriques

Comme on vient de l'illustrer, il y a beaucoup d'informations à surveiller sur un tel système, et si l'on multiplie ces derniers par des centaines de systèmes - et il existe plusieurs grands comptes chez qui on peut trouver ce genre de déploiement - la seule masse des données (et leur fréquence d'émission) peut suffire à faire s'écrouler de nombreuses solutions de supervision.

On peut bien évidemment y remédier en déployant plusieurs instances, ou en mettant à profit la capacité de la solution à monter en charge d'une manière ou d'une autre. On peut aussi déployer une instance de supervision par environnement et agréger les résultats sur une instance « maître ». Toutes ces solutions sont envisageables mais ne font qu'augmenter la complexité du système ainsi que le nombre de machines à utiliser.

Pour s'en convaincre, faisons un rapide calcul. S'il faut un serveur de supervision pour surveiller vingt systèmes, il faudra probablement dix autres serveurs de supervision pour surveiller deux cents systèmes. Bref, la supervision a rapidement un coût considérable en terme de machines.

En outre, presque toutes les solutions de supervision ont besoin, pour être complètes, de déployer un agent sur le système cible. Ces agents peuvent être relativement consommateurs en ressource, et nécessitent parfois, comme c'est le cas pour Tivoli ou RHQ **[2]**, leur propre machine virtuelle Java. Sur de gros systèmes hébergeant de nombreux applicatifs Java, la mémoire dédiée à ces agents peut atteindre jusqu'à plusieurs Gb. Toujours autant de ressources de consommées, juste à des fins de supervision.

1.3 Comment s'en sortir ?

Depuis l'émergence du *Cloud*, mais aussi de solutions distribuées comme Git, il apparaît de plus en plus évident que la meilleure approche pour tenir une charge est d'avoir un système distribué, dont la capacité croît au fur et à mesure que le nombre d'instances augmente.

Regardons justement le cas de Git. Avant l'apparition de Git, les serveurs SVN des entreprises devenaient rapidement des goulots d'étranglement dans le processus de développement. Les fusions de code étaient compliquées, le déclenchement de processus pré ou post *commit* ralentissaient le système et ne pouvaient donc pas être utilisées de manière systématique. En outre, retrouver le *commit* coupable d'une régression ne pouvait pas être fait en exécutant simplement un test unitaire sur un sous-ensemble d'entrées dans l'historique de la base de code. Bref, aussi bien en terme de montée en charge qu'en terme de fonctionnalités, le modèle centralisé de SVN avait atteint ses limites. Et le succès de Git, et des DVCS en général, en a découlé naturellement.

En effet, Git a déporté la plupart des traitements à effectuer du serveur central au poste du développeur, ne laissant à l'éventuelle machine dédiée à héberger le « point de référence » commun, que la simple tâche d'accepter une série de changements (ou non). Tout le travail difficile est géré sur le poste du développeur. Plus ils sont nombreux, plus chacune de leurs machines travaille, sans surcharger le référentiel. Il reste bien évidemment une limite physique - un certain nombre de connexions concurrentes peuvent toujours rendre le serveur « référentiel » indisponible, mais la limite est largement plus haute qu'avant, et au pire des cas, le développeur peut continuer son travail, et simplement repousser l'envoi des changements à un moment où le serveur sera de nouveau disponible.

2 Supervision distribuée

2.1 Architecture de haut niveau

Fort de ce constat, voyons comment nous pourrions appliquer la même stratégie à la supervision. Au lieu de centraliser la remontée des métriques, nombreuses et coûteuses en espace, vers un serveur central, nous allons tout d'abord réduire le rôle de ce dernier au minimum en déléguant le travail au système cible en lui-même.

Dans cette vision, le système central se limite à vérifier que le serveur est toujours *up* (en gros qu'il répond à la commande `ping`, plus quelques tests SNMP pour être sûr que le système d'exploitation n'a pas « planté »). Le reste est délégué à un processus local.

Le processus local, finalement voisin de l'agent évoqué plus haut, se charge de plusieurs tâches :

1. Surveiller la disponibilité des processus applicatifs et remonter une alerte si ces derniers ne répondent plus ;
2. Relever les différentes métriques systèmes (CPU, espace disque...) et applicatifs (mémoire de la JVM, nombre de connexions à la base de données utilisées...) ;

3. Redémarrer les processus indisponibles ou autre action similaire en cas de détection d'une anomalie.

2.2 Limitations

On peut tout de suite relever quelques limites évidentes à cette approche. Tout d'abord, l'absence de centralisation des données. Si l'on souhaite connaître la consommation moyenne en CPU d'un serveur, il faut s'y connecter pour regarder les données collectées. C'est un faible prix à payer, quand on réfléchit à la quantité de données à envoyer pour bénéficier du confort d'une approche « centralisée ».

Dans le même ordre d'idée, il n'y a aucune vision d'ensemble de type *reporting*. Par exemple, on ne peut pas facilement, en accédant au serveur central, répondre à la question « Quelle est la consommation moyenne de mémoire des serveurs Java ». Mais là encore, est-il pertinent d'épuiser l'ensemble des systèmes en mettant en place une lourde infrastructure de supervision pour répondre à une question finalement épisodique ? Ne serait-il pas plus judicieux, lorsque la question se pose, d'effectuer un relevé et de calculer cette moyenne ?

Un autre désavantage dans la décentralisation de cette approche réside dans le fait que les réactions à certains événements ne peuvent être que locales. Par exemple, on peut imaginer à l'aide d'outils tels que RHQ, si la charge de plusieurs systèmes augmente, pouvoir démarrer des instances supplémentaires. Dans notre cas, ce n'est pas possible.

Nous verrons un peu plus loin comment compenser ces limites. Pour le moment, place à la pratique, et essayons de construire, à l'aide d'une solution *Open Source* bien sûr, une telle supervision distribuée.

Note

On notera néanmoins que Monit a une limite claire par rapport à des agents Java, comme celui de RHQ évoqué plus haut. Monit n'est pas aussi portable que ce dernier et il ne fonctionnera donc que sur des systèmes Linux. Cette opinion n'engage que l'auteur de cet article, mais ce n'est pas une limite très importante dans le cas de la gestion de serveurs. En outre, il semble exister des alternatives, telles que Munin **[4]**.

Un autre avantage d'utiliser un agent dédié à la plateforme, plutôt qu'un applicatif portable s'exécutant sur la JVM est bien évidement la moindre consommation de ressource système.

3 Supervision locale avec Monit

Pour démontrer la validité de cette approche, nous allons mettre en place un outil *Open Source*, du nom de Monit **[3]**, qui permet justement de surveiller des processus locaux, mais aussi d'effectuer des relevés de métriques systèmes et de redémarrer des services si nécessaire.

3.1 Puppet

En plus d'expliquer la configuration de Monit, nous allons, au fur et à mesure de l'article, réaliser la configuration Puppet **[5]** associée. Le propos de cet article n'étant pas de décrire Puppet, on laissera le soin au lecteur de se reporter à la documentation existante sur cet outil, si nécessaire, ou sinon de simplement ignorer ces ajouts qui ne sont pas nécessaires à la compréhension du sujet de cet article.

3.2 Installation de Monit

3.2.1 Installation Manuelle

C'est une étape assez triviale, le paquet est très souvent disponible dans la plupart des distributions. Pour un système RHEL, il suffit d'ajouter le dépôt logiciel EPEL **[6]** adapté à sa version :

```
# yum install -y http://ftp.uni-bayreuth.de/linux/fedora-epel/6/
i386/epel-release-6-8.noarch.rpm
```

On notera néanmoins que, pour ne pas compromettre son système, et s'assurer qu'on n'installe pas, par erreur, des paquets logiciels issus de ces dépôts plutôt que ceux fournis par Red Hat, il ne faudra pas oublier de désactiver les dépôts EPEL après l'installation de Monit :

```
# sed -i /etc/yum.repos.d/epel.repo -e 's/enabled=.*$/enabled=0/'
```

C'est déjà prudent mais en fait, on peut même faire mieux et simplement ajouter, à la définition du dépôt logiciel, une instruction `includepkgs` qui limitera le périmètre des paquets à installer et à mettre à jour depuis ce dépôt :

```
/etc/yum.repos.d/epel.repo
[epel] name=Extra Packages for Enterprise Linux 6 - $basearch
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-6&arch=$amp;basearch
failovermethod=priority
enabled=1
gpgcheck=1
includepkgs=puppet*,monit*
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6
```

Une fois le dépôt logiciel en place, il suffit donc d'installer le paquet logiciel associé à Monit :

```
# yum install -y monit
```

3.2.2 Mise en place avec Puppet

Le premier élément à mettre en place est bien évidemment la définition du dépôt logiciels EPEL :

```
//modules/epel/manifests/init.pp:
class epel::repo {
  $gpgkey_epel='/etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6'
  yumrepo { 'epel':
    name => 'Extra Packages for Enterprise Linux 6 - $basearch',
    enabled => 1,
    failovermethod => 'priority',
    mirrorlist => 'https://mirrors.fedoraproject.org/metalink?repo=epel-6&arch=$basearch',
    gpgcheck=> 1,
    includepkgs => 'puppet*,monit*',
    gpgkey => "file://$gpgkey_epel",
    require => File[$gpgkey_epel]
  }

  file { $gpgkey_epel:
    ensure => present,
    source => puppet://modules/epel/RPM-GPG-KEY-EPEL-6,
   }
}
```

Une fois la définition du dépôt ajoutée, il suffit de demander à Puppet de garantir la présence du paquet `monit` :

```
// manifests/site.pp:
node 'myhost' {
  include epel::repo

  package { 'monit':
    ensure => installed,
    require => Yumrepo['epel'],
  }
}
```

3.3 Mise en place du service

3.3.1 Mise en place manuelle

`monit` est un service comme un autre et il faut donc le démarrer de manière tout à fait traditionnelle:

```
# service monit status monit is stopped
# service monit start Starting monit: [ OK ]
# service monit status monit (pid 1586) is running...
```

Il est bien sûr important que ce service soit lancé dès le démarrage de la machine, donc n'oublions surtout pas d'exécuter la commande suivante :

```
# chkconfig monit --level=345 on
```

3.3.2 Mise en place avec Puppet

La mise en place du service avec Puppet est assez triviale, mais nous allons tout de même, pour garder la configuration la plus propre possible, déplacer toutes les ressources spécifiques à Monit dans un module dédié - ce qui nous servira par la suite de toute manière :

```
// modules/monit/manifests/init.pp
class monit::monit($yumrepo='epel') {
  package { 'monit':
     ensure => installed,
     require => Yumrepo[$yumrepo],
  }

  service { 'monit':
     ensure => running,
   enable => true,
   require => Package['monit'],
  }
}
```

Note

Nous avons ajouté l'option de redéfinition du dépôt Yum dont dépend Monit pour faciliter le déploiement de Monit sur d'autres systèmes n'utilisant pas le dépôt EPEL.

Regardons maintenant à quoi ressemble notre fichier manifeste :

```
// manifests/site.pp:
node 'myhost' {
  include epel::repo
  include monit::monit
}
```

3.4 Configuration du service

3.4.1 Configuration par défaut

Sans surprise, le service Monit se configure simplement à l'aide de fichiers déployés dans le répertoire **/etc/monit.d/** - comme tout bon service « à la Unix » qui se respecte. Lors de l'installation du service, le fichier présent dans ce répertoire est le fichier **logging** qui définit simplement dans quel fichier Monit journalise ses actions :

```
$ cat /etc/monit.d/logging
# log to monit.log
set logfile /var/log/monit
```

Et voici le contenu habituel de ce fichier :

```
# cat /var/log/monit
[CEST Aug 17 12:59:06] info : monit: generated unique Monit id
8a8cc0a7d85fb3e85f9d91deaec81970 and stored to '/root/.monit.id'
[CEST Aug 17 12:59:06] info : 'myhost' Monit started
```

3.4.2 Ajout du serveur de mail

Cette configuration est minimale et il lui manque un élément crucial : l'adresse du serveur STMP à partir duquel Monit pourra, le cas échéant, envoyer des messages d'alerte. Commençons donc par ajouter un fichier **mailserver** à notre configuration :

```
# cat /etc/monit.d/mailserver
set mailserver smtp.gmail.com port 587 username "monit@gmail.
com" password "isupervisesoftware" using tlsv1
```

Note : par souci de simplicité, nous avons utilisé comme STMP le service de Google GMail.

3.4.3 Définition du destinataire des alertes

Maintenant que Monit peut envoyer des mails d'alerte, il reste encore à définir à qui - ainsi que le format du mail. Ceci se fait très aisément avec l'ajout d'un autre fichier dans notre répertoire de configuration :

```
$ cat /etc/monit.d/alert
set mail-format {
   from: $HOST@gmail.com
   subject: Monit a effectué un redémarrage du service $SERVICE
exécuté sur $HOST
   message: Vous êtes notifié par ce message d'une action
entreprise automatiquement par Monit sur le service $SERVICE sur
$HOST: $DESCRIPTION

}

set alert belaran@gmail.com
```

À ce stade, il est prudent de redémarrer le service et de vérifier, dans son fichier journal, qu'il n'y a pas d'erreur dans la configuration et que Monit a pu démarrer sans incident :

```
# service monit restart
Shutting down monit: [ OK ]
Starting monit: [ OK ]
# tail -f /var/log/monit
[CEST Sep 23 16:17:05] info : 'myhost' Monit started
```

Il nous reste un dernier élément à mettre en place dans la configuration générale de Monit. Monit expose une partie de son API via un *Web Service* et vient donc avec un (petit) serveur Web. Ce dernier permet aussi d'accéder à des données de supervision sur l'hôte, comme nous le verrons plus loin.

Bref, cette fonctionnalité est fort pratique, et nous allons donc l'activer, en ajoutant un autre fichier au répertoire **/etc/monit.d/** :

```
// /etc/monit.d/monit.web
set httpd port 2812 and use address localhost
allow localhost
```

On notera que l'on autorise uniquement les accès locaux (**allow localhost**) pour des raisons de sécurité. Néanmoins, si votre réseau est déjà sécurisé, vous pouvez souhaiter autoriser certaines adresses IP ou sous réseau, pour vous connecter à distance à Monit.

3.4.4 Configuration avec Puppet

Nous allons étendre notre module Puppet pour inclure une paire de fichiers patrons (*templates*) qui permettront de générer une configuration appropriée selon les environnements :


```
// modules/monit/templates/mailserver.erb
set mailserver <%= @smtp_host %> port <%= @smtp_port %> username
"<%= @smtp_user %>" password "<%= @smtp_pass %>"
<% if (defined(@smtp_use_tlsv1)) %>
using tlsv1
<% end if %>

// modules/monit/templates/alert.erb
set mail-format {
  from: $HOST@gmail.com
  subject: Monit a effectué un redémarrage du service $SERVICE
exécuté sur $HOST
  message: Vous êtes notifié par ce message d'une action entreprise
automatiquement par Monit sur le service $SERVICE sur $HOST:
$DESCRIPTION
}

set alert <%= @alert_recipient %>

// modules/monit/templates/monit.web.erb
set httpd port <%= @httpd_port %> and use address <%= @monit_bind_address %>
allow <%= @allowed_host %>
```

On ajoute ensuite une classe, que l'on peut configurer si besoin est, mais définie avec les paramètres par défaut appropriés à notre module Monit :

```
// modules/monit/manifests/init.pp:
...
class monit::alert($mail_receipient='belaran@gmail.com', $smtp_
host='smtp.gmail.com', $smtp_port='587', $smtp_user='monit@gmail.
com', $smtp_pass='isupervisesoftware', $smtp_use_tlsv1='true',
$httpd_port='2812', $monit_bind_address='localhost', $allowed_
host='localhost') {
  file { '/etc/monit.d/alert':
    content => template("monit/alert.erb'),
    require => Package['monit'],
    notify => Service['monit'],
  }

  file { '/etc/monit.d/mailserver':
    content => template('monit/mailserver.erb'),
    require => Package['monit'],
    notify => Service['monit'],
  }

  file { '/etc/monit.d/monit.web':
    content => template('monit/monit.web.erb'),
    require => Package['monit'],
    notify => Service['monit'],
  }
}
```

Ce qui donne, dans le fichier manifeste :

```
// manifests/site.pp:
node 'myhost' {
  include epel::repo
  include monit::monit
  include monit::alert
}
```

On en profite d'ailleurs pour généraliser encore plus notre manifeste, en définissant un node **monit-node** dont notre serveur de test, **myhost**, va hériter :

```
node 'monit-node' {
  include epel::repo
  include monit::monit
  include monit::alert
}

node 'myhost' inherits 'monit-node' {}
```

4 Mise en place des services sous le contrôle de Monit

4.1 Apache HTTPd

4.1.1 Installation du serveur

Le temps où l'on recompilait soi-même son serveur Apache étant révolu (enfin, on l'espère pour le lecteur), l'installation d'un serveur Apache est relativement simple :

```
# yum install -y httpd # sur RHEL
```

N'oublions pas d'installer **mod_proxy**, pour permettre à Apache d'effectuer une balance de charge entre les deux instances Wildfly, et aussi le **mod_status**, qui permet de bénéficier d'informations supplémentaires sur l'état du serveur. Sur RHEL, ils le sont par défaut, mais si vous utilisez une autre distribution il sera peut être nécessaire d'installer des paquets logiciels supplémentaires.

Quid de mod_jk et mod_proxy ?

Si le lecteur a déjà expérimenté avec Wildfly, ou même des versions antérieures, quand le serveur s'appelait encore JBoss, il aura sans doute entendu parler de **mod_jk** et **mod_cluster** qui sont des modules dédiés à l'utilisation de Apache HTTPd avec justement des serveurs Web Java. Néanmoins, par souci de simplicité et pour rester au cœur du sujet, nous utiliserons le module **mod_proxy**.

En effet, ce dernier est un *reverse proxy* dont le fonctionnement est simple - les requêtes sont dirigées vers les instances Wildfly, et il ne nécessite donc pas, à l'inverse des modules cités à l'instant, d'explication supplémentaire.

4.1.2 Configuration de Apache HTTPd

Il n'y a que peu de configuration à effectuer sur le serveur, tout du moins à ce stade. En effet, il suffit, comme toujours, de proprement définir son nom (**ServerName**) – ne serait-ce que pour retirer le désagréable message d'erreur à ce sujet au démarrage du service, et d'activer la page **/server-status** :

```
// /etc/httpd/conf/httpd.conf
...
ServerName myhost:80
...
<Location /server-status>
  SetHandler server-status
</Location>
```

4.1.3 Mise sous contrôle de Monit

Pour que Monit contrôle le processus de HTTPd, il est nécessaire d'ajouter un fichier décrivant ce nouveau service à surveiller au sein du répertoire **/etc/monit.d/**:

```
/etc/monit.d/httpd.service:

check process httpd with pidfile /var/run/httpd/httpd.pid
start program = "/sbin/service httpd start"
stop program = "/sbin/service httpd stop"
if failed host localhost port 80 protocol HTTP request / then restart
if 5 restarts within 5 cycles then timeout
```

Redémarrons le service **monit** pour s'assurer que les configurations sont correctes et que le service est désormais sous contrôle :

```
# service monit reload # vérifie que la configuration est correcte,
sans interrompre le service
# service monit restart # prend en compte la nouvelle configuration
```

4.2 Vérification du bon fonctionnement de Monit

Nous allons maintenant simplement simuler un *crash* de HTTPd en exécutant la commande **kill** sur son PID :

```
# kill -9 28497
```

Regardons maintenant le journal de Monit :

```
$ tail -f /var/log/monit.log
```

On constate bien, sans surprise, que Monit a détecté l'absence du processus et a redémarré automatiquement le serveur HTTPd. Assurons-nous maintenant que Monit cesse bien de redémarrer le serveur si celui-ci est constamment en échec. Pour démontrer ceci, nous allons créer un petit script en charge d'éteindre le service, toutes les secondes, pour forcer Monit à le redémarrer :

```
/root/keep-service-down.sh
#!/bin/bash

readonly SERVICE_NAME=${1:-'httpd'}
readonly SCAN_INTERVAL=${2:-'1'}

while true
do
  echo "Kill processes ${SERVICE_NAME} and wait for ${SCAN_INTERVAL}s."
  service "${SERVICE_NAME}" 'stop'
  sleep "${SCAN_INTERVAL}"
done
```

Une fois ce script lancé observons le journal de Monit :

```
[CEST Sep 25 16:27:07] info : 'myhost' Monit started
[CEST Sep 25 16:28:10] error : 'apache' process is not running
[CEST Sep 25 16:28:12] info : 'apache' trying to restart
[CEST Sep 25 16:28:12] info : 'apache' start: /sbin/service
[CEST Sep 25 16:28:42] error : 'apache' failed to start
[CEST Sep 25 16:29:43] error : 'apache' process is not running
[CEST Sep 25 16:29:43] info : 'apache' trying to restart
[CEST Sep 25 16:29:43] info : 'apache' start: /sbin/service
...
[CEST Sep 25 16:39:14] error : 'apache' failed to start
[CEST Sep 25 16:40:14] error : 'apache' process is not running
[CEST Sep 25 16:40:14] info : 'apache' trying to restart
[CEST Sep 25 16:40:14] info : 'apache' start: /sbin/service
[CEST Sep 25 16:40:44] error : 'apache' failed to start
[CEST Sep 25 16:41:44] error : 'apache' service restarted 5
times within 5 cycles(s) - unmonitor
```

Comme on peut le voir, après plusieurs essais dans le temps imparti, Monit cesse de surveiller le processus (action qu'il désigne sous le nom de *unmonitor*). Bien évidemment, Monit envoie une notification pour indiquer ceci.

À partir de maintenant, nous pouvons librement éteindre ou démarrer HTTPd, Monit ne surveillant plus le processus. Ce comportement peut sembler étrange de prime abord, mais il s'agit en fait d'une fonctionnalité cruciale. En effet, redémarrer sans cesse un service peut avoir de fâcheuses conséquences. Par exemple, certains logiciels au démarrage créent un ensemble de jeux de données de seulement quelques méga-octet, dans un répertoire temporaire, généré automatiquement dans **/tmp**. Si on redémarre le logiciel à l'infini, on peut aisément consommer tout l'espace disponible sur cette partition et faire purement et simplement s'effondrer le système !

Ceci étant dit, arrêtons notre script « tueur de service » et demandons à Monit de surveiller de nouveau HTTPd :

```
# monit monitor httpd start
# tail -f /var/log/monit
[CEST Sep 26 10:09:22] info : 'myhost' Monit started
[CEST Sep 26 10:10:17] debug : monitor service 'httpd' on user request
[CEST Sep 26 10:10:17] info : monit daemon at 21451 awakened [CEST
Sep 26 10:10:17] info : Awakened by User defined signal 1
[CEST Sep 26 10:10:17] info : 'apache' monitor action done
```

4.3 Mise en place avec Puppet

Là encore, la configuration de Apache avec Puppet est triviale :

```
// modules/httpd/manifests/init.pp

class httpd::httpd {
  package { 'httpd': ensure => installed, }

  file { '/etc/httpd/conf/httpd.conf':
    content => template('httpd/httpd.conf.erb'),
    require => Package['httpd'],
    notify => Service['httpd'],
  }

  service { 'httpd':
    ensure => running,
    enable => true,
    require => File['...'],
  }
}

// modules/httpd/templates/httpd.conf.erb
...
ServerName <%= @hostname %>:80
...
<Location /server-status>
  SetHandler server-status
</Location>
...
```

La précédente configuration ne tient compte néanmoins que d'une installation « standard » d'Apache. On ne mentionne nulle part la mise sous contrôle de ce service par Monit. Pour ce faire, nous allons rajouter, dans notre module Monit, une nouvelle définition de ressource :

```
// modules/monit/manifests/init.pp
...

define monit::service($service_name=$name,$pid_file, $http_
port=80, $bind_ip='localhost' $nb_failed_restart=5, $nb_cycles=5)
{
  file { "/etc/monit.d/$service_name":
    content => template('monit/service.erb'),
    require => Package['monit'],
  }
}
...

// modules/monit/templates/service.erb
check process <%= @service_name %> with pidfile <%=@pid_file %>
start program = "/sbin/service <%= @service_name %>
start" stop program = "/sbin/service <%= @service_name %> stop"
if failed host <%= @bind_ip %> port <%= @http_port %> protocol
HTTP request / then restart
if <%= @nb_failed_restarts §> restarts within <%= @nb_cycles %>
cycles then timeout
```

Une fois ce module défini, il ne nous reste plus qu'à mettre à jour notre fichier manifeste, et à utiliser notre nouvelle définition de ressource :

```
// manifests/site.pp:
node 'myhost' {
    include epel::repo
    include monit::monit
    include monit::alert
    include httpd::httpd

    monit::service { 'httpd':
    pid_file => '/var/run/httpd/httpd.pid',
    require => Service['httpd'],
  }
}
```

4.4 Installation de Wildfly

4.4.1 Préparation du système

Wildfly étant un serveur d'applications Java/Java EE, la première étape à effectuer est bien évidemment l'installation d'un compilateur et d'une machine virtuelle Java. Si, pendant de longues années, la machine virtuelle de référence était celle fourni par Sun (*HotSpot*), désormais fournie par Oracle, son équivalent *Open Source* OpenJDK a désormais bien rattrapé son prédécesseur et est largement aussi performante. Nous allons donc installer cette dernière (avec son compilateur) :

```
# yum install -y openjdk-devel
```

4.4.2 Installation du serveur

Sur RHEL, si vous disposez d'une souscription pour JBoss EAP, vous pouvez simplement utiliser **yum** pour effectuer l'installation :

```
# yum install jbossas
```

Sinon, comme peu de distributions fournissent des paquets logiciels pour les logiciels *Open Source*, pour des raisons qui nécessiteraient probablement un article à part à expliciter, il faudra effectuer une installation à partir de l'archive ZIP fournie sur le site de Widfly :

```
# wget http://download.jboss.org/wildfly/8.1.0.Final/wildfly-
8.2.0.Final.tar.gz
```

Il suffit ensuite de décompresser cette archive dans un répertoire. Comme le RPM fourni par Red Hat sur les systèmes RHEL installe ce serveur dans **/usr/share/jbossas**, nous ferons de même pour permettre, pour le reste de l'article, d'avoir la même configuration.

```
# tar xvzf  wildfly-8.1.0.Final.tar.gz
wildfly-8.2.0.Final/
wildfly-8.2.0.Final/.installation/
wildfly-8.2.0.Final/appclient/
...
wildfly-8.2.0.Final/standalone/tmp/auth/
```

Une fois l'archive décompressée (ou le RPM installé), il suffit de mettre en place ce serveur comme un service. Fort heureusement, un script est fourni à cet effet avec l'archive : **bin/init.d/jboss-as-standalone.sh**. Ce script utilise par défaut, comme fichier de configuration du serveur et de sa machine virtuelle, le fichier situé dans le même répertoire que lui **bin/init.d/jboss-as.conf**. On peut bien évidemment, et nous allons le faire, redéfinir ce fichier :

```
# grep -e 'jboss-as.conf' bin/init.d/jboss-as-standalone.sh
config: /etc/jboss-as/jboss-as.conf
JBOSS_CONF="/etc/jboss-as/jboss-as.conf"
```

Commençons déjà par utiliser le script fourni, pour vérifier que le serveur a été proprement installé et démarre sans problème :

```
# ln -s /usr/share/jbossas/bin/init.d/jboss-as-standalone.sh /
etc/init.d/wildfly
# service wildfly start # service wildfly status jboss-as is
running (pid 1460)
```

4.4.3 Mise en place de la seconde instance

Comme décrit plus haut, nous souhaitons disposer des deux instances de Wildfly, chacune s'exécutant dans une machine virtuelle Java séparée et dédiée - et utilisant une série de ports différents. Pour ce dernier point, c'est très aisé à faire puisque le serveur dispose d'un argument permettant de décaler de manière consistante l'ensemble des ports qu'il utilise.

Pour ce qui est de démarrer une seconde instance, il est nécessaire, d'une part de placer les données de chaque instance dans des répertoires distincts, et d'autre part de disposer de services distincts du point de vue système. Néanmoins, plutôt que de dupliquer les fichiers de services, et devoir maintenir manuellement les deux instances pourtant identiques, nous allons avoir recours à une astucieuse utilisation des liens symboliques.

```
/etc/init.d/wildfly

#!/bin/bash

readonly JBOSS_USER='jboss'
readonly JBOSS_CONF='/etc/jbossas/jbossas.conf'
export JBOSS_CONF

readonly JBOSS_HOME='/usr/share/jbossas/'
readonly SERVICE_CONF_DIR="${JBOSS_HOME}/standalone/configuration"

if [ "${0}" == '/etc/init.d/wildfly' ]; then
  echo 'This script can not be invoked directly. Please create a
symlink, with an ID value:'
  echo "# ln ${0} ${0}-1"
  exit 1
fi

readonly INSTANCE_ID=$(echo ${0} | sed -e 's/^.*-//')
readonly SERVICE_BASE_DIR="/usr/share/jbossas-${INSTANCE_ID}/"
readonly SERVICE_LOG_DIR="/var/log/jbossas/${INSTANCE_ID}"
readonly JBOSS_PIDFILE="/var/run/jboss-as/wildfly-${INSTANCE_ID}.pid"
readonly JBOSS_CONSOLE_LOG="${SERVICE_LOG_DIR}/console.log"
readonly PROG="wildfly-${INSTANCE_ID}"
readonly PORT_OFFSET=$(expr 100 * $(expr "${INSTANCE_ID}" - 1) )

export JAVA_OPTS="-Djboss.server.base.dir=${SERVICE_BASE_DIR} \
-Djboss.server.log.dir=${SERVICE_LOG_DIR} \ -Djboss.server.config.
dir=${SERVICE_CONF_DIR}\ -Djboss.socket.binding.port-offset=${PORT_
OFFSET}" "${JBOSS_HOME}/bin/init.d/jboss-as-standalone.sh" ${@}
```

Ce script de démarrage est relativement simple à comprendre. En essence, on se sert d'un numéro, ajouté au nom du lien symbolique, pour déterminer le décalage de port, le nom du service et où se situent les répertoires de données de l'instance. On modifie aussi la définition du fichier de configuration qui pointe désormais sur le fichier **/etc/jbossas/jboss.conf**, et dont le contenu est vide.

On notera qu'il ne faut pas oublier de créer l'utilisateur **jboss**, et les répertoires associés aux instances. En outre, il faudra attribuer à cet utilisateur la propriété de ces répertoires. Plus bas nous réaliserons cette tâche avec Puppet, mais ici nous le ferons tout d'abord à l'aide d'une série de commandes Bash :


```
# export JBOSS_USER='jboss'
# useradd -s /sbin/nologin "${JBOSS_USER}"
# mkdir /usr/share/jbossas-{1,2}
# mkdir /var/log/jbossas/{1,2}
# mkdir /var/run/jboss-as/
# chown -R "${JBOSS_USER}:${JBOSS_USER}" /usr/share/jbossas-* /
var/log/jbossas/* /var/run/jboss-as/
```

Il ne reste plus maintenant qu'à créer les deux liens symboliques vers le fichier de service, et démarrer nos instances :

```
# ln -s /etc/init.d/wildfy /etc/init.d/wildfy-1
# ln -s /etc/init.d/wildfy /etc/init.d/wildfy-2
# service widlfly-1 start # service widlfly-2 start
# service widlfly-1 status wildfly-1 is running (pid 1890)
# service widlfly-1 status wildfly-2 is running (pid 1895)
# curl -s http://$(hostname):8080 # pour vérifier l'accessibilté
de la première instance # curl -s http://$(hostname):8180 # pour
vérifier l'accessibilité de la seconde instance
```

Et le fameux *Domain Mode* ?

Introduit depuis JBoss AS 7, soit quelques versions avant le renommage du projet en Wildfly, ce mode de fonctionnement - très inspiré par celui du concurrent propriétaire du projet, WebLogic, permet de gérer plusieurs instances, qu'elles soient localisées sur une même machine ou sur plusieurs. Il forme donc un excellent moyen de mettre en place plusieurs instances sur une machine mais il nécessiterait plus d'explications spécifiques pour en décrire le fonctionnement, ce qui nous éloignerait de beaucoup du propos de l'article.

4.4.4 Mise en place avec Puppet

Bien que certains le pensent - et malheureusement parfois s'en servent ainsi, Puppet n'est pas un outil de déploiement, mais un outil destiné à maintenir un serveur dans un état défini. Ainsi, Puppet ne prend pas réellement en charge l'installation d'un logiciel - son déploiement pour être clair - mais utilise seulement les outils à sa disposition selon le système (ici **yum** pour gérer les RMS) pour vérifier la présence ou l'absence d'un paquet logiciel.

Tout ceci pour dire que si vous ne disposez pas d'un RPM pour Wildfly, la première étape est d'en réaliser un. Il s'agit d'un paquet sans architecture (**noarch**) et sans dépendance, plus complexe que la machine virtuelle Java que l'on souhaite utiliser. C'est plutôt trivial, et il existe plusieurs exemples en ligne - à commencer par le Git de l'auteur de cet article **[7]**. Nous n'allons donc pas nous écarter du sujet de l'article et supposer que, soit vous disposez d'un tel RPM, soit en tant que client de Red Hat vous avez accès au RPM fourni par la société.

La mise en place de Wildly/JBoss CAP par Puppet est donc assez « classique » et ressemble à ce que l'on a pu voir jusqu'à maintenant. Nous allons juste définir une ressource dynamique pour faciliter la mise en place des différentes instances :

```
class widlfly::wildfy($jboss_user='jboss') {
 user { $jboss_user:
    ensure => present,
    shell => '/sbin/nologin',
 }

 package { 'jbossas':
    ensure => installed,
    require => User[$jboss_user],
 }

 file { '/var/run/jboss-as/':
    ensure => directory,
    require => Package['jbossas'],
  }

  file { '/etc/init.d/wildfly':
    source => 'puppet:///modules/wildfly/wildfly',
    mode => 644, owner => 'root', group => 'root',
    require => Package['jbossas'],
  }

 file { '/etc/jbossas/jboss.conf':
    content => '', # le fichier est vide, car la logique est placé
dans le script de démarrage
    owner => root, group => root, mode => 755,
    require => Package['jbossas'],
  }
}

define wildfly::instance($id=$name, $jboss_home='/usr/share/
jbossas', $jboss_log_dir='/var/log/jbossas/' $jboss_user='jboss') {

 file { "$jboss_home-$id":
    ensure => directory,
    owner => $jboss_user,
    group => $jboss_user,
  }

  file { "$jboss_log_dir":
    ensure => directory,
  }

  file { "$jboss_log_dir/$id":
    ensure => directory,
    require => File[$jboss_log_dir],
  }

  $service="wildfly-$id"
  file { "/etc/init.d/$service":
    ensure => link,
    target => '/etc/init.d/wildfly',
    require => File['/etc/init.d/wildfly'],
  }

  service { "wildfly-$id":
    ensure => running,
    enable => yes,
    require => File["/etc/init.d/$service"],
 }
}
```

Mettons maintenant à jour notre fichier manifeste pour y déclarer les deux instances de Wildfly :

```
// manifests/site.pp:

node 'myhost' {
  include epel::repo
  include monit::monit
  include monit::alert
  include httpd::httpd
  include widlfly::wildfy

  wildfly::instance { ['1','2']: }
  monit::service { 'httpd':
     pid_file => '/var/run/httpd/httpd.pid',
     require => Service['httpd'],
  }
}
```

4.4.5 Mise sous contrôle de Wildfly par Monit (avec Puppet)

Grâce à la ressource que nous avons défini précédemment, il est très aisé de mettre sous contrôle de Monit nos instances Wildfly. Ajoutons donc maintenant une paire de nouvelles ressources à notre fichier manifeste :

```
// manifests/site.pp:

node 'myhost' {

  include epel::repo
  include monit::monit
  include monit::alert
  include httpd::httpd
  include widlfly::wildfy

  wildfly::instance { ['1','2']: }
  monit::service { 'httpd':
     pid_file => '/var/run/httpd/httpd.pid',
     require => Service['httpd'],
  }

  monit::service { 'wildfly-1':
     pid_file => '/var/run/jboss-as/wildfly-1.pid',
     http_port => 8080,
  }

  monit::service { 'wildfly-2':
     pid_file => '/var/run/jboss-as/wildfly-2.pid',
     http_port => 8180,
  }
}
```

4.5 Console de supervision

Bien que notre stratégie de supervision distribuée ne permette pas, par définition, de disposer d'une console de supervision globale, Monit propose néanmoins une simple console Web, locale, dont nous avons configuré l'interface et le port d'écoute plus haut. Nous allons maintenant rapidement présenter cette dernière.

Pour y accéder, il suffit donc de se connecter au port 2812 sur l'hôte `localhost`. On pourrait bien évidemment exposer cette console vers l'extérieur, mais comme cette dernière permet d'effectuer quelques opérations - comme par exemple désactiver le monitoring, ce n'est pas une approche recommandée.

Sécuriser l'accès à la console de Monit

Si l'on souhaite pourvoir accéder à la console de Monit depuis une machine distante, sans recours à des tunnels SSH - ou plus simplement si on souhaite mettre à disposition certaines de ses opérations à des utilisateurs authentifiés, il est recommandé de placer une instance Apache - ou un serveur Web plus léger, comme nginx **[8]** ou lighttpd **[9]** en frontal. Ces serveurs sont dotés des fonctionnalités nécessaires pour implémenter une authentification forte, et peuvent donc faire le relais entre la console graphique, disponible seulement en local, et le monde extérieur.

Le premier écran de la console donne une vision globale du système. Elle indique l'utilisation de ses ressources (CPU, Mémoire), mais aussi l'état des services que Monit supervise (voir figure 1) . Si l'on clique sur le lien associé au système, on obtient quelques informations plus détaillées, mais surtout un bouton permettant d'activer ou désactiver la console, comme le montre la figure 2. On peut aussi accéder aux informations, à la disposition de Monit, sur l'état d'un processus, comme par exemple, à une instance de JBoss EAP (voir figure 3). Le plus utile est sans doute, non pas les informations sur le processus en tant que tel, mais la possibilité d'effectuer des opérations sur le processus à travers Monit. Ceci peut se révéler très pratique, pour permettre à des développeurs ou des testeurs d'effectuer des tâches de maintenance sur un système applicatif sans pour autant leur donner les accès privilégiés (*root*).

Conclusion

Après un tour d'horizon, très technique, de notre cas d'utilisation, voici venu le temps de tirer quelques conclusions. La première étant que nous avons déjà démontré la faisabilité d'une telle supervision distribuée, à l'aide de Monit.

Une première limite, qui n'est pas liée au concept de supervision distribuée, mais plutôt à l'implémentation utilisée, chez Monit, est l'utilisation d'email pour remonter les alertes. S'il est relativement aisé aujourd'hui d'avoir une infrastructure de transfert

Fig. 1: Vision globale du système.

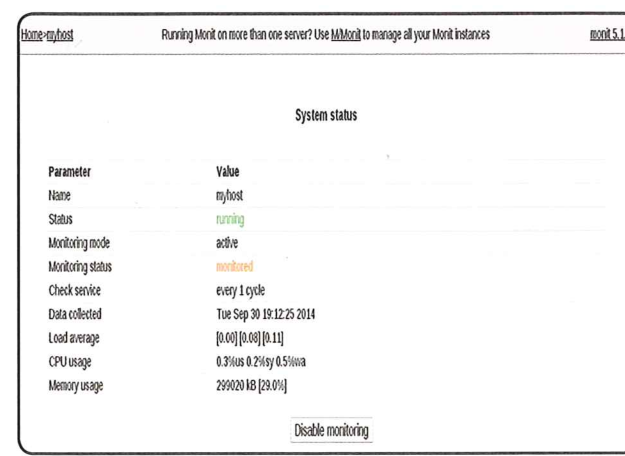


Fig. 2: Informations détaillées et bouton d'activation/désactivation de la console.

Fig. 3: État d'un processus (instance JBoss EAP).

de message résistante et hautement disponible, il n'en reste pas moins qu'une alerte ne peut jamais quitter le système en panne.

Mais là encore, il est important de noter que l'idée n'est pas de supprimer le système de supervision centrale, mais d'en limiter le périmètre au maximum. On peut donc laisser à ce dernier la charge de vérifier que le système cible est toujours en marche et que le serveur de mail utilisé est toujours accessible.

À l'inverse, comme l'a montré l'article, mettre en place Monit pour superviser des processus est très, très simple, surtout avec une gestion de configuration déléguée à un outil comme Puppet. Cette supervision est non seulement robuste, et capable de réagir à des changements d'état du service, mais elle tiendra également la mise à l'échelle, puisque chaque système supplémentaire déployé prendra en charge lui-même le coût de sa supervision. ■

Références

[1] Java Management Extensions : http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html

[2] RHQ : http://rhq-project.github.io/rhq/

[3] Monit : https://mmonit.com/monit/

[4] Munin : http://munin-monitoring.org/

[5] Puppet : http://puppetlabs.com/

[6] Dépôts logiciels EPEL: https://fedoraproject.org/wiki/EPEL

[7] RPM JBoss : https://github.com/rpelisse/eap6-rpm-infra

[8] Nginx : http://nginx.org/

[9] Lighttpd : http://www.lighttpd.net/

REST IN PEACE : PRÉSENTATION DE DJANGO REST FRAMEWORK

par **Jean-Michel Armand** [Cocréateur Hybird, Développeur Crème CRM et Djangonaute]

Les plus vieux d'entre vous, ou les plus malchanceux, savent que dans les temps immémoriaux du Web, il existait une horrible façon de mettre en place des Webservices, le protocole SOAP. Heureusement pour nous, REST est arrivé.

Cet article sera l'occasion de voir comment l'on peut mettre en pratique des webservices REST **[1]** en utilisant un framework Web Python bien connu dans ces pages : Django **[2]** (un HS **[3]** lui a été dédié en 2012 et des articles en parlent régulièrement). Nous commencerons par une petite présentation théorique des Webservices REST, ensuite nous mettrons rapidement en place un environnement de test avant de rentrer dans le vif du sujet en présentant l'une des applications phares en ce qui concerne la mise en place de webservices dans l'écosystème Django : Django REST Framework **[4]** (assez souvent dans les prochaines pages le nom Django REST Framework sera raccourci en DRF)

1 Mise en place de l'environnement de développement

1.1 Webservice REST : présentation

Il existe plusieurs façons de mettre en place des Webservices. Il y a quelques années maintenant la technique majoritairement utilisée était la mise en place de Webservices SOAP. Et puis certains se sont dit qu'il serait bien de raison garder, que le développement de webservices SOAP était loin d'être simple et qu'il y avait sûrement une façon moins complexe et plus intéressante de faire des Webservices.

Cette façon plus simple, ce sont les Webservices REST. Ceux-ci partent d'un principe simple : le protocole HTTP contient en son sein tout ce qui est nécessaire pour définir l'accès à des Webservices, surtout s'ils sont orientés ressources. Les URL permettent en effet de définir sur quel ensemble on veut travailler. L'url https://api.github.com/toto/issues/ est explicite : elle permet d'avoir la liste des issues de `toto`. Pour ce qui est de dire au Webservice ce que l'on souhaite faire comme action sur une url donnée, on utilise les méthodes HTTP (GET, POST, PUT, DELETE, HEAD étant les plus fréquentes pour indiquer ce que l'on fait) :

- GET vous permet de récupérer une liste de choses ou une unique chose ;
- POST vous permet de créer des choses ;
- PUT de les modifier ;
- DELETE de les supprimer ;
- et HEAD de demander des informations.

Enfin, quand c'est nécessaire, on peut utiliser les headers HTTP pour configurer certaines métadonnées utiles. Pour ce qui est du serveur, en plus du corps de sa réponse, il a seulement besoin d'utiliser les statuts de retour HTTP pour informer les clients du bon résultat ou pas des demandes.

La chose la plus importante dans un Webservice REST est donc au final la définition des URL. Par exemple, imaginons un site multi-langues mettant en place un Webservice REST. Comment permettre aux utilisateurs du Webservice de définir la langue dans laquelle ils veulent une ressource ? Est ce que cela doit être explicite dans l'url ou défini dans les headers HTTP ?

1.2 L'environnement virtuel python

Avant toute chose, on va commencer par ce qui est la base de tout démarrage d'un projet python, la création d'un virtualenv spécifique au dit projet. Pour cela, on va utiliser `virtualenwrapper`. Une fois notre environnement virtuel prêt, on pourra y installer dedans tout ce qui sera nécessaire à notre application RestFul.

Virtualenv et virtualenwrapper

Virtualenv **[5]** vous permet de créer de multiples environnements virtuels pour vos projets python. En ayant un environnement spécifique par projet, vous pouvez gérer finement les versions des librairies que vous voulez utiliser. De plus, grâce aux virtualenvs, vous pourrez installer des paquets python en espace utilisateur. Il peut arriver également que plusieurs de vos projets aient des dépendances incompatibles : les virtualenvs résoudront ce problème. Comme virtualenv en lui-même est un peu frustre d'utilisation, on l'utilise souvent à travers virtualenwrapper **[6]** qui fournit quelques commandes de plus haut niveau.

La création d'un virtualenv est vraiment enfantine (n'oubliez pas d'installer `virtualenvwrapper` et `virtualenv` si vous ne l'avez pas encore fait) :

```
$ mkvirtualenv drf
```

On va ensuite, tout simplement installer les librairies qui nous sont nécessaires pour le projet :

```
[drf]$ pip install django
[drf]$ pip install djangorestframework
[drf]$ pip install markdown
[drf]$ pip install django-filter
```

Markdown et Django-filter sont deux dépendances optionnelles de DRF. Markdown permettra de mettre en place une API dans laquelle vos utilisateurs pourront naviguer tandis que django-filter vous permettra de supporter des opérations de filtrage intéressantes.

Note

Si vous avez cloné les sources de l'article, vous pouvez aussi tout simplement faire un :

```
[drf]$ pip install -r requirements.txt
```

L'ensemble des dépendances se trouve en effet listé dans le fichier `requirements.txt`.

1.3 Le projet Django en lui-même

Il semblait plus judicieux de présenter Django REST Framework au sein d'un vrai projet plutôt que de simplement montrer de petits morceaux de code sans rapport entre eux. Le concept qui servira d'exemple est tout simple : une application Web de gestion de jeux de cartes s'appelant *hall of cards*. Les joueurs pourront construire des *decks* à partir de leurs cartes, celles-ci pouvant être de plusieurs types : les vertes étant celles des créatures des forêts, les rouges celles des dragons et des gobelins.

Note

Précision importante, si vous avez cloné le dépôt git des sources de l'article, vous avez déjà l'architecture du projet et vous n'avez donc pas besoin de faire les étapes suivantes.

Nous allons commencer par créer notre projet Django. Pour cela il suffit de taper :

```
[drf]$ django-admin startproject hall_of_cards
```

Cela va créer l'arborescence de base de votre projet ; celle-ci se compose pour l'instant du strict minimum : un répertoire `hall_of_cards` qui contient un fichier `manage.py` et un répertoire s'appelant lui aussi `hall_of_cards` dans lequel se trouvent plusieurs fichiers. Vous allez avoir les fichiers :

- `urls.py` qui définit le premier routage des urls vers les vues de votre application. C'est un peu la table des matières de votre projet. Chaque requête de vos utilisateurs passera forcément par le routage défini dans ce fichier.
- `settings.py` qui vous permet de configurer votre projet.

- **wsgi.py** qui est un point d'entrée pour les serveurs compatibles WSGI. Ce fichier ne sert que lorsque vous voulez mettre en production (si vous utilisez un serveur WSGI, bien sûr).

Pour l'instant, pour vérifier que tout fonctionne bien, nous allons simplement tester que l'interface d'administration s'affiche. Pour simplifier les choses, nous resterons sur une base de données sqlite.

La seule chose que nous allons donc modifier pour l'instant c'est la configuration de la langue et le fuseau horaire de notre projet pour que celui-ci soit en français et à l'heure de Paris (fichier **settings.py**) :

```
66: # Internationalization
67: # https://docs.djangoproject.com/en/1.7/topics/i18n/
68:
69: LANGUAGE_CODE = 'fr'
70:
71: TIME_ZONE = 'Europe/Paris'
```

Maintenant il va falloir créer notre première base de données et notre premier utilisateur. Pour la création de la base de données, attention, les choses ont changé depuis la sortie de Django 1.7. La commande **syncb** n'est en effet plus utilisée depuis que South **[8]** a été intégré dans Django. Pour initialiser votre projet il faut maintenant lancer la commande **migrate**. Allons-y donc :

```
[drf]$ cd hall_of_cards
[drf]$ python manage.py migrate
```

Si tout se passe bien vous devriez avoir une sortie ressemblant à celle-ci :

```
Operations to perform:
  Apply all migrations: admin, contenttypes, auth, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying sessions.0001_initial... OK
```

Pour ceux d'entre vous qui ont pu faire du Django avant 1.7, vous constaterez qu'il y a une amélioration dans la clarté des informations.

Avant de tester que notre Django est bien déployé, il nous faut créer un utilisateur. Cela sera chose faite en lançant :

```
[drf]$ python manage.py createsuperuser
```

vous pouvez maintenant tester que votre Django est bien installé en lançant le serveur de dev (avec la commande ci-après) et en allant, grâce à votre butineur Web préféré, à l'adresse http://127.0.0.1:8000.

```
[drf]$ python manage.py runserver
```

Si tout s'est bien passé, vous devriez voir un formulaire d'authentification qui, une fois passé, vous laissera admirer l'interface d'administration de Django. Le socle de notre projet étant prêt, il est maintenant temps de se retrousser les manches et de commencer à utiliser Django Rest Framework.

2 Django REST Framework, premier contact

2.1 Notions importantes

Aux notions de vues et d'urls de Django, DRF rajoute une notion importante, celle de « sérialiseur ». Pour pouvoir envoyer des données à vos consommateurs d'API il faut en effet que DRF sache comment transformer vos données projet en données « API ». Ce sont les sérialiseurs qui vont s'occuper de cela.

Concernant la construction des vues, DRF vous offre plusieurs possibilités. Vous pouvez utilisez suivant vos besoins des *Functions Based Views* (FBV) ou des *Class Based Views* (CBV). Il faut bien reconnaître que DRF vous offre bien plus d'outils pour construire des CBV, mais l'utilisation des FBV n'est pas impossible. Au niveau des CBV, DRF vous propose un ensemble complet de vues génériques à la manière de Django. Vous aurez donc plusieurs CBV utilisables de la même manière que celles de Django. Vous disposez bien entendu d'une bonne tripoté de *mixin* pour construire vos propres CBV. Enfin dans le cas où vous ne pouvez pas utiliser les CBV génériques de Django REST Framework, vous pourrez dériver de la class **APIView** (les CBV génériques de DRF dont toutes les classes dérivent de **GenericApiView**, elle-même étant une **APIView** et issue d'un certain nombre de *mixin*) pour construire vos propres vues.

En plus des CBV et des CBV génériques, DRF vous propose la notion de *ViewSet* et de routeurs. Les *Viewset* sont, comme leurs noms l'indiquent, des ensembles de vues. En une seule classe vous allez pouvoir

regrouper plusieurs vues génériques et vous simplifier ainsi la tâche. Quant aux routeurs, ils vont permettre de générer un ensemble homogène d'urls en fonction d'un *ViewSet* et, bien entendu, ils se connectent au mécanisme de routage d'urls de Django.

2.2 Premières utilisations et application exemple

La première chose à faire pour utiliser DRF, c'est de l'activer en temps qu'application Django. Pour cela il suffit de modifier votre **settings.py** en y ajoutant **rest_framework** :

```
32: INSTALLED_APPS = (
33:     'django.contrib.admin',
34:     'django.contrib.auth',
35:     'django.contrib.contenttypes',
36:     'django.contrib.sessions',
37:     'django.contrib.messages',
38:     'django.contrib.staticfiles',
39:    'rest_framework',
40: )
```

Maintenant, il va nous falloir avoir des choses à servir à nos utilisateurs. Nous allons donc commencer à développer notre application de gestion de jeux de cartes. Pour cela nous créons une application Django :

```
[drf]$ django-admin startapp cardsgame
```

Dans cette application, nous allons définir deux modèles : un pour gérer les cartes et un autre pour gérer les types de cartes (le code ci dessous est limité à la déclaration des modèles, le fichier complet est disponible dans le code de l'article) :

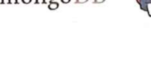
```
01: @python_2_unicode_compatible
02: class CardType(models.Model):
03:     name = models.CharField(_("Type of Card"), max_length=250, unique=True)
04:
05:     def __str__(self):
06:         return self.name
07:
08: @python_2_unicode_compatible
09: class Card(models.Model):
10:     name = models.CharField(_("Card's Name"), max_length=250, unique=True)
11:     description = models.TextField(_("Description"))
12:     mana_cost = models.PositiveIntegerField(_("Mana Cost"), default=0)
13:     life = models.PositiveIntegerField(_("Life"), default=0)
14:     damage = models.PositiveIntegerField(_("Damage"), default=0)
15:     card_type = models.ForeignKey(CardType, verbose_name=_("Type of Card"))
16:
17:     def __str__(self):
18:         return " %s, type : %s " % (self.name, self.card_type)
```


LINAGORA
Recrute !
Nos Offres d'emplois
DEVELOPPEUR SYMFONY
COMMUNITY MANAGER
BUG HUNTER
RESPONSABLE TECHNIQUE
CONTRIBUTEUR JBOSS/TOMCAT/JEE
DEVELOPPEUR DRUPAL
ARCHITECTE
OPENLDAP
INGENIEUR COMMERCIAL
INGENIEUR R&D
EXPERT
LEADER TECHNIQUE
INTEGRATEUR
CHEF DE PROJET
CONSULTANT TECHNIQUE ET FONCTIONNEL
DEVELOPPEUR FRONT/BACK
DEVELOPPEUR OPENSTACK
ADMINISTRATEUR SYSTEME & RESEAUX
Nos Produits

GESTION ET FÉDÉRATION DES IDENTITÉS

PARTAGE DE FICHIERS

PLATEFORME COLLABORATIVE CLOUD
Nos Technos


HTML5 node ANGULARJS openstack git Symfony CSS3 python puppet labs mongoDB Jenkins
We want you !

job@linagora.com

Les lignes 1, 8, ainsi que les fonctions `__str__` (lignes 5 à 6 et 17 à 18) servent à gérer la compatibilité python 2.7 et python 3.4. Pour le reste, on est dans des modèles classiques Django. On a un type de carte et des cartes, celles-ci possédant divers attributs (nom, description, type, points de vie et points d'attaque).

D'ici quelques lignes nous allons pouvoir proposer une liste de cartes en REST. Pour commencer, il faut créer les nouvelles tables. Nous allons tout d'abord activer notre application dans `settings.py`. Rien de plus simple on ajoute `'cardsgame'` au dessus de `'rest_framework'` dans le tuple des `INSTALLED_APPS`. Ensuite il faut créer les tables, là aussi les choses ont changé depuis l'intégration de South dans Django. Maintenant il suffit de créer les migrations en lançant :

```
[drf]$ python manage.py makemigrations
```

Puis de lancer celles-ci en lançant à nouveau :

```
[drf]$ python manage.py migrate
```

Une dernière chose avant de passer à la configuration de Django Rest Framework : il vous faut créer des données de test. Le dépôt de code de l'article contient les fichiers nécessaires pour configurer l'administration Django pour pouvoir simplement ajouter vos données, n'hésitez pas à aller le consulter si nécessaire ou faites votre propre configuration dans le fichier `admin.py` de votre application `cardsgame`

Maintenant que nous avons préparé les choses, il faut configurer DRF. Comme on a pu le voir plus haut, DRF a besoin de sérialiseurs pour pouvoir fonctionner. Nous allons donc créer deux sérialiseurs ; un pour chacun de nos modèles. Par convention, les sérialiseurs se trouvent dans un module (ou un paquet si vous en avez beaucoup) appelé `serializers.py`. Voici le nôtre :

```
01: from .models import Card, CardType
02: from rest_framework import serializers
03:
04: class CardSerializer(serializers.HyperlinkedModelSerializer):
05:     class Meta:
06:         model = Card
07:         fields = ('name', 'description', 'mana_cost', 'life',
08:                   'damage', 'card_type')
09:
10:
11: class CardTypeSerializer(serializers.HyperlinkedModelSerializer):
12:     class Meta:
13:         model = CardType
14:         fields = ('name',)
```

On a quelque chose de très simple et qui rappelle beaucoup les formulaires Django. Lignes 4 et 11 on commence par déclarer nos nouveaux sérialiseurs. On les fait dériver d'un des sérialiseurs fournis par DRF (on verra qu'il y en a plusieurs possibles). Ensuite, lignes 5 et 12, on retrouve la définition d'une classe `Meta` exactement comme pour un formulaire. Dans celles-ci on va définir à chaque fois (lignes 6 et 13) sur quel module on travaille et lignes 7, 8 et 14 quels sont les champs de notre modèle que le sérialiseur va prendre en compte. Là encore, rien de bien différent d'un formulaire Django.

Nous avons mis en place la première étape, maintenant il va falloir pouvoir utiliser les sérialiseurs que nous avons définis et, pour cela, il faut coder des vues. Voici le fichier `views.py` avec les deux vues en question :

```
01: from rest_framework import viewsets
02: from .models import Card, CardType
03: from .serializers import CardSerializer, CardTypeSerializer
04:
05:
06: class CardViewSet(viewsets.ModelViewSet):
07:     queryset = Card.objects.all()
08:     serializer_class = CardSerializer
09:
10:
11: class CardTypeViewSet(viewsets.ModelViewSet):
12:     queryset = CardType.objects.all()
13:     serializer_class = CardTypeSerializer
```

Après les imports nécessaires (lignes 1 à 3), on définit nos deux vues (lignes 6 et 11). Pour chacune des vues on définit le `queryset` sur lequel elles vont travailler (lignes 7 et 12) puis on leur déclare le sérialiseur qu'elles vont devoir utiliser (lignes 8 et 13). Et c'est tout. Aussi simple qu'une `class base views` de Django (voire plus simple d'ailleurs).

Il ne nous reste plus que deux petites choses à faire. La première est de définir les url de l'API. La deuxième est simplement de configurer l'accès à celles-ci.

Les urls se définissent, comme toujours en Django, dans le fichier `url.py`, voici le nôtre (pour l'application `cardsgame`) :

```
01: from django.conf.urls import url, include
02: from rest_framework import routers
03: from .views import CardTypeViewSet, CardViewSet
04:
05: router = routers.DefaultRouter()
06: router.register(r'card', CardViewSet)
07: router.register(r'cardstype',CardTypeViewSet)
08:
09: urlpatterns = [
10:     url(r'^', include(router.urls)),
11:     url(r'^api-auth/', include('rest_framework.urls',
namespace='rest_framework'))]
```


Encore une fois, on commence par les imports nécessaires au fonctionnement du code. On définit ensuite le routeur qui sera utilisé par DRF pour construire nos urls d'API. Ici, on définit en ligne 6 que pour travailler sur les cartes, l'url sera card et que pour travailler sur les types de carte l'url sera cardstype. Enfin lignes 9 à 11 on connecte le routeur DRF au mécanisme de routage d'url de Django : on définit tout d'abord ligne 10 le point d'entrée global pour l'utilisation de l'API au travers du routeur et ligne 11 on rajoute l'url d'authentification.

Il ne reste maintenant plus que quelques configurations à faire dans le fichier `settings.py` :

```
01: REST_FRAMEWORK = {
02:     'DEFAULT_PERMISSION_CLASSES': ('rest_framework.permissions.AllowAny',),
03:     'PAGINATE_BY': 10
04: }
```

La configuration de DRF se fait grâce au dictionnaire `REST_FRAMEWORK`. Ici on va configurer deux choses : le nombre d'items par page que l'on met à 10 en ligne 3 et ligne 2 on va configurer les droits par défaut au niveau des accès à l'API. Ici on définit la permission `AllowAny` qui permet de donner accès à tout le monde. On aurait pu utiliser les choix suivants :

- `rest_framework.permissions.IsAdminUser` qui limite l'accès à un administrateur ;
- `rest_framework.permissions.IsAuthenticated` qui oblige à être authentifier.

Bien entendu cette définition par défaut des permissions peut ne pas être suffisante, on verra plus tard comment ajouter des permissions par vues. Il est en effet maintenant temps de tester notre application. On commence par lancer le serveur de développement de Django (qui sera pour l'exemple lancé sur le port `8001`) :

```
[drf]$ python manage.py runserver 127.0.0.1:8001
```

Et ensuite, avec `curl` j'appelle une url de l'api :

```
$ curl -H 'Accept: application/json; indent=4'
http://127.0.0.1:8001/cardgame/card/
{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "name": "Gobelins grenade",
            "description": "Transporte un tonneau de grenade",
            "mana_cost": 2,
            "life": 2,
            "damage": 3,
            "card_type": "http://127.0.0.1:8001/cardgame/cardstype/1/"
        },
        {
            "name": "Zombie Affamé",
            "description": "il veut vous manger le cerveau",
            "mana_cost": 1,
            "life": 1,
            "damage": 1,
            "card_type": "http://127.0.0.1:8001/cardgame/cardstype/3/"
        }
    ]
}
```

On le voit, notre API nous renvoie l'intégralité des cartes qui sont définies dans l'application, soit pour l'exemple deux cartes. Plusieurs choses sont intéressantes dans cette sortie console. Tout d'abord on découvre que l'on peut avoir accès au type de carte grâce à l'url d'accès à tous les types de carte que l'on fait suivre d'une valeur de clé primaire. Par extrapolation, on peut en déduire que c'est la même chose pour les cartes et, effectivement l'url `http://127.0.0.1:8001/cardgame/card/1` nous donne accès à la première des cartes. Vous vous demandez peut-être pourquoi nous avons une url en valeur pour la clé `card_type`. C'est tout simplement parce que nous avons défini nos sérialiseurs comme étant des classes filles de `serializers.HyperlinkedModelSerializer`. Si nous avions défini que nos sérialiseurs étaient des classes filles de `serializers.ModelSerializer` nous aurions eu les clés primaires en valeur pour les champs clés étrangères. Voici un exemple avec un sérialiseur de cartes en mode *Model* :

```
$ curl -H 'Accept: application/json; indent=4'
http://127.0.0.1:8001/cardgame/card/1/
{
    "name": "Gobelins grenade",
    "description": "Transporte un tonneau de grenade",
    "mana_cost": 2,
    "life": 2,
    "damage": 3,
    "card_type": 1
}
```

Vous trouverez dans le code de l'article les deux sérialiseurs dans les deux modes *Model* et *HyperLink*.

Notre application est maintenant capable de nous donner la liste des cartes et des types de cartes ainsi que d'en récupérer une seule instance. Il serait temps de mettre en place la modification et la création d'instances. Pour cela nous allons commencer par changer d'outil de test d'API.

curl est en effet un peu trop épuisant à utiliser dès que l'on veut faire plus qu'un simple appel GET sur une URL. Nous allons donc utiliser un outil plus récent et plus simple nommé httpie **[8]**. Pour cela, un simple petit appel à **pip** suffit, httpie étant développé en python.

```
[drf]$ pip install httpie
```

Un certain nombre de dépendances sera installé par la même occasion, ne vous inquiétez pas. httpie permet de faire très simplement des appels POST ou PUT sur des urls. Et si nous essayions de créer une carte ?

```
[drf] $ http --form POST 127.0.0.1:8001/cardgame/modelcard/
name='test3' description="desc test" mana_cost=1 life=2 damage=3
card_type=1
HTTP/1.0 201 CREATED
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Date: Mon, 02 Feb 2015 01:04:45 GMT
Server: WSGIServer/0.1 Python/2.7.5+
Vary: Accept, Cookie
X-Frame-Options: SAMEORIGIN

{
    "card_type": 1,
    "damage": 3,
    "description": "desc test",
    "life": 2,
    "mana_cost": 1,
    "name": "test3"
}
```

Cela fonctionne sans rien faire de plus, grâce à la magie des *ViewSet*. Essayons de modifier une carte pour voir si cela fonctionne également (dans les captures de retour console suivantes, nous avons enlevé toute la partie entête HTTP pour plus de clarté) :

```
[drf] $  http  127.0.0.1:8001/cardgame/modelcard/4/
HTTP/1.0 200 OK
...
{
    "card_type": 1,
    "damage": 3,
    "description": "desc test",
    "life": 2,
    "mana_cost": 1,
    "name": "test3"
}
[drf] $  http --form PUT 127.0.0.1:8001/cardgame/modelcard/4/
name='dragonet' description="petit dragon rose" mana_cost=2
life=2 damage=1 card_type=1
HTTP/1.0 200 OK
...
{
    "card_type": 1,
    "damage": 1,
    "description": "petit dragon rose",
    "life": 2,
    "mana_cost": 2,
    "name": "dragonet"
}
[drf] $  http  127.0.0.1:8001/cardgame/modelcard/4/
HTTP/1.0 200 OK
...
{
    "card_type": 1,
    "damage": 1,
    "description": "petit dragon rose",
    "life": 2,
    "mana_cost": 2,
    "name": "dragonet"
}
```

Effectivement tout tourne correctement. Un dernier test : imaginons que nous avons une autre carte dont le nom serait **test4** et que nous voulions modifier notre carte **4** pour lui donner également le nom **test4**. Cela devrait normalement ne pas fonctionner du fait que le champ **name** de notre modèle est défini avec **Unique=True**. Testons cela :

```
[drf] $ http --form PUT 127.0.0.1:8001/cardgame/modelcard/4/
name='test4' description="petit dragon rose" mana_cost=2 life=2
damage=1 card_type=1
HTTP/1.0 400 BAD REQUEST
...
{
    "name": [
        "This field must be unique."
    ]
}
```

Nous recevons bien un statut de retour d'erreur **400** et la cause de l'erreur, ici très claire.

Vous l'avez peut-être remarqué en faisant vos propres tests, mais chaque appel PUT pour modifier une valeur nous oblige à donner à minima tous les champs obligatoires. C'est un peu lourd comme procédé. Heureusement il existe la méthode HTTP PATCH qui fait en sorte que tous les champs deviennent optionnels. On peut donc faire ceci :

```
[drf] $ http --form PATCH 127.0.0.1:8001/cardgame/modelcard/5/
mana_cost=4
```

Avant d'aller plus loin parlons d'une chose intéressante, la découverte par simple navigation HTTP de votre API. En effet, si vous utiliser un navigateur pour vous rendre à l'url de votre API vous allez voir une page Web telle que celle présentée en Figure 1.

DÉCOUVREZ NOS NOUVELLES OFFRES D'ABONNEMENTS !

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

www.ed-diamond.com

LES COUPLAGES PAR SUPPORT :

VERSION **PAPIER**
Retrouvez votre magazine favori en papier dans votre boîte à lettres !

VERSION **PDF**
Envie de lire votre magazine sur votre tablette ou votre ordinateur ?

ACCÈS À LA **BASE DOCUMENTAIRE**
Effectuez des recherches dans la majorité des articles parus, qui seront disponibles avec un décalage de 6 mois après leur parution en magazine.

Sélectionnez votre offre dans la grille au verso 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 :	

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

❑ 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.

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

Prix TTC en Euros / France Métropolitaine

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

CHOISISSEZ VOTRE OFFRE !

SUPPORT — Prix en Euros / France Métropolitaine

Offre	ABONNEMENT	PAPIER Réf	Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	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° MISC + 6n° HS + 2n° HS + 6n° HK* + 4n° OS + 6n° LP + 3n° HS	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.

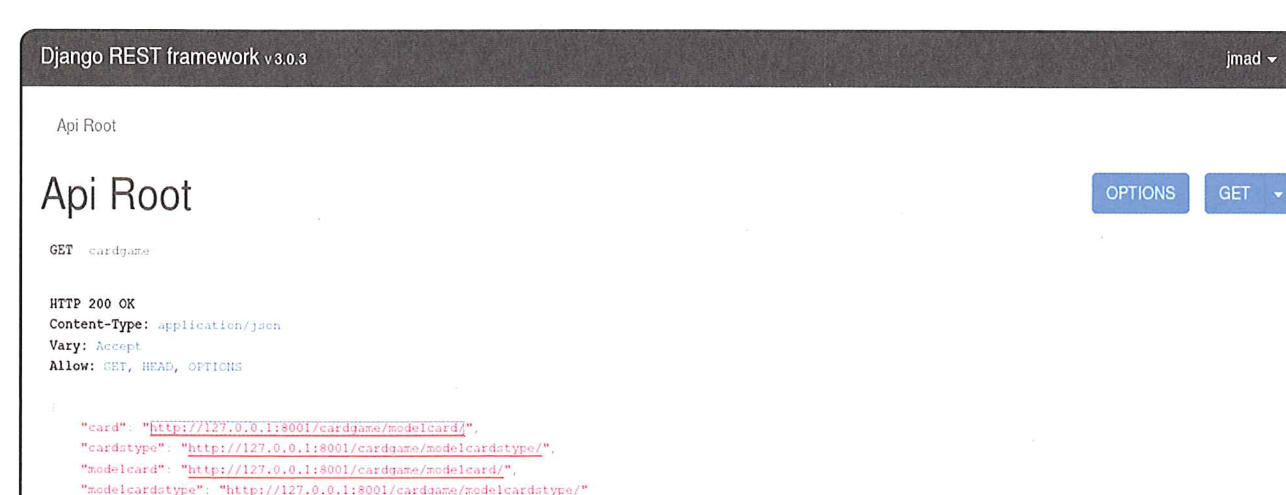


Fig. 1 : Découverte de votre API à travers votre navigateur

En suivant les liens donnés par cette page, vous allez pouvoir découvrir votre API et l'utiliser directement à travers votre navigateur. Aller à l'url http://127.0.0.1:8001/cardgame/modelcard/ vous permettra d'accéder à la liste des cartes, mais aussi d'en créer de nouvelle en utilisant un formulaire Web !

2.2 Personnalisation de ViewSets et de sérialiseurs

Jusqu'à présent nous nous sommes contentés d'utiliser les ressources fournies directement par DRF. Mais les mécanismes disponibles ne sont pas forcément suffisant. Imaginons par exemple que nous voulons pouvoir stocker en interne le nombre de fois où une carte a été modifiée. Ce champ ne devrait pas pouvoir être modifié par l'API que ce soit en création ou en modification. Il faudrait pourtant pouvoir le mettre à **0** automatiquement à la création et le mettre à jour à chaque modification. Commençons par ajouter le champ dans notre modèle de carte et par effectuer les migrations :

```
class Card(models.Model):
...
    modified = models.PositiveIntegerField(_("Modified"), null=True)
```

Ensuite il va nous falloir modifier notre sérialiseur. Celui-ci possède deux fonctions **create** et **update** que nous allons modifier pour qu'elles conviennent à nos besoins. Voici le code après modification :

```
01: class ModelCardSerializer(serializers.ModelSerializer):
02:     class Meta:
03:         model = Card
04:         fields = ('name', 'description', 'mana_cost', 'life',
05:                   'damage', 'card_type')
06:
07:     def create(self, validated_data):
08:         return Card.objects.create(modified=0, **validated_data)
09:
10:     def update(self, instance, validated_data):
11:         instance.name = validated_data.get('name', instance.name)
12:         instance.description = validated_data.
get('description', instance.description)
13:         instance.mana_cost = validated_data.get('mana_cost',
instance.mana_cost)
14:         instance.life = validated_data.get('life', instance.life)
15:         instance.damage = validated_data.get('damage',
instance.damage)
16:         instance.card_type = validated_data.get('card_type',
instance.card_type)
17:         instance.modified = instance.modified + 1 if instance.
modified else 1
18:         instance.save()
19:         return instance
```

La fonction **create** définie dans les lignes 7 et 8 est toute simple : elle récupère les données validées et lance la création d'une carte en ajoutant la valeur de **modified**. La fonction **update**, bien que plus longue (lignes 10 à 19), est tout aussi triviale. On se contente de récupérer les nouvelles valeurs dans le dictionnaire **validated_data** si celles-ci existent et de faire les modifications. Ligne 17, on ajoute la modification manuelle de **modified** (qui est définie comme pouvant être à **null** dans notre modèle d'où le **if** uni-ligne).

On teste ensuite les choses en demandant une modification d'une carte :

```
[drf] $ http --form PUT 127.0.0.1:8001/cardgame/modelcard/6/
name='Nouvelle carte' description="test create in serializer"
mana_cost=3 life=1 damage=0 card_type=1
HTTP/1.0 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Date: Mon, 02 Feb 2015 20:09:47 GMT
Server: WSGIServer/0.1 Python/2.7.5+
Vary: Accept, Cookie
X-Frame-Options: SAMEORIGIN

{
    "card_type": 1,
    "damage": 0,
    "description": "test create in serializer",
    "life": 1,
    "mana_cost": 3,
    "name": "Nouvelle carte"
}
```

Est ce que cela a marché ? Il va falloir aller voir directement dans l'administration de Django pour le savoir.

Ce qui serait intéressant c'est de pouvoir afficher cette valeur **modified**. C'est possible avec l'attribut **read_only_fields** des sérialiseurs. Modifions légèrement notre sérialiseur pour lui ajouter cette liste :

```
01: class ModelCardSerializer(serializers.ModelSerializer):
02:     class Meta:
03:         model = Card
04:         fields = ('name', 'description', 'mana_cost', 'life',
05:                   'damage', 'card_type', 'modified')
06:         read_only_fields = ('modified',)
```

Comme vous le voyez en ligne 4, il ne faut pas seulement ajouter votre champ à la liste des champs en *read only*, il faut aussi l'ajouter à la liste des champs (**fields**) qui seront pris en compte par votre sérialiseur. Ensuite tentons une nouvelle modification de carte :

```
[drf] $ http --form PUT 127.0.0.1:8001/cardgame/modelcard/5/
name='test4' description="decdec" mana_cost=3 life=1 damage=0
card_type=1
...

{
    "card_type": 1,
    "damage": 0,
    "description": "decdec",
    "life": 1,
    "mana_cost": 3,
    "modified": 4,
    "name": "test4"
}
```

Une précision intéressante : tous les champs qui sont définis en **editable=False** dans votre modèle sont automatiquement considérés par DRF comme étant en lecture seule. Nul besoin de les ajouter en plus à **read_only_fields**.

Enfin, on peut vouloir ajouter d'autres fonctionnalités à notre API et ne pas la limiter à des fonctions par défaut. Il va ici falloir modifier notre *ViewSet*. Imaginons que nous voulons ajouter une url permettant de connaître le nombre de cartes et que nous voulons également avoir une url qui donne uniquement la valeur du champ **modified** d'une carte. Nous allons alors modifier notre *ViewSet* pour avoir au final le code suivant :

```
01: class ModelCardViewSet(viewsets.ModelViewSet):
02:     queryset = Card.objects.all()
03:     serializer_class = ModelCardSerializer
04:
05:     @list_route()
06:     def count(self, request):
07:         return Response(Card.objects.count())
08:
09:     @detail_route()
10:     def modified(self, request, pk):
11:         try:
12:             card = Card.objects.get(pk=pk)
13:             return Response(card.modified)
14:         except ObjectDoesNotExist:
15:             return Response('Card not Exist', status=status.
HTTP_400_BAD_REQUEST)
```

On définit donc nos deux fonctions. La première, la fonction **count()** travaille sur les listes. On la décore avec **list_route()** en ligne 5. Quand à la fonction **modified()**, elle travaille sur une seule carte et on la décore donc avec **detail_route()** (ligne 9). Il faut de plus gérer le fait que l'utilisateur peut nous donner une clé primaire qui n'existe pas. Dans ce cas là, on lui renvoie en ligne 15 une réponse en statut d'erreur en lui donnant un message expliquant l'erreur. Quand les choses se passent bien, on lui renvoie un objet **Response** contenant l'information qu'il a demandé. Les décorateurs **list_route()** et **detail_route()** routent par défaut les fonctions sur la méthode GET HTTP. Pour ajouter d'autres méthodes HTTP, il faut utiliser l'argument **methods** comme par exemple dans **detail_route(methods=['post', 'put'])**. Enfin les deux décorateurs acceptent un autre argument, **permission_classes** qui est une liste de permissions que l'utilisateur qui appelle cette fonction doit valider.

3 Allons un peu plus loin avec DRF

3.1 Faire des tests

On le sait tous, un code qui n'est pas testé est un code qui ne fonctionne pas. Comment donc tester les choses avec DRF ? En fait, DRF met à notre disposition plusieurs classes qui sont un peu des miroirs de celles de Django. Tout d'abord on a deux classes qui permettent d'interagir avec l'API. La première, **APIRequestFactory**, étend la **RequestFactory** [9]. Quand à la seconde, **APIClient**, elle étend la classe **Client** de Django. Ensuite on a les classes de *TestCase* qui sont **APISimpleTestCase**, **APITransactionTestCase**, **APITestCase**, et **APILiveServerTestCase**.

Un test de notre vue de création de cartes ressemblerait à cela :

```
01: class CardTests(APITestCase):
02:     def setUp(self):
03:         CardType.objects.create(name='Type1')
04:
05:     def test_create_cards(self):
06:         url = reverse('card-list')
07:         card_data = {'name': 'CarteTest', 'description': 'desc', 'mana_cost': 1,
08:                      'life': 1, 'damage': 2, 'card_type': 1}
09:         response = self.client.post(url, card_data, format='json')
10:         self.assertEqual(response.status_code, status.HTTP_201_CREATED)
11:         card_data['modified'] = 0
12:         self.assertEqual(response.data, card_data)
```

On commence par définir une fonction de **setUp** pour créer un type de carte (lignes 2 et 3), et ensuite on crée notre test dans les lignes 5 à 10. On récupère l'url en ligne 6 (vous remarquerez à ce sujet que l'on n'a jamais défini de nom pour nos urls, DRF l'a fait pour nous suivant sa convention), puis on construit notre dictionnaire définissant notre carte et on appelle l'url en **POST** en ligne 9. En ligne 10 on teste que le statut de retour est bon, et en ligne 12 que les données de la réponse sont égales à celles que l'on avait envoyé. Comme on rajoute la donnée **modified** dans le retour, il faut l'ajouter aussi dans le dictionnaire qui nous sert de test, ce que l'on fait ligne 11.

3.2 Jouer avec les permissions

On peut vouloir modifier les permissions pour une *View* ou un *ViewSet*. Dans le cas d'une *View* CBV ou d'un *ViewSet* cela se fait avec l'attribut de classe **permission_classes** qui accepte une liste de permissions à valider. Si on voulait restreindre l'accès de notre *ViewSet* aux utilisateurs authentifiés, on ferait donc ainsi :

```
from rest_framework.permissions import IsAuthenticated
class CardViewSet(viewsets.ModelViewSet):
    permission_classes = (IsAuthenticated,)
    queryset = Card.objects.all()
    serializer_class = CardSerializer
```

Malheureusement la définition des permissions se fait par *View* ou par *ViewSet* pour ce qui est des méthodes classiques. On ne peut donc pas simplement les définir pour un seul *ViewSet* qui serait utilisé à la fois par les utilisateurs authentifiés et par les utilisateurs anonymes. Une solution possible serait de définir un *ViewSet* complet dérivant de **ModelViewSet** pour les utilisateurs authentifiés et utiliser un **ReadOnlyModelViewSet** qui lui se limiterait à proposer les actions en lecture seule à savoir **list()** et **retrieve()**

3.3 Valider les données

Il peut être utile de valider les données que l'on reçoit des utilisateurs. Par exemple dans notre système de carte, on peut vouloir valider que le coût des cartes est forcément supérieur ou égal à la somme des points de vie et des points de dommages. On va donc mettre en place une classe de validation pour le sérialiseur :

```
01: class CardsValidator(object):
02:
03:     def __call__(self, attrs):
04:         if attrs['mana_cost'] < attrs['life'] + attrs['damage']:
05:             raise serializers.ValidationError("Mana cost must be > at life + damage ")
06:
07: class ModelCardSerializer(serializers.ModelSerializer):
08:     class Meta:
09:         model = Card
10:         fields = ('name', 'description', 'mana_cost', 'life',
11:                   'damage', 'card_type', 'modified')
12:         read_only_fields = ('modified',)
13:
14:         validators = [CardsValidator(), ]
```

Lignes 1 à 5 on crée notre validateur. Pour être une classe de validateur, une classe doit simplement implémenter la méthode `__call__()`. Ensuite on déclare tout simplement notre validateur dans le sérialiseur à la ligne 14.

Si l'on teste maintenant la création d'une carte :

```
[drf] $ http --form POST 127.0.0.1:8001/cardgame/modelcard/
name='TestCreateValidators' description="test" mana_cost=0
life=1 damage=1 card_type=1HTTP/1.0 400 BAD REQUEST
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Date: Mon, 02 Feb 2015 23:51:05 GMT
Server: WSGIServer/0.1 Python/2.7.5+
Vary: Accept, Cookie
X-Frame-Options: SAMEORIGIN

{
    "non_field_errors": [
        "Mana cost must be > at life + damage "
    ]
}
```

On voit bien que notre ajout a été refusé et que l'on a bien reçu le message d'erreur défini dans le validateur. Une précision à propos des validateurs : dans DRF, la totalité de la validation se fait dans les validateurs du sérialiseur alors que dans Django la validation est à la fois faite dans le formulaire et dans le modèle.

3.4 Les nouveautés de DRF 3.0.4

La version 3.0.4 de Django REST Framework est sortie dans les touts derniers jours de janvier. Cette nouvelle version apporte quelques nouveautés intéressantes comme le support de Django 1.8a1 (sachant que la version 1.8 de Django est prévue pour sortir, si tout se passe bien, fin Mars 2015). Cette gestion 1.8 de Django implique la gestion des **`UUIDField`** et des **`HstoreField`**.

Conclusion

Cette présentation, bien que comportant un nombre certain de pages, est loin d'être exhaustive en ce qui concerne les possibilités de Django REST Framework. Vous avez toutefois les bases pour pouvoir aller plus loin et mettre en place des Webservices REST qui ne vous feront pas rougir. N'hésitez donc pas à tester et plonger dans la documentation qui très bien faite. Et si vous avez encore des questions, n'hésitez pas à joindre le chan irc **`#restframework`** sur freenode pour échanger avec les développeurs. Si vous n'êtes pas très anglophones, vous pouvez même venir sur **`#django-fr`** (toujours sur freenode), on saura vous y aider. ■

Références

[1] REST par Wikipedia : http://fr.wikipedia.org/wiki/Representational_State_Transfer

[2] Django : http://djangoproject.com

[3] T. Colombo et J.-M. Armand, « Django, le framework Python pour les perfectionnistes pressés... et les poneys aux pouvoirs magiques », GNU/Linux Magazine HS n°59, mars/avril 2012

[4] Django Rest Framework : http://www.django-rest-framework.org/

[5] VirtualEnv : http://www.virtualenv.org/en/latest/

[6] virtualenwrapper : http://www.doughellmann.com/docs/virtualenvwrapper

[7] South : http://south.aeracode.org/

[8] httpie : https://github.com/jakubroztocil/httpie

[9] Django Request Factory : https://docs.djangoproject.com/en/1.7/topics/testing/advanced/#django.test.client.RequestFactory

PROFESSIONNELS !

DÉCOUVREZ NOS NOUVELLES OFFRES D'ABONNEMENTS ...

PDF COLLECTIFS

PDF COLLECTIFS		PROFESSIONNELS					
		1 - 5 lecteurs		6 - 10 lecteurs		11 - 25 lecteurs	
OFFRE	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC	Réf	Tarif TTC
PROLM2	11 n° GLMF	PRO LM2/5	260,-	PRO LM2/10	520,-	PRO LM2/25	1040,-
PROLM+2	11 n° GLMF + 6 n° HS	PRO LM+2/5	472,-	PRO LM+2/10	944,-	PRO LM+2/25	1888,-

Prix TTC en Euros / France Métropolitaine

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 DOCU

ACCÈS COLLECTIFS BASE DOCU		PROFESSIONNELS					
		1 - 5 connexion(s)		6 - 10 connexions		11 - 25 connexions	
OFFRE	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC	Réf	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,-

Prix TTC en Euros / France Métropolitaine

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

...EN VOUS CONNECTANT À L'ESPACE DÉDIÉ AUX PROFESSIONNELS SUR : www.ed-diamond.com

ANDROID TV ET TV INPUT FRAMEWORK

*par **Benjamin Zores** [Directeur technique @ Alcatel-Lucent Enterprise]*

Fraîchement disponible depuis Android 5.0 (Lollipop), Android TV est le nouvel élan de Google pour s'imposer dans le monde déjà très fermé des Set-Top-Box et autres consoles de jeux. Comme à l'accoutumée, découvrons dès à présent les entrailles de la bête.

1 Android TV

Les plus fervents lecteurs l'ont en partie découvert au sein de GLMF n°178 **[1]**, Google cherche avec Android TV à reconquérir le marché des TVs connectées et autres Set-Top-Box (STB). Il faut dire que son premier essai, **GoogleTV**, résonne encore comme un profond échec de la part du géant. Mais les choses pourraient être différentes aujourd'hui. Le système a grandement mûri, aussi bien en interne, que dans les périphériques externes qu'il supporte et le catalogue **Google Play** force désormais le respect, permettant aux développeurs de porter leurs applications sur cette variante destinée aux TVs. Le système s'ouvre clairement aux industriels et autres diffuseurs de contenus vidéo et multimédia (tels que **Hulu** aux Etats-Unis et **Netflix**, maintenant également disponible en France), mais également aux particuliers, qui pourront envoyer leurs propres contenus personnels à leurs box grâce au support de la technologie de diffusion sans-fil **ChromeCast**. Mais Google ne s'arrête pas là et souhaite clairement créer un véritable centre multimédia avec Android TV, en s'attaquant au marché des consoles de jeux. Android supportant désormais nativement claviers, souris et autres manettes de jeux et ces derniers étant désormais foison sur le store, pourquoi s'en priver. Avec **Lollipop**, Google a donc annoncé (uniquement disponible à la vente aux États-Unis pour l'instant) sa propre Set-Top-Box Android TV, appelée **Nexus Player [2]**. De nombreux constructeurs chinois se sont également lancés dans leurs propres variantes. Notez également que de grandes marques telles qu'**Asus** et **Razer** se sont lancées dans une déclinaison de la STB plus orientée « *gamer* ». Du côté des constructeurs de TV, c'est **Sony**, **Sharp** et **TP Vision** qui ont annoncé un partenariat avec Google, afin de fournir des TVs reposant nativement sur Android TV (sans box additionnelle donc). Visuellement, le système séduit, comme vous pourrez le voir sur la figure 1.

Séduisant, certes, mais pas nécessairement innovant. En effet, nous ne sommes finalement pas très éloignés de ce que peut proposer une **AppleTV [3]** ou un MediaCenter libre tels que **XBMC / Kodi [4]**.

2 Adaptations Matérielles

Comme déjà précisé, le système Android TV se base sur les sources d'Android 5.0 (**Lollipop**), mais a réellement été conçu pour créer des applications « utilisables » sur ce type de périphériques (i.e. sans écran tactile). Comme déjà vu précédemment, en s'attardant rapidement aux sources AOSP, au sein du répertoire `device/google/atv`, nous remarquons d'ores et déjà les propriétés système suivantes :

Fig. 1 : Interface graphique du Nexus Player.

```
hw.screen=no-touch
hw.keyboard=yes
hw.battery=no
hw.accelerometer=no
hw.camera.back=none
hw.camera.front=emulated
hw.sensors.proximity=no
hw.sensors.magnetic_field=no
hw.sensors.orientation=no
hw.sensors.temperature=no
```

Que déduire de ces dernières ? Simplement que le système a enfin été modifié pour être modulaire, selon le type de périphérique (fixe, mobile, avec ou sans batterie, avec ou sans écran tactile, avec ou sans caméra ...) et qu'il est de moins en moins nécessaire de hacker les sources pour transformer le système pour un périphérique « non standard ». Typiquement, notre système Android TV tournant sur une TV (un appareil toujours alimenté électriquement, disposant d'un unique écran non-tactile, pilotable par une télécommande, assimilable à un clavier), nous sommes ici très loin de la configuration matérielle typique d'un smartphone. Le gros challenge sera donc d'adapter les applications pour qu'elles continuent de fonctionner sans les périphériques habituels (GPS, accéléromètre, écran tactile, caméra, boussole ...), mais surtout qu'elles proposent une interface utilisable. Nul doute ici que si presque toutes les applications pourront « tourner » sur Android TV, seules celles qui auront été adaptées au cas d'usage de la plateforme deviendront phares.

3 TV Input Framework (TIF)

Mais intéressons-nous désormais au changement principal avec le support du **TV Input Framework** (ou **TIF**), permettant aux applications de recevoir des flux vidéos de multiples sources, telles que des entrées HDMI, des tuners TV ou encore des récepteurs IPTV (les « box » de nos fournisseurs d'accès Internet par exemple). Le TIF simplifie l'accès aux contenus multimédias depuis Android TV, fournissant aux constructeurs une API permettant de créer différents modules d'entrée vidéo (source) permettant de contrôler Android TV mais également permettant aux diffuseurs de publier dynamiquement des métadonnées liées au contenu sur lesquelles le spectateur pourra effectuer des recherches. Mais Android TIF facilite surtout la vie aux équipementiers, leurs permettant de créer des STB à moindre frais sans être forcés d'implémenter chacun de leur côté les différentes normes de diffusions numériques et leurs spécificités locales (DVB, ATSC ...).

D'une manière générale, vous retrouverez au sein du schéma 2 une présentation de l'architecture logicielle du **TV Input Framework**. Les blocs logiques en vert sont mis à disposition par Google au sein de **Lollipop**. Les bleus sont à fournir par le constructeur de la STB et sont très liés au matériel et les oranges sont à fournir par le diffuseur de contenu (e.g. **Netflix**) et sont, quant à eux, très liés au contenu lui-même (et peuvent donc être chiffrés / protégés).En quelques mots, le framework TIF repose sur une application, la TV App. Il s'agit là d'une application système unique (il ne peut y en avoir qu'une, fournie par chaque constructeur de périphérique), qui ne peut être remplacée par une application tierce par l'utilisateur et qui a pour tâche l'affichage de toute l'interface graphique. Cette dernière communique ensuite avec différents modules TV au travers du **TV Input Manager,** qui gère toute interaction avec l'utilisateur (typiquement via une télécommande). Pour résumer les choses, les différents blocs logiques utilisés sont les suivants :

- La **TV App** (`com.android.tv.TvActivity`) qui, comme son nom le laisse supposer (**Activity**), propose l'interface graphique et gère l'interaction avec l'utilisateur.
- Le **TV Provider** (`com.android.providers.tv.TvProvider`), qui consiste en une base de données des différentes chaînes TV, programmes et autres permissions associées.
- Le **TV Input Manager** (`android.media.tv.TvInputManager`), qui permet d'interfacer (par abstraction) la TV App avec les différentes sources de type TV Inputs.
- Les différentes sources de type **TV Inputs**, des applications représentant des sources d'entrée vidéo, qu'elles soient physiques (e.g. un tuner TNT ou une entrée vidéo HDMI) ou logiques (e.g. un flux IPTV).
- La **TV Input HAL** (un module système de type `tv_input`), une couche d'abstraction matérielle permettant aux applications TV Input d'accéder au matériel (e.g. un tuner TV), sans rien connaître de l'implémentation de son pilote de périphérique.

Notez également que, sécurité oblige (du point de vue des diffuseurs du moins), Android fournit désormais tous les moyens nécessaires pour permettre aux TV Inputs de diffuser du contenu chiffré et protégé par DRM. Pour le grand public, le TIF propose également des fonctionnalités de contrôle parental (permettant de bloquer l'accès à certaines chaînes TV ou à un certain type de contenu) mais également le support de la technologie (généralisée) **HDMI-CEC**, permettant le contrôle de différents périphériques compatibles au sein d'une seule et unique télécommande). Si vous avez bien suivi, vous aurez noté que le **TV Provider** dispose d'une base de données contenant des informations privées (pour vous ou vos diffuseurs). Ainsi, afin d'en protéger l'accès, seules les applications **TV App** et

TV Inputs de type **signatureOrSystem** (et donc fournies par le constructeur de l'appareil) disposent d'un accès complet à la base et peuvent recevoir des événements. De même, seules les **TV Inputs** fournies par le système peuvent accéder au matériel, via la HAL. Inutile donc d'espérer créer un **TV Input** vous-même, pour accéder au tuner DVB-S et essayer de décrypter le module CAM sans avoir payé votre abonnement à votre fournisseur de contenu satellite. De même, si la base de données du **TV Provider** est commune à toutes les applications **TV Inputs**, cette dernière est segmentée pour que chaque entrée ne puisse accéder qu'aux données qui lui sont propres. Et s'il est possible à chacun de créer son propre **TV Input** (mais pas d'accéder au matériel, comme vu plus haut), il ne leur est possible d'afficher que leur propre contenu ou, à défaut, celui arrivant en direct (et non crypté) d'une entrée de type HDMI. Aussi lourdes et contraignantes que puissent paraître ces différentes sécurités mises en place, elles sont néanmoins nécessaires pour convaincre les différents équipementiers et surtout les diffuseurs de contenus de la robustesse de la plateforme Android TV. Elles sont donc garantes du succès de la plateforme.

4 TV App

Mais voyons plus en détail les différents composants du TIF, en commençant par la **TV App**. Comme vu précédemment, elle fait office d'interface entre l'utilisateur et le reste du framework et ce, aussi bien visuellement qu'en terme de contrôle. Elle est fournie par le système, signé en conséquence et ne peut être remplacée par l'utilisateur (sauf probablement sur un appareil « rooté »). Son rôle est de fournir aux différents **TV Inputs** (un à la fois cela dit), les différentes actions de l'utilisateur (appui sur une touche de la télécommande, contrôle du volume, passage d'une chaîne à l'autre ou encore d'une source à l'autre) au travers du **TV Input Manager**. Elle fournit également à l'utilisateur un moyen de rechercher des informations sur les chaînes et les programmes disponibles (informations stockées dans la base du **TV Provider**) et d'interagir avec (au travers du paquetage **com.android.tv.search.TvProviderSearch**). Ces informations peuvent être aussi simples que la liste des chaînes (permettant ainsi de basculer d'une chaîne à l'autre) ou des données associées au contenu actuellement en cours de diffusion (nom du film, durée, acteurs, synopsis ...). Notez qu'il est indispensable pour les constructeurs de fournir une **TV App** implémentant la fonctionnalité de recherche.

La **TV App** fournit les fonctionnalités suivantes : auto-détection des sources de type **TV Input**, mise en service d'une chaîne par un **TV Input**, support des paramètres de contrôle parental, de contrôle global de la TV, accès et navigation parmi les différentes chaînes TV, accès aux informations du programme en cours, support de multiples pistes audio et des sous-titres ... Qu'importe l'implémentation retenue par chaque constructeur et le look&feel graphique de l'application, ces fonctionnalités se doivent d'être présentes pour que l'application puisse être certifiée.

5 TV Provider

La base de données du **TV Provider** stocke les informations, chaînes et programmes des différentes sources de type **TV Input** et en gère les permissions d'accès, garantissant une isolation des données en provenance de chaque source. Le **TV Provider** a également la lourde tâche de classifier les informations des chaînes et ce, en canonisant de façon standard les codes de classification propre à chaque standard de diffusion. Typiquement, un utilisateur voudra peut être lister toutes les chaînes de sport. Au niveau du **TV Provider**, cela correspond au type canonique **android.provider.TvContract.Genres.SPORT**. Mais, dans le cas du standard de diffusion numérique américain ATSC A/65, ce type correspondra au code **0x25** du champ indiquant le genre, inclus dans les métadonnées de diffusion. Le **TV Provider** effectuera donc ce lourd travail d'association pour nous, mais surtout pour les développeurs, leurs évitant de ré-implémenter encore une fois la roue.

La base de données du TV Provider propose deux types de tables : une pour y stocker les chaînes (**android.provider.TvContract.Channels**) et une pour y stocker les programmes (**android.provider.TvContract.Programs**). Chacun des **TV Inputs** remplit ses tables avec les informations qui lui sont propres et l'application **TV App** est en mesure de les relire. Ces tables disposent d'un certain nombre de types de champs standards :

- **Display** : les champs de ce type contiennent des informations susceptibles d'être présentées à l'utilisateur, telles que le nom d'une chaîne, son numéro ou le titre du programme en cours de diffusion.
- **Metadata** : les champs de ce type contiennent des métadonnées internes utilisées pour l'identification du contenu, spécifique à chaque standard. Cela peut correspondre à l'identifiant du flux MPEG-TS, l'identifiant du réseau de diffusion ou encore celui du service. Rien de très utile pour l'utilisateur ici.
- **Internal Data**: les champs de ce type sont utilisés pour des paramètres spécifiques à chaque **TV Input**. Certains champs (e.g. **COLUMN_INTERNAL_PROVIDER_DATA**) contiennent des données binaires (ou BLOB) pouvant contenir n'importe quoi ou presque. Il peut s'agir d'informations très spécifiques comme la fréquence du tuner TV associée à une chaîne ou tout autre buffer écrit dans un protocole binaire propre au **TV Input**.
- **Flag** : les champs de ce type indiquent si une chaîne peut être recherchée, parcourue ou tout simplement visionnée. Ces champs n'ont pas de sens au niveau d'un programme et ces derniers héritent donc naturellement des champs de la chaîne dont ils dépendent. Ils prennent tout leurs sens en fonction des spécificités régionales (e.g. le diffuseur **Hulu** interdit toute diffusion ailleurs qu'aux États-Unis, sauf en passant par un proxy) ou des paramètres de contrôle parental (chaînes restreintes à l'accès par code PIN).

En terme de sécurité, chaque ligne de la base de données du **TV Provider** se voit associée une colonne **PACKAGE_NAME**, affichant le nom de l'application ayant écrit cette ligne. Ainsi, seule l'application **TV Input** correspondante pourra accéder aux informations qui lui sont propres.

6 TV Input Manager

Le **TV Input Manager** fait le lien entre l'application **TV App** et les différents **TV Inputs** par le biais de sessions de type **TVIM**. Cela permet à la **TV App** de lister les différentes sources TV Inputs disponibles ainsi que de vérifier leurs statuts (typiquement une **TV Input** proposant des flux IPTV sera désactivée si une connexion Internet n'est pas disponible). Chaque session **TVIM** permettra à la **TV App** d'être notifiée d'événements en provenance d'un **TV Input** (et réciproquement). Ceci se fait très simplement au sein de la **TV App**, par création et enregistrement d'une **TvInputCallback** auprès du **TV Input Manager**. A chaque changement d'état d'une **TV Input**, la **TV App** sera ainsi notifiée.

7 TV Input

Les **TV Inputs** constituent des applications Android tout ce qu'il y a de plus classique, disposant d'un fichier **AndroidManifest.xml** et pouvant être pré-installées par le constructeur ou être installées à posteriori par l'utilisateur via le **Play Store** ou « à la main » par installation manuelle d'un APK. Leur but est de fournir au framework TIF une nouvelle source de contenu TV. Selon leurs provenances, les **TV Input** seront signés ou non comme applications systèmes et pourront accéder à plus ou moins d'informations du **TV Provider** ou encore aux couches matérielles.

Vous retrouverez au sein du schéma 3 la forme la plus simple de source TV, à savoir celle utilisant le mode *passthrough* (ou direct), utilisé par une source HDMI.

Cette source **TV Input**, accédant au matériel (HDMI oblige) doit vous être fournie par le constructeur de l'appareil. Elle ne fournit cependant aucune information sur une quelconque chaîne ou programme TV, son unique but étant d'afficher le flux brut reçu sur l'interface HDMI. Il n'y a donc aucune réelle intelligence ni métadonnée associée. L'application se doit cependant d'être signée par le système car accédant à la **HAL TV-Input** ainsi qu'à la **HAL HDMI-CEC** de manière à pouvoir dialoguer avec le bus HDMI pour être contrôlée par la télécommande.

Un autre type simple, présenté via le schéma 4, est l'exemple d'un tuner intégré (typiquement DVB-T, également appelé TNT en France).

Comme précédemment, le module **TV Input** adéquat sera fourni par le constructeur de votre appareil, lui seul sachant comment accéder au pilote de

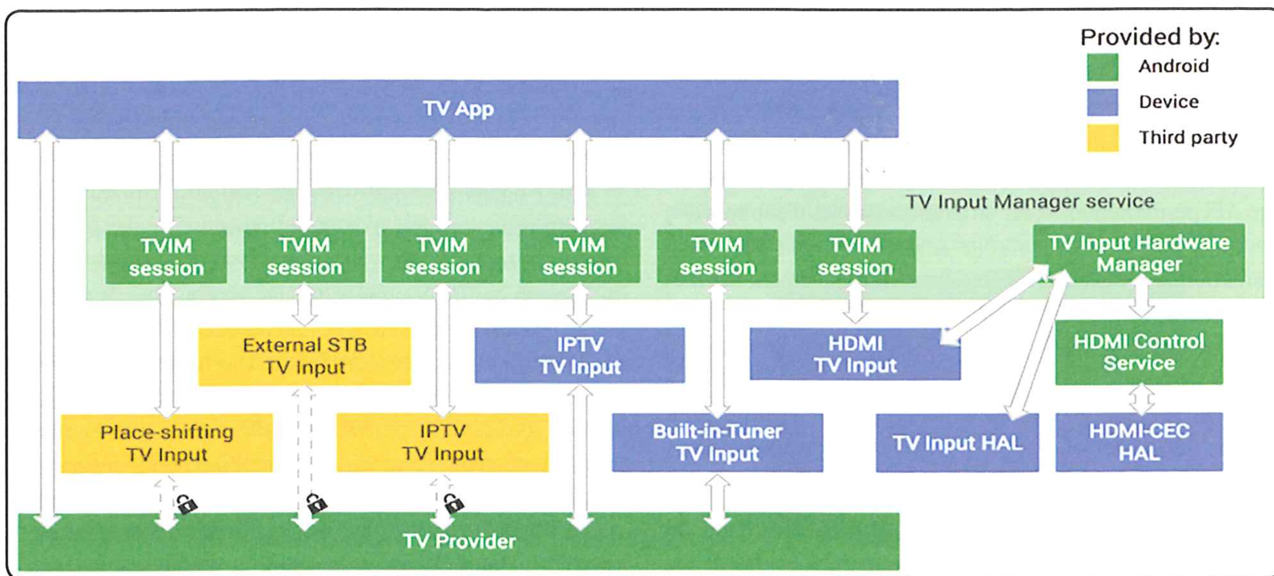


Fig. 2 : Architecture logicielle de l'Android TV Input Framework (TIF).

périphérique matériel du tuner, comment le configurer, lui associer une fréquence pour chaque chaîne TV etc ... et ceci au travers de la HAL qu'il aura pris le soin d'écrire.

Il existe ensuite bien d'autres possibilités pour les développeurs de créer leurs propres applications de type **TV Input** (par exemple pour accéder à du contenu Internet de type IPTV ou pour afficher le contenu d'une STB externe). Si ces dernières nécessitent l'accès à une ressource matérielle telle que l'HDMI, elle devront cependant y accéder au travers du **TV Input Manager**, ne disposant pas des privilèges suffisants.

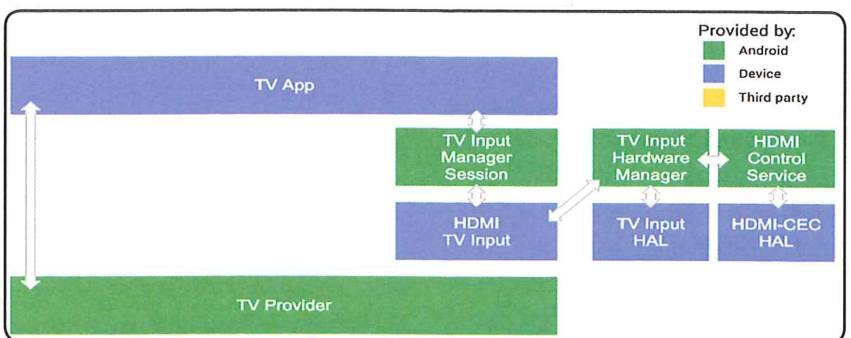


Fig. 3 : Architecture logicielle du TV Input HDMI Passthrough.

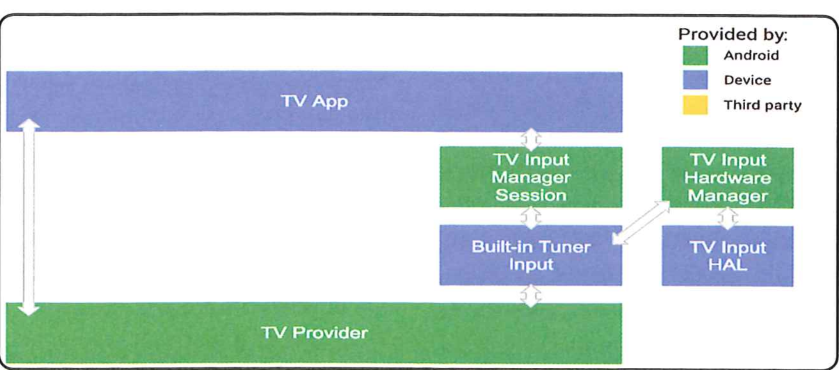


Fig. 4 : Architecture logicielle du TV Input Tuner.

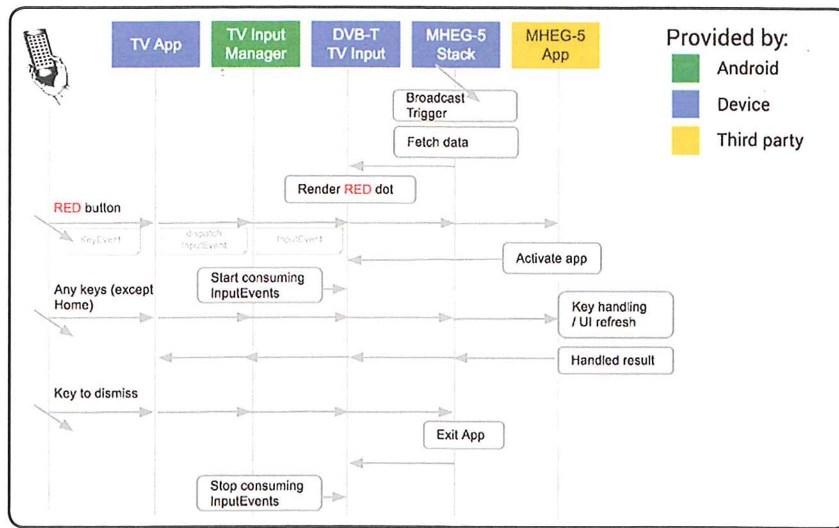


Fig. 5 : Interaction entre l'utilisateur et l'application TV Input.

Enfin, des applications complexes peuvent néanmoins être écrites. Il n'est en effet pas rare de retrouver des box capables d'afficher des applications propriétaires au dessus du flux vidéo sur demande de l'utilisateur, par appui sur une touche de la télécommande. Typiquement, cela permet l'affichage de métadonnées spécifiques au contenu que vous êtes en train de visualiser (par exemple une page Web avec le score ou les statistiques sportives en cours). Au niveau d'Android TV, cela se matérialise sous la forme présentée par la figure 5.

Dans cet exemple, c'est la **TV App** qui recevra l'appui de touche de la part de l'utilisateur qui propagera cet événement à l'application **TV Input** active au travers du **TV Input Manager**. L'application **TV Input** traitera l'événement de la manière qui lui est propre. Dans notre exemple, cela peut correspondre au démarrage d'une application propriétaire d'affichage des informations demandées. Toutes les interactions futures de la part de l'utilisateur (e.g. navigation via la télécommande) seront ensuite gérées par notre **TV Input**, qui les transférera à son application, et ainsi de suite, jusqu'à demande explicite de l'utilisateur d'arrêt du programme.

8 TV Input HAL

La **TV Input HAL**, comme toutes les autres HAL du système Android, permet aux développeurs d'accéder au matériel de manière abstraite, via une API unique, quel que soit le matériel exact utilisé. Dans les sources, cela se traduit par une nouvelle HAL, dont vous trouverez les entêtes au sein du fichier `hardware/libhardware/include/hardware/tv_input.h`. Très jeune (le premier commit date du 3 Mars 2014), le framework autorisera les différents constructeurs à créer leurs propres « box » compatibles « Android TV » et à y diffuser leur contenu en toute simplicité.

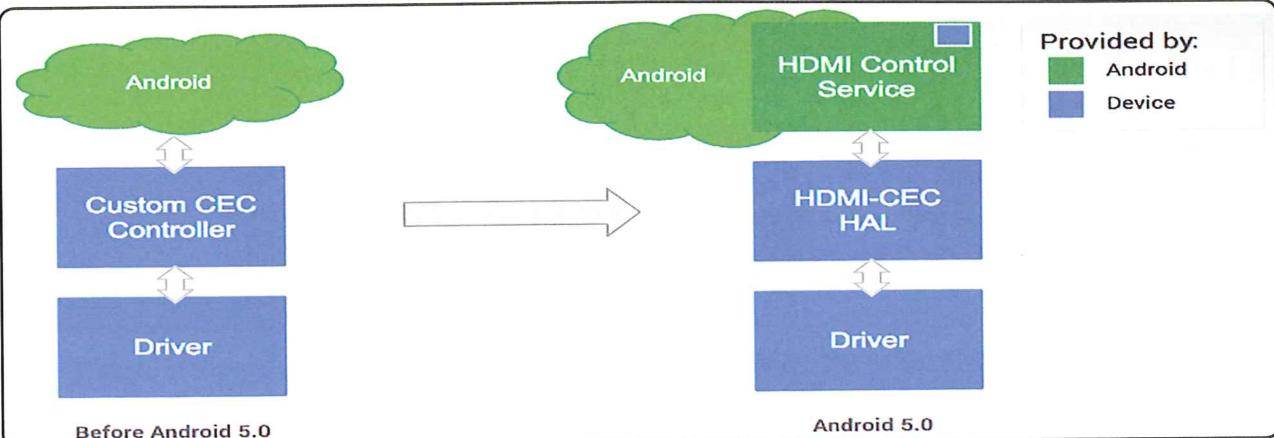


Fig. 6 : HDMI Control Service.

9 HDMI-CEC

Finissons notre découverte d'Android TV avec le support de la norme **HDMI-CEC**, pour « *Consumer Electronics Control* » ou contrôle d'appareils électroniques grand public. Cette norme permet à différents appareils compatibles avec la norme d'échanger des informations l'un avec l'autre. L'usage typique est de permettre d'utiliser la même télécommande pour piloter à la fois votre TV, votre STB, votre lecteur Blu-ray, votre ensemble Hi fi / audio etc ... et donc de vous simplifier la vie. Android TV se veut donc compatible avec cette norme. Mais comme toutes les normes, chaque constructeur y va de sa propre implémentation, qui respecte plus ou moins le standard, rendant deux produits compatibles **HDMI-CEC** pas nécessairement aussi compatibles que prévu. L'un des buts du framework TIF introduit avec Android 5.0 se veut ainsi de minimiser les problèmes de compatibilité entre équipements, via le service `UdmiControlService`, comme le montre la figure 6.

En faisant ceci, Android propose une implémentation standard de la norme **HDMI-CEC** que les différents constructeurs pourront utiliser au travers de la HAL adéquate. Le but est également de garantir une implémentation de référence rigoureusement testée et validée comme conforme auprès du plus grand nombre d'équipements numériques du marché. Enfin, notez que ce service est connecté à la fois à l'`AudioService` et au `PowerManager` et ce afin d'implémenter l'ensemble des fonctionnalités de la norme CEC (par exemple changer le volume ou encore mettre en veille le périphérique).

Pour implémenter le support CEC, il vous suffit d'adapter le support de la HAL CEC, dont vous trouverez les entêtes au sein du fichier `hardware/libhardware/include/hardware/hdmi_cec.h`. Les constructeurs d'appareils Android devront ainsi ajouter la directive suivante au sein du fichier `device.mk` afin d'autoriser le support **HDMI-CEC** :

```
PRODUCT_COPY_FILES += frameworks/native/data/etc/android.hardware.hdmi.cec.xml:system/etc/permissions/android.hardware.hdmi.cec.xml
```

Enfin, la propriété système `ro.hdmi.device_type` devra être positionnée pour permettre à l'`HdmiControlService` de fonctionner correctement, tout en différenciant si le système agit sous la forme d'un consommateur ou fournisseur de contenu. Typiquement, pour une STB (source de contenu), il vous faudra ajouter la directive suivante :

```
PRODUCT_PROPERTY_OVERRIDES += ro.hdmi.device_type=4
```

Pour une TV (consommatrice de contenu), il vous faudra ajouter la directive suivante :

```
PRODUCT_PROPERTY_OVERRIDES += ro.hdmi.device_type=0
```

Conclusion

Voici donc qui met fin à notre présentation de la plateforme Android TV, introduite avec Android 5.0 Lollipop. Comme nous venons de le voir, la plateforme gagne encore en maturité et en fonctionnalités et essaie de conquérir le marché déjà fermé des SetTopBox en fournissant aux industriels tous les moyens de s'y lancer à moindre frais. À très bientôt pour de nouvelles aventures « Androidesques » ! ■

Références

[1] B. Zores, « Inside Android : Lollipop 5.0 », GNU/Linux Magazine n°178, janvier 2015, p. 48 à 53.

[2] Nexus Player : http://www.google.fr/nexus/player/

[3] Apple TV : https://www.apple.com/fr/appletv/

[4] Kodi Media Center : http://kodi.tv

JBOSS FORGE2, JAVA EE FACILE, TRÈS FACILE

par **Jérôme Baton** *[Arrière-arrière petit-fils de maréchal-ferrant]*

Le framework JBoss Forge offre de puissantes possibilités dans le domaine de la création d'applications Java EE et son extensibilité est un atout pour chaque développeur Java.

Forge est un outil et un framework pour le développement rapide d'applications Java EE basées sur Maven. Il ne s'agit pas d'une nouveauté de l'éditeur de WildFly : la v1.0 du projet date de février 2012 mais aujourd'hui, avec la branche 2.x, Forge a acquis une maturité qui le rend encore plus intéressant pour tous les développeurs Java, qu'ils soient débutants ou experts, pressés ou bien bidouilleurs.

Si créer des applications est son rôle premier, JBoss Forge, de par sa construction modulaire, est très ouvert à l'imagination. Nous verrons d'abord la productivité exceptionnelle que ce framework offre, avant de le découvrir plus profondément et de jouer un peu avec. Parce que l'on aime cela, telle est notre joie.

1 Pré-requis

L'unique pré-requis technique pour utiliser Forge est d'avoir un JDK de type Java 7 ou supérieur. Comprenez bien que c'est suffisant pour utiliser Forge dans un but de découverte mais pas pour exécuter l'application Java EE que nous allons créer. Pour suivre le déroulement de cet article, des connaissances en Java EE, Maven, serveurs d'applications et base de données sont préférables.

L'application Java EE que nous allons créer nécessitera un serveur d'applications pour son exécution et une base de données pour conserver en base les informations portées par les entités JPA (*Java Persistence API*) utilisées. Par conséquent, il vous faudra une base de données et un serveur d'applications accessibles à votre projet pour tester le résultat, soit installer l'environnement intégré. Ceci n'est pas obligatoire et l'installation d'une base de données et d'un serveur d'applications est en dehors du cadre de cet article.

2 Installation

Le premier pas de ce voyage initiatique dans les territoires de la génération de code est l'installation du logiciel. Trois types d'installations sont proposés :

- Télécharger JBoss Developper Studio, une version d'Eclipse version Java EE augmentée des JBoss Tools, packagée par Red Hat JBoss ;
- Télécharger les JBoss Tools pour les installer dans un Eclipse pré-existant ;
- Télécharger une version utilisable en ligne de commande.

La suite de cet article utilisera d'abord la version en ligne de commande puis JBoss Developper Studio. Pour télécharger ces deux versions, rendez-vous sur http://forge.jboss.org/download.

Si vous n'avez pas encore de serveur d'applications et de base de données installés sur votre machine, ou simplement pas l'envie de les configurer, je vous conseille d'utiliser JBoss Developper Studio car il est accompagné de WildFly, un serveur full Java EE et de H2, une base de données simple et rapide, adaptée à ce que nous allons faire. Ainsi, environnement de code, serveur et base seront déjà configurés... c'est toujours 2 jours-homme de gagnés :)

Pour l'Installation de la version en ligne de commandes, décompressez l'archive dans un répertoire de votre choix et ouvrez un shell pour déclarer ce répertoire comme valeur de la variable d'environnement `FORGE_HOME` en éditant le fichier `.profile` ou `.bashrc` de votre répertoire personnel. Ajoutez `$FORGE_HOME/bin` à votre variable `PATH`, puis exécutez :

```
$ source nomDuFichierModifié
```

Sur Ubuntu et ses variantes, cela correspond aux commandes suivantes si vous utilisez le bash :

```
$ nano .bashrc
$ source .bashrc
```

Voilà, cela en est fini pour l'installation, votre base de registre n'est pas polluée de nouvelles clefs inutiles :)

3 Et maintenant, créons une application Java EE

Maintenant, positionnez-vous dans le répertoire où vous souhaitez créer votre projet Java EE le plus rapidement de toute votre vie de codeur jusque là.

Dans votre terminal, tapez

```
$ forge
```

Un nouveau prompt s'affiche, encore plus puissant que Bash. Et avec lui aussi, la touche de tabulation **<Tab>** est votre fidèle second.

```
jerome@bosphore ~ $ forge

    _____
   |  ___|__  _ __ __ _  ___
   | |_ / _ \| `__/ _` |/ _ \  \\
   |  _| (_) | | | (_| |  __/  //
   |_|  \___/|_|  \__, |\___|
                   |___/

JBoss Forge, version [ 2.9.2.Final ] - JBoss, by Red Hat, Inc.
[ http://forge.jboss.org ]

[jerome]$
```

Parce que, comme on peut le lire et l'entendre dire à longueur de temps, c'est très long et très très lourd de faire du Java EE, nous allons donc voir le contraire dans le but de vous faire apprécier la puissance de cet outil. Les heureux lecteurs présents lors de l'événement Devoxx France 2014 auront déjà pu voir le Java Champion Antonio Goncalves en faire la démonstration avec son calme légendaire.

Dans le prompt de Forge, tapez ce que vous êtes : un pro !

```
[jerome]$ pro
```

Puis appuyez sur **<Tab>** et la commande se complète en :

```
[jerome]$ project-new
```

Inattendu. Forge vous fait revivre ce moment de bonheur où vous êtes passés du prompt DOS à un vrai shell. Appuyez à nouveau sur **<Tab>** et ce sont les paramètres de la commande qui vous sont proposés. Magique.

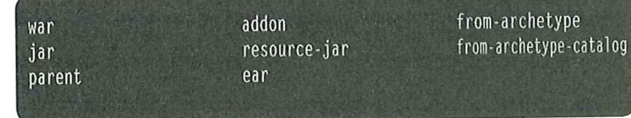
```
[jerome]$ project-new
--named              --version        --targetLocation   --buildSystem
--topLevelPackage    --finalName      --type
```

Dans votre shell, sauf à utiliser un moniteur monochrome parce que vous poussez la « hipsteritude » un peu loin, le paramètre `--named` sera en couleur pour indiquer qu'il est obligatoire. Il apparaît ci-dessus en gras pour le démarquer. Comme il se doit, les autres paramètres prendront les valeurs par défaut si vous ne les renseignez pas.

Saisissez ensuite :

```
[jerome]$ project-new --named mesNews --topLevelPackage org.moi --type
```

Appuyez sur **<Tab>**, et ce sont les valeurs possibles pour le type qui s'affichent :



Formidable ! Pas besoin de trouver la documentation en ligne. J'adore. Vous aussi, j'en suis certain. Prenons-en de la graine pour nos projets.

Choisissons le type `war` pour continuer notre guerre aux idées reçues. La commande finale est :

```
[jerome]$ project-new --named mesNews --topLevelPackage org.moi
--type war --finalName newsAngularJs
***SUCCESS*** Project named 'mesNews' has been created.
```

Petit aparté, maintenant qu'un projet est créé, pour une commande à essayer, qui porte bien son nom : **`command-list`**. Je vous invite à jeter un œil sur la sortie écran qu'elle génère avant de passer aux prochaines commandes. Ne l'oubliez pas, elle vous servira ultérieurement à vérifier la disponibilité de commandes supplémentaires que vous aurez installées. Aparté dans l'aparté, les commandes de Forge visibles à un instant t dépendent du contexte (ici, il faut qu'un projet soit créé). Surprenant, on peut filtrer avec **`more`** :

```
[jerome]$ command-list | more
```

Maintenant, faites exécuter les commandes suivantes une à une :

```
[jerome]$ jpa-setup --provider Hibernate --container JBOSS_EAP6
[jerome]$ jpa-new-entity --named News --targetPackage org.moi.model --idStrategy AUTO
[jerome]$ jpa-new-field --named titre --type String
[jerome]$ jpa-new-field --named texte --type String
[jerome]$ jpa-new-field --named quand --type java.util.Date
```

Les paramètres de la commande **`jpa-setup`** sont optionnels, par défaut Forge créera une base H2 en mémoire. Ces paramètres sont assurés de fonctionner pour un déploiement vers WildFly.

En quelques lignes de commandes, vous avez généré la classe suivante :

```
package org.moi.model;

import javax.persistence.Entity;
import java.io.Serializable;
import javax.persistence.Id;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Column;
import javax.persistence.Version;
import java.lang.Override;
import java.util.Date;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.xml.bind.annotation.XmlRootElement;

@Entity
@XmlRootElement
public class News implements Serializable
{

  @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   @Column(name = "id", updatable = false, nullable = false)
   private Long id;
   @Version
   @Column(name = "version")
   private int version;
   @Column
   private String titre;

   @Column
   private String texte;

   @Column
   @Temporal(TemporalType.DATE)
   private Date quand;

   public Long getId()   {
      return this.id;
   }

   public void setId(final Long id)   {
      this.id = id;
   }

   public int getVersion()   {
      return this.version;
   }

   public void setVersion(final int version)   {
      this.version = version;
   }

  @Override
   public boolean equals(Object obj)   {
      if (this == obj)
      {
         return true;
      }
      if (!(obj instanceof News))
      {
         return false;
      }
      News other = (News) obj;
      if (id != null)
      {
         if (!id.equals(other.id))
         {
           return false;
         }
      }
      return true;
   }

   @Override
   public int hashCode()   {
      final int prime = 31;
      int result = 1;
      result = prime * result + ((id == null) ? 0 :
 id.hashCode());
      return result;
   }

   @Override
   public String toString()   {
      String result = getClass().getSimpleName() + " ";
      if (id != null)
         result += "id: " + id;
      result += ", version: " + version;
      if (titre != null && !titre.trim().isEmpty())
         result += ", titre: " + titre;
      if (texte != null && !texte.trim().isEmpty())
         result += ", texte: " + texte;
      if (quand != null)
         result += ", quand: " + quand;
      return result;
   }

   // Getters et setters supprimés pour un minimum de concision
}
```

Excellent. Un peu verbeux.
Les méthodes **`hashCode`** et **`toString`** sont générées aussi. Bien.

Je vous invite à jeter un coup d'œil au fichier **`pom.xml`** également.

Maintenant, nous allons faire générer une interface Web pour nos news, par la technique du scaffolding, et pour céder à la mode, elle sera basée sur **AngularJS**.

Note

Le *scaffolding* consiste en de la génération automatique de code. Le logiciel utilise les informations à sa disposition pour créer une application de type CRUD basique. Généralement, c'est une base de données existante qui sert .... de base :)

Ce n'est pas dans les capacités initiales de Forge mais grâce à son extensibilité et au travail de la communauté, il existe un addon de *scaffolding* AngularJS. Pour l'installer directement à partir de Github, saisissez cette commande :

```
[jerome]$ addon-install-from-git --url https://github.com/forge/
angularjs-addon.git --coordinate org.jboss.forge.addon:angularjs
```

Bienvenue dans un monde nouveau ! Cette installation va prendre un certain temps, par un effet de pelote typique de Maven, qui va remplir un peu plus votre dépôt **`M2`**.

Coupez le chrono et allez prendre une pause ; heureusement, que cette installation n'est à faire qu'une seule fois...

Passons maintenant à la génération de l'interface utilisateur à proprement parler :

```
[jerome]$ scaffold-setup --provider AngularJS
[jerome]$ scaffold-generate --provider AngularJS  --targets org.
moi.model.News
```

Puis, on lance la construction avec, je vous le donne en mille, Émile :

```
[jerome]$ build
```

L'application est générée et est prête à être déployée. Le **`war`** se trouve dans le répertoire **`target`** sous le nom **`newsAngularJs.war`**, ce qui est la valeur du paramètre **`finalName`** donnée lors de la création du projet. Rien ne se perd. Allez jeter un œil dans le répertoire du projet pour satisfaire votre curiosité.

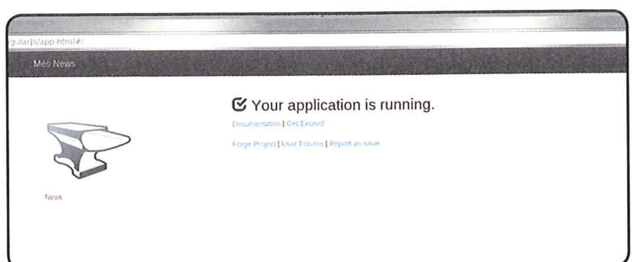


Fig. 1 : Page d'accueil de l'application générée

Fig. 2 : Page de création d'une nouvelle

L'artefact déployable généré est basé sur des fichiers source qui ont été générés aussi et constituent donc un projet de type Maven. Vous pouvez importer le projet dans Eclipse/JBoss Developper Studio par ***Import > Existing Maven Project*** pour effectuer toute modification ou ajout que vous jugerez nécessaire. Et puis, bien entendu, vous pouvez l'ajouter à votre serveur d'application Java EE pour l'exécuter

L'application que vous venez de générer ressemble à celle présentée en figure 1.

En cliquant sur « News », sous l'enclume, on arrive à un écran d'administration visible en figure 2.

Impressionnant, mais il faut admettre que l'on ne peut pas mettre cela en ligne sans quelques retouches pour rebrander le résultat, comme mettre son logo et enlever quelques liens. Certes, mais cela a été généré en quelques minutes. Et un pro du front pourra embellir ce premier jet, un peu brut de fonderie.

Si vous avez eu le souffle coupé, reprenez votre respiration avant de devenir bleu et essayez d'estimer le temps qu'il vous serait nécessaire pour créer l'équivalent. Fascinant outil à mon humble avis.

Si vous déployez cette archive et que la création d'une news ne fonctionne pas, pas d'inquiétude, c'est normal. Utilisez le scaffolding JSF, redéployez et cela marchera. L'application AngularJS utilise des points d'entrée REST que nous n'avons pas encore créés.

3.1 Bonus : point d'accès REST

Pour générer un point d'accès REST à nos news, les commandes sont :

```
[jerome]$ rest-setup --applicationPath rest --targetPackage org.
moi.rest --config APP_CLASS
[jerome]$ rest-generate-endpoints-from-entities --targets org.
moi.model.News  --persistenceUnit mesNews-persistence-unit
--generator JPA_ENTITY
```

Et là, Forge a créé le paquet **org.moi.rest**, dans lequel se trouvent deux classes. La première est la classique **RestApplication** qui permet de définir le point d'entrée de l'interface REST de toute l'application :

```
@ApplicationPath("rest")
public class RestApplication extends Application { }
```

Et la deuxième classe, **NewsEndpoint** est le point d'entrée vers nos news.

Voici une version expurgée du code source, sans aucun corps de méthode, qui montre le respect du paradigme REST, avec l'utilisation conforme des méthodes HTTP GET, POST, PUT et DELETE pour la manipulation de cette entité :

```
@Stateless
@Path("/news")
public class NewsEndpoint
{
   @PersistenceContext(unitName = "mesNews-persistence-unit")
   private EntityManager em;

   @POST
   @Consumes("application/json")
   public Response create(News entity)

   @DELETE
   @Path("/{id:[0-9][0-9]*}")
   public Response deleteById(@PathParam("id") Long id)

   @GET
   @Path("/{id:[0-9][0-9]*}")
   @Produces("application/json")
   public Response findById(@PathParam("id") Long id)

   @GET
   @Produces("application/json")
   public List<News> listAll(@QueryParam("start") Integer
startPosition, @QueryParam("max") Integer maxResult)

   @PUT
   @Path("/{id:[0-9][0-9]*}")
   @Consumes("application/json")
   public Response update(News entity)
}
```

Comme vous le voyez, ce code est très classique et vous auriez certainement écrit au moins la même chose. Il faut voir ce code comme une base de travail à laquelle il est possible d'ajouter ce que vous souhaitez comme des annotations pour définir les autorisations par exemple.

La méthode **listAll**, qui retournera au format JSON la liste (paramétrée) de toutes les news pourra être accessible par votre navigateur préféré à l'adresse suivante :

Note

D'ailleurs, il est probable que vous obteniez des erreurs en voulant créer des news (au moment du clic sur le bouton ***Save***). Pas d'affolement, c'est normal, ne rejetez pas la forge, c'est uniquement de la configuration à revoir en fonction de ce sur quoi vous l'avez déployé. Votre environnement ne possède peut-être pas de *datasource* au nom par défaut utilisé par Forge. Et, sans points d'entrée REST, l'application en AngularJS, qui s'exécute dans votre navigateur, n'a aucune communication possible avec le serveur. C'est quand cela ne fonctionne pas que toutes vos compétences Java EE s'avèrent nécessaires.

http://localhost:8080/newsAngularJs/rest/news?start=0&max=1610. Mais à priori, et à fortiori aussi, cela ne renverra rien si vous n'avez pas joué à créer des entrées avec l'interface utilisateur Web.

Voilà pour l'utilisation de Forge en mode ligne de commande.

Les téléchargements divers de la première utilisation ayant (fortement) ralenti le processus de création de l'application, je vous propose de recommencer un nouveau projet afin d'apprécier pleinement la rapidité de cet outil. Pourquoi ne pas créer un projet de blagues, avec un texte, un auteur, une date, un booléen pour indiquer si elle est destinée aux adultes,... et les informations qui vous semblent pertinentes. Nous générerons ensuite une version JSF (*scaffolding Faces*) et une version AngularJS.

3.2 Scripting

Il est possible de faire exécuter un script de commandes Forge à la console Forge. Pour cela, utilisez la commande **run**, suivie du nom du fichier.

Pour obtenir la liste des commandes exécutées et en faire un script, tournez-vous vers le fichier **~/.forge/history**.

Mais Forge, ce n'est pas seulement un prompt puissant.

3.3 Texte, GUI, faites votre choix !

Vous vous souvenez de la promesse de Java, « Écrire une fois, exécuter partout » : écrire un code qui sera exécuté identiquement sur différentes plate-formes matérielles. C'est une des forces de Java et de son moteur d'exécution (runtime) : la machine virtuelle Java (JVM) et hormis des cas limites, cela est vrai.

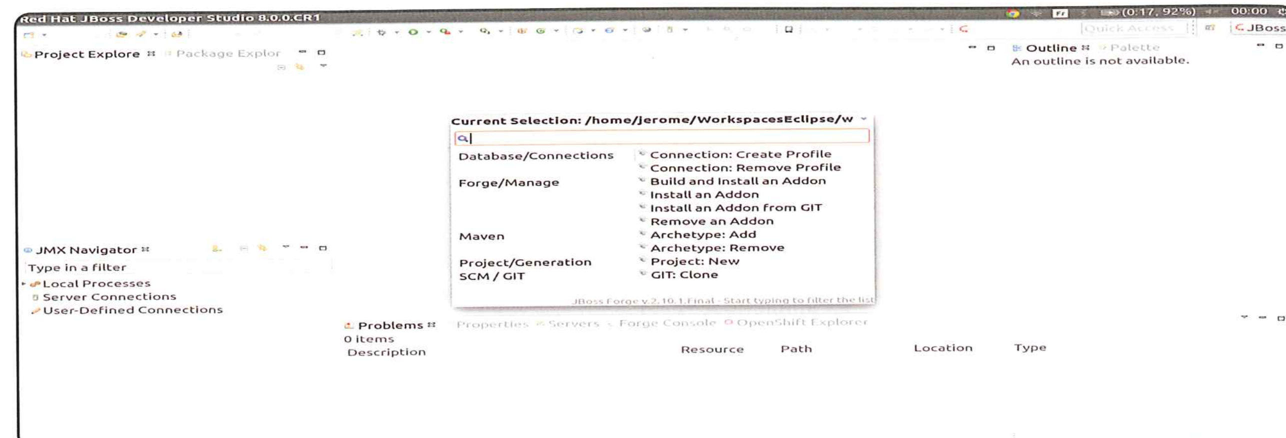


Fig. 3 : La fenêtre Forge

Dans le même esprit que Java, Forge possède deux plate-formes d'exécution pour ses plugins (*addons*), l'une est textuelle comme nous venons de le voir, l'autre est graphique, fondue dans Eclipse. Faites votre choix, vous ne serez pas pris entre le marteau et l'enclume. La première façon d'utiliser Forge de façon graphique est d'ajouter les JBoss Tools à son IDE. La deuxième façon est d'utiliser JBoss Developper Studio.

4 JBoss Developper Studio (JDS)

Note

À l'heure de l'écriture de cet article, la version 7 est la version stable mais elle intègre Forge 1.x, aussi préférez une version 8 (en bêta à cette date) pour bénéficier de Forge 2.x

Fussiez-vous adorateur d'Emacs ou apôtre d'IntelliJ, cette mouture d'Eclipse va vous étonner.

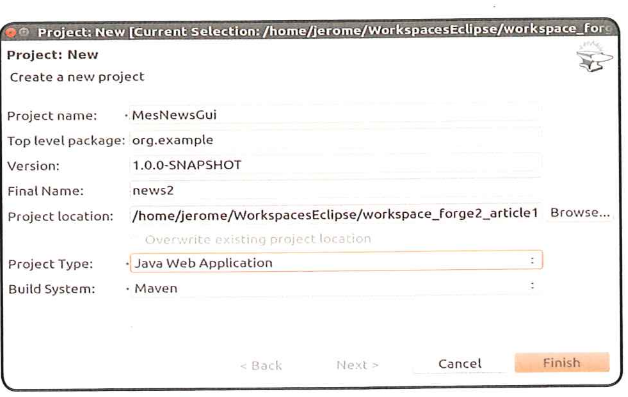


Fig. 4 : Création du projet

Pour la comparaison, nous allons recréer l'application **MesNews** de façon graphique. Une fois JDS lancé, appuyez sur la combinaison de touches **<Ctrl> + <4>** (enfin, **<Ctrl> + <Shift> + <'>**).

Vous devriez voir s'afficher la fenêtre présentée en figure 3 qui est le *wizard* (assistant) de Forge.

Note

La fenêtre Forge n'est pas la console Forge (dans Eclipse) qui n'est pas Forge en mode console (shell).

Si ce n'est pas le cas, c'est que le raccourci clavier a encore changé (j'ai expérimenté le fait qu'il diffère entre deux versions sur deux OS différents). Dans ce cas, il faut aller à la pêche à l'information : allez dans le menu ***Window/Preferences*** puis dans le choix ***General/Keys***. Dans le champ ***Filter***, saisissez **forge**, puis cliquez sur la ligne ***Run a Forge Command*** dont la colonne **Binding**

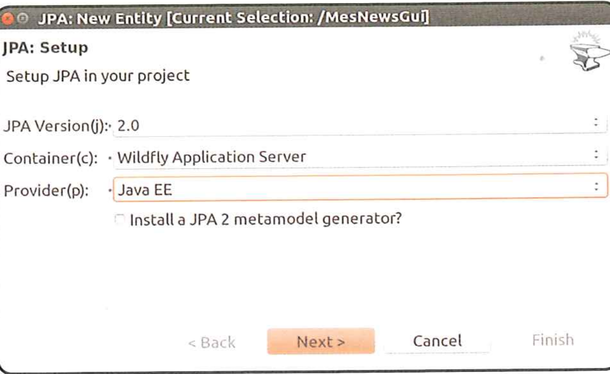


Fig. 5 : Configuration de JPA

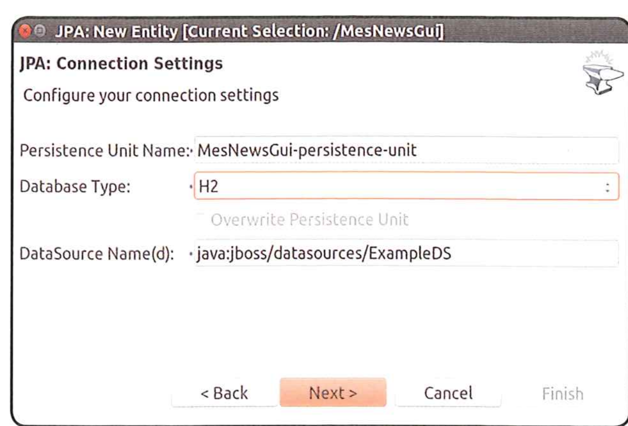


Fig. 6 : Configuration de JPA

Fig. 7 : Création d'un champ de l'entité JPA

vous indiquera la combinaison de touches à réaliser. Si cette combinaison n'a aucune valeur, à vous de la définir selon votre envie et les disponibilités.

Cette fenêtre est votre interface avec Forge au sein d'Eclipse. Elle affiche l'équivalent de la commande `command-list` disponible en ligne de commandes. Il faut cliquer sur les commandes pour les exécuter : simple.

Voilà pour la configuration, reprenons la création de l'application. Dans la zone de filtrage (avec la loupe), tapez `new` puis cliquez sur ***New Project***. Vous allez voir apparaître la fenêtre présentée en figure 4 qu'il vous faudra compléter.

Relancez la fenêtre de commande de Forge (`<Ctrl> + <Shift> + <'>`) et, dans le filtre, saisissez `entity` puis cliquez sur ***JPA:New Entity*** et choisissez WildFly et Java EE comme le montre la figure 5.

Cliquez ensuite sur ***Next*** et laissez les valeurs par défaut dans la fenêtre suivante (voir figure 6). Jetez quand même un œil à Database Type pour voir l'ensemble des bases supportées. Gardez le choix par défaut (H2) et cliquez sur ***Next***.

Laissez les valeurs par défaut. Notamment le nom de la source de données. Mais jetez un œil à Database Type pour voir l'ensemble des bases supportées. Gardez le choix du presque voyageur galactique. Cliquez sur Next.

Dans la fenêtre suivante reprenons ensuite le nom `News` pour la classe puis cliquez sur ***Finish*** pour créer l'entité. Il faut désormais lui ajouter des champs.

Dans le filtre de la fenêtre de commande, saisissez `field` pour afficher le choix ***JPA : New Field*** (voir figure 7) et ajoutez le champ `texte`.

Note

Si comme moi, vous trouvez que c'est beaucoup trop de clics pour créer un bête champ, sachez qu'il est possible d'utiliser la « Forge Console », un shell Forge dans JDS. MAIS, et c'est un gros « mais », cette console n'est pas synchronisée sur l'éditeur actif d'Eclipse. Ils sont indépendants l'un de l'autre. Aussi, il vous faudra « cd » jusque dans la classe `Entity` voulue pour exécuter une commande `jpa-new-field`.

Faites de même pour le champ `titre`, puis pour le champ `quand` il faudra utiliser le type `java.util.Date`.

Pour le *scaffolding*, à ré-exécuter à chaque changement devant être visible dans l'interface, c'est la commande ***Scaffold : Generate***. Pour le build, sans surprise, c'est aussi la commande ***build***.

Et là, c'est le drame... Nous venons de créer une application géniale mais comment la déployer ? Au début de l'article, j'écrivais que JDS offrait un environnement complet, mais en fait il n'y a pas de serveur configuré dans la nouvelle version que j'utilise. J'espère que la vôtre en dispose. Pour le cas où votre version serait également sans aucun serveur pré-configuré, voyons comment ajouter un serveur WildFly 8.

Si l'onglet ***Servers*** de votre installation n'est pas vide, chanceux que vous êtes, vous pouvez passer à la section suivante. Sinon, cliquez sur le lien indiqué dans la fenêtre (voir figure 8).

Vous devrez ensuite définir un nouveau serveur (sélectionnez *WildFly 8.x*), configurer l'adaptateur pour WildFly (serveur *Local* contrôlé par *Filesystem and shell operation*), choisissez le téléchargement et l'installation de la version la plus élevée de WildFly, comme d'habitude, faites comme si vous aviez lu la licence et acceptez-la puis lancez l'installation en choisissant un chemin clair pour vous y retrouver ultérieurement (une longue attente se dessine à l'horizon). Enfin vous pourrez cliquer sur ***Finish*** !

Il faut désormais ajouter l'application au serveur nouvellement créé. Dans l'onglet ***Servers***, faites un clic droit et sélectionnez ***Add and remove***. Vous pourrez alors faire basculer votre application de la liste *Available* vers *Configured* en cliquant sur le bouton ***Add***, puis cliquez sur Finish (voir figure 9).

Maintenant démarrez le serveur (clic droit sur le serveur puis ***Start***) et naviguez à l'URL correspondante : http://localhost:8080/MesNewsGui/.

5 Et il y en a encore

Je ne peux pas finir cet article sans évoquer les collections. Nous allons ajouter un auteur pour plusieurs news par exemple. À cet auteur, donnons lui un nom et un prénom (vous savez comment faire). Ensuite il faut lier la classe `Auteur` et la classe `News`. Dans la classe `Auteur`, ajoutez :

```
@OneToMany(mappedBy = "auteur", cascade = CascadeType.ALL, fetch
= FetchType.LAZY)
  private Set<News> news = new HashSet<News>();
```

Il faut ensuite ajouter dans la classe `News` :

```
@ManyToOne(cascade = CascadeType.ALL, fetch=FetchType.EAGER)
@JoinColumn(name="auteur")
  private Auteur auteur;
```

Puis, pour chacun de ces deux fichiers, faites générer les *getters* et les *setters* correspondants. Relancez le *scaffolding*, le *build*. Vous obtiendrez un meilleur résultat avec le *scaffolding* JSF (Faces) qu'avec celui produisant l'application AngularJS. Cette dernière étant assez avare d'informations en n'affichant que l'identifiant pour choisir l'auteur de la News dans une liste déroulante. Essayez les deux et comparez les.

Note

Vous remarquerez que le produit des deux *scaffolding* est disponible dans l'application. Le dernier utilisé est celui actif mais les artefacts produits par l'autre *scaffolder* restent présents dans le projet.

6 Et maintenant ?

Et maintenant que vous êtes abasourdi par cet outil, vous vous demandez ce que vous allez pouvoir faire de toute la puissance qui vous est offerte.

Parce que le quotidien de la majorité de nous autres, développeurs Java, est plutôt fait de maintenance (quelqu'un d'autre fait aussi de la maintenance sur des projets Struts 1.x ?) que de démarrage de projets avec les dernières nouvelles technos, à quoi peut bien servir Forge au quotidien ?

Mis à part prototyper une application comme nous venons de le faire, Forge peut aussi être utilisé pour

- faire générer dans un projet de brouillon un bout de code que l'on a la flemme d'écrire pour son projet parce qu'on l'a déjà écrit vingt fois avec dix frameworks différents ;
- démarrer sa *startup*, en allant chercher des capitaux avec un prototype à montrer aux investisseurs ;
- créer ses outils ;
- créer ses usages.

6.1 Démarrer sa startup ?

Vous avez une idée à même de faire des millions de bénéfice ? Bien, avec Forge et sa productivité, vous pouvez créer et recréer à l'envie le backend pour votre startup.

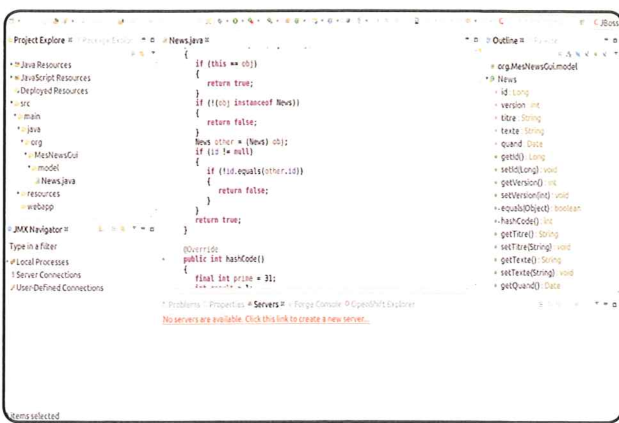


Fig. 8 : L'onglet Servers

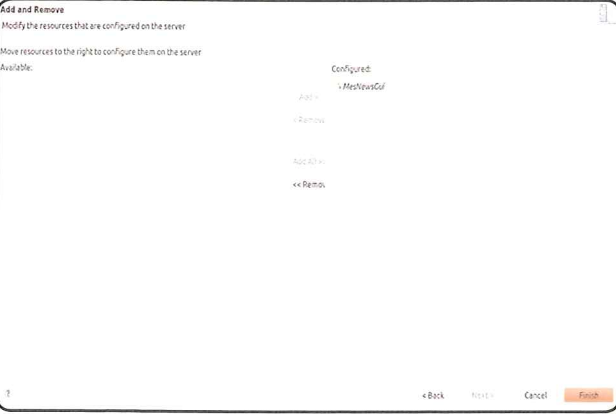


Fig. 9 : Publication de l'application

Note

Petit conseil : trouvez-vous un(e) ami(e) designer pour améliorer le design par défaut.

Créez les *beans entities* qui portent le métier de votre site, modifiez-les, recréez-les puis faites (re)générer les API REST (attention au *versioning*). Un peu de *scaffolding* comme vu précédemment, et vous avez un site Web à montrer à des investisseurs, ou à votre responsable (dans le cas « prototyper une application »).

Testez et déployez. Vous pouvez utiliser JDS pour déployer votre application vers le *cloud* de Red Hat : OpenShift. Naturellement, cela est intégré à l'outil. Le déploiement vers d'autres nuages est tout à fait possible également mais il faudra prévoir du temps de re-configuration.

6.2 Créer ses outils ?

Oui, créer ses outils. Wow. Rien que cela. Vaste programme. Parce que nous, programmeurs, sommes parfois fainéants (surtout les jours en -di) et que nous n'aimons pas changer de fenêtre à répétition, il existe pléthore d'extensions pour Eclipse. De l'alarme à thé au modeleur UML en passant par le prompt shell, ces extensions couvrent un large pourcentage des besoins de la majorité des développeurs.

Seulement voilà, il reste un pourcentage de tâches qui sont locales à votre environnement de projet, votre projet. Et un vrai développeur n'aime pas les tâches répétitives puisque c'est pour les réaliser que l'on a inventé les ordinateurs. Et chacun, dans nos bureaux, nous avons des choses à faire qui sont « locales » (sans pour autant impliquer de l'UTF-8). Et pour ces tâches, on a tous nos astuces. Que cela soit une feuille de tableur qui permet de générer du SQL pour initialiser sa base, un script shell qui lance une classe Java qui va *scraper* un site pour augmenter un fichier des fortunes ou une base de données, ou même une page AngularJS qui affiche le code Java pour des classes et un DAO complet à partir du nom d'une procédure stockée (juré je l'ai fait), on a tous nos trucs pour nous aider dans notre boulot quotidien, celui qui est *closed-source*, se fait derrière les portes de nos clients et qui paye les factures.

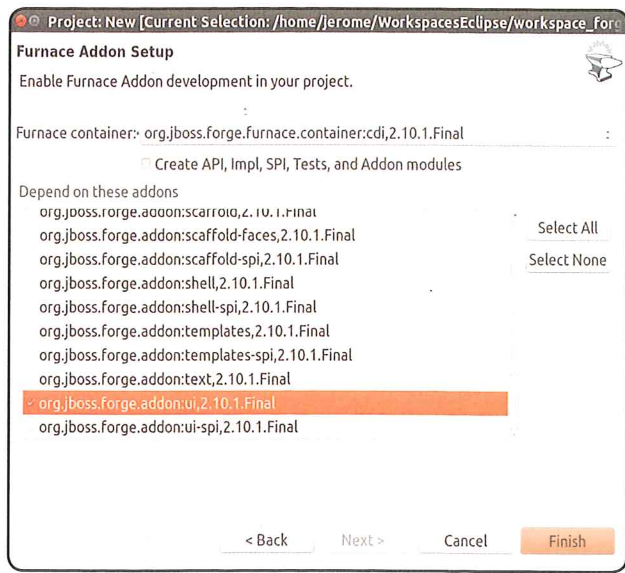


Fig. 10 : Choix des addons utilisés par l'addon

Seulement, ces exemples peuvent ressembler à du bricolage et interpeller quant à leurs maintenabilités les gens de peu de foi, un peu plus haut dans la hiérarchie. Alors, un meilleur type de solution serait de créer un plugin pour Eclipse et de le mettre sur le gestionnaire de sources de la boite. Seulement voilà, les plugins Eclipse sont (très) loin d'être simples à développer pour le peu de ce que j'ai pu en voir avant de fuir vite et loin. Tant le démarrage d'un projet d'application RCP que l'ordonnancement de ses menus dans l'ordre désiré m'ont semblé dingues de complexité, et par conséquent, cette plate-forme très perfectible.

Alors Forge met à votre disposition un cadre (framework) qui permet de créer des actions personnalisées, utilisables sans quitter l'IDE. Ce qui est génial si votre client, comme cela m'est souvent arrivé, vous met à disposition une machine sous-dimensionnée avec un OS lourd et propriétaire plombé par un antivirus troué et sous-doté en RAM. Ainsi vous évitez un changement de contexte, des accès disque, etc...

Avec ces actions personnalisées, vous pourrez programmer ce que vous seul utilisez, que cela soit propre à votre projet ou à votre entreprise. Et comme ces actions (plugins) s'écrivent en Java, c'est dire que vous disposez de nombreuses API pour effectuer toute tâche imaginable. Bien entendu, vous pouvez aussi développer des *addons* utiles à tous et les partager avec vos pairs. L'open source, il ne faut pas seulement en profiter.

6.3 Philosophie

Forge se base réellement sur son mécanisme d'addons et de ce fait, les commandes disponibles après l'installation sont elles-mêmes des addons. Je rappelle aux plus curieux que les sources sont disponibles ;) Il est aussi possible d'utiliser ces addons dans nos addons. Inception. Collaboration.

7 Le premier addon

Parce que c'est en forgeant que l'on devient forgeron, pour ce premier addon, souscrivons à la tradition du Hello World et écrivons un « Bonjour Dennis Ritchie. Merci pour les jouets que tu as offert au monde. Entre le C, Unix et ta barbe, tu étais le père noël des geeks ! », pour honorer ce géant, pas assez idolâtré à mon avis, sur les épaules duquel nous sommes tous assis.

Note

Comme le sujet est vaste, la création d'un addon sera survolée avec un exemple simple qui n'a pour but que vous donner envie de lire le prochain article.

Après avoir fermé tous les projets ouverts dans JDS, affichez la fenêtre Forge et choisissez ***Project : New***. Cela permet d'avoir le choix pour utiliser des ***addons*** disponibles (voir figure 10). Dans un premier temps, il ne nous faut que *core* et *UI*.

Sinon, pour créer un projet d'addon dans la console Forge, exécutez :

```
[jerome]$ project-new --named addonDR --type addon
```

Cette commande créera le projet et le fera apparaître *automatiquement* dans JDS (mais il faudra ajouter les *addons* en tant que dépendance Maven).

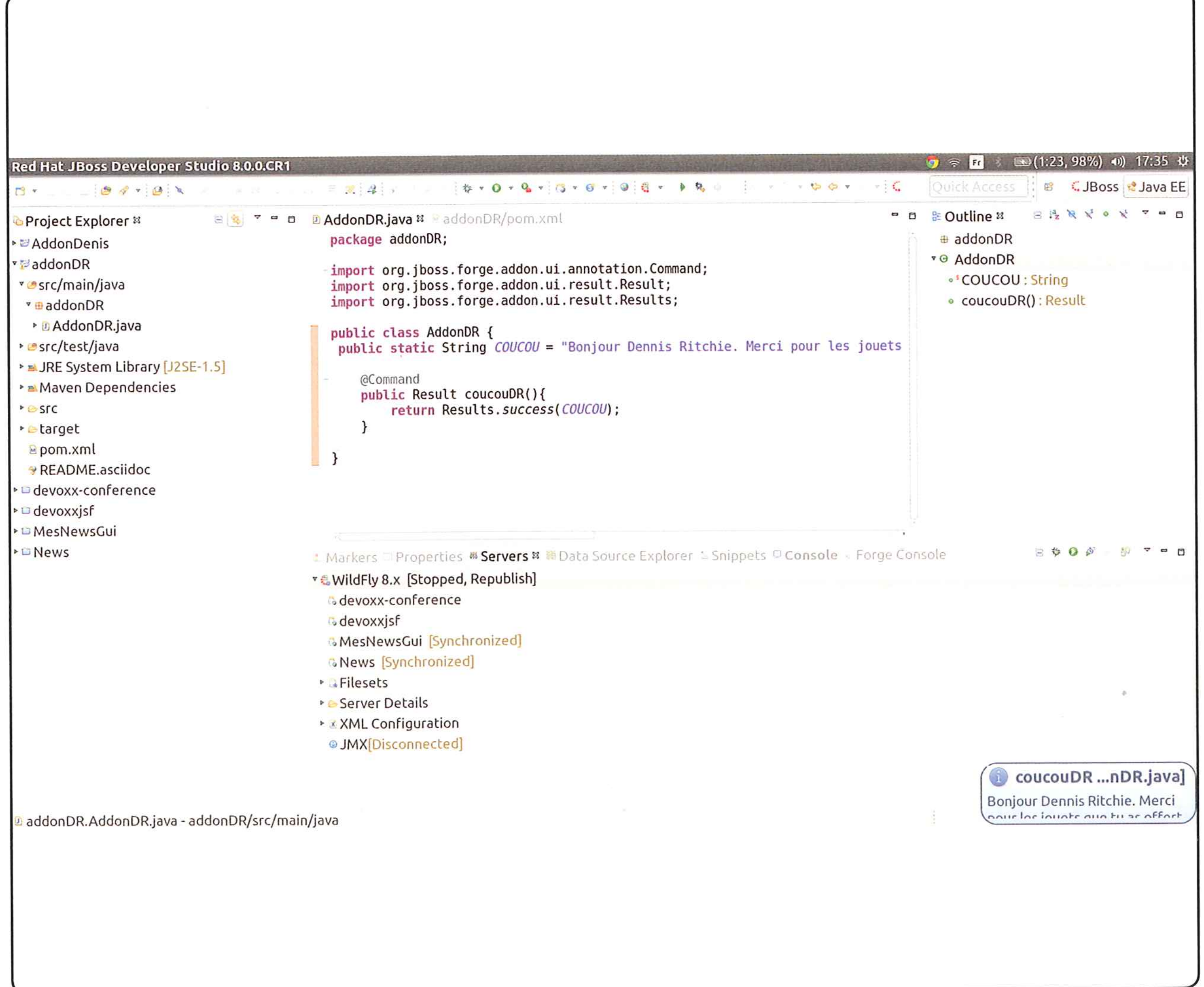


Fig. 11 : Affichage de la popup avec l'hommage

Créez ensuite une classe, et copiez-y le code suivant :

```
package addonDR;


import org.jboss.forge.addon.ui.annotation.Command;
import org.jboss.forge.addon.ui.result.Result;
import org.jboss.forge.addon.ui.result.Results;


public class AddonDR {
    public static String COUCOU = "Bonjour Dennis Ritchie. Merci
pour les jouets que tu as offert au monde. Entre le C, Unix et ta
barbe, tu étais le père noël des geeks !";


    @Command(categories="LeKiff")
    public Result coucouDR(){
        return Results.success(COUCOU);
    }
}
```

Maintenant, dans la fenêtre Forge, choisissez ***Build and Install an Addon***. Sélectionnez le chemin du projet et patientez un (long) moment pendant le build.

Voilà, votre *addon* est créé et disponible sous le nom de **coucouDR**. Dans la fenêtre Forge, choisissez **coucouDR** et à l'exécution, vous obtiendrez votre message dans le coin en bas à droite de l'écran (voir figure 11).

Et là, vous pouvez légitimement vous dire que le message était trop long pour la popup. Certes. Mais cet exemple simple et concis représente la porte ouverte à toutes les fenêtres de commandes que vous voudrez vous créer. Vous pouvez aussi exécuter cette commande dans la console Forge pour relire le texte dans son intégralité et toute sa splendeur.

Conclusion

Il est indéniable que Forge vous offre une productivité impressionnante. Il fait le boulot, le slogan annoncé est tenu. En très peu de temps, un projet Maven est initialisé, câblé, JPA et REST sont configurés, les fichiers obligatoires ne sont pas oubliés. Aucune bête erreur de syntaxe ou d'oubli. Le tout en un minimum de commandes et d'informations transmises au programme. Même s'il n'est pas parfait, je juge cet outil excellent. Sa disponibilité en mode texte et en mode graphique est un vrai plus. La version graphique peut se révéler plus lente d'utilisation que la version en mode texte, notamment pour l'ajout de champs à une entité JPA. Mais, redevenons autonome, il est possible de les ajouter à la main ou même par copier/coller pour encore plus de rapidité :)

Si les interfaces graphiques générées en JSF ou AngularJS ne conviennent pas, il est possible de créer son propre générateur ou bien de les créer séparément tout en exploitant les points d'entrée REST générés.

À défaut de prendre l'ensemble de ce qui est généré, il est aussi possible de n'utiliser que des parties et de les assembler dans un autre projet.

Forge n'est pas la *silver bullet* du développement Java EE, il arrive que cela ne marche pas comme on le voudrait notamment avec les collections. Pire, j'ai essayé des versions de JDS qui n'ouvraient pas du tout la fenêtre Forge. Ma solution de contournement rapide a été de me tourner vers une version précédente de JDS.

Au-delà des projets simplistes, avoir des compétences Java EE est obligatoire pour utiliser le projet produit. Néanmoins, Forge est un pas supplémentaire sur le chemin de la simplification de la programmation qui, après les cartes perforées, a commencé par l'assembleur, et a continué avec le C, le Java à la main et finira pourquoi pas demain, avec de la programmation « parlée » en discutant avec un avatar anthropomorphe du cloud de notre hébergeur.

Cela en est fini pour ce premier article introductif à Forge, nous creuserons plus profondément la création d'un addon plus ambitieux dans un prochain article.

Note

Pour me permettre d'avoir votre feedback sur cet article, vous pouvez répondre au sondage http://goo.gl/ijqi1r.

Remerciements

Je remercie Antonio Goncalves (@agoncal), Philippe Prados et Alexis Maillot pour leur travail de relecture de cet article.