N°162 JUILLET/AOÛT 2013

GNU LINUX MAGAZINE / FRANCE

ACTU / BIG

Découvrez comment le mouvement Big Data impacte les architectures informatiques, la distribution et le traitement des données

p.04

ADMINISTRATION ET DÉVELOPPEMENT SUR SYSTÈMES OPEN SOURCE ET EMBARQUÉS

KERNEL / 3.9

Découvrez les nouveautés du noyau 3.9 : tcpdump/BPF avec iptables, stockage, Btrfs, gestion d'énergie, etc.

p.12

PKI / ANDROID

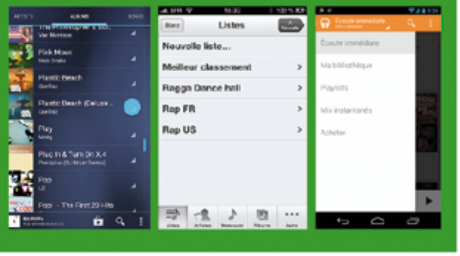
Explorez les technologies publiques et cachées d'Android pour la gestion de certificats

p.43

GUI / SMARTPHONES

Confrontez les fonctionnalités Android à iOS et Windows Phone sur le terrain de l'ergonomie mobile

p.60



WEB / CMS

Créez un site statique en conservant le confort d'un CMS pour gérer le contenu et sa présentation

p.18

PROJET / GESTION

Changez pour l'intelligence collective et la méthode de développement Kanban pour vos projets

p.38

SYSADMIN / SERVEURS

Vous avez des serveurs GNU/Linux et Windows un peu partout qui « log » dans leur coin ?

CENTRALISEZ LA GESTION DES LOGS

p.24

1. Rappel sur les journaux d'activité
2. Déportez et centralisez les logs GNU/Linux
3. Ajoutez les logs des machines Windows
4. Configurez votre serveur rsyslog et regroupez l'ensemble
5. Connectez la solution à votre supervision avec NRDP

PHP / DÉPENDANCE

Composer, enfin une solution de gestion des dépendances de vos projets PHP souple et efficace !

p.74

CODE / LEGACY

Reprendre, adapter, traiter et étendre du code existant, vieux, non testé, non supporté, ça s'apprend

p.78

L 19275 - 162 - F: 7,50 € - RD

France METRO : 7,50 € – CH : 13 CHF – BEL/PORT.CONT : 8,50 € – DOM : 8 € – CAN : 13,75 $ cad – NCAL/S : 1000 CFP – POL/S : 1100 CFP – POL/A : 1500 CFP – TUNISIE : 16,50 TND – MAR : 95 MAD

JDEV

Journées Développement Logiciel de l'Enseignement Supérieur et de la Recherche

Conception graphique : Benoît Colas, Université de Toulouse-Le Mirail / CPRS - [illegible] Photographie : D.R.

Webcast

4 - 5 - 6 septembre 2013, **École polytechnique**

http://devlog.cnrs.fr/jdev2013

Contact : contact-jdev2013@services.cnrs.fr

SOMMAIRE N°162

GNU/Linux Magazine France
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

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

Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Secrétaire de rédaction : Véronique Sittler
Réalisation graphique : Jérémy Gall

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
Service des ventes : Distri-médias : Tél. : 05 34 52 34 01
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, N° **ISSN :** 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,50 €

MIXTE Papier issu de sources responsables FSC® C015136 www.fsc.org

Membre April www.april.org

ÉDITORIAL

Vous avez vu ?!

Mais si ! En couverture... juste làààà, en gros, la démonstration qu'il est possible de faire beaucoup, bien propre, multi-plateforme et modulaire sans pour autant brutalement remplacer quelque chose qui fonctionne à merveille par une usine à gaz^W^W^Wtechnologie avancée permettant de royalement « gratter » une poignée de secondes au démarrage (je ne vise personne en particulier, promis).

Attention, je ne dis pas pour autant que les avancées et les évolutions sont forcément ou majoritairement néfastes, loin de là. Regardez, par exemple, la manière dont gPXE/iPXE a su étendre à la fois EtherBoot et les spécifications PXE sans pour autant traumatiser des légions de sysadmins. Vous aimez TFTP ? Gardez TFTP ! Mais, quand vous aurez le temps, essayez le *boot* HTTP(S), iSCSI, FCoE, AoE, ...

C'est ainsi que les choses changent et s'améliorent dans le monde UNIX. C'est une évolution naturelle, sélective, darwinienne, fluide... et non des « sauts » bouleversant jusqu'aux fondations, ce qui existe, au point d'écœurer ou de contraindre les utilisateurs. Le risque encouru avec ces sauts (forcés) est, dans le meilleur des cas, une utilisation à contre-cœur, dans le pire, un abandon pur et simple de la plateforme et, entre les deux, un changement de distribution ou une succession d'efforts permettant de conserver l'élément initialement utilisé. Disons-le franchement, l'utilisateur lambda, lui, ne cherchera pas longtemps avant de jeter l'éponge. Celui-ci, fuyant l'interface Metro de Windows 8 et les problèmes de compatibilité, se retrouvera dans sa Ubuntu dernier cri, avec Unity-bling-bling, d'autres problèmes de compatibilité ainsi que des annonces publicitaires dans ses recherches... Tout ce qu'il voulait c'était retrouver la métaphore du bureau, ses « pochettes » et plein plein plein d'icônes masquant son fond d'écran. Peu importe que la « fête aux icônes » soit ou non une bonne chose, c'est ce qu'il voulait et il ne l'a pas eu alors qu'il était prêt à sauter le pas.

Si nous en revenons au présent numéro, rsyslog tout comme syslog-ng sont des implémentations du protocole Syslog. Personne n'est venu tout casser avec de grandes théories et des concepts révolutionnaires (enfin si, Systemd l'a fait, mais finalement au même titre que le remplacement total ou partiel de SysVinit, de l'automount, de la gestion ACPI, de cron, etc). Pourtant, rsyslog tout en restant classique, va vous permettre de faire de grandes choses comme vous pourrez le constater par vous-même dans l'article en question.

Pourquoi donc réinventer la roue ? En particulier si la nouvelle version est carrée pour faciliter le rangement, alors qu'il suffit naturellement de l'améliorer en la laissant bien circulaire et l'équipant d'un roulement à bille, d'une suspension et/ou d'un garde-boue. Personnellement, je préfère garder mes roues améliorées (e17, X, SysVinit, Syslog, LVM/ext4, Vim, etc.) car pour l'heure, tout ce que j'utilise quotidiennement est le fruit d'une évolution progressive (parfois d'un *fork*) et non d'un chamboulement faisant fi des enseignements et de l'expérience passés, et par la même occasion, du temps que j'ai investi.

Denis Bodor

PS : Si le présent numéro d'été de GLMF, plus Open Silicium 7 disponible dans le même temps, ne vous suffisent pas à occuper tout le temps de vos vacances, et que vous êtes lycéen, sachez que nos confrères du magazine Tangente organisent un sympathique concours d'intelligence artificielle sur les jeux logiques. C'est le concours Bernard Novelli et les inscriptions sont ouvertes jusqu'au 31 juillet. Plus d'informations sur : http://www.infinimath.com/.

LES ARCHITECTURES BIG DATA

par Thomas VIAL [Architecte Senior @ OCTO Technology]

Le contenu de cet article est issu de nombreux ateliers de réflexion que nous avons menés chez mon employeur, aboutissement de nos activités de veille, de R&D et de communication sur le sujet.

Impossible ces temps-ci de passer une journée sans entendre parler de big data. Les consultants comme nous, nos clients, les éditeurs de logiciels, les constructeurs, les grands du web, les adeptes de la théorie du complot qui voient *Big Brother* à chaque coin de rue : tout le monde en parle ! Et cet article n'échappe pas à la règle.

Bien sûr, chacun met ce qu'il veut derrière ce terme, selon qu'il veut mettre en avant l'origine de big data, ses leviers, son potentiel ou ses dangers. Ici, nous nous intéresserons à la manière dont ce mouvement et les architectures informatiques s'influencent mutuellement. Les capacités de stockage et de traitement, à coût donné, sont en croissance constante, ce qui suscite de nouvelles possibilités jusqu'alors inenvisagées ; à l'inverse, ces possibilités font émerger des idées nouvelles qui stimulent la définition d'architectures toujours plus performantes. Comme nous le verrons, ce cycle continue de tourner, et nul doute que les évolutions des coûts le déplaceront dans l'avenir. Cela ne nous empêche pas de faire une photo à la date d'aujourd'hui. Nous commencerons par détailler des principes généraux (distribution, parallélisation et commoditization), avant de reconstruire couche par couche l'architecture d'un système big data tel qu'on peut le rencontrer aujourd'hui.

Dans ce beau modèle théorique où les possibilités de traitement finissent par tendre vers l'infini à coût nul, une variable nous ramène toutefois à la réalité : le débit des I/O disques et dans une moindre mesure, des réseaux. Malgré l'arrivée des technologies SSD, le débit disque croît moins vite que les autres indicateurs. Le coût de la RAM étant encore élevé par rapport à celui du stockage "dur" et non volatile, celui-ci reste nécessaire à l'absorption de l'avalanche de données nécessaire aux projets. Les projets se voient ainsi limités par le goulet d'étranglement des I/O. Stocker et traiter toutes ces données sur un ou quelques serveurs gonflés aux stéroïdes n'est pas possible à des coûts raisonnables. De toute façon, la croissance constante des volumes de données fait que cette stratégie ne serait qu'une fuite en avant. Le critère à prendre en compte n'est pas tant la capacité de stockage ou de traitement, ou le coût des infrastructures capables d'assurer ces tâches, que le coût ramené au GB de données stockées ou à la requête unitaire.

1 Distribution, parallélisation et commoditization

Pour contourner cette limitation, et se libérer d'une équation de coûts trop pessimiste par rapport aux besoins d'innovation suscités par le big data, il a fallu changer radicalement les architectures. Celles mises en oeuvre dans tous les projets big data répondent par trois stratégies complémentaires : distribution des données, parallélisation des traitements et *commoditization*

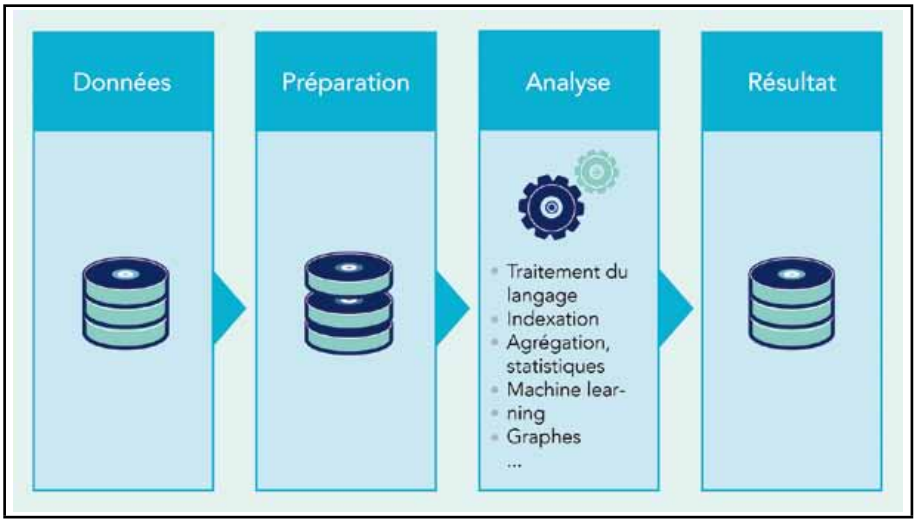


Figure 0

des infrastructures. Ces stratégies sont directement inspirées des travaux des géants du web (www.geantsduweb.com), Google, Facebook et autres Yahoo!, qui ont été les premiers à faire face aux limites mentionnées plus haut.

Distribution des données : pour fixer les idées, disons que l'on parlera de big data à partir d'un ordre de grandeur de 10 To de données en ligne. Si un serveur ou un SAN n'est pas capable, de manière économique, d'héberger le volume nécessaire aux traitements, alors il faut distribuer ce volume sur plusieurs serveurs. Cette distribution est au minimum horizontale, c'est-à-dire que les ensembles d'objets à stocker sont regroupés en paquets -- des *shards* -- répartis sur les nœuds du cluster. Le critère de répartition est souvent le résultat d'une opération de hachage sur les données. À cette stratégie de distribution, le *sharding*, s'ajoute parfois un partitionnement des données individuelles ; cette fois, les objets eux-mêmes peuvent être découpés et répartis sur plusieurs nœuds (c'est le cas, par exemple, des fichiers HDFS, le système de fichiers d'Hadoop). Pour récupérer un objet complet, il faut alors identifier les nœuds qui hébergent chacune de ses parties et les recomposer.

Parallélisation des traitements : les besoins tendent aussi vers une réponse de plus en plus rapide aux problématiques métiers. Au vu des volumes en jeu, les traitements séquentiels ont des temps d'exécution qui sont prohibitifs : la parallélisation est nécessaire. Notons déjà que ce point entretient un rapport étroit avec le précédent : les goulets d'étranglement se situant au niveau des I/O des disques et des équipements réseau, une stratégie de parallélisation efficace doit forcément tenir compte de l'emplacement des données manipulées. C'est la notion de *colocalisation* (nous y reviendrons juste après).

Commoditization des infrastructures : derrière cet anglicisme se trouve l'idée qu'il est plus économique, à terme, de remplir des armoires avec de nombreux serveurs de série (*commodity hardware*), qu'avec quelques serveurs hauts de gamme. Ces serveurs "basiques" assument le double rôle de distribution du stockage et du traitement que nous venons d'évoquer. Encore une fois, il faut considérer le coût ramené au GB de données. Cette stratégie confère aux clusters ainsi constitués une propriété d'élasticité : on peut ajouter des noeuds de stockage et de traitement au fur et à mesure de l'évolution des besoins, progressivement, sans faire de gros investissements par à-coups. On peut aussi en enlever ! C'est ainsi que les géants du web ont constitué, petit à petit, des clusters de plus en plus gros pour faire face à leurs traitements de plus en plus gourmands.

1.1 Nouvelles stratégies, nouveaux défis

Si ces stratégies s'avèrent payantes tout en semblant reposer sur des principes simples, il ne faut pas négliger non plus les défis qu'elles suscitent.

Tout d'abord, au niveau des plateformes techniques, il s'agit ni plus ni moins que de mettre en place des systèmes distribués, avec leur cortège de problématiques : cohérence et équilibrage des données distribuées, tolérance aux pannes, distribution multi-datacenters, cohabitation de traitements transactionnels et analytiques, etc. En particulier, les plateformes distribuées embarquent nécessairement des mécanismes de résilience face à la perte d'un noeud, d'autant plus lorsque l'infrastructure est de type commodity, plus encline à la panne.

La résilience pour le stockage implique nécessairement la duplication de la donnée entre deux ou plusieurs noeuds. Cette duplication offre aussi un avantage non négligeable pour l'exécution : elle offre une grande souplesse dans l'affectation des ressources de calcul aux nœuds, en appliquant le principe de colocalisation évoqué précédemment. Ce principe veut que, plutôt que de rapatrier un shard à traiter sur un nœud de calcul distant, c'est le calcul qui est envoyé vers le nœud qui héberge une des répliques du shard, par exemple le noeud qui est le moins sollicité à ce moment. Les I/O réseau sont minimisés lorsque c'est possible. La duplication des shards a ainsi un double avantage, pour un inconvénient mineur étant donné le coût des disques : une

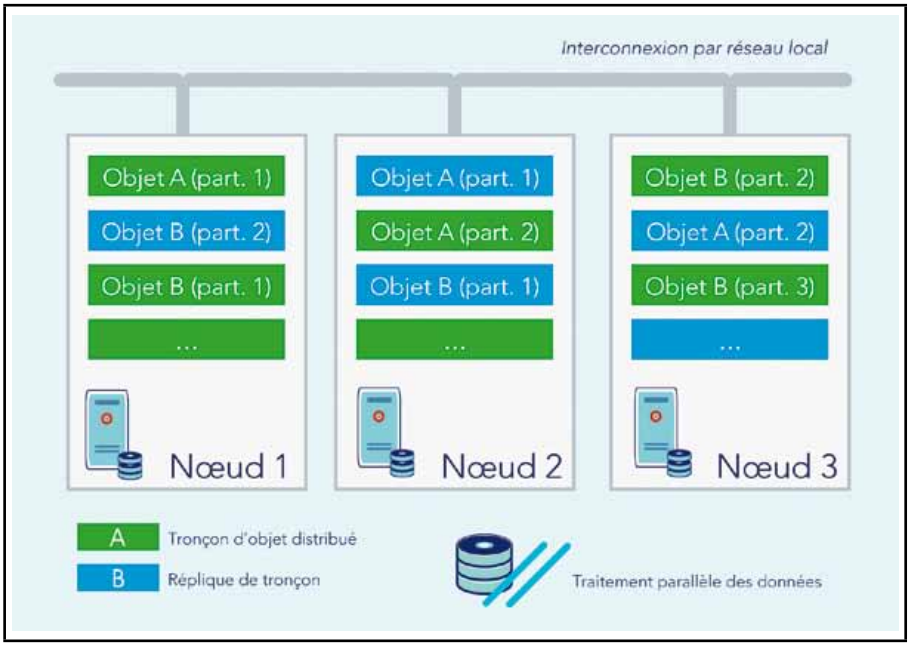


Figure 1

empreinte de stockage multipliée par le facteur de réplication. Il faut toutefois en tenir compte au moment de dimensionner la plateforme !

Nous pouvons résumer par un schéma le couplage entre le stockage et les traitements (Figure 1 page précédente).

Deuxième défi, les changements d'architecture amenés par ces nouveaux paradigmes interviennent au niveau le plus bas. Il a donc fallu revoir toutes les couches qui s'appuient dessus, et pour ainsi dire réinventer la roue à tous les niveaux. C'est ce que nous allons voir plus en détail maintenant, en visitant l'une après l'autre les couches d'architecture qui constituent un système "big data" générique (nous ne ferons qu'à la fin la distinction entre les systèmes transactionnels -- type NoSQL -- et analytiques -- type Hadoop). Nous commencerons par le bas, pour empiler les couches.

2 L'infrastructure

Le stockage et les traitements s'appuient évidemment sur une infrastructure qui en fournit les ressources : dans le cas le plus général, ce sont des disques, des cœurs de calcul, de la RAM et du réseau.

Nous avons vu que ces ressources étaient nécessairement distribuées, du fait des volumes en jeu (10 To et au-delà) et de la lourdeur des traitements qui opèrent dessus. L'infrastructure d'un système big data est une infrastructure de cluster. Elle a besoin d'une interconnectivité réseau entre les noeuds du cluster. La topologie réseau n'est pas anodine, car ces mêmes volumes de données sont échangés de manière importante entre les nœuds. Les liens doivent être suffisants pour ne pas être un goulet, et la plateforme doit idéalement tenir compte de la topologie pour optimiser le placement des données et des traitements en minimisant les échanges (par exemple en favorisant les transferts entre nœuds d'un même rack).

Enfin, dans le cas d'un hébergement sur machines physiques, une couche de virtualisation peut être nécessaire. La virtualisation va en quelque sorte à l'encontre de l'approche mentionnée plus haut, car elle permet la consolidation des ressources de stockage ou de calcul, et non leur extension. La virtualisation, lorsqu'elle existe, ne répond pas tant à un besoin d'utilisation optimale des ressources que d'élasticité et de facilité de provisioning des machines.

3 Stockage vs modélisation des données

L'observation la plus évidente que l'on peut faire est la séparation nette entre le stockage et la modélisation des données : cela constitue une rupture par rapport aux systèmes relationnels classiques, où la base de données embarque son schéma.

En effet, le besoin de traiter des gros volumes de données, hétérogènes (voire non structurés) et variables dans le temps, milite pour une modélisation lâche ou même inexistante. Le cas extrême se retrouve du côté des bases de données NoSQL en clefs/valeurs, qui ne sont ni plus ni moins que des tables de hachage distribuées. Leur API est réduite au minimum : `get(key)` et `put(key, val)` ; les valeurs sont pour la base des BLOBs opaques et charge à l'application utilisatrice d'interpréter leur contenu. D'autres bases de données, orientées document (où les structures de données sont mémorisées d'un seul tenant, comme des objets complets) ou en familles de colonnes (où les attributs de chaque ligne sont stockés ensemble par groupe, les familles), proposent un stockage un peu plus structuré quoique encore bien plus lâche que les modèles relationnels. Les solutions de traitement en masse, comme Hadoop, n'échappent pas à cette règle puisqu'elles ne "voient" que des fichiers, i.e. des flux indifférenciés d'octets.

Toutes ces situations ont une caractéristique commune : quand il existe, c'est l'application qui "fait" le modèle, le maintient et assure la compatibilité ascendante des données qu'elle rencontre. Pour faire face aux évolutions inévitables de ce modèle dans le temps, plusieurs stratégies sont possibles ; selon les frameworks utilisés et les patterns d'accès à la donnée, telle ou telle stratégie sera plus adaptée.

3.1 Évolutivité des modèles de données

Le plus simple est une reprise d'historique des données in-situ lors des migrations, comme aujourd'hui avec un SGBDR -- mais avec des volumes bien plus importants bien sûr.

Une deuxième stratégie, appelée *schema on read*, est l'interprétation à la lecture du modèle de la donnée lue. C'est la stratégie la plus courante dans les environnements de type Hadoop, car elle se prête aussi bien aux données structurées et non structurées, et permet à plusieurs applications ou requêtes d'interpréter chacune une même donnée à sa manière. Elle induit bien sûr un surcoût répété à la lecture, qui peut être pénalisant sur des accès massifs.

La troisième stratégie est plus adaptée aux systèmes transactionnels, soumis à des lectures unitaires fréquentes qui doivent rester performantes. Elle s'appuie sur la stratégie *schema on read* pour transformer la donnée de l'ancien modèle vers le nouveau, et la réécrire immédiatement afin de faciliter les lectures suivantes. C'est une sorte de reprise d'historique opportuniste...

3.2 Modélisation et distribution

Nous avons présenté le découplage stockage / modélisation comme une nécessité liée à la forme et à l'évolution des données elles-mêmes. Toutefois, on peut justifier ce choix

1&1 SERVEUR CLOUD DYNAMIQUE

PUISSANCE FLEXIBLE
PRIX MAÎTRISÉ

à partir de **0,03 €** HT PAR HEURE*

CONTRÔLE TOTAL DES COÛTS

- **NOUVEAU : sans engagement !**
- **NOUVEAU : sans frais de mise en service !**
- **NOUVEAU : sans prix de base !**
- **Durée limitée : jusqu'à 30 € de crédit offert !****
- **Transparence totale** grâce à la facturation horaire à l'usage.
- **Trafic illimité** sans réduction de bande passante.
- **Parallels® Plesk Panel 11 inclus,** noms de domaine illimités.

HAUTE FLEXIBILITÉ

- vCores, RAM et espace disque configurables séparément : **seulement 0,01 € HT par heure et par unité matérielle !***
- **NOUVEAU : jusqu'à 8 vCores et 32 Go de RAM.**
- Ajoutez jusqu'à 99 machines virtuelles en un seul clic - sans migration !

SÉCURITÉ OPTIMALE

- Disques durs et unités de calcul redondés afin de protéger votre serveur cloud contre toute défaillance.

ACCÈS ROOT COMPLET

- Droits d'administrateur et ressources dédiées pour chaque VM.

OPTERON AMD · SUSE · Parallels Plesk Panel

1&1

MEMBRE DE united internet

DOMAINES | EMAIL | HÉBERGEMENT | E-COMMERCE | SERVEURS

0970 808 911 (appel non surtaxé)

1and1.fr

* Configuration minimale : 1 vCore, 1 Go de RAM et 100 Go d'espace disque, soit 0,03 € HT/heure (0,036 € TTC). Le prix final varie en fonction de la configuration choisie : simulation en ligne sur hosting.1and1.fr/cloud-server-config.

** Crédit de 30 € déduit du montant HT de la 1ère facture, pas de report du crédit non consommé. Détails disponibles sur 1and1.fr.

d'architecture par d'autres critères, plus techniques : la contrainte de distribuer et dupliquer les données conduit de toute façon à abandonner la modélisation relationnelle, trop contraignante.

Certes, les SGBDR modernes savent faire du sharding, mais au prix de certaines contorsions pour respecter le modèle relationnel tout en gardant des performances acceptables. Ainsi, le respect des contraintes de clefs étrangères, pour être efficace, suppose que des enregistrements corrélés soient stockés ensemble ce qui, en jargon de modélisation NoSQL, n'est pas très loin d'un modèle orienté document... Ou bien certaines tables, comme les tables de référence, devront être dupliquées pour satisfaire les relations sur tous les nœuds séparés.

En revanche, des modèles de stockage en clefs/valeurs ou documents se prêtent très bien à la distribution, car les contraintes de clefs étrangères n'existent plus. C'est bien sûr à l'application d'assurer les règles d'intégrité. Quant aux systèmes à base de fichiers, comme HDFS, la question ne se pose pas : ce sont des flux d'octets que l'on peut couper où l'on veut, pour recoller les morceaux à la lecture.

4 Traitement des données

Nous venons de voir que si la distribution physique des données est assurée par la plateforme de stockage, et donc transparente pour l'application, ce n'est pas le cas de sa structure.

Lorsque la donnée stockée est exploitable directement (par exemple elle est déjà assez structurée, et le traitement consiste en un calcul de statistiques simples comme un comptage ou une moyenne), le code du traitement peut directement opérer dessus. Selon les plateformes, l'implémentation devra tenir compte ou non d'une exécution parallèle du traitement par le framework. C'est le cas des plateformes s'appuyant sur un framework MapReduce, par exemple, qui imposent une structure particulière aux programmes qui l'utilisent.

Cependant, dans de nombreux cas, la donnée issue du système de stockage n'est pas exploitable en l'état par le coeur du traitement. Il faut d'abord la faire passer dans un pipeline d'interprétation, pour lui donner une forme consommable par l'algorithme. Ce pipeline peut bien sûr être implémenté par le programme lui-même, ou faire appel à des librairies pré-existantes que le programme se contentera de coordonner.

Avant de citer quelques unités de traitement possibles pour un tel pipeline, notons que ce dernier peut s'exécuter de différentes manières : en tâche de fond sur de gros volumes (mode *batch*), sur des volumes faibles ou avec des temps de réponse garantis faibles (mode *interactif*), ou au fur et à mesure de l'arrivée des données unitaires dans le système (mode *événementiel*).

4.1 Les étapes du pipeline

Une première étape quasiment obligatoire, avec la variété des sources de données à laquelle on est confrontée, est la **préparation des données**. Il faut en général les nettoyer, les enrichir (avec des données de référence ou des valeurs par défaut pour les portions manquantes), afin qu'elles atteignent une qualité suffisante pour être réellement traitées.

L'indexation des données est une étape nécessaire lorsqu'un besoin de recherche dans les données est identifié. Il n'est en effet pas question de faire une recherche brutale sur des volumes aussi importants, pour retrouver une donnée précise. Les solutions de type NoSQL, qui stockent des objets, offrent des index de type clef primaire. On observe cependant que les besoins plus élaborés (index secondaires, recherches plein-texte) sont en général confiés à des moteurs de recherche tiers (type Solr, Lucene) intégrés avec le stockage, et qui opèrent directement sur leurs données.

Pour les **données non ou faiblement structurées** (texte, images, vidéo...), il est nécessaire de faire une extraction de métadonnées, ou des traitements spécifiques au type de document. À titre d'exemple, pour du texte on trouvera une extraction de mots, des statistiques sur les paragraphes, voire du traitement élaboré de langage naturel (NLP -- *Natural Language Processing*) et une interprétation sémantique. Pour des images, on peut envisager d'en calculer la taille, de faire de l'OCR, de la détection de visages ou de logos... Cette première étape du pipeline permet de doter la donnée brute d'une structure exploitable par les étapes suivantes. Elle peut aussi alimenter un moteur de recherche.

Dans de nombreux cas, on est amené à appliquer des méthodes statistiques plus poussées que de simples sommes, afin d'extraire plus d'information ou de construire des modèles statistiques. C'est le domaine du **machine learning**, qui n'est ni plus ni moins que l'incarnation moderne et (partiellement) automatisée des techniques traditionnelles de data mining. Il sert trois problématiques : décrire, expliquer et prédire. Les algorithmes font appel à des combinaisons de techniques statistiques, d'intelligence artificielle, et bien sûr au flair de l'analyste de données (le fameux data scientist) qui va choisir les meilleures stratégies en fonction du contexte.

On peut voir les **traitements des graphes** comme un cas particulier du machine learning, mais il a aussi ses caractéristiques propres. De nombreux problèmes peuvent être formulés avec une représentation en graphes. C'est une représentation compacte d'objets et de leurs relations, qui offre l'avantage d'être outillée par une théorie mature. La panoplie offerte par cette théorie est vaste, et couvre un spectre d'analyse très large : analyse descriptive globale sur les propriétés du graphe, calcul d'influence des sommets au sein du réseau, calcul de chemins, etc. Les défis, avec les graphes, viennent des arcs dans les scénarios de parcours. Lorsque le graphe doit être distribué sur plusieurs

noeuds de stockage, l'efficacité du partitionnement dépend directement de la répartition de ces arcs.

Voici la vue d'ensemble d'un pipeline complet (Figure 2).

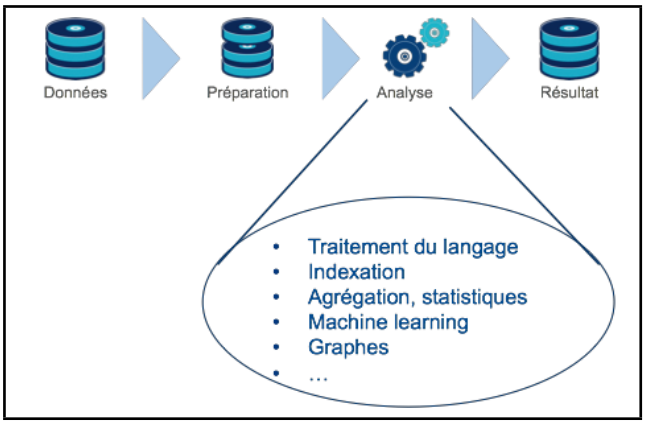


Figure 2

5 Requêtage et visualisation

Au-dessus des étages de traitement et de préparation, nous trouvons l'exploitation des données proprement dites. Les patterns de mise à disposition des données n'ont pas tellement changé : préparation de dashboards statiques, mise à disposition d'outils d'exploration des données, « push » de mises à jour sur des données qui changent fréquemment, ou tout simplement ouverture d'un infocentre à des requêtes ad hoc.

Des données, brutes ou structurées par le biais d'une étape précédente, peuvent tout d'abord être **requêtées**. Il s'agit ici de transposer, sur une plateforme big data, les requêtes que l'on fait depuis longtemps dans le monde SGBDR classique : recherches, accès par clef primaire, enrichissement (jointures), statistiques descriptives simples (comptage et somme typiquement) avec regroupements. L'apport de la plateforme big data consiste à brasser des quantités de données inaccessibles aux SGBDR. Entre les systèmes NoSQL, essentiellement transactionnels, et les systèmes BI comme Hadoop et Hive, il existe des différences considérables dans les types de requêtes que l'on peut faire de manière efficace -- même si une convergence est à l'oeuvre -- nous y reviendrons en fin d'article.

Le volume des données étant en général énorme, en faire une représentation fidèle et synthétique, qui sache présenter et expliquer, est un défi ; c'est le domaine de la **visualisation**. Les tableaux et les camemberts de papa ont encore de beaux jours devant eux, mais ils ne suffisent plus, car ils sont purement descriptifs (ils n'expliquent rien des traitements élaborés qui ont conduit à la production des données), ne permettent pas de traduire la richesse des analyses multi-variables ou des données connectées (graphes). Et tout simplement, ils ne sont pas assez "jolis" pour des utilisateurs de plus en plus exigeants. Les éditeurs, et les contributeurs open source, proposent des catalogues de représentations très riches, statiques ou interactives. À titre d'exemple, on pourra consulter la galerie du site D3[1].

Même au niveau restreint de la couche de visualisation, les architectures évoluent. Si la préparation de reporting prédigérés peut être vue comme une étape de traitement parmi d'autres sur la plateforme big data, les visualisations à base d'exploration et de requêtage ad hoc se heurtent aux problèmes de performance que les architectures batch ne savent pas toujours contourner. Cette limitation est levée par l'emploi d'architectures mixtes batch/TP, qui sont en train d'émerger et d'enrichir considérablement l'offre logicielle des plateformes.

6 Intégration au SI...

... ou à l'extérieur du SI ! Pas de big data sans données, c'est une évidence : il faut bien amener ces données sur la plateforme, avant de l'utiliser. Et dans l'autre sens, on veut

[1] http://www.d3js.org

pouvoir exporter les données traitées vers des fichiers ou d'autres systèmes, par exemple stocker des agrégats de haut niveau dans des entrepôts relationnels classiques. Pour tout cela, des connecteurs sont nécessaires.

On trouve dans cette catégorie des outils d'import/export depuis/vers les bases de données du SI ou des fichiers. Les API, et outils en ligne de commandes, sont aussi de cette catégorie, car ils vont permettre à des applications externes d'interroger les données stockées sur la plateforme (voire de lancer des traitements).

7 Management de cluster

On retrouve dans cette couche les outils habituels de gestion d'infrastructure, adaptés aux composants opérationnels de la plateforme big data. Les outils et composants sont assez inégaux de ce point de vue ; nous allons illustrer ce point avec deux exemples.

Tout d'abord la supervision : les solutions offrent en général peu de métriques de haut niveau permettant de comprendre l'utilisation du cluster, d'un point de vue de la charge de travail. Le dimensionnement, le capacity planning, sont donc encore assez artisanaux... Ou bien ces métriques existent, mais ne sont pas centralisées facilement (c'est le cas d'Hadoop).

La sécurité, elle aussi, est souvent laissée de côté. Elle constitue pourtant un enjeu à elle seule, car l'enjeu de big data étant l'exploitation massive de toutes les données à disposition (en caricaturant), la protection de ces données est cruciale. Là encore, Hadoop dispose d'un système de sécurité encore embryonnaire, mais que les prochaines versions d'HDFS vont compléter -- en introduisant également des politiques de gestion de cycle de vie de la donnée.

8 Synthèse : l'architecture d'un système big data

Nous sommes maintenant en mesure de dresser le portrait général d'un système big data, en reprenant les couches que nous avons vues, sous forme d'un fond de carte (Figure 3).

Bien entendu, tous les systèmes n'implémentent pas toutes les couches ; en particulier les systèmes transactionnels et analytiques se distinguent par leurs couches hautes. Du moins était-ce le cas jusqu'à récemment... la situation devient plus floue.

Nous avons déjà mentionné la convergence entre les systèmes doués pour le transactionnel, et ceux plus portés sur l'analytique. Elle se manifeste par le fait que les deux classes d'architecture, différentes dans leur conception, se voient augmentées de capacités empiétant sur l'autre royaume. Du côté des bases NoSQL, de plus en plus se dotent de frameworks MapReduce (MongoDB, Riak) ou proposent une intégration avec Hadoop en se substituant au système de fichiers HDFS (l'édition DataStax de Cassandra, avec CFS). À l'inverse, côté Hadoop, une compétition féroce est lancée pour savoir qui proposera le framework de requêtage interactif de demain (Impala chez Cloudera, Hive Stinger chez Hortonworks, Pivotal HD chez VMWare, sans compter des améliorations de MapReduce comme Tez ou Spark).

Cependant, il n'y aura pas d'architecture technique unique estampillée "big data", au-delà de la description fonctionnelle que nous venons de faire. Chaque implémentation étend son périmètre, mais au prix de compromis (sur le stockage, la duplication de données ou les performances absolues). Le choix de la plateforme reste très influencé par le jeu de requêtes auquel les données vont être soumises, et comme toujours, le système idéal pour votre cas d'utilisation sera un hybride intelligent du meilleur de chaque plateforme.

Pour conclure, je tiens à remercier une bonne quinzaine de mes collègues qui participent aux ateliers de réflexion et de rédaction pour le livre blanc sur le big data que je coordonne actuellement. ■

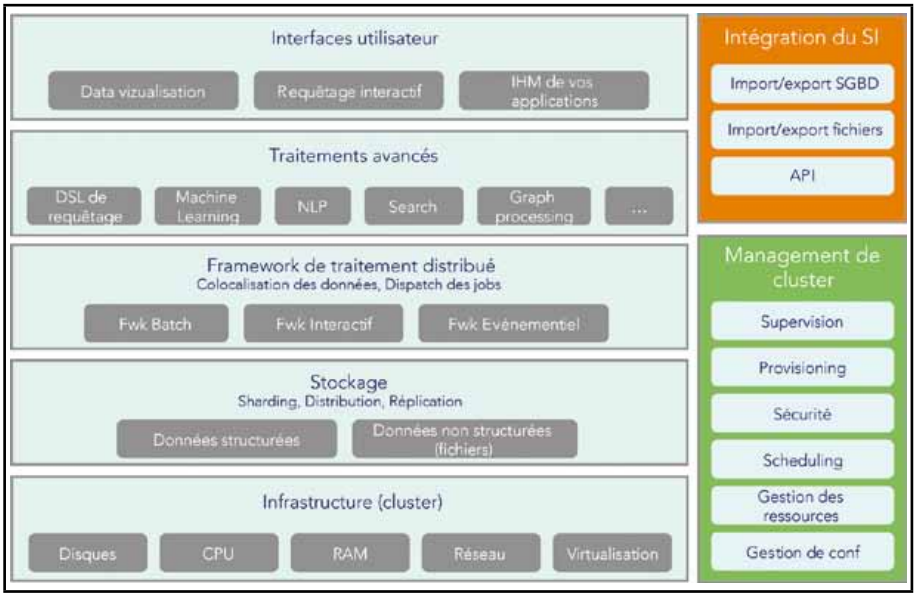


Figure 3

CHOISISSEZ ~~ENTRE~~
LA TECHNOLOGIE & LE SERVICE

À PARTIR DE
19,99€ HT/MOIS

Processeur Intel dernière génération*

* à partir du G460

RAM jusqu'à 96 Go DDR3

Disque jusqu'à 8 To SATA ou 4 x 240 Go SSD
Disques additionnels jusqu'à 3 To

TECHNOLOGIE DE POINTE | DATACENTER PROPRIÉTAIRE | SERVICE CLIENT ★★★★

Support Technique
Nos équipes sont disponibles sur site en 24x7.

Gérez votre serveur sur une interface intuitive
Reboot, Reverse DNS, Rescue MX, Backup, RAZ...

Disponible sur votre Espace Client

RENDEZ-VOUS SUR : express.ikoula.com/serveur-dedie

01 84 01 02 50
sales@ikoula.com

ikoula Hosting Services

NOM DE DOMAINE | MESSAGERIE | HÉBERGEMENT | CERTIFICAT SSL | CLOUD | SERVEUR DÉDIÉ

NOUVEAUTÉS DU NOYAU LINUX 3.9

par Matthieu Barthélemy

Ce noyau 3.9 est rempli de nouveautés qui touchent tous les domaines ; tellement que nous ne ferons, comme à notre habitude, que présenter celles qui nous ont semblé être les plus pertinentes.

1 Virtualisation : KVM sur ARM

Ça y est, vous pouvez transformer votre tablette ou smartphone en serveur de virtualisation ! Ou presque.

Concrètement, dans ce noyau 3.9, la couche de virtualisation KVM fonctionne sur les processeurs ARM dotés d'instructions de virtualisation (famille ARMv7). Actuellement, les puces Cortex A15 sont prises en charge.

KVM étant une couche de virtualisation minimale que l'on couple en général avec Qemu, la prise en charge d'ARM nécessite une version récente de ce dernier. Ça tombe bien, Qemu 1.5 est fraîchement éclos !

2 Le réseau

2.1 Écouter à plusieurs sur le même port

Une nouvelle option, nommée **SO_REUSEPORT**, peut être passée en paramètre lors de la création d'un socket. Elle autorise à écouter sur un port déjà occupé par un autre processus ou thread ; sans cette option, la réaction est d'habitude immédiate : **Address already in use**.

Mais alors, que se passe-t-il lorsqu'une connexion cliente arrive ? Qui répond ?

Chaque connexion cliente qui démarre est redirigée au hasard vers l'un des sockets en écoute. Une clef unique basée sur l'adresse et le port source est générée. Grâce à celle-ci, les requêtes suivantes provenant du même client seront envoyées au même socket qui a reçu et initié la session.

À partir de là, on devine donc le but de cette option : faire de la répartition de charge entre plusieurs instances d'un même service. Mais quel est l'avantage à utiliser cette méthode ? En effet, il existe déjà des techniques éprouvées : ainsi, un serveur web possède généralement un point d'entrée qui se charge uniquement d'écouter et d'accepter les connexions clientes, puis les dispatche sur un ensemble de threads ou processus fils pour traitement.

Premièrement, l'auteur du patch, qui travaille pour Google, indique que sous forte charge c'est le processus ou thread « central », chargé d'accepter et de répartir les connexions, qui devient le goulot d'étranglement.

Deuxièmement, en ayant plusieurs threads ou processus distincts, écoutant chacun les connexions entrantes, cela permet de les répartir bien mieux sur les cœurs ou processeurs du système, selon les observations de l'auteur.

Troisièmement, on peut se dire que cette méthode offre dans certains cas une robustesse améliorée : un seul processus qui crashe n'entraîne pas de coupure du service. En pratique, pour l'instant, les connexions TCP en cours d'initialisation peuvent être perturbées si, dans le même intervalle, des processus se partageant un socket sont stoppés ou ajoutés.

Enfin, cette option existe déjà depuis longtemps sur les OS de la famille BSD.

Cette option doit être spécifiée pour tous les sockets destinés à se partager un port, y compris pour le tout premier. De plus, pour sécuriser un peu le tout et éviter qu'un processus malicieux profite de l'option pour s'y greffer, tous les processus se partageant ce port doivent tourner sous le même compte utilisateur.

2.2 Utiliser des règles tcpdump/BPF avec iptables

Tcpdump est un outil bien connu permettant de filtrer et capturer des paquets réseau selon les règles qu'on lui indique. Ces règles agissent comme des filtres et obéissent à une syntaxe appelée BPF (Berkeley Packet Filter). On retrouve un système de filtrage utilisant cette même syntaxe sur nombre d'OS de famille Unix, ce qui n'est pas étonnant étant donné qu'elle date du début des années 90. Il en existe même une implémentation pour les systèmes Microsoft.

À partir de ce noyau, il est possible d'utiliser des règles BPF compilées comme règles **iptables** (avec **iptables** >= 1.4.19).

Prenons un exemple concret : filtrer tous les paquets arrivant en UDP sur le port 9000. La syntaxe BPF correspondante serait :

```
udp dst port 9000
```

Le nouveau module (nommé **xt_bpf**) doit recevoir une expression BPF déjà parsée, ce qui n'est pas un problème si l'on dispose de **tcpdump** :

```
# tcpdump -i any -ddd udp dst port 9000 | tr '\n' ','
```

L'opton **-ddd** indique qu'on souhaite que la règle soit parsée et renvoyée sous forme décimale, et nous remplaçons, grâce à **tr**, les retours à la ligne par des virgules.

La réponse de **tcpdump**, quoique laide, n'est est pas moins parfaitement prête à être transformée en règle **iptables** ; on peut par exemple l'intégrer ainsi (option **--bytecode**) :

```
# iptables -A INPUT -m bpf \
--bytecode "`tcpdump -i any \
-ddd udp dst port 9000 | \
tr '\n' ',' `" -j LOG
```

Le noyau dispose, depuis plusieurs versions, d'un compilateur à la volée (JIT) d'expressions BPF, qui sera utilisé pour notre règle.

Nous avons volontairement utilisé un exemple de filtre très simple pour faciliter la compréhension. En réalité, il est même tellement peu évolué qu'une règle **iptables** « classique » aurait ici parfaitement fait l'affaire. Mais tout l'intérêt de ce nouveau module est que les filtres BPF acceptent des règles bien plus complexes et/ou évoluées : cf **man tcpdump** :-)

Pour les connaisseurs, il est effectivement déjà possible de faire dans certains cas la même chose avec le module **iptables u32,** mais sa syntaxe est plus absconse, et est en tout cas moins connue que celle de BPF. Signalons également que quelques tests rapides des développeurs du nouveau module indiquent de meilleures performances, pour une même règle, qu'avec **u32**.

2.3 Netfilter/Iptables et les labels

Ce noyau 3.9 comporte une nouvelle extension Netfilter qui permet de nommer et classifier les flux en leur attribuant un label (ou étiquette, ou tag, ou...). Il est ainsi plus facile de s'y retrouver parmi de nombreuses règles **iptables**. Le noyau ne voit, de son côté, qu'un identifiant numérique ; les labels eux-mêmes sont stockés en espace utilisateur, dans le fichier **/etx/xtables/connlabel.conf**.

Concrètement, pour nommer un flux SSH entrant sur l'interface eth0, nous ajouterions une ligne dans ce fichier en utilisant le premier identifiant numérique non utilisé (par défaut, le fichier contiendra déjà cinq entrées) :

```
# echo '6 SSH depuis exterieur' >> \
/etc/xtables/connlabel.conf
```

Maintenant, muni d'**iptables** en version minimale 1.4.19, nous pouvons définir notre règle de blocage des connexions SSH entrantes et lui indiquer de porter ce label :

```
# iptables -A INPUT -i eth0 -p \
tcp --dport 22 -m connlabel --label \
"SSH depuis exterieur" -j DROP
```

Désormais, le label apparaît bien lorsqu'on liste les règles :

```
# iptables -L | grep SSH
 DROP  tcp  --  anywhere    anywhere   tcp dpt:ssh connlabel 'SSH
depuis exterieur'
```

Nous pensons que cette nouvelle fonctionnalité d'étiquetage des règles aura, par exemple, toute son utilité avec le module **xt_bpf** précédemment décrit, étant donné qu'il sera vite difficile de s'y retrouver parmi plusieurs règles BPF entrées au format numérique.

2.4 Bridge et vlans

Le module **bridge**, permettant de réaliser une sorte de switch virtuel groupant plusieurs interfaces réseau, prend maintenant en charge les VLANs, dans le sens où il permet de définir pour chacun de ses ports (interfaces membres du bridge) ceux qui sont autorisés à transiter. Si un VLAN n'est pas autorisé sur un port, le trafic marqué avec cet identifiant de VLAN ne passera tout simplement pas par ce port. En outre, le module prend aussi en charge PVID, qui permet de définir un VLAN par défaut pour tout trafic qui n'est pas tagué.

Ces fonctionnalités sont soumises à l'utilisation d'une version très récente des utilitaires **iproute2** (version au moins égale à celle du noyau), car la commande **bridge** a reçu des options supplémentaires (**bridge vlan {show, add, delete}**) pour les prendre en charge.

3 Stockage et systèmes de fichiers

3.1 Device mapper et cache sur SSD

Le gestionnaire de volumes du noyau s'enrichit encore une fois d'une nouveauté importante : dm-cache, développée par Redhat, qui permet de définir un périphérique de stockage rapide (un SSD, par exemple) comme cache de données d'un périphérique « lent » (disque traditionnel, volume iSCSI)... Il regroupe les blocs de données en ensembles plus gros (taille configurable de 256KB à 1MB) qui sont l'unité minimale pouvant être mise en cache sur (ou retirée du) volume rapide.

L'idée est jusqu'ici simple à comprendre. Mais la partie la plus complexe est sans doute de déterminer ce qui doit, ou pas, être mis en cache. En effet, cette décision est totalement dépendante du type de données et de leur utilisation : derniers blocs lus, parce qu'on pense qu'on aura besoin d'y accéder à nouveau sous peu? Blocs les plus fréquemment accédés ? Mais ne sont-ils pas déjà dans le cache en mémoire du noyau, rendant inutile un cache sur disque ? Et, si des

blocs fréquemment utilisés ne le sont plus pendant une certaine période, faut-il envisager de les retirer du cache au profit d'autres données ? Cherche-t-on à améliorer plutôt les performances en lecture ou les écritures ?

Partant de ce constat, la décision et la gestion de ce qui est cachable (ou non) sont entièrement déléguées à des modules, qu'il est de notre responsabilité de choisir. Pour l'instant, un seul module, nommé mq, a été inclus.

L'idée n'est pas nouvelle : plusieurs solutions ont récemment été travaillées et de sérieux challengers comme bcache ou EnhancedIO par exemple, visent l'inclusion dans le noyau. Quant à dm-cache, il est inspiré d'une première implémentation d'IBM datant de 2006.

DM-cache est, en langage Device Mapper, une « cible » comme une autre. Elle a besoin d'un petit volume dédié pour garder quelques métadonnées liées à la mise en cache, et réutilise pour ce faire la cible metadata incluse en même temps que le thin-provisionning dans le noyau 3.2.

Si l'on voulait tester rapidement la création d'un volume dm-cache, il nous faudrait donc disposer de 3 volumes. Prenons la configuration suivante :

- **/dev/sda10** est notre volume lent, qui contiendra toutes les données ;
- **/dev/sda11** est sur un disque classique, il nous servira de volume de métadonnées ;
- **/dev/sdb1** est sur un SSD rapide comme l'éclair, nous l'utiliserons donc comme cache.

La commande permettant de créer notre volume **dm-cache** regroupant tout ce petit monde serait :

```
# dmsetup create mon_cache --table '0 \
$(blockdev --getsize /dev/sda10) \
cache /dev/sda11 /dev/sdb1 /dev/mapper/sda10 \
512 1 writeback mq 0'
```

En attendant que les commandes LVM (**lvcreate**...) prennent en charge ce nouveau type de volume, nous sommes obligés de passer par **dmsetup** à la syntaxe complexe, mais qui fonctionne avec tous les composants Device-Mapper.

Dans l'ordre, nous avons spécifié à **dmsetup create** :

- **mon_cache** : le nom de notre volume dm-cache, qui apparaîtra dans **/dev/mapper/**. C'est lui qu'il faudra ensuite formater, monter et utiliser !
- Ensuite, on précise les blocs de début et de fin de notre périphérique de cache. Ici, nous utiliserons la même taille que notre disque « lent », ce qui semble logique : on démarre au début du volume (**0**) et on l'utilise jusqu'à la fin, d'où l'utilisation de la commande **blockdev** pour savoir combien de blocs compte notre disque lent.
- **cache**, on l'aura deviné, correspond au type de volume que l'on crée (le type de **dm-cache** est en effet tout simplement cache). Ensuite, nous précisons, respectivement : le volume de métadonnées, le disque rapide et enfin le disque lent.
- Nous indiquons légalement la taille d'un bloc mis en cache (rappelons-nous que **dm-cache** groupe les blocs disque en ensembles plus gros), nous avons ici choisi 512. Le chiffre **1** indique que nous allons ensuite passer un seul paramètre, qui sera le mode d'écriture des données. Deux modes existent, au choix writeback (on attend seulement que les données soient écrites sur le SSD) ou writethrough qui exige qu'une écriture soit répercutée à la fois sur le disque rapide de cache et sur le disque d'origine (le lent).
- Enfin, on termine avec le choix du gestionnaire de politique de mise en cache (**mq**) et le nombre de paramètres qu'on lui passe (ici **0**).

3.2 Btrfs

Prévue et annoncée depuis longtemps, la prise en charge des modes RAID5 et 6 est désormais effective.

Le mode RAID5, très populaire, nécessite 3 disques au minimum, et utilise un espace constant, équivalent à 1 disque (quel que soit le nombre de disques total de l'ensemble) pour stocker les infos de parité permettant de récupérer les données en cas de besoin. Les données sont découpées en « bandes » (fixées à 64 KB dans le cas de Btrfs), chacune écrite sur un disque différent. Un ensemble RAID 5 tolère la perte d'un seul disque, mais est très apprécié pour ses bonnes performances et le peu d'espace consommé par les données de redondance.

Moins utilisé, le mode RAID6, quant à lui, maintient deux fois plus d'informations de parité, afin de tolérer la perte de 2 disques maximum. Le coût de maintien de ces informations est nettement plus élevé qu'en RAID5 et l'impact sur les performances plus important.

Rappelons que gérer la redondance RAID au niveau système de fichiers, comme le fait Btrfs, et non au niveau volume comme les solutions traditionnelles de RAID, offre des avantages indiscutables : le système de fichiers sait mieux que personne quelles sont les données réellement à copier/synchroniser, on évite ainsi de copier des zones inutilisées du volume. Cela permet également de détecter et réparer à chaud les données corrompues.

Autant calmer les ardeurs tout de suite, ce n'est pas encore le moment de déployer du Btrfs en RAID5 à tour de bras, la fonctionnalité est expérimentale (rappelons que le système de fichiers est également toujours dans ce cas) et partielle. Par exemple, la correction d'erreurs grâce aux sommes de contrôles CRC, qui est normalement réalisée lors d'un **btrfs scrub**, n'est pas encore implémentée. La taille de bande (stripe) est pour le moment fixée à 64KB, et il y a un risque de corruption (non réparable) des informations de parité lors d'un crash. Enfin, le remplacement d'un disque défaillant ne fonctionne pas encore ; autant de points rédhibitoires pour une utilisation réelle, mais qui devraient être réglés dans les tout prochains noyaux. Par contre, les premiers tests

GREAT PLACE TO WORK Best Workplaces 2013 France

UNIVERSUM TOP 100 IDEAL EMPLOYER 2013 STUDENT SURVEY

IMAGINER ET CONCEVOIR UN MONDE SANS CONTRAINTE

Technologies de pointe, challenges complexes en systèmes embarqués, ambitions internationales. Avec Parrot, faites le pari de l'innovation dans l'univers des périphériques sans fil.

BOUSCULEZ TOUS LES CODES SUR
WWW.RECRUTE.PARROT.COM

UN MONDE D'INNOVATIONS AU CŒUR DE PARIS

Parrot

de Chris Mason (fondateur et développeur principal du projet Btrfs) montrent des performances largement supérieures à celles du RAID logiciel générique du noyau (MD).

Pour récapituler, Btrfs prend maintenant en charge les modes RAID0, 1, 10, 5 et 6, avec possibilité de configurer une redondance différente pour les données et les métadonnées. Dans un futur proche, on peut s'attendre à voir s'ajouter à cette liste les assemblages en « N-miroir » (volumes répliqués complètement, comme en RAID1, mais avec N supérieur à deux volumes).

Autre nouveauté, moins tape-à-l'œil, mais importante dans le cadre d'une utilisation réelle de Btrfs : lancer une défragmentation n'entraînera désormais plus de surconsommation d'espace disque. Le comportement par défaut de Btrfs fait que, si l'on prend un snapshot d'un système de fichiers, tant que les données ne sont pas modifiées, les données du snapshot pointent sur les blocs « réels ». C'est ce qui permet que la création de snapshots soit instantanée et sans surcoût de base en espace disque.

Cependant, jusqu'à présent, lancer une défragmentation brisait le partage des blocs de données entre système de fichiers « normal » et snapshot et entraînait leur duplication. Problème désormais réglé.

4 Gestion de l'énergie

4.1 Suspend-freeze

Linux possède deux modes principaux de mise en veille : le suspend-to-disk (hibernation) et la mise en veille en RAM. Le premier éteint complètement le système, qui ne consomme plus aucune énergie électrique. Le second ne laisse alimentés que la mémoire et son contrôleur. Le noyau 3.6 a ajouté une mise en veille hybride qui est la combinaison des deux précédentes (RAM + disque). Cette fois, c'est une nouvelle manière de faire qui est proposée : extinction des périphériques, gel des processus du système, mais le processeur et la RAM sont toujours actifs (juste mis en veille). C'est beaucoup plus économe qu'un système complètement éveillé, moins qu'un système dont seule la mémoire reste alimentée ; et surtout, le système est beaucoup plus rapide à « réveiller », à revenir dans un état pleinement opérationnel. Ce mode est appelé freeze ; de manière similaire aux autres modes existants, il peut être déclenché via un simple **`echo freeze > /sys/power/state`**.

4.2 Intel Powerclamp

Le nouveau pilote Powerclamp fait partie des modules de régulation thermique. Il permet de définir manuellement un pourcentage de temps pendant lequel on forcera le processeur à ne rien faire : si le taux d'inoccupation du CPU passe en dessous de la valeur fixée, un thread noyau le fera passer en veille (en utilisant les C-states des processeurs Intel) pendant un certain laps de temps. Cette solution permet de limiter le dégagement thermique d'un processeur, mais aussi sa consommation d'énergie de pointe. Pour l'activer, il faut trouver l'entrée sysfs correspondante, elle aura un dossier du type **`/sys/class/thermal/cooling_deviceX/`**, contenant une entrée type de valeur **`intel_powerclamp`**. On écrira le pourcentage de mise en sommeil voulu dans l'entrée **`cur_state`**, par exemple :

```
# echo 25 > /sys/class/thermal/cooling_device5/cur_state
# dmesg | tail -1
intel_powerclamp: Start idle injection to reduce power
```

5 Affichage graphique

5.1 Nvidia et gestion multi-GPU

Nous l'avons déjà évoqué à l'occasion de la sortie de noyaux précédents, une infrastructure permettant de prendre en charge les ordinateurs portables dotés de deux puces graphiques (l'une performante et gourmande, l'autre moins performante, mais plus économe en énergie) se met doucement en place dans le noyau. Dans un premier temps, l'objectif est de pouvoir choisir manuellement quelle puce doit être active, et de permettre de faire transiter les données à afficher jusqu'à l'écran s'il est connecté à l'autre puce.

Cette infrastructure contient des fonctions génériques formant une ABI qui peut être utilisée pour tous les pilotes graphiques. Les développeurs noyau, vigoureusement appuyés par Linus Torvalds, ont choisi de n'en autoriser l'accès qu'aux pilotes dont le code est sous licence GPL. En effet, Linux permet de définir explicitement une fonction comme faisant partie de son ABI (elle est alors « exportée »), tout en précisant quels types de codes peuvent l'utiliser, grâce aux fonctions **`EXPORT_SYMBOL()`** (tous) ou **`EXPORT_SYMBOL_GPL()`** (code GPL uniquement). Bien sûr, ce choix a délibérément été fait pour empêcher le pilote propriétaire de Nvidia d'y accéder.

Ce noyau intègre donc une couche d'abstraction supplémentaire, développée par Nvidia , qui sépare complètement l'implémentation réelle (du gestionnaire de mémoire graphique du noyau, GEM) des pilotes ; cette nouvelle interface peut bien évidemment être utilisée par du code non-GPL. En parallèle, Nvidia a également sorti une nouvelle version de son pilote propriétaire qui peut utiliser cette nouvelle interface.

5.2 Nouveau et ATI

Le pilote pour puces Nvidia **`nouveau`** (celui qui est libre) commence à pouvoir gérer les ventilateurs, pour les cartes Geforce 6 à 9 et 100 à 300. Si jusqu'à présent ils vrombissaient en permanence au maximum, un mode automatique peut désormais être testé via un **`echo AUTO > /sys/class/hwmon/hwmonX/pwm1_enable`**.

Le pilote pour puces ATI gère quant à lui les chipsets graphiques Radeon HD 8500, 8600 et les futures puces Richland. ■

À NE PAS RATER !

OPEN SILICIUM N° 7

DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

JEKYLL : CRÉEZ UN SITE STATIQUE AVEC LE CONFORT D'UN CMS

par Stéphane Mourey

Les langages des scripts tels que PHP, les bases de données telles que MySQL et même les exécutables CGI écrits en C auront beau faire, ils ne seront jamais aussi rapides que de simples pages statiques. Malheureusement, il est souvent laborieux d'en maintenir un grand volume et une légère modification de design peut tourner au défi titanesque. Jekyll, écrit en Ruby, a été conçu pour pallier cette difficulté vous offrant les outils nécessaires tant pour gérer votre contenu que sa présentation.

1 Installation

Si votre distribution fournit un package pour Jekyll, vous essayerez de l'installer comme à l'ordinaire. Ici, la commande d'installation pour Ubuntu :

```
$ sudo apt-get install jekyll
```

Mais, au moins en ce qui concerne Ubuntu, je ne vous le recommande vraiment pas. En effet, la version qui se trouve dans les dépôts est antédiluvienne au moment où j'écris ces lignes, il s'agit d'une version 0.10, alors la version disponible par ailleurs est la 1.0.0. Et Jekyll a eu le temps de bien changer dans l'intervalle et rien de ce qui sera expliqué dans cet article n'a la moindre chance de fonctionner avec l'ancienne version.

Avant de procéder à l'installation de Jekyll selon la méthode recommandée, il sera peut-être nécessaire d'installer le paquet de développement associé à Ruby ainsi que **gem**, l'utilitaire d'installation d'application Ruby, s'il ne l'est pas déjà :

```
$ sudo apt-get install ruby1.9.1-dev rubygems
```

Ensuite, vous pourrez utiliser **gem** pour installer Jekyll :

```
$ sudo gem install jekyll
```

Après cela, vous aurez peut-être besoin de réinitialiser votre environnement bash en vous déconnectant ou en ouvrant une nouvelle console. Je n'ai pas investigué pour en déterminer les causes, mais il se trouve que la commande **jekyll**, bien que disponible depuis l'auto-complétion de la ligne de commandes, provoque une erreur indiquant son indisponibilité si elle est exécutée immédiatement après l'installation...

Si vous envisagez de publier du code sur votre nouveau site, vous aurez peut-être envie d'installer le module de coloration syntaxique :

```
$ sudo apt-get install python-pygments
```

Enfin, si vous envisagez de parler de mathématiques, vous serez heureux d'apprendre qu'un module existe qui vous permettra de convertir automatiquement du LaTeX en PNG pour la publication de vos formules sur le web. Nous n'explorerons pas cette possibilité dans cet article, qui m'a d'ailleurs paru assez peu documentée. Si cela vous intéresse, le point d'entrée est ici : http://jekyllrb.com/docs/extras/#latex_support

2 Première utilisation

Jekyll est un outil en ligne de commandes. Point de jolie interface pour vous assister dans la réalisation de vos designs, Jekyll se situe plus loin dans la chaîne de production web, et il vous faudra maîtriser l'intégration web si vous voulez personnaliser quelque peu vos pages.

Avant d'aller plus loin, vérifions si tout marche bien, créons un projet de site :

```
$ jekyll new mon_site
```

Et lançons le serveur web de développement intégré :

```
$ cd mon_site
$ jekyll serve
```

Observons le résultat dans notre navigateur, en ouvrant l'URL **http://localhost:4000** (voir Figure 1).

Figure 1

Nous découvrons un site pour le moins fruste, demandant encore de nombreux paramétrages, avec un lien vers un article « Welcome to Jekyll ». Vous remarquerez que Jekyll propose d'emblée une orientation blog à votre site. Il n'est nullement obligatoire de la suivre, mais cela vous donne un cadre pour commencer. Si vous prenez le temps de lire l'article de bienvenue, vous découvrirez qu'il est possible de paramétrer votre serveur web de développement pour qu'il répercute immédiatement vos modifications en le lançant avec l'option **`-w`** ou **`--watch`**. Cela vous permettra de voir le résultat sans avoir à reconstruire le site de nombreuses fois avant la construction pour publication. Attention toutefois : cela n'est pas vrai pour le fichier **`_config.yml`**, qui contient la configuration globale du site, nous en reparlerons. Pour que des modifications dans ce fichier soient prises en compte, il faudra redémarrer votre serveur Jekyll.

Avant de voir comment ajouter du contenu et améliorer sa présentation, donnons quelques indications sur la marche à suivre pour publier ce contenu.

3 Publication simple

Pour générer votre site, dans le répertoire de votre projet, il suffit de lancer la commande :

```
$ jekyll build
```

Vous verrez alors apparaître un répertoire **`_site`** contenant toutes les pages HTML statiques de votre site. Si vous affichez le fichier **`index.html`** dans votre navigateur, vous remarquerez vite que les fichiers CSS manquent à l'appel. Un rapide examen du code source vous expliquera le problème : l'emplacement des fichiers CSS est indiqué depuis la racine du site, ce qui ne convient pas pour une consultation hors ligne, ni pour un site qui se trouverait dans un sous-répertoire.

La solution consiste à modifier le fichier **`_layouts/default.html`** qui contient le modèle de page. Il faut y définir une balise **`base`** dans le **`head`** du fichier **`_layouts/default.html`** (nous en reparlerons au chapitre 4.1.2 sur les liens vers des ressources) :

```
<base href="{{ site.url }}" />
```

Dès lors, toutes vos URL relatives prendront le contenu de la variable **`{{ site.url }}`** comme point de départ, ce qui allégera d'autant la syntaxe. Il ne vous reste plus qu'à régénérer le site.

Une fois votre site construit d'une façon satisfaisante, il ne vous reste plus qu'à copier le contenu du répertoire **`_site`** dans l'arborescence de votre serveur web à l'endroit approprié. Remarquez que vous n'êtes pas contraint de publier votre site sur un serveur web, mais que vous avez une arborescence autonome que vous pourrez utiliser pour une consultation off-line, comme sur un CD par exemple.

Il est à noter que la commande **`jekyll build`** supporte également l'option **`--watch`**, ce qui permet de reconstruire le site automatiquement dès qu'une modification a été détectée dans les sources, solution que sous la forme d'un fichier je ne recommande pas pour un site de production... Il est également possible de préciser à cette commande le répertoire du projet avec l'option **`--source`**, ainsi que le répertoire de publication avec l'option **`--destination`**.

Il est possible de ne pas avoir à préciser les différentes options à chaque fois qu'on utilise une commande, car celles-ci peuvent être spécifiées dans un fichier de configuration **`_config.yml`**. La configuration par défaut est présentée sur le site officiel de Jekyll sous la forme d'un texte obéissant à la même syntaxe à l'adresse http://jekyllrb.com/docs/configuration/. Cette même page vous présentera de manière exhaustive toutes les options supportées par **`jekyll`**. **Attention ! N'utilisez pas de tabulation dans les fichiers de configuration de Jekyll, sans quoi une erreur sera levée, et Jekyll reprendra les paramètres par défaut.**

Le format utilisé pour ce fichier de configuration est YAML, acronyme récursif de YAML Ain't Markup Language (Yaml n'est pas un langage de balisage). YAML est né d'une critique constructive

de XML : Clark Evans trouvait qu'il était possible de créer un format de fichier ayant les mêmes fonctionnalités que le XML tout en étant plus lisible, tant pour un être humain que pour un ordinateur, et il concrétisa cette idée avec YAML. Pour découvrir les bases de YAML, la page Wikipédia sur le sujet (**http://fr.wikipedia.org/wiki/YAML**) est un bon point de départ., vous pourrez ensuite approfondir depuis le site officiel (http://www.yaml.org). Nous verrons que Jekyll en fait usage en plusieurs endroits, sans qu'il soit pour autant nécessaire d'en faire une étude approfondie.

4 Ajouter du contenu

4.1 Écrire des articles

Jekyll étant d'emblée orienté blog, et bien qu'il soit possible de l'utiliser avec une orientation différente, le plus naturel est d'abord d'examiner comment ajouter de nouveaux posts. Si vous avez pris le temps de lire l'article « Welcome to Jekyll », vous aurez aussi découvert que pour ajouter un nouvel article à votre blog, il suffit d'ajouter un fichier dans le répertoire **_posts** de votre projet en suivant la convention de nommage pour votre fichier **YYYY-mm-dd-mon-titre.ext**. Ce fichier devra commencer par un entête dont la structure est déterminée :

```
---
layout : post
title : "Mon titre"
date : 2013-05-07 15:30:28
categories : news
---
Ici, vous placez le contenu de votre article.
```

L'entête comporte quatre lignes séparées du reste par trois tirets sur une ligne au début et à la fin. La première ligne indique le modèle de page ou layout à utiliser pour l'affichage du contenu. La seconde indique le titre de l'article. La troisième indique sa date de publication, et la quatrième la catégorie de l'article. À la suite de cet entête, vous placez le reste de l'article.

En réalité, cet entête est ce que les concepteurs de Jekyll appellent un front-matter, un bloc Yaml permettant la définition de variables, dont certaines sont prédéfinies. Aux précédentes, nous pouvons ajouter :

- published : définir à false si vous souhaitez que l'article ne soit pas publié ;
- permalink : permet de définir l'URL d'accès à l'article si vous voulez utiliser un format différent de celui par défaut ;
- tags : fonctionne comme les catégories.

Ces variables sont prédéfinies également pour les pages statiques que nous allons voir un peu plus loin.

Cet entête Yaml permet la définition de variables personnalisées que vous pourrez ensuite utiliser dans vos layouts. Par exemple, si vous définissez la variable **mavar**, vous pourrez l'appeler dans votre template avec la syntaxe **{{ page.mavar }}**. Pour traiter les layouts, Jekyll utilise Liquid (**https://github.com/Shopify/liquid/wiki**), un moteur de templates libre développé par le service en ligne Shopify.

Bon, vous allez me dire que publier du texte sur des pages HTML, c'est bien, mais que nous avons là une vision un peu réductrice des capacités du Web et que, ma foi, tout cela manque un peu de balises. Et vous auriez bien raison. C'est pour cela que Jekyll supporte non pas un, mais trois langages de balisage : le HTML habituel, le markdown (http://daringfireball.net/projects/markdown) et le Textile (http://textile.sitemonks.com)...

4.1.1 Markdown

Comme il y a de fortes chances que vous connaissiez déjà bien le HTML avant de lire cet article, prenons le temps de dire quelques mots concernant l'utilisation de Markdown, qui vous permettra sans aucun doute de gagner du temps lors de vos saisies.

Markdown a été conçu pour être facile à écrire et, pour cela, s'appuie sur des conventions de texte existantes. Le texte ainsi produit est destiné à être ensuite converti en HTML pour publication sur le web.

Sans vouloir faire une description extensive de la chose, signalons quelques éléments qui vous permettront d'en faire déjà un usage intéressant.

La première chose à savoir est que vous pouvez très bien mettre un bloc de texte en HTML pur dans un fichier Markdown, cela ne posera pas de problème si vous prenez la précaution de séparer ce bloc des autres paragraphes par au moins une ligne vide. Il faut savoir qu'alors votre bloc HTML ne sera pas interprété par Markdown, mais copié à l'identique. Cela dit, cette restriction ne s'applique qu'aux éléments de type bloc tels que **div**, **p**, **table**, **pre**, etc. En ce qui concerne les éléments en ligne tels que **span** ou **em**, par contre le mélange peut très bien être opéré sans difficulté, et le Markdown sera traité à l'intérieur des balises.

Un retour à la ligne est converti en retour à la ligne HTML (**br**). Une ou plusieurs lignes vides servent à séparer des paragraphes.

Pour mettre un texte en italique, entourez-le de deux astérisques ou de deux tirets :

```
_en italique_ *aussi*
```

Doublez pour mettre en gras :

```
__en gras__ **aussi**
```

Pour créer un titre de niveau 1, il suffit de le « souligner » de signes « égal » ou encore de le précéder d'un dièse:

```
1. Mon titre de haut niveau
==================
# 2. Mon second titre de haut niveau
```

Pour un titre de niveau 2, de simples tirets suffisent ou encore deux dièses :

```
1.1. Mon titre d'un peu moins haut niveau
-------------------------------------------
# 1.2. Un autre titre d'un peu moins haut niveau
```

Pour atteindre les titres de niveaux moindres, il reste plus qu'à ajouter autant de dièses que de niveaux, jusqu'à six, le maximum permis par le HTML :

```
###### 1.2.3.4.5.6. Un tout petit titre...
```

Il n'y a plus à partir du troisième niveau inclus de syntaxe à base de soulignement.

Vous pouvez créer un bloc de citation en utilisant le signe « supérieur à » en début de ligne :

```
> elle pleura pour toute l'humanité, sur la fragile beauté des corps
auxquels ils avaient renoncé, il y avait des trillions d'années.
```

Les citations peuvent être imbriquées, en utilisant autant de signes « supérieur à » que de niveaux d'imbrication, séparés par un espace :

```
> > Qui était-il ?
> Et Elvex répondit :
> > Ce robot, c'était moi.
```

Les citations peuvent contenir d'autres éléments Markdown sans souci.

Il est possible de créer des listes tout aussi facilement : pour une liste non-ordonnée, on utilisera les signes astérique, plus ou encore tiret ; pour les listes ordonnées, il suffit de numéroter les lignes, en faisant suivre le numéro d'un point.

```
1. Une pierre
2. deux maisons
3. trois ruines
4. quatre fossoyeurs
. un jardin
. des fleurs
```

L'ordre des numéros que vous indiquez n'aura pas d'incidence sur le HTML produit. Le principe est qu'un nombre suivi d'un point en début de ligne entraîne le HTML correspondant à un nouvel item d'une liste ordonnée : **<li>**.

Si vous souhaitez placer du contenu Markdown dans une liste, il suffit de l'indenter (une tabulation ou quatre espaces, pas deux) :

```
1. Première loi de la robotique
  > Un robot ne peut porter atteinte à un être humain, ni, restant
passif, permettre qu'un être humain soit exposé au danger.
```

Pour produire un bloc de code, il faut indenter la première ligne (du coup, à l'intérieur d'une liste, il faut indenter deux fois...). Un bloc de code se termine à la première ligne qui n'est indentée. Pour traiter ce bloc, Markdown placera les deux balises HTML **pre** et **code**. À l'intérieur d'un bloc de code, les signes spéciaux nécessaires sont convertis pour que le code HTML soit affiché, et non traité. Vous pouvez donc facilement donner des exemples de code HTML, mais ne pas mettre de HTML dans un bloc de code...

Si vous voulez indiquez qu'une portion de texte correspond à du code, mais sans produire un bloc de code pour autant, utilisez les guillemets obliques :

```
Utilisez la fonction `fopen` pour ouvrir un fichier.
```

Vous obtiendrez une ligne de séparation horizontale (**<hr />**) en mettant seuls sur la ligne au moins trois astérisques ou tirets, éventuellement séparés par des espaces.

Pour créer un lien, il faut placer le texte du lien entre crochets et faire suivre l'URL de destination entre parenthèses. Éventuellement, vous pouvez ajouter un texte pour l'infobulle le plaçant dans la parenthèse, entre guillemets, après l'URL :

```
Le [site des Éditions Diamond] (http://www.unixgarden.com "Retrouvez
le site de Gnu/Linux Magazine France")
```

Toutefois, si vous voulez simplement indiquer l'URL sans texte, vous pouvez le faire en l'entourant simplement de chevrons :

```
<http://www.unixgarden.com>
```

Cette URL peut très bien être une adresse mail :

```
<stephane.mourey@exemple.com>
```

Pour indiquer une image, la syntaxe est très comparable à celle d'un lien, à la différence que les crochets sont précédés d'un point d'exclamation :

```
![Texte de remplacement](http://exemple.com/monImage.png "Texte pour
l'infobulle")
```

Enfin, vous aurez peut-être besoin d'un caractère d'échappement pour pouvoir empêcher le traitement par Markdown d'éléments que vous voulez voir affichés. Ce caractère est la barre oblique inversée (\).

Pour en savoir plus, consultez le site officiel : http://daringfireball.net/projects/markdown, ou une explication détaillée en français de la syntaxe : http://michelf.ca/projets/php-markdown/syntaxe

Enfin, dernière chose à savoir pour pouvoir utiliser Markdown avec Jekyll : Jekyll se base sur l'extension de fichier pour savoir avec quel langage il doit l'interpréter. Ainsi, dans notre premier exemple en HTML, nous avions un fichier dont l'extension était **.html**. Pour utiliser Markdown, il suffit d'utiliser l'extension **.markdown** ou **.md**.

4.1.2 Ajouter des images et d'autres ressources

Nous avons vu que Markdown permettait d'ajouter des liens, qui peuvent pointer vers vos ressources, et d'insérer des images. Mais où faut-il placer les fichiers et comment rédiger les liens pour que Jekyll les génère correctement ?

Jekyll est de ce point de vue agnostique. Vous pouvez les placer où bon vous semble du moment qu'il ne s'agit pas d'un des répertoires de travail de Jekyll (ceux dont le nom commence par un underscore). En fait, Jekyll prend tous vos fichiers et tous vos répertoires et les recopie à l'emplacement équivalent dans le site généré. Bien sûr, le mieux est sans doute de placer les images dans un répertoire dédié (**/img** pour la suite de notre article), de même que pour les autres ressources. Vous n'avez de ce point de vue pas à vous adapter à Jekyll, c'est Jekyll qui s'adapte à vous.

En ce qui concerne l'écriture des liens, pour obtenir l'URL correspondante à l'emplacement réel des fichiers, vous utiliserez la variable Liquid **site.url** :

```
![Le joli logo]({{ site.url }}/img/logo.png)
```

Cette solution est celle recommandée par la documentation officielle de Jekyll. L'autre se base sur l'URL d'origine par défaut que nous avons ajoutée au chapitre 3 pour lier les fichiers CSS. Ainsi, le code précédent peut être remplacé par celui-ci :

```
![Le joli logo](/img/logo.png)
```

4.1.3 Coloration syntaxique

Si vous avez installé le module Pygments, voici la syntaxe à utiliser pour obtenir la coloration syntaxique :

```
{% highlight php linenos %}
<?php
function test($a) {
  // ceci est une fonction de test
  print "Hello world !";
  return true;
}
{% endhightlight %}
```

L'instruction Liquid **highlight** déclenche la coloration syntaxique, elle est suivie immédiatement du nom du langage dans lequel le bloc à colorer a été écrit. Enfin, optionnellement, vous pouvez ajouter l'argument **linenos** qui permet d'ajouter les numéros de lignes. En passant, vous pouvez dès lors vous dispenser de l'indentation prévue par Markdown pour identifier un bloc de code.

Malheureusement, cela ne suffira pas pour que cela fonctionne, car il vous manque les fichiers CSS nécessaires. Vous pouvez les obtenir en exécutant la commande suivante dans le répertoire de votre projet :

```
$ pygmentize -S default -f html > stylesheets/
pygments.css
```

Il faudra alors inclure ce nouveau fichier CSS dans **_layouts/default.html**.

À noter que si vous utilisez Twitter Bootstrap, il y a une petite incompatibilité en son CSS et celui que vous venez de produire concernant l'arrière-plan de la balise code. Une solution consiste à modifier la ligne suivante du **bootstrap.min.css** de sorte à en évacuer l'attribut **background**.

```
code{background-color:#fee9cc;color:rgba(0,
0, 0, 0.75);padding:1px 3px;}
```

4.2 Ajouter ou simuler des pages statiques

4.2.1 Ajouter

Les pages statiques obéissent aux mêmes principes que les articles de blog, seule change la position des fichiers dans l'arborescence. Jekyll conservant les fichiers à leur emplacement à l'exception des répertoires réservés, il serait théoriquement possible de les placer n'importe où, à l'exception de la page d'accueil qui doit être **index.html** à la racine du projet. Toutefois, il est sans aucun doute préférable de suivre l'une des deux méthodes préconisées par la documentation : soit de créer toutes vos pages dans des fichiers HTML à la racine du site, soit de créer un dossier pour chacune d'elles dans un répertoire dédié et de placer le contenu dans un fichier **index.html** de ce nouveau répertoire. La principale différence entre les deux méthodes tient à l'URL qui va permettre d'accéder aux pages. Avec la première méthode, l'URL se terminera nécessairement par l'extension **.html**, alors qu'avec la seconde méthode, il n'y aura pas cette extension.

Le problème qui se pose alors est que ces pages statiques ne sont pas listées automatiquement, à aucun endroit. Vous pouvez naturellement écrire une liste à la main, mais vous y perdez en confort et l'intérêt de Jekyll comme système de gestion de contenu est alors très limité, pour peu que vous ne vouliez pas réaliser un blog.

4.2.2 Simuler

Dans le cas où vous abandonnez complètement toute idée d'un blog, il est possible de configurer Jekyll de telle sorte que ce qu'il considère comme des articles de blog apparaît comme des pages statiques sur le site généré. Il y a même assez peu de choses à faire pour le réaliser : il suffit de modifier trois fichiers.

Tout d'abord, nous allons changer l'URL d'adressage des articles pour que la date n'y apparaisse plus. Pour cela, il faut modifier ou ajouter la directive **permalink** du fichier **_config.yml** comme suit :

```
permalink: /:title
```

ce qui aura pour effet que l'URL ne comprendra plus que le titre de l'article. Si vous souhaitez que la catégorie apparaisse, vous pouvez utilisez le raccourci suivant :

```
permalink: none
```

Il nous faut également modifier la liste des articles de la page d'accueil en modifiant le fichier **index.html**, dans lequel il faudra supprimer la chaîne de caractères suivante :

```
<span>{{ post.date | date_to_string }}</span>
```

ce qui aura pour effet de supprimer l'affichage de la date de l'article.

Enfin dans **_layouts/post.html**, on supprimera la chaîne :

```
<p class="meta">{{ page.date | date_to_string }}</p>
```

ce qui aura pour effet de supprimer l'indication de date dans le détail de l'article.

Une fois tout cela effectué, la liste des articles ressemblera à s'y méprendre à une liste de pages statiques. Mais vous

voudrez sans doute contrôler l'ordre de ces pages dans la liste. Pour cela, il suffit de se rappeler que Jekyll affiche les articles dans l'ordre inversement chronologique. La date prise en compte pour ce tri est d'abord celle définie par le nom de fichier selon la convention de nommage que nous avons vue au chapitre 4.1, puis par le champ date de l'entête YAML du fichier, cette dernière valeur écrasant la première. Personnellement, dans le cas de pages statiques, je vous recommanderai de ne plus utiliser ce champ, mais seulement le nom du fichier. Vous aurez ainsi une visualisation directe de l'ordre de vos pages en examinant les fichiers de votre répertoire triés par nom.

4.3 Arborescence autonome

Jekyll est également un outil intéressant pour produire une arborescence de pages hypertextes consultables hors ligne sans serveur, sur un CD par exemple, mais il faut alors prendre quelques précautions. En particulier, il n'est plus possible de s'appuyer sur la balise HTML `base` pour définir l'emplacement à partir duquel les URL sont définies par la suite. De même, on ne peut pas utiliser la variable `{{ site.url }}` qui ne peut pas être définie dans ce contexte. La solution la plus simple consiste à définir un layout différent pour chaque niveau possible de l'arborescence, a priori un pour l'index, un pour les pages fixes si vous adoptez la solution "un répertoire par page", et un pour les articles. Dans chacun d'eux, il faudra indiquer l'emplacement des fichiers CSS relativement à la position des fichiers générés.

Conclusion

Dans cet article, nous vous avons donné un aperçu des grandes possibilités offertes par Jekyll. Certaines directions mériteraient d'être approfondies, comme l'utilisation de brouillons, d'articles dans le futur ou la pagination (qui ne fonctionne malheureusement qu'avec les fichiers HTML...).

En particulier, le fait que Jekyll stocke l'intégralité de ses données dans des fichiers fait qu'il fonctionne particulièrement bien avec des outils de gestion de versions et en particulier avec Git. Si ensuite vous paramétrez un hook au commit, vous aurez vite fait de mettre en place un système de publication particulièrement rapide.

Notons également qu'il existe déjà un nombre important de plugins à utiliser avec Jekyll qui pourront vous permettre d'en étendre les possibilités. Vous pourrez faire votre marché sur cette page : http://jekyllrb.com/docs/plugins/#available_plugins

Enfin, sachez qu'il existe des méthodes de migration depuis nombre d'autres systèmes de blogs : WordPress, Movable Type, Blogger, Tumblr... ■

CENTRALISATION ET SUPERVISION DES LOGS AVEC RSYSLOG ET NAGIOS/NRDP

par Stanislas LEVEAU
[Administrateur Systèmes et Réseaux au Rectorat de Caen]

La supervision est un vaste sujet, avoir une vue d'ensemble de son infrastructure en temps réel, localiser les problèmes, les résoudre, limiter le temps d'indisponibilité des services... tout un programme.

Nous avons souvent tendance à utiliser des scripts pour faire des contrôles et nous fournir un code de retour OK WARNING CRITICAL, mais une fâcheuse tendance à oublier les logs, vous savez ces petits fichiers qui fournissent une mine d'informations sur l'état de votre système, tout ce qui transite, les mises à jour, les défaillances, les tentatives d'intrusions... Aujourd'hui, nous allons leur accorder un peu plus d'importance en les centralisant, les analysant et en avertissant notre administrateur systèmes et réseaux en cas de problème.

1 Rappel

1.1 Qu'est-ce que les logs

Les logs sont un historique d'événements enregistrés séquentiellement dans un fichier ou une base de données (application, activité d'un réseau informatique...).

Le journal (en anglais log file ou plus simplement log) désigne alors le fichier contenant ces enregistrements.

Plusieurs intérêts de centraliser les logs : l'administrateur système et/ou réseau recevra un seul rapport quotidien plutôt que n rapports de n machines. En cas d'intrusion d'une machine, même si le pirate parvient à effacer ses traces, nous aurons de toute façon une copie de celles-ci.

1.2 Le format des logs

Syslog est un protocole définissant un service de journaux d'événements d'un système informatique. C'est aussi le nom du format qui permet ces échanges.

Un journal au format syslog comporte dans l'ordre les informations suivantes : la date à laquelle a été émis le log, le nom de l'équipement ayant généré le log (hostname), une information sur le processus qui a déclenché cette émission, le niveau de gravité du log, un identifiant du processus ayant généré le log et enfin un corps de message.

Exemple :

```
Sep  3 21:14:02 nagios rsyslogd: [origin software="rsyslogd"
swVersion="4.6.4" x-pid="3372" x-info="http://www.rsyslog.com"] (re)start
```

La structure d'un message de logs est la suivante : 2 types d'information sont utilisés pour caractériser un message de log.

- Niveaux de gravité

→ Les niveaux de gravité Syslog, souvent appelés level sont au nombre de huit représentés par un chiffre de 0 (Emergency) à 7 (Debug) :

→ **0 Emerg (emergency)** : système inutilisable ;

→ **1 Alert** : une intervention immédiate est nécessaire ;

→ **2 Crit (critical)** : erreur critique pour le système ;

→ **3 Err (error)** : erreur de fonctionnement ;

→ **4 Warning** : avertissement ;

→ **5 Notice** : événement normal méritant d'être signalé ;

→ **6 Info (informational)** : pour information seulement ;

→ **7 Debug** : message de mise au point.

- Facilités

→ Outre les niveaux de gravité, les messages sont orientés en fonction de leur origine, dont les codes sont regroupés suivant 24 types énumérés ci-dessous :

→ type (facilities) : **authpriv**, **cron**, **daemon**, **kern**, **local0 -> local7**, **lpr**, **mail**, **news**, **syslog**, **user**, **uucp**, **ftp**, **mark**

→ **Auth/authpriv** : traces sécurité/identifiant ;

→ **cron** : traces d'un cron ;

→ **daemon.*** : trace d'un daemon ;

→ **kern.*** : traces du noyau ;

→ **lpr.*** : traces du système d'impression ;

→ **mail** : traces du système de messagerie ;

→ **news** : traces d'un service de news/réseau ;

→ **syslog** : traces du service syslog lui-même ;

→ **user** : trace des processus utilisateur ;

→ **local0 à 7** : traces issues des klogd.

Pour plus d'informations : **[1]**

1.3 Les logs et la loi

Poursuivons avec un peu de droit sur la conservation des logs.

Les logs de messagerie : les entêtes des messages (entêtes SMTP, RFC 2822, incluant le champ objet, et les entêtes d'extensions MIME ou propriétaires) doivent être conservés pendant un an à compter du jour de l'enregistrement.

Les données de connexion : les opérateurs télécoms, FAI et cybercafés devront dorénavant conserver pendant un an les données relatives au trafic des communications électroniques.

2 Objectifs

Centraliser, filtrer, alerter. Ces trois verbes résument bien le but de cet article, mais entrons un peu plus dans les détails.

L'objectif est de centraliser sur un même serveur plusieurs sources de logs : les logs venant de serveurs Linux gérés par syslog ou non. Logs provenant de serveurs Windows du gestionnaire d'événements ou bien de différentes applications.

Une fois les logs centralisés, ils sont filtrés à l'aide de règles de filtrage rsyslog initialisées par nous-mêmes en fonction de différents critères de recherche. Une fois les logs correspondants à notre filtre sélectionnés, l'administrateur est alerté.

Pour illustrer cet article, nous allons nous appuyer sur un exemple. Nous avons un système anti-virus qui contrôle les flux de mails entrants et sortants. L'antivirus est composé du logiciel libre Clamav qui, dans notre configuration, met à jour toutes les heures sa base antivirale avec l'aide de l'outil Freshclam. À chaque mise à jour, il va marquer sa mise à jour dans les logs ET chose intéressante, il précise lorsque la version du logiciel utilisé (donc ici nous parlons bien de la version d'installation de Clamav et non de la version de sa base antivirale) - tout le monde suit encore ?? - est obsolète ou non par une ligne qui correspond à ceci : **Your ClamAV installation is OUTDATED**.

Une non mise à jour de ce logiciel peut représenter une faille de sécurité au niveau de notre système d'analyse antivirale d'où l'importance d'en être averti rapidement et de pallier à cette potentielle faille.

Il faut donc centraliser (pas forcément, mais cela est plus simple lorsqu'on a plusieurs serveurs dont nous voulons filtrer les logs. Cela permet de centraliser les règles de filtrage et également d'avoir une copie en cas d'intrusions et d'effacement des logs sur le serveur piraté.) ses logs sur un serveur de logs et les filtrer. Si notre règle de filtrage correspond à cette chaîne de caractères, en avertir Nagios qui nous enverra une alerte mail ou SMS en s'appuyant sur NRDP pour faire le lien entre le serveur de logs et l'outil de supervision Nagios. Voici l'architecture mise en place...

3 Architecture

3.1 Outils

- Logiciel de supervision : Nagios 3.x **[2]**

- Logiciel de gestion de logs : Rsyslog **[3]**

- Agent passif pour faire le lien entre Rsyslog et Nagios : NRDP **[4]**

3.2 Serveurs

L'architecture comporte :

- Un serveur Nagios opérationnel dont nous ne verrons pas l'installation et un agent NRDP (@IP : 192.168.1.1 DNS : serveur-nagios) ;

- Un serveur de logs Rsyslog et un client NRDP (@IP : 192.168.1.2 – DNS : serveur-rsyslog) ;

- Une passerelle anti-virus Clamav et un client Rsyslog (@IP : 192.168.1.3 – DNS : serveur-clamav).

La distribution utilisée est une RedHat 6.3.

Les commandes pour la distribution Debian seront également indiquées.

3.3 Schémas

Voir Figures 1 et 2.

1 – Mise à jour des passerelles antivirus avec Clamav.

2 – Centralisation des logs vers un serveur de logs (Rsyslog).

3 – Filtrage des logs et remontée de l'information à Nagios lorsque le filtre correspond.

4 – Nagios alerte l'administrateur par divers moyens (mail, SMS...).

Nous allons passer en revue plusieurs méthodes pour centraliser les logs en fonction du type de système (Linux, Windows) et du type de logs (gérés ou non par Syslog). Gardons bien en tête que ceux qui nous intéressent pour la mise en place de notre système sont ceux de Clamav.

4 Centraliser les logs d'un serveur Linux

Par défaut, les distributions RedHat sont configurées avec le logiciel Syslog, sauf depuis la dernière version RedHat 6.0, où le logiciel Rsyslog est devenu le daemon Syslog par défaut comme sous Debian.

Donc tout au long de cet article, nous allons nous appuyer sur le logiciel Rsyslog ainsi que sur la partie cliente étant donné qu'il devient la référence en matière d'archivage des logs sur les différentes distributions.

Attention, tous les logs ne sont pas gérés par Rsyslog, certaines applications gèrent elles-mêmes leurs logs comme Apache... Nous verrons comment les utiliser avec le protocole Syslog et les centraliser.

4.1 Logs gérés par le protocole rsyslog

4.1.1 Installer

Si le logiciel Rsyslog n'est pas déjà installé sur votre distribution préférée, vous pouvez l'installer à la place de Syslog en désactivant ce dernier. Rien de plus simple sous RedHat.

Pour désinstaller Syslog :

```
# yum erase syslog
```

Pour installer Rsyslog :

```
# yum install rsyslog
```

La commande **Yum** nous permet d'aller chercher le RPM sur le dépôt de RedHat accessible grâce à notre enregistrement auprès de RedHat Network ainsi que notre connexion Internet bien évidemment!

Sous Debian :

```
# apt-get remove syslog
```

Pour installer Rsyslog :

```
# apt-get install rsyslog
```

Lancer le daemon :

```
# service rsyslog start
```

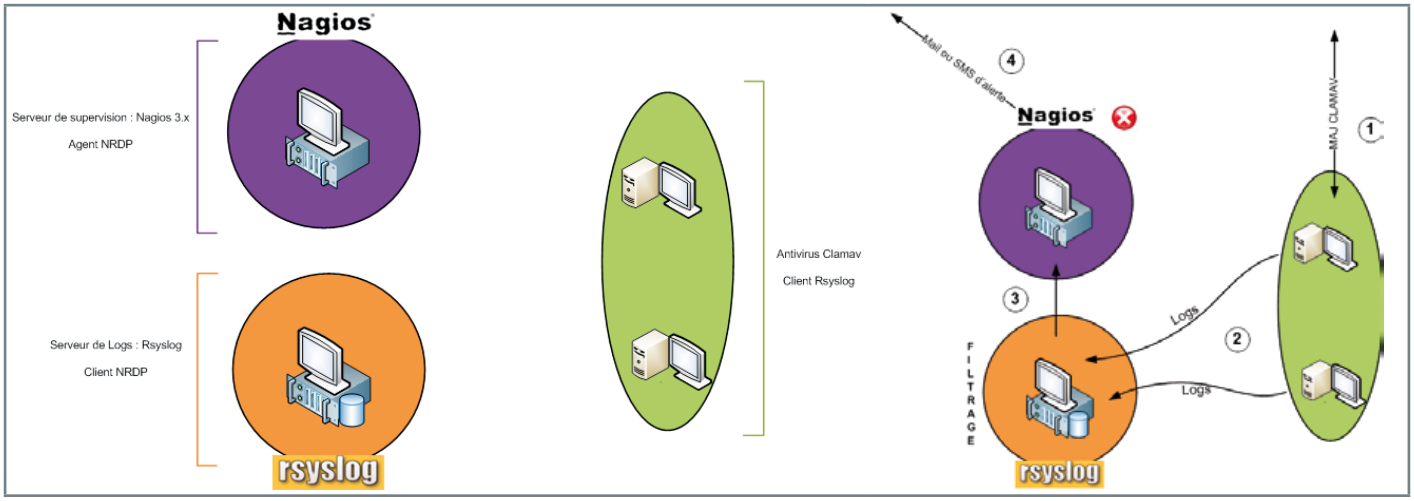


Fig. 1 : Architecture logicielle mise en place pour réaliser le filtrage de logs

Fig. 2 : Architecture des flux mise en place pour réaliser le filtrage de logs

Sous Debian :

```
# /etc/init.d/rsyslog start
```

4.1.2 Configurer

Les logs de Clamav ont la possibilité d'être gérés par le protocole Syslog.

Clamav s'appuie sur l'outil de mise à jour Freshclam qui se charge de mettre à jour sa base antivirale. Il se lance toutes les heures grâce à un script se trouvant dans **/etc/cron.hourly/**

Note

Sous Debian, c'est le daemon **clamav-freshclam** qui gère les mises à jour. Avec la directive Checks dans le fichier de configuration **freshclam.conf**.

```
# Check for new database 24 times a day
Checks 24
DatabaseMirror db.local.clamav.net
DatabaseMirror database.clamav.net
...
```

Le fichier de configuration de Freshclam : **freshclam.conf** permet de faire appel à une directive pour s'appuyer sur le protocole Syslog pour la gestion des logs. Pour cela, il vous suffit d'éditer le fichier de configuration **/etc/clamav/freshclam.conf**.

Pour activer les logs via Syslog, il suffit de mettre à **true** le paramètre suivant :

```
LogSyslog true
```

et d'indiquer la directive d'utilisation de Rsyslog :

```
LogFacility LOG_LOCAL6
```

Pour envoyer la totalité des logs vers notre serveur de logs, il reste juste à le renseigner dans le fichier de configuration de Rsyslog **/etc/rsyslog.conf** :

```
*.* @192.168.1.2
```

Rien de plus simple, la totalité des logs part en copie vers le serveur de logs. Bien sûr, il y a certaines choses à paramétrer sur ce dernier pour bien classer les logs par hôte. Nous verrons sa configuration plus loin.

Si nous voulons affiner les logs qui sont centralisés, il suffit de spécifier quel type de logs nous allons envoyer, seulement les logs de la mise à jour de Clamav qui correspondent au type local6.

```
local6.* @192.168.1.2
```

Nous envoyons ici les logs concernant freshclam, aussi bien les logs d'info, de warm et d'erreur grâce à *

- **local6.info**
- **local6.warm**
- **local6.erreur**

Redémarrer le service rsyslog :

```
# service rsyslog restart
```

Sous Debian :

```
# /etc/init.d/rsyslog restart
```

Vous pouvez forcer la mise à jour de Clamav en forçant le script suivant **/etc/cron.hourly/freshclam** :

Nous pouvons voir dans le fichier de logs **/var/log/clamav/freshclam.log** que la version du logiciel Clamav n'est pas à jour :

```
Sep 20 12:01:05 serveur-clamav
freshclam[16152]: ClamAV update process
started at Thu Sep 20 12:01:05 2012
Sep 20 12:01:05 serveur-clamav
freshclam[16152]: Your ClamAV installation
is OUTDATED!
Sep 20 12:01:05 serveur-clamav
freshclam[16152]: Local version: 0.97.3
Recommended version: 0.97.6
Sep 20 12:01:05 serveur-clamav
freshclam[16152]: DON'T PANIC! Read http://
www.clamav.net/support/faq
```

Ainsi vous savez comment utiliser Rsyslog pour gérer les logs et les centraliser sur un serveur de logs. Nous verrons dans la prochaine partie, comment configurer le serveur Rsyslog qui réceptionnera les logs. En attendant, nous allons voir comment centraliser des logs qui ne sont pas gérés par syslog, par exemple les logs d'Apache. Nous reviendrons un peu plus tard sur le filtrage des logs de Clamav, filtrage qui nous intéresse pour la suite de cet article.

4.2 Logs non gérés par le protocole rsyslog

4.2.1 Installer

Même procédure que ci-dessus pour installer et lancer le service Rsyslog. Néanmoins, la configuration va légèrement se compliquer.

4.2.2 Configurer

Comme je vous l'ai précisé précédemment, si vous avez lu l'article dans le bon sens sans sauter des lignes voire des paragraphes comme nous savons si bien le faire... nous allons nous appuyer sur Rsyslog pour les logs d'Apache ou autres logs non gérés par Syslog.

Il suffit de configurer de cette manière-là le fichier de configuration **/etc/rsyslog.conf**. Nous verrons ensuite les différentes directives utilisées ci-dessous :

```
$ModLoad imfile.so
$InputFileName /var/log/fichier-de-logs.log
$InputFileTag tag1:
$InputFileStateFile stat-file-apache
$InputFileSeverity error
$InputFileFacility local7
$InputRunFileMonitor
local7.* @192.168.1.2:514
```

Quelques explications sur les paramètres utilisés :

- **$ModLoad** charge le module permettant de gérer les fichiers.
- **$InputFileName** indique le nom du fichier de logs à analyser.
- **$InputFileTag** définit le nom du tag utilisé dans le fichier Rsyslog.
- **$InputFileStateFile** est un identifiant interne à Rsyslog.
- **$InputFileSeverity** indique le niveau des logs. Par défaut, ils y sont tous.
- **$InputFileFacility** définit le type de logs.

- `$InputRunFileMonitor` active l'écoute sur ce fichier.
- `Local7.* @192.168.1.2:514` permet d'envoyer le contenu de local7 vers le serveur rsyslog centralisé.

Par défaut, le service Syslog utilise le port 514 du protocole UDP.

Note

La directive InputFileStatFile doit être unique pour chaque fichier que vous allez monitorer.

La directive `InputRunFilemonitor` est absolument nécessaire. Si vous l'oubliez, il n'y aura pas de contrôle des logs.

Ceci est un aperçu de la façon d'utiliser Rsyslog pour gérer et centraliser des logs non gérés par Syslog.

5 Centraliser les logs de Windows

5.1 Logs du gestionnaire d'événements

Pour réaliser la centralisation des logs de Windows, nous avons besoin d'un outil tiers qui est l'agent SNARE de l'entreprise InterSectAlliance. L'outil SNARE fonctionne de la même manière que Rsyslog, il redirige instantanément chaque nouvelle entrée du journal des événements Windows vers un serveur de type Syslog (voir Figure 3).

5.1.1 Installer SNARE

Télécharger le fichier d'installation disponible ici : http://www.intersectalliance.com/download.html?link=http://prdownloads.sourceforge.net/snare/SnareForWindows-4.0.1.2a-MultiArch.exe.

L'installation est relativement simple, il vous suffit de lancer l'exécutable et de cliquer sur « Next » plein de fois (NDLR : installation typique sous Windows) (voir Figures 4 et 5).

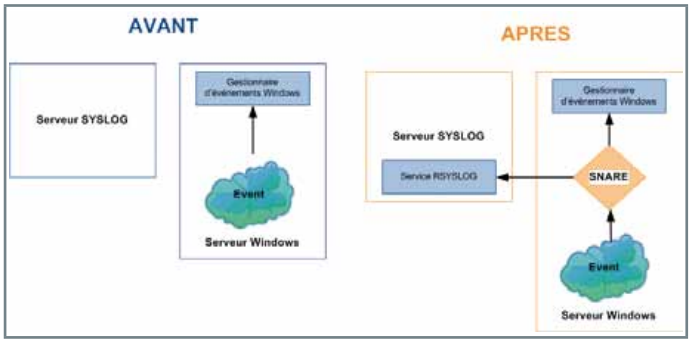


Fig. 3 : Architecture avec Snare

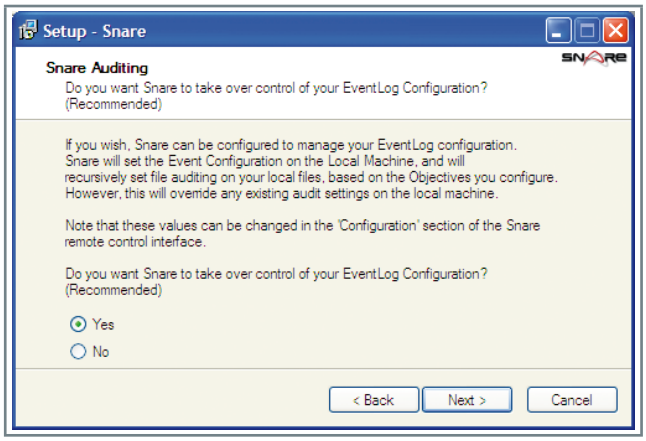


Fig. 4 : Snare prend la main sur le gestionnaire d'événements Windows

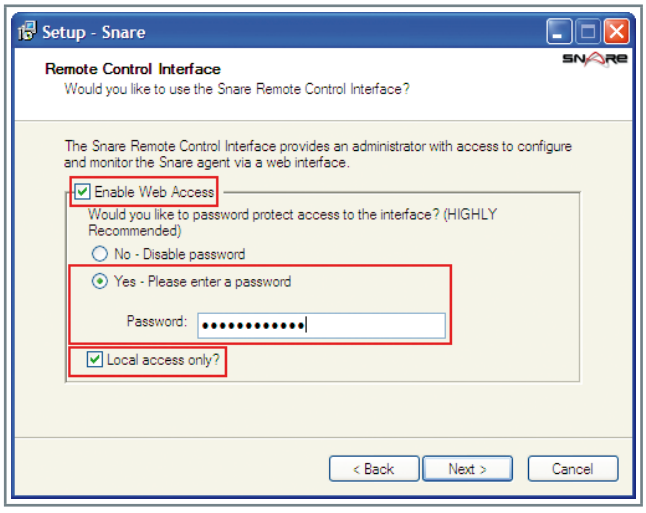


Fig. 5 : Activer l'interface graphique

Note

La présence de l'interface est indispensable pour tester le fonctionnement de l'application, mais en production il sera préférable de la désactiver. Les informations sont stockées dans la base de registres.

Le logiciel est installé, vous n'êtes pas obligé de redémarrer votre serveur Windows.

5.1.2 Configurer SNARE

Pour accéder à l'interface graphique, il suffit de lancer son navigateur préféré et se connecter à l'adresse http://localhost:6161.

Entrez le nom d'utilisateur : snare et le mot de passe associé si vous en avez mis un. Sélectionnez sur le menu de gauche : « Network Configuration » pour configurer le lien vers le serveur SYSLOG.

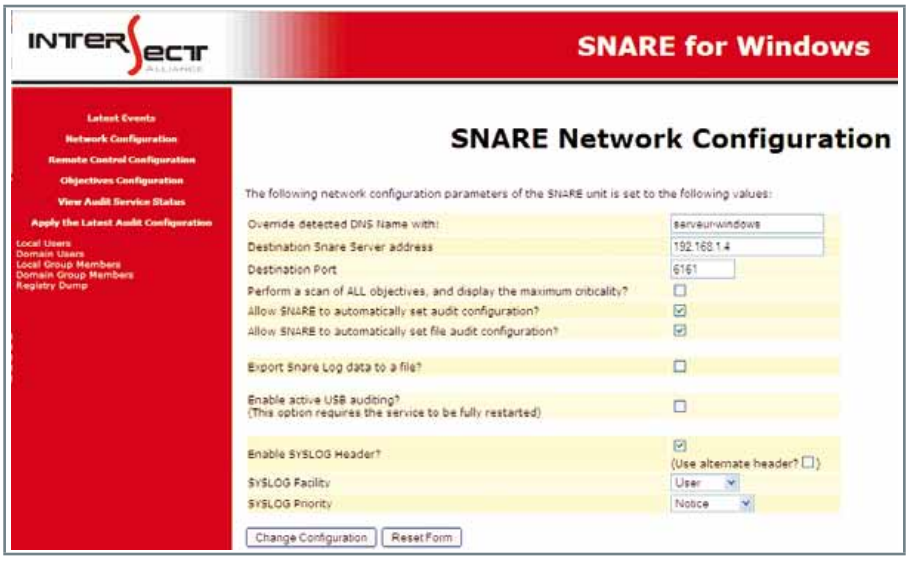


Fig. 6 : Network Configuration

Vous devez indiquer plusieurs éléments :

- « Override detected DNS Name with » : Le nom que l'on souhaite envoyer vers le serveur Rsyslog.
- « Destination Snare Server adress » : Adresse IP du serveur Rsyslog.
- « Destination port » : Le port d'écoute de ce dernier.
- « Enable SYSLOG header » : Activer «Enable SYSLOG header» pour enregistrer de manière standard les entrées.

Cliquez sur « Change Configuration » pour valider les informations.

5.1.3 Configurer le niveau d'alerte

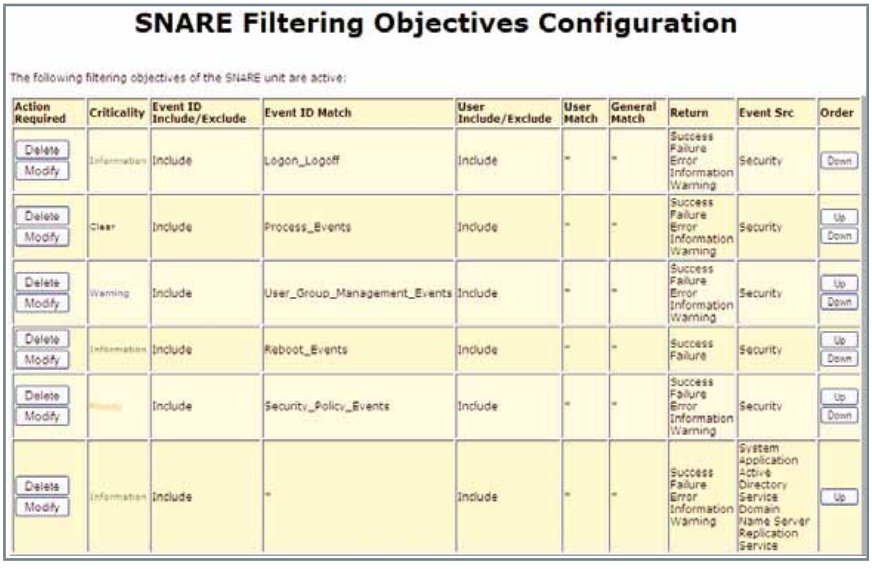


Fig. 7 : Configuration du niveau d'alerte

Il est possible de modifier les règles préétablies ou bien d'en créer d'autres. Nous allons en créer une en cliquant sur Add.

Ici, nous avons créé une règle dans laquelle nous avons seulement sélectionné les Error de sécurité à retourner au serveur de logs Rsyslog avec un niveau d'alerte Critical (voir Figure 8 page suivante).

5.1.4 Tester SNARE

Maintenant que Snare est configuré, nous pouvons générer une fausse alerte vers notre serveur Rsyslog, mais il faudra au préalable configurer ce dernier pour qu'il puisse filtrer les éléments venant du serveur Windows. Mais attendons un peu la suite de l'article, pour l'instant je présente les différentes solutions pour rediriger les logs (syslog, non gérés par syslog, logs windows, logs windows applicatifs) vers un serveur de logs.

Il conviendra d'ajouter les lignes suivantes dans la configuration du serveur rsyslog (que nous verrons par la suite) pour stocker dans un fichier journalier chacune des entrées du journal des événements Windows.

```
$template Windows, "/var/log/%HOSTNAME%.%DAY
%%$MONTH%%$YEAR%.log"
fromhost-ip, isequal, '192.168.1.4' then
-?Windows
```

L'adresse IP correspond à celle du serveur Windows.

Et il faudra générer une fausse alerte Windows pour vérifier qu'elle est bien dirigée vers le serveur de logs Rsyslog.

Pour générer une alerte, nous allons nous appuyer sur la commande « EventCreate ».

```
# EVENTCREATE /T ERROR /ID 100 /L
APPLIACTION /D "Création d'une entrée ERROR
dans le journal des applications"
Opération réussie : un événement de type
error est créé dans le journal ou la source
'application '.
```

Nous devons récupérer le résultat sur notre serveur de Logs et le stocker dans **`/var/log/%HOSTNAME%.%DAY%%$MONTH%%$YEAR%.log`**.

Je ne rentre pas trop dans les détails sur la configuration du serveur Rsyslog

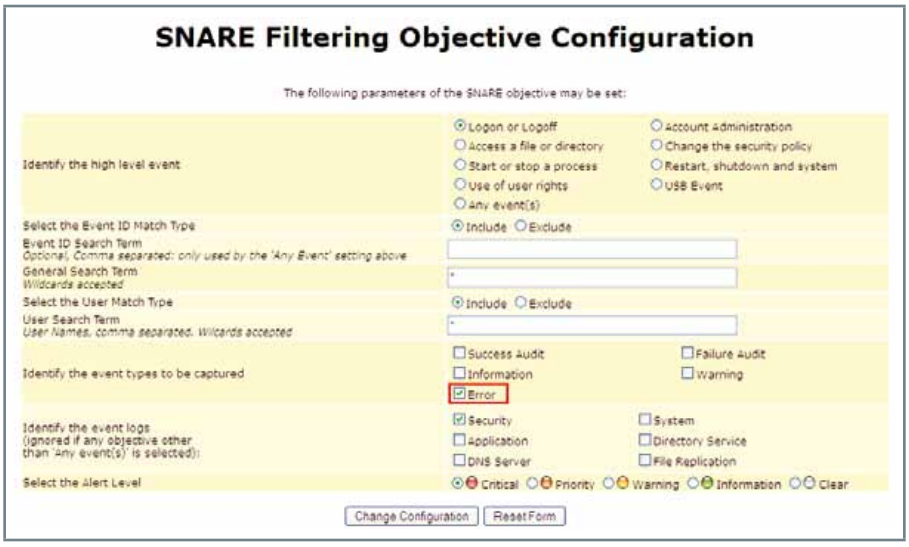


Fig. 8 : Ajouter une règle

étant donné que cela fait l'objet d'un prochain chapitre.

5.2 Logs applicatifs

Pour réaliser cette opération, nous allons utiliser les outils libres de l'entreprise InterSectAlliance. Nous utiliserons l'agent Epilog sur Windows. L'outil Epilog a pour objectif de rediriger de manière instantanée chaque nouvelle entrée d'un fichier supervisé (voir Figure 9).

5.2.1 Installer Epilog

Téléchargez le fichier d'installation disponible ici : http://sourceforge.net/projects/snare/files/Epilog%20for%20Windows/1.6.0/EpilogSetup-1.6.0-MultiArch.exe/download.

L'installation est relativement simple, il vous suffit de lancer l'exécutable et de cliquer sur « Next » à chaque étape.

Note

La présence de l'interface est indispensable pour tester le fonctionnement de l'application, mais en production il sera préférable de la désactiver. Les informations sont stockées dans la base de registres.

Comme pour le logiciel SNARE, il n'est pas utile de redémarrer le serveur Windows.

5.2.2 Configurer EPILOG

Pour accéder à l'interface graphique, il suffit de lancer son navigateur préféré et se connecter à l'adresse http://127.0.0.1:6162.

Entrez le nom d'utilisateur : **snare** et le mot de passe associé si vous en avez mis un.

Comme pour SNARE, il faut configurer : l'adresse IP et le port d'écoute du serveur syslog.

Allez sur Network Configuration. Vous devez indiquer plusieurs éléments :

- « Override detected DNS Name with » : nom que l'on souhaite envoyer au serveur Rsyslog.
- « Destination Snare Server address » : adresse IP du serveur Rsyslog.
- « Destination port » : le port d'écoute de ce dernier : 514.
- « Enable SYSLOG header » : activer «Enable SYSLOG header» pour enregistrer de manière standard les entrées.

Cliquez sur « Change Configuration » pour valider les informations

Note

Attention, la directive **Override detected DNS Name with** est identique entre SNARE et EPILOG, vous allez mélanger dans un même fichier des événements Windows qui sont dans un format **X** et des logs applicatifs qui sont dans un format **Y**. C'est pour cela que je vous invite à séparer les contenus.

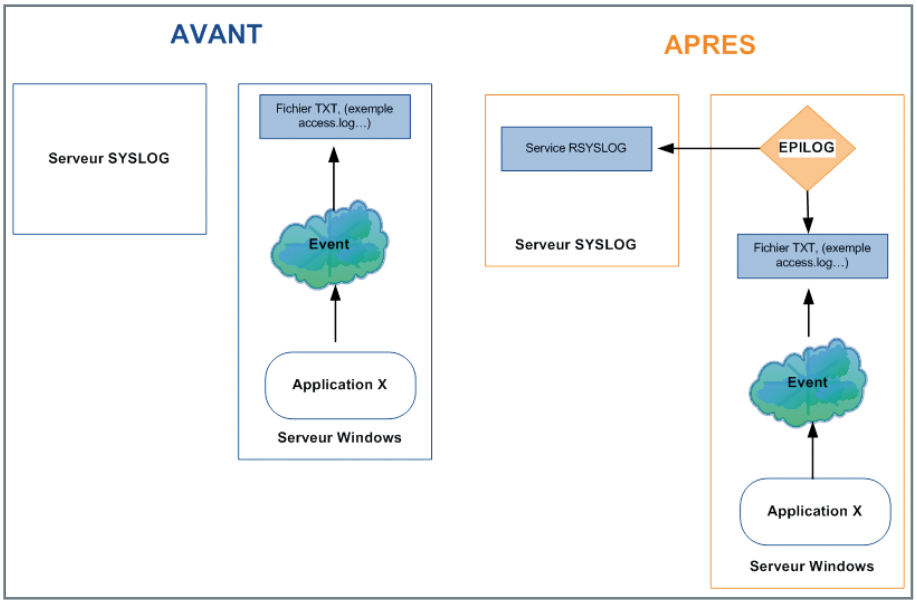


Fig. 9: Architecture avec Epilog

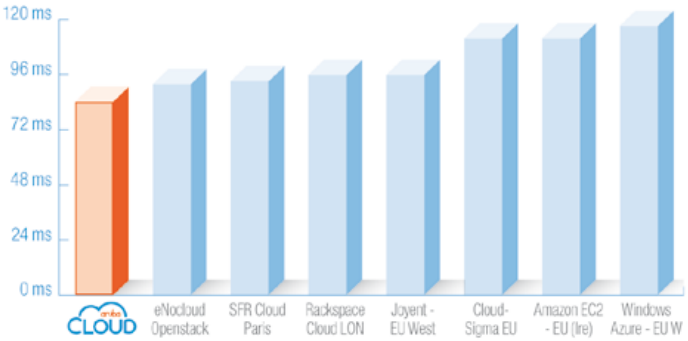
Aruba Cloud, la meilleure offre de Cloud Computing du marché

Le plus performant et le moins cher des Clouds Publics

Fournisseur	Coût mensuel estimé	Performances	Sécurité	Flexibilité
Meilleure proposition	aruba CLOUD	aruba CLOUD	Rackspace	Amazon Web Services
aruba CLOUD	69 €	★★★★½	★★★☆☆	★★★½☆
Windows Azure	87 €	★★★★☆	★★★½☆	★★★★☆
Amazon Web Services	96 €	★★★½☆	★★★★☆	★★★★½
CloudSigma	135 €	★★★★☆	★★★½☆	★★★☆☆
Rackspace	157 €	★★★★☆	★★★★☆	★★★★☆

Source www.cloudscreener.com
sur les paramètres standards d'évaluation des offres de Cloud Computing.

Classé n°1 en temps de réponse, temps de connexion et disponibilité



Source : Analyse Cedexis réalisée du 5 mai au 5 juin 2013 depuis la France.
www.cedexis.com

TESTEZ GRATUITEMENT NOTRE OFFRE

Aruba, le bon partenaire pour bénéficier de la puissance d'un acteur majeur qui considère que chaque client, dans chaque pays, est unique. **MY COUNTRY. MY CLOUD.**

aruba CLOUD
arubacloud.fr | TÉL : 0810 710 300
(COÛT D'UN APPEL LOCAL)

Fig. 10 : Network Configuration

5.2.3 Configurer le niveau d'alerte

Sélectionnez dans le menu : Log Configuration, puis cliquez sur Add (voir Figure 10).

Vous devez indiquer plusieurs éléments :

- « Select the Log Type » : Custom Event Log.
- « Log File or Directory » : fichier de logs des applications pour lesquelles vous voulez centraliser les logs.

Cliquez sur « Change configuration » pour valider les informations.

Tout comme SNARE, il est également possible de filtrer le contenu de ce que l'on veut rediriger vers notre serveur de logs.

Je ne rentrerai pas plus dans les détails sur la centralisation des logs Windows, vous savez que ça existe.

5.2.4 Tester EPILOG

Pour tester, il suffit simplement de se connecter à l'application pour laquelle nous centralisons les logs et aller vérifier si les traces de connexions sont présentes dans le fichier access.log.

Désormais, vous savez envoyer différents types de logs vers un serveur de logs en utilisant syslog. Maintenant, voyons comment configurer le serveur de logs qui va les réceptionner pour les stocker soit dans des fichiers plats, soit dans une base de données du type mysql ou bien encore faire appel à un script (solution qui nous intéresse pour faire le lien avec Nagios).

6 Serveur de logs

6.1 Installer

Le serveur Rsyslog permet de centraliser les logs des différents serveurs.

L'installation du service **Rsyslog** se fait en utilisant la commande **Yum** installée par défaut sur les RedHat EL5.0 qui permet l'accès à RedHat Network.

```
# yum install rsyslog rsyslog-server
```

Si vous désirez utiliser un serveur mysql local pour stocker les logs et une interface web pour les visualiser, installez les paquets suivants, sinon ce n'est pas la peine et allez directement au chapitre suivant 6.2 Configurer.

```
# yum install mysql-server
# yum install httpd php php-mysql php-gd
```

Si mysql n'est pas lancé, lancez-le :

```
# /etc/init.d/mysqld start
```

Première chose à faire, modifier le mot de passe root :

```
 #/usr/bin/mysqladmin -u root password
********
```

6.2 Configurer

Éditer le fichier de configuration :

```
# vi /etc/rsyslog.conf
```

Activer la réception des logs distants en éditant le fichier de configuration **/etc/rsyslog.conf** et en décommentant les lignes correspondantes :

Activer les connexions UDP : réception des logs via UDP sur le port 514.

```
$ModLoad imudp
$UDPServerRun 514
```

Même punition pour les connexions TCP.

```
$ModLoad imtcp
$InputTCPServerRun 514
```

Ajouter l'option -r pour activer l'écoute de syslog sur le port par défaut UDP 514.

```
SYSLOGD_OPTIONS="-m 0 -r"
```

Penser à relancer le service rsyslog :

```
# service rsyslog restart
```

Notre serveur de logs est prêt à réceptionner les logs venant des serveurs que nous avons configurés au début de cet article. Maintenant, il va être bon de lui indiquer ce qu'il faut faire des logs. Pour cela, nous allons mettre en place un système de filtrage. Nous allons faire différents filtres, un qui classera les logs dans un fichier plat, un autre qui les stockera dans une base mysql et un dernier qui exécutera un script (c'est celui-ci qui nous intéressera pour faire le lien avec Nagios/NRDP).

6.2.1 Filtre simple

Nous allons commencer par créer un filtre simple qui va filtrer sur une chaîne de caractères et mettre les logs dans un fichier plat tout simplement.

Éditer le fichier de configuration :

```
# vi /etc/rsyslog.conf
```

et ajouter les lignes suivantes :

```
$template clamav, "/var/log/%HOSTNAME%/clamav.log"
if $syslogfacility-text == 'local6' and $fromhost-
ip == '192.168.1.3' then ?clamav
```

Explication du filtre : si les logs sont du type (**syslogfacility-text) local6** et proviennent du serveur qui a l'adresse IP (**$fromhost-ip**)192.168.1.3, alors j'exécute le template clamav qui va classer les logs dans le fichier correspondant **/var/log/serveur-clamav/clamav.log**.

Nous verrons plus loin d'autres directives de filtrage, mais si vous êtes pressés, je vous invite à vous rendre directement ici **[5]**.

6.2.2 Filtre et intégration dans une base mysql

Nous allons voir comment stocker les logs dans une base de données du type Mysql. Pour ceux qui veulent utiliser ce système de stockage, j'espère que vous n'avez pas sauté l'étape d'installation des paquets décrite au-dessus.

Créer la base de données :

```
# mysql < /usr/share/doc/rsyslog-mysl-3.22.1/createDB.sql
```

Donner des droits à l'utilisateur syslog sur la base Syslog :

```
mysql> grant all on Syslog.* to syslog@localhost identified by 'password';
mysql> flush privileges ;
```

Il faut activer le module MySQL de Rsyslog en décommentant ou en ajoutant la ligne suivante dans le fichier de configuration **/etc/rsyslog.conf** :

```
$ModLoad ommysql
```

Ensuite, pour que Rsyslog intègre automatiquement les logs dans la base de données, il faut lui donner les informations de connexion à cette base :

```
mail.* :ommysql:localhost,Syslog,syslog,password
*.* :ommysql:database-server,database-name,database-userid,database-password
```

Ici, les logs qui ont les facilities **mail.*** iront se stocker dans la base mysql.

Faire un lien entre

```
# ln -s /usr/lib64/rsyslog/ommysql.so /usr/lib64/rsyslog/ommysql
```

Redémarrer le service rsyslog

```
# /etc/init.d/rsyslog restart
```

Les logs viendront se stocker dans la base mysql. L'intérêt est de pouvoir utiliser des applications tiers qui permettent de traiter les événements (messages) syslog.

On connaît l'éternel PHPLogCon **[6]**, sans oublier 8pussy **[7]**, Splunk **[8]** et des potentiels remplaçants n'utilisant pas Mysql, mais d'autres bases de données:

- graylog2 **[9]** ;
- Logstash **[10]** ;
- log.io **[11]**.

Nous pourrions rédiger un chapitre dessus, mais là n'est pas le sujet, je vous laisse les liens de chaque application si vous voulez étendre vos connaissances.

6.2.3 Filtre et appel à un script

Venons-en à ce qui nous intéresse pour interfacer l'ensemble avec Nagios, un filtre faisant appel à un script qui lui même fait appel à NRDP pour passer les informations à Nagios. Voici le filtre qui va exécuter une commande si celui-ci correspond à la chaîne de caractères définie :

```
$template OUTDATED, "%HOSTNAME%;Service VERSION CLAMAV;2;%msg%"
if $syslogfacility-text == 'local6' and $fromhost-ip == '192.168.1.3'
and $msg contains 'Your ClamAV installation is OUTDATED' then ^/usr/
local/scripts/rsyslog-nagios.sh;OUTDATED
& ~
```

Voyons en détail ce que fait ce filtrage rsyslog :

```
$template OUTDATED, "%HOSTNAME%;Service VERSION CLAMAV;2;%msg%"
```

- **$template** : nous avons créé un gabarit qui fixe le format du message pour le script **rsyslog-nagios.sh**
- **%HOSTNAME%** : le nom du serveur qui doit être identique à celui associé au service dans Nagios.
- **Service VERSION CLAMAV** : le nom du service qui doit être identique à celui défini dans Nagios.
- **2** : le statut retourné à Nagios, ici CRITICAL.
- **%msg%** : un message de retour.

Vient ensuite la directive qui fixe le filtre et ce qui est fait des messages correspondants.

```
if $syslogfacility-text == 'local6' and $fromhost-ip == '192.168.1.3'
and $msg contains 'Your ClamAV installation is OUTDATED' then ^/sql/
log/scripts/script.sh;OUTDATED
& ~
```

Explication du filtre : Si les logs sont du type (**syslog-facility-text) local6** et proviennent du serveur qui a l'adresse IP (**$fromhost-ip**)192.168.1.3 (serveur-clamav) et si dans le corps du message nous tombons sur la chaîne de caractères "**Your ClamAV installation is OUTDATED**", alors il est nécessaire d'exécuter le script **rsyslog-nagios.sh**.

Le script va prendre en paramètre les informations définies dans le template, c'est-à-dire le nom de la machine de l'hôte, le nom du service, un code retour et le contenu du message.

Le script **rsyslog-nagios.sh** :

```
#!/bin/bash
CMD="/usr/bin/php /usr/local/nrdp/send_nrdp.php"
PART1=$(echo $1 | cut -d";" -f1)
PART2=$(echo $1 | cut -d";" -f2)
PART3=$(echo $1 | cut -d";" -f3)
PART4=$(echo $1 | cut -d";" -f4)
$CMD --url=http://serveur-nagios/nrdp --token=mysecrettoken
--host="$PART1" --service="$PART2" --state="$PART3" --output="$PART4"
```

Vous pouvez voir dans le script qu'il y a la commande **send_nrdp.php**. C'est la partie cliente de NRDP qui va faire le lien entre mon serveur de logs et mon serveur de supervision Nagios pour l'informer lorsque le filtre correspond et lui envoyer les informations interprétables par Nagios. Nous allons voir dans la partie suivante l'installation de NRDP et comment ce dernier fonctionne.

7 NRDP

Maintenant que nous avons centralisé les logs, il est temps d'interagir avec Nagios. Nous allons pour cela utiliser l'outil NRDP.

Le principe est simple : il est constitué d'une partie serveur (**/usr/local/nrdp/server/config.inc.php**) qui est installé sur le serveur Nagios et d'un client NRDP (**/usr/local/nrdp/send_nrdp.cfg**) installé sur le serveur de logs effectuant des vérifications passives.

Pourquoi l'utilisation de NRDP ? Pour plusieurs raisons ! En effet, NRDP a plusieurs intérêts par rapport à NSCA (outil permettant également de faire des vérifications passives).

NRDP utilise des ports standards, ce qui permet de simplifier les règles de filtrage au niveau du firewall.

NRDP utilise le serveur web apache pour fournir en option le chiffrement SSL et l'authentification.

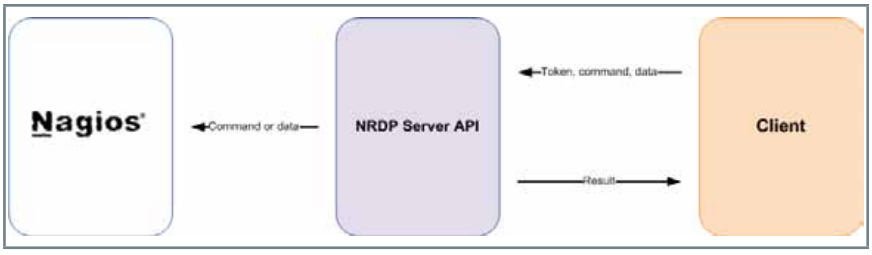


Fig. 11 : Architecture NRDP

Fonctionnement de NRDP :

- Étape 1 :
  → Un client distant envoie une demande à l'API du serveur NRDP. Le client doit lui soumettre :
    → Un jeton valide qui a été autorisé dans le fichier de configuration du serveur NRDP.
    → La commande appelée par le serveur NRDP
    → Les données associées à la commande.
- Étape 2 :
  → NRDP vérifie le jeton et transmet la demande du client au plugin NRDP.
- Étape 3 :
  → Le plugin NRDP traite la demande du client et transmet les données à Nagios ou une autre application.
- Étape 4 :
  → NRDP retourne le résultat au client au format XML.

7.1 Partie serveur

7.1.1 Installation

Télécharger l'archive de NRDP et la décompresser sur votre serveur Nagios :

```
# cd  /usr/local/
# wget http://assets.nagios.com/downloads/
nrdp/nrdp.zip
# unzip nrdp.zip
```

Donner les bons droits aux répertoire et fichiers de NRDP :

```
# chown -R nagios.nagios /usr/local/nrdp
```

7.1.2 Configuration

Éditer le fichier de configuration principal de NRDP :

```
# vi /usr/local/nrdp/server/config.inc.php
```

Ajouter un ou plusieurs jetons dans la variable **$cfg['authorized_tokens']**.

```
$cfg['authorized_tokens'] = array(
"asd7fjk3134",
"df23m7jadI34",
);
```

Vérifier également les chemins de Nagios et les adapter en fonction de votre configuration, notamment le chemin d'accès au fichier de commandes de Nagios.

```
$cfg["command_file"]="/usr/local/nagios/var/
nagios.cmd";
```

Copier le fichier de configuration d'Apache pour NRDP :

```
# cp nrdp.conf /etc/httpd/conf.d
```

sous debian :

```
# cp nrdp.conf /etc/apache2/conf.d
```

Note

Optionnel : pour configurer les paramètres permettant de restreindre l'accès à une adresse ou activer SSL ou encore utiliser une authentification basique, il vous suffit d'éditer le fichier **/etc/httpd/conf.d/nrdp.conf**

Redémarrer Apache :

```
# /etc/init.d/httpd restart
```

Le serveur NRDP est désormais installé. Nous allons tester notre installation pour vérifier si cette dernière fonctionne correctement.

7.1.3 Tester l'API de NRDP

Vous pouvez tester l'application en vous connectant sur le lien suivant : http://serveur-nagios/nrdp.

Premier test, nous allons désactiver les notifications du serveur Nagios (penser à les réactiver ensuite). Il suffit de renseigner un jeton valide et de spécifier l'hôte pour lequel nous voulons désactiver les notifications, ici localhost.

Fig. 12 : Tester NRDP (Submit Nagios Command)

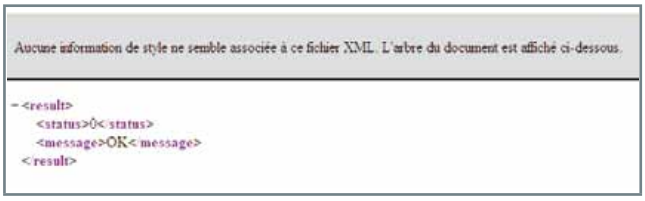


Fig. 13 : Résultat du test NRDP (Submit Nagios Command)

Après avoir validé, vous pouvez vérifier par l'interface de Nagios si les notifications sont bien désactivées pour l'hôte spécifié.

Second test, nous allons modifier l'état du service SSH du serveur Nagios en le passant à CRITICAL. Il suffit de renseigner un jeton valide, de spécifier le nom du service (ici SSH) et l'hôte pour lequel nous voulons changer le statut du service.

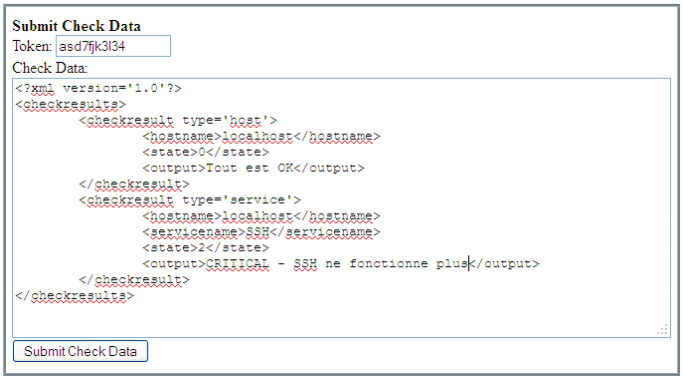


Fig. 14 : Tester NRDP (Submit Check Data)

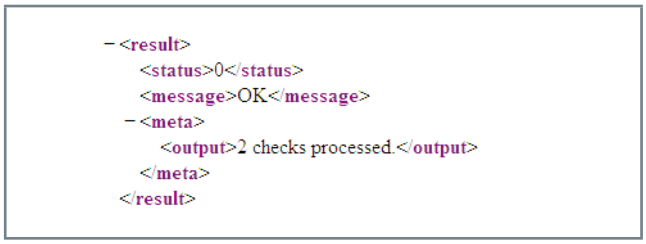


Fig. 15 : Résultat du test NRDP (Submit Check Data)

Vous pouvez vérifier par l'interface de Nagios si l'état du service SSH a bien été modifié pour l'hôte spécifié.

Maintenant que les tests sont concluants, nous allons configurer la partie cliente se trouvant sur le serveur Rsyslog (notre serveur de logs) qui va permettre d'envoyer vers Nagios les informations correspondantes au filtre Rsyslog.

7.2 Partie cliente

Il suffit de récupérer le fichier **send_nrdp.php** se trouvant dans **/usr/local/nrdp/clients** et le transférer sur le serveur de logs qui va faire des vérifications passives.

Comment fonctionne cette commande ? Sur le même principe que **send_nsca.cfg** avec NSCA pour ceux qui connaissent cet outil, sinon pour les autres, il suffit de lui passer des paramètres -> l'hôte concerné, le service défini dans Nagios, le statut à renvoyer...

Voici les différentes options possibles à passer en paramètres :

```
Utilisation : /usr/local/nrdp/clients/send_nrdp.php --url=<url> --token=<token>
--host=<hostname> [--service=<servicename>] --state=<state> --output=<output>
<url> = L'url pour accéder à l'agent NRDP.
<token> = Le jeton utilisé pour accéder à l'agent NRDP.
<hostname> = Le nom de l'hôte associé au service passif qui doit être contrôlé.
<servicename> = Le nom du service associé au check passif.
<state> = Le status du service que l'ont veut renvoyer.
<output> = Le texte qui doit être envoyé au check passif.
```

Nous allons tester la commande à la main pour exécuter un check passif. Nous faisons le même test que par l'interface, mais cette fois-ci à distance en utilisant la partie cliente de NRDP **send_nrdp.php**.

```
# /usr/bin/php /usr/local/nrdp/clients/send_nrdp.php --url=http://
serveur-nagios/nrdp --token=asd7fjk3l34  --host=serveur-nagios -
service=SSH --state=2 -output="CRITICAL - SSH ne répond plus"
```

Il faut présenter le token se trouvant dans la configuration du serveur NRDP vu précédemment, le nom de l'hôte, le nom du service à impacter dans Nagios (SSH), le code retour et le message à afficher dans la console de Nagios.

Vous pouvez exécuter la commande et vérifier sur votre serveur Nagios que votre service SSH passe du vert au rouge, si c'est le cas le test distant est concluant.

Vous allez me dire que tout ça est bien beau, mais quel est le lien avec Rsyslog, les filtres de Rsyslog, le script **rsyslog-nagios.sh ?** Hé bien, si vous vous souvenez bien, nous avons fait appel à la commande **send_nrdp.php** dans le script **rsyslog-nagios.sh** qui lui-même est appelé par le filtre lorsque ce dernier rencontre la chaîne de caractères spécifiée dans sa configuration... voilà, tout se recoupe.

C'est NRDP qui reçoit ce message (message composé du token, nom d'hôte, nom du service à impacter...) et transforme ce message en commande externe Nagios. Pour que cette commande soit bien prise en compte par Nagios, il faut bien entendu avoir l'hôte et le service précisé présents dans les fichiers de configuration.

Nous allons voir ensuite comment configurer Nagios pour accueillir les paramètres transmis via NRDP à ce dernier. Pour cela, il suffit de créer un service passif.

8 Configuration de Nagios

8.1 Créer un service

Créer un service passif sur Nagios en utilisant la commande **check_dummy**.

```
define service{
  use generic-service
  service_description  Service VERSION CLAMAV
  host_name serveur-clamav
  active_checks_enabled 0
  passive_checks_enabled 1
  is_volatile 1
  max_check_attempts 1
  check_freshness 1
  freshness_threshold 4200
  check_command check_dummy!0
}
```

Nous créons un fichier qui définit un service pour serveur-clamav appelé « Service VERSION CLAMAV ». Sa particularité est d'être un service passif grâce aux directives **active_check_enabled** à 0 et **passive_checks_enabled** à 1.

max_check_attempts est à 1 pour passer le service directement en état HARD.

check_freshness à 1 indique que nous souhaitons utiliser la fonction de contrôle de fraîcheur d'un service de nagios.

freshness_threshold à 4200 indique que nous déclencherons ce contrôle 4200 secondes soit 70 minutes après la réception du dernier contrôle.

check_command est la commande check_dummy, nous lui passons comme paramètre 0 pour état OK et le message qui s'affiche dans la console d'administration.

En résumé : si le système de filtrage de logs ne voit pas passer la chaîne de caractères définie (**Your ClamAV installation is OUTDATED**), le script ne s'exécutera pas et donc nagios ne sera pas alerté avec le code retour 2 (CRITICAL). Nagios considérera que le logiciel Clamav est à jour, il exécutera donc la commande check_dummy avec la valeur 0 en argument, c'est-à-dire OK.

Par contre, si le système de filtrage voit passer la chaîne de caractères définie, alors il va envoyer le statut défini dans le script **rsyslog-nagios.sh**, c'est-à-dire la valeur 2 (CRITICAL) et le service passera au rouge.

Si au bout de 3600 secondes (cela correspond à la mise à jour de Clamav qui s'effectue toutes les heures), la chaîne de caractères ne réapparaît pas, cela sous-entend que Clamav a été mis à jour entre temps et donc au bout de 4200 secondes (juste un peu plus que 3600 secondes) Nagios exécutera donc la commande **check_dummy** avec la valeur 0 en argument, c'est-à-dire OK. Le service repassera à OK.

Ne pas oublier de créer la commande correspondante dans votre fichier **commands.cfg** :

```
define command{
command_name check_dummy
command_line $USER1$/check_dummy $ARG1$
}
```

Voici l'état du service en temps normal : (voir Figure 16).

8.2 Test depuis la machine cliente - simuler l'envoi de trap

Maintenant que notre service passif sur Nagios est configuré, nous allons donc pouvoir faire nos tests et voir en réel si cela marche.

Nous allons tester de deux manières :

Exécuter NRDP manuellement :

```
# /usr/bin/php /usr/local/nrdp/clients/send_nrdp.php --url=http://serveur-nagios/nrdp --token=asd7fjk3134
--host=serveur-clamav -service=Service VERSION CLAMAV --state=2 -output="Clamav n'est pas à jour"
```

Il faut présenter le token, le nom de l'hôte, le nom du service, le code retour et le message à afficher dans la console et la description du service à impacter dans Nagios.

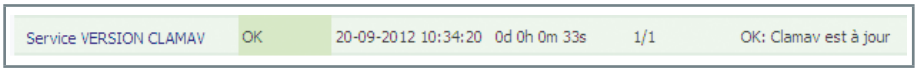


Fig. 16 : Service VERSION CLAMAV OK

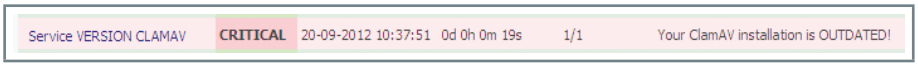


Fig. 17 : Service VERSION CLAMAV KO

Exécuter toute la chaîne :

Pour tester en production, deux solutions : soit vous attendez que votre service freshclam sur votre serveur-clamav vienne se lancer pour mettre à jour Clamav, soit vous le forcez en lançant tout simplement la commande suivante :

```
# /etc/cron.hourly/freshclam.
```

Dans le cas manuel ou automatique, vous devriez voir dans l'interface de Nagios votre service nommé «Service VERSION CLAMAV» passer du vert au rouge si votre installation de Clamav est obsolète (voir Figure 17).

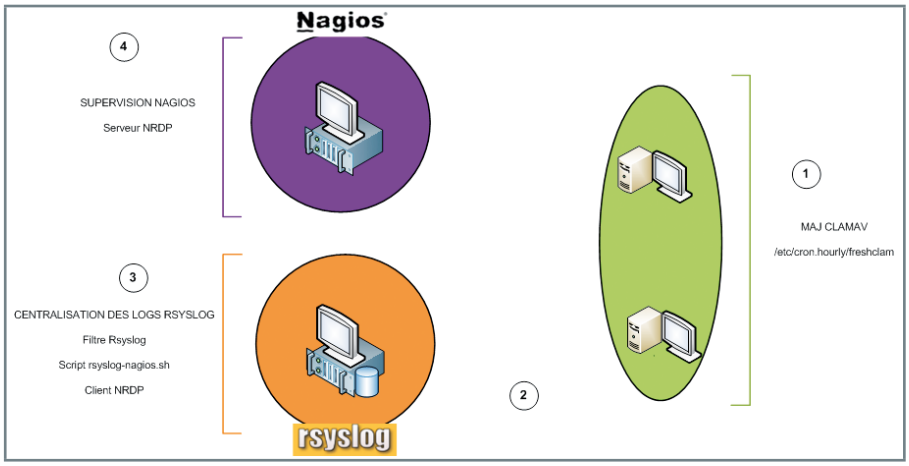


Fig. 18 : Schéma de la chaîne

9 En résumé

Un petit récapitulatif avant de conclure (voir Figure 18) :

Récapitulons :

1 - Clamav met à jour sa base antivirale en exécutant le cron **/etc/cron.hourly/freshclam**. Une trace dans les logs de clamav apparaît concernant la version obsolète de Clamav : **'Your ClamAV installation is OUTDATED'**

2 - Les logs de Clamav sont centralisés sur le serveur de logs grâce à la configuration du client **rsyslog.conf** (serveur-clamav) :

```
local6.* @192.168.1.2
```

3 - Sur le serveur de logs (serveur-rsyslog), les logs sont filtrés et si le filtre correspond, il exécute le script rsyslog-nagios.sh :

```
$template Alert, "%HOSTNAME%;Service VERSION CLAMAV;2;%msg%"
if $syslogfacility-text == 'local6' and $fromhost-ip == '192.168.1.3'
and $msg contains 'Your ClamAV installation is OUTDATED' then ^/usr/
local/scripts/rsyslog-nagios.sh;Alert
& ~
```

Le script **rsyslog-nagios.sh** fait appel à NRDP en lui passant en paramètre les valeurs **"serveur-clamav;Service VERSION CLAMAV;2;Your ClamAV installation is OUTDATED"** interprétables par Nagios pour les envoyer au serveur Nagios (serveur-nagios).

4 - Nagios récupère ces informations (l'hôte, le nom du service, le statut du service, et le message de sortie) grâce à son service passif «Service VERSION CLAMAV» et le **check_dummy**. Étant donné que le statut retourné a pour valeur 2, le service «Service VERSION CLAMAV» passe au rouge c'est-à-dire **CRITICAL5**. Si dans l'heure qui suit vous avez mis le service Clamav à jour, la chaîne de caractères ne se représentera pas dans les logs et donc le filtre ne matchera pas et ne renverra pas le status 2 concernant le service « **Service VERSION CLAMAV** », donc **check_dummy** s'exécutera avec la valeur 0 et remettra le service à OK...

Conclusion

À travers cet article, j'ai essayé de vous illustrer la mise en place de cette architecture de supervision de logs avec un cas concret. Cela peut vous sembler un peu fastidieux au premier abord, mais une fois vos logs centralisés sur un même serveur, il n'y aura plus qu'à jouer avec les règles de filtrage et faire remonter l'information voulue à Nagios pour être alerté en cas de problème. À vos serveurs... ■

Liens intéressants

[1] http://ram-0000.developpez.com/tutoriels/reseau/Syslog/
[2] http://www.nagios.org
[3] http://www.rsyslog.com/
[4] http://assets.nagios.com/downloads/nrdp/docs/NRDP_Overview.pdf
[5] http://www.rsyslog.com/doc/rsyslog_conf_filter.html
[6] http://loganalyzer.adiscon.com/
[7] http://www.8pussy.org
[8] http://fr.splunk.com/
[9] http://graylog2.org/
[10] http://logstash.net/
[11] http://logio.org/

Remerciements

Un très grand merci à mes deux relecteurs : Emeline et Emmanuel LEVEAU.

COMMENT KANBAN A CHANGÉ MA VIE

par Cyrille DERUEL
[Delivery Manager @ OCTO Technology]

Rassurez-vous, je ne vais pas vous parler de ma vie personnelle, mais plutôt de ce qui a changé dans ma vie de chef de projet depuis que je mets en place Kanban dans les projets.

Le métier de chef de projet n'est pas si simple, vous devez satisfaire votre client en lui livrant une application qui correspond à son besoin, respecter votre budget ainsi que les délais. Le chef de projet a aussi un autre challenge à relever : communiquer auprès de son équipe afin de garder une bonne ambiance générale ! Pas toujours si facile !

J'oubliais le plus important : votre application doit avoir un très haut niveau de qualité sinon vous allez passer la majorité de votre temps à corriger des bugs au lieu de développer de nouvelles fonctionnalités.

On distingue 4 notions autour du mot kanban :

- Un système kanban (avec un k minuscule), système inventé par Taiichi Ohno dans les années 50 au sein des ateliers de Toyota ;
- La méthode de développement logiciel Kanban avec un K majuscule, inventée par David Anderson dans les années 2000 ;
- Un kanban board (souvent appelé kanban), avec un k minuscule ;
- Personal kanban, avec un k minuscule, la méthode inventée par Jim Benson dans les années 2000.

Les objectifs d'un système kanban et la méthode de développement logiciel Kanban restent les mêmes : éliminer le gaspillage par le maintien de la taille des stocks intermédiaires à un niveau raisonnable. Avant de détailler ce que m'a apporté Kanban, regardons les 5 grands principes de cette méthode.

Un système Kanban possède 5 pratiques :

- Visualiser les éléments de travail et le processus ;
- Limiter le travail en cours (WIP : *Work In Progress*). Chaque activité (spécification, développement ou tests) possède une limite de capacité permettant d'obtenir un flux tiré ;
- Manager le flux afin de s'assurer de la fluidité des activités. Cette fluidité va nous permettre d'améliorer notre prédictibilité de notre système ;
- La mise en place de règles explicites de votre processus ;
- S'améliorer collaborativement, c'est-à-dire que tous les membres du projet sont acteurs dans l'amélioration de leur système Kanban.

Regardons l'impact et la mise en œuvre de ces 5 pratiques à travers un retour d'expérience (Figure 1).

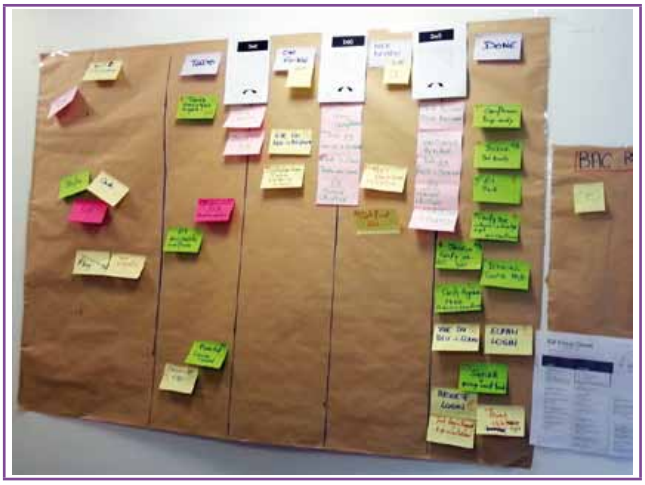


Figure 1

Tout a commencé début juin, j'intervenais dans un projet avec 7 développeurs pour sécuriser la livraison du projet le 31 décembre suite à l'arrivée d'une nouvelle loi réglementaire. Le projet avançait globalement bien, mais l'intégration de nouveaux développeurs dans l'équipe s'était mal passé les mois précédents. Le plan initial prévoyait l'intégration de 10 autres développeurs en août pour construire des mini-sites autonomes, mais tous interconnectés entre eux.

Mon premier travail a consisté à identifier le mode de fonctionnement actuel du projet : le métier rédige des spécifications (plus ou moins simples), ces spécifications sont analysées avec les « LeadDeveloppers », elles sont ensuite priorisées dans la liste des fonctionnalités à développer. Les développeurs présentent les écrans et ajustent les

spécifications avec le métier, puis les interfaces et règles de gestion sont codées et livrées pour recette au métier.

Mon premier kanban se dessinait avec les étapes suivantes :

- Spécification ;
- Développement des écrans ;
- Développement des règles de gestion et des interfaces ;
- Recette globale de la fonctionnalité ;
- Prêt pour livrer aux utilisateurs ;
- Une dernière étape lorsque la fonctionnalité est livrée au métier.

Voici sa représentation en figure 2.

Bien sûr, vous vous demandez pourquoi le développement s'est fait en 2 fois, la première fois pour les écrans et la deuxième pour le restant de la fonctionnalité. Je m'étais posé la question, mais il est important d'avancer par petits pas et donc de ne pas remettre en cause ce qu'il se passe actuellement.

À partir du moment où nous avions notre Kanban affiché, l'équipe, le métier et moi avons placé les fonctionnalités sur le Kanban et nous avons tous eu nos premières surprises :

- des fonctionnalités que l'équipe pensait terminée et ne l'étaient pas par le métier ;
- des fonctionnalités que le métier pensait en cours de développement étaient en attente de spécification par l'équipe de développement ;
- etc.

J'étais content, car tous les acteurs du projet partageaient le même état du projet même si je me serai bien passé de certaines surprises !

Le plus important : nous avions passé l'étape 1 de la mise en place de notre Kanban : « Visualiser les éléments de travail et le processus et avoir une vision partagée de l'avancement du projet ».

Durant 3 ou 4 semaines, nous n'avons rien changé, nous utilisions ce Kanban uniquement pour savoir où en était le projet, c'est après cette période que nous avons intégré une nouvelle règle : clarifier les conditions permettant à une fonctionnalité de changer de colonne. Il devait y avoir environ 20 fonctionnalités en attente de recette et l'équipe continuait à développer et pousser de nouvelles fonctionnalités dans la colonne « Test ». Jusqu'au jour où le métier a commencé à tester ! Le métier a remonté énormément de bugs, pas des plantages, mais des coquilles comme des fautes d'orthographes, des erreurs d'arrondi, des erreurs d'intégrations graphiques, etc.

Toutes ces erreurs ne concernaient pas le fonctionnel, mais la qualité du logiciel, au final le métier a remonté plus de 50 ajustements.

Malgré le fait que tous ces retours étaient justifiés, l'équipe de développement avait deux autres problèmes :

- nos estimations des dates d'arrivée devenaient intenables et ces ajustements remettaient en cause la fluidité de notre organisation ;
- les développeurs « switchaient » entre plusieurs tâches entraînant un agacement palpable au sein de cette équipe.

Après cette « mini-tempête », nous avons mis en place une limite de travail (WIP) par colonne, nous avons tâtonné pour trouver les 4 limites sur les 4 grandes colonnes de notre tableau.

La mise en place des limites de travail en cours sur chaque colonne « Spécification », « Dev écran », « Développement » et « Test » s'est déroulée assez naturellement.

Les limites de Kanban interdisent d'ajouter des actions (tâche ou fonctionnalités) dans une colonne si elle est pleine. Il ne sert à rien d'ajouter des choses à faire tant que les suivantes ne sont pas terminées. Les rôles des intervenants peuvent alors être ponctuellement redistribués pour libérer une colonne.

L'équipe peut être amenée à travailler sur les fonctionnalités en cours de test pour débloquer une situation.

Cette limite d'encours vous permet de basculer d'un flux poussé à un flux tiré. Dans un flux poussé, les développeurs « déversent » leurs fonctionnalités dans la colonne à tester quoi qu'il arrive alors que dans un flux tiré, les développeurs ne pourront développer des fonctionnalités uniquement si le niveau de tâche dans la colonne suivante est en dessous du seuil (WIP).

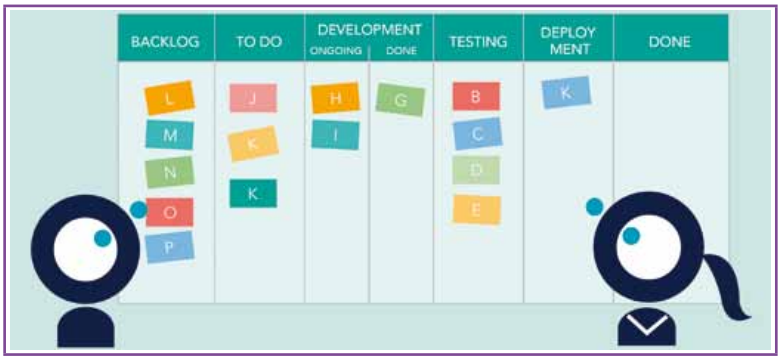


Figure 2

Ce « mini » changement a aussi un autre effet sur l'équipe projet. Avant la mise en place du WIP, chaque personne ne regardait que sa propre colonne :

- Le métier surveillait la colonne « spécification » et la colonne « test » ;
- Les développeurs étaient concentrés sur les colonnes « dev écran » et « développement ».

Après la mise en place du WIP, tous les acteurs du projet analysaient le Kanban dans sa globalité à la recherche du goulot d'étranglement, mais aussi des solutions à apporter.

Comme chaque colonne possédait maintenant un stock faible de fonctionnalités, je me suis concentré sur les fonctionnalités qui remontaient le Kanban, pas les fonctionnalités qui se déplaçaient de la gauche vers la droite, mais celles qui revenaient dans des colonnes en amont. Le fait d'avoir des fonctionnalités qui remontent le Kanban indique des problèmes/manques sur les règles permettant la transition d'une colonne à une autre. Mon objectif était simple : faire prendre conscience aux acteurs du projet que les règles permettant le passage d'une fonctionnalité d'une colonne à une autre n'étaient pas gravées dans le marbre. Les fonctionnalités remontant le flux étaient donc un excellent prétexte pour identifier ces problèmes. Les listes entre chaque colonne ont continué à évoluer tout au long du projet et l'intégration de nouveaux développeurs sur le projet s'était grandement facilitée.

J'avais gagné bien plus que la possibilité de connaître le véritable avancement du projet ; j'avais une équipe qui connaissait seule l'avancement du projet, les difficultés en cours, les prochaines fonctionnalités qui allaient arriver. Je n'étais plus tout seul à porter le projet, mais toute l'équipe avait un avis, une stratégie et des choix à proposer pour tenir le délai. Cette dynamique a émergé dès les premières semaines de la mise en place du Kanban. Elle a aussi eu un effet très positif par les acteurs du métier, ils arrivaient à savoir si l'équipe était assez « alimentée » en spécification.

Comme vous l'avez vu, la mise en place d'un Kanban n'est pas quelque chose de très compliqué. Personnellement, il m'a permis de trouver d'autres activités sur le rôle de chef de projet : je suis passé du « contremaître » responsable de l'avancement à un rôle de facilitateur d'équipe qui cherche à garantir les conditions propres à la bonne réalisation du projet.

Ma fierté à cette époque n'était pas d'avoir mis en place un Kanban, mais d'être content d'aller travailler, d'aller rejoindre une équipe aussi impliquée et heureuse de travailler sur ce projet, d'avoir un client aussi serein malgré toutes les difficultés que nous avons rencontrées.

Au final, nous avons livré la majorité des fonctionnalités critiques du projet courant novembre et nous avons continué à livrer de nouvelles fonctionnalités courant décembre pour déployer l'application en production avant les fêtes de fin d'année.

Je pense que la réussite de ce projet n'est pas liée à la mise en place du Kanban. Un système Kanban n'est qu'un moyen. La réussite de ce projet est due à la forte collaboration de tous les acteurs du projet, du métier jusqu'au développeur et à l'ambiance positive qui régnait sur le plateau.

L'une des plus grandes difficultés dans la mise en place d'un Kanban est de réussir à instaurer un climat de confiance à tous les acteurs du projet, de leur faire oublier leur côté « Command & Control », accepter que l'intelligence collective soit plus forte qu'un chef de projet, même très expérimenté.

Bien sûr, la chance m'a aidé sur ce projet, la chance d'avoir une équipe sur un grand plateau projet, la chance d'avoir un responsable métier qui m'a fait confiance rapidement.

Si vous souhaitez assimiler rapidement les grands principes du Kanban, il existe un jeu de plateau « GetKanban » permettant de simuler le développement d'un logiciel. Ce jeu est disponible gratuitement en version 2 à l'adresse suivante : http://GetKanban.com. Je co-anime ce jeu depuis plus d'un an et je ne peux que vous conseiller de le découvrir. J'essaie d'animer une session tous les 2 mois dans le cadre de formations ou de soirées découvertes du Kanban (Figure 3).

Vous allez peut-être vouloir utiliser un logiciel pour gérer vos tâches dans votre Kanban. Il existe une multitude d'offres répondant à ce besoin. Attention, certaines applications ne vous offrent pas la possibilité de gérer votre limite de tâches en cours (WIP) cela sera à vous de vous limiter.

Personnellement, j'utilise Trello depuis quelque temps : http://www.trello.com

Trello m'offre plusieurs fonctionnalités intéressantes ; je peux créer autant de Kanban que je le souhaite, ce qui me permet d'avoir un Kanban pour mes différentes activités. Trello vous autorise le partage d'un Kanban avec d'autres utilisateurs, vous pouvez facilement travailler à plusieurs sur le même Kanban pour partager l'avancement d'une activité. Dernier point, Trello a développé des applications sur iPhone et Android.

J'espère vous avoir donné envie d'essayer de mettre en place un système Kanban dans votre projet et j'espère pouvoir échanger avec vous très prochainement sur les difficultés et succès que vous avez rencontrés. ■

Figure 3

Abonnez-vous !

Profitez de nos offres d'abonnement spéciales disponibles au verso !

Téléphonez au 03 67 10 00 20 ou commandez par le Web

Économisez plus de 25%*

* Sur le prix de vente unitaire France Métropolitaine

11 Numéros de GNU/Linux Magazine

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez GNU/Linux Magazine chaque mois chez vous ou dans votre entreprise.
- Économisez 24,50 €/an ! (soit plus de 3 magazines offerts !)

4 façons de commander facilement :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur www.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-18h au 03 67 10 00 20
- par fax au 03 67 10 00 21

ABONNEMENT

11 GNU/LINUX MAGAZINE / FRANCE

58 €*

au lieu de 82,50 €* en kiosque

Économie : 24,50 €*

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE
Pour les tarifs hors France Métropolitaine, consultez notre site : www.ed-diamond.com

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

Tournez SVP pour découvrir toutes les offres d'abonnement >>

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
e-mail :	

☐ Je souhaite recevoir les offres promotionnelles et newsletter des Éditions Diamond.

☐ Je souhaite recevoir les offres promotionnelles de nos partenaires.

GNU/LINUX MAGAZINE / FRANCE

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 :

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **www.ed-diamond.com/cgv** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

PROFITEZ DE NOS OFFRES D'ABONNEMENT SPÉCIALES POUR LIRE PLUS ET FAIRE DES ÉCONOMIES !

➡ Abonnement

offre 1 — ABONNEMENT — 11 LINUX Magazine France — **58€*** au lieu de 82,50€** en kiosque — Economie : 24,50 €

Vous pouvez également vous abonner sur :

www.ed-diamond.com

ou par Tél. : +33 (0)3 67 10 00 20 / Fax : +33 (0)3 67 10 00 21

NOUVEAU !

numerique.ed-diamond.com
pour vous abonner et acheter vos magazines en format numérique (PDF)

unixgarden.com
pour retrouver une sélection d'articles des Éditions Diamond

* Tarifs France Métro (F)
** Base tarifs kiosque zone France Métro (F)

➡ Voici nos offres d'abonnements groupés incluant GLMF

offre 3 — ABONNEMENTS GROUPÉS — 11 LINUX Magazine France + 6 LINUX Pratique — **85€*** au lieu de 121,50€** en kiosque — Economie : 36,50 €

offre 4 — ABONNEMENTS GROUPÉS — 11 LINUX Magazine France + 6 Hors-séries — **89€*** au lieu de 130,50€** en kiosque — Economie : 41,50 €

offre 5 — ABONNEMENTS GROUPÉS — 11 LINUX Magazine France + 6 MISC — **90€*** au lieu de 133,50€** en kiosque — Economie : 43,50 €

offre 6 — ABONNEMENTS GROUPÉS — 11 LINUX Magazine France +6 HS + 6 LINUX Pratique — **119€*** au lieu de 169,50€** en kiosque — Economie : 50,50 €

offre 7 — ABONNEMENTS GROUPÉS — 11 LINUX Magazine France +6 HS + 6 MISC — **124€*** au lieu de 181,50€** en kiosque — Economie : 57,50 €

offre 8 — ABONNEMENTS GROUPÉS — 11 LINUX Magazine France +6 HS + 6 LINUX Pratique + 6 MISC — **154€*** au lieu de 220,50€** en kiosque — Economie : 66,50 €

offre 9 — ABONNEMENTS GROUPÉS — 11 LINUX Magazine France +6 HS + 6 LINUX Pratique + 6 MISC + 6 LINUX Essentiel — **184€*** au lieu de 259,50€** en kiosque — Economie : 75,50 €

offre 12 — ABONNEMENTS GROUPÉS — 11 LINUX Magazine France +6 HS + 6 LINUX Pratique +3 HS + 6 MISC +2 HS + 6 LINUX Essentiel — **215€*** au lieu de 301,50€** en kiosque — Economie : 86,50 €

offre 2 — ABONNEMENTS GROUPÉS — 6 LINUX Essentiel + 6 LINUX Pratique — **60€*** au lieu de 78,00€** en kiosque — Economie : 18,00 €

➡ Voici nos autres offres d'abonnements groupés

offre 10 — ABONNEMENTS GROUPÉS — 6 MISC + 2 Hors-séries — **48€*** au lieu de 69,00€** en kiosque — Economie : 21,00 €

offre 11 — ABONNEMENTS GROUPÉS — 6 LINUX Pratique + 3 Hors-séries — **48€*** au lieu de 63,00€** en kiosque — Economie : 15,00 €

offre 15 — ABONNEMENTS GROUPÉS — 6 LINUX Essentiel + 6 LINUX Pratique +3 HS — **78€*** au lieu de 102,00€** en kiosque — Economie : 24,00 €

➡ Nos Tarifs s'entendent TTC et en euros

		F France Métro	D DOM	T TOM	Zone 1 Europe	Zone 2 Afrique / Orient	Zone 3 Amérique	Zone 4 Asie / Océanie
1	Abonnement GLMF	58 €	78 €	94 €	80 €	87 €	84	80 €
2	Abonnement LPE + LP	60 €	86 €	105 €	88 €	96 €	92 €	89 €
3	Abonnement GLMF + LP	85 €	120 €	145 €	123 €	134 €	129 €	124 €
4	Abonnement GLMF + GLMF HS	89 €	122 €	147 €	125 €	136 €	131 €	126 €
5	Abonnement GLMF + MISC	90 €	128 €	151 €	130 €	141 €	136 €	131 €
6	Abonnement GLMF + GLMF HS + Linux Pratique	119 €	164 €	198 €	168 €	183 €	176 €	170 €
7	Abonnement GLMF + GLMF HS + MISC	124 €	172 €	204 €	175 €	190 €	183 €	177 €
8	Abonnement GLMF + GLMF HS + MISC + LP	154 €	214 €	255 €	218 €	237 €	228 €	221 €
9	Abonnement GLMF + GLMF HS + MISC + LP + LPE	184 €	258 €	309 €	263 €	286 €	275 €	266 €
10	Abonnement MISC + MISC HS	48 €	66 €	76 €	66 €	72 €	69 €	68 €
11	Abonnement LP + LP HS	48 €	65 €	78 €	66 €	72 €	70 €	68 €
12	Abonnement GLMF + GLMF HS + MISC + MISC HS + LP + LP HS + LPE	215 €	297 €	355 €	302 €	329 €	317 €	307 €
15	Abonnement LPE + LP + LP HS	78 €	109 €	132 €	111 €	121 €	117 €	113 €

- ZONE 1 : Allemagne, Belgique, Danemark, Italie, Luxembourg, Norvège, Pays-Bas, Portugal, Suède, Autriche, Espagne, Finlande, Grande Bretagne, Grèce, Islande, Suisse, Irlande, Estonie, Croatie, Slovenie, Slovaquie, Republique Tcheque, Pologne, Bielorussie, Bosnie Herzegovine, Bulgarie, Chypre, Georgie, Hongrie, Lettonie, Lituanie, Macedoine, Malte, Moldova, Roumanie, Russie, Serbie, Ukraine, Albanie, Armenie, ...
- ZONE 2 : Algerie, Maroc, Tunisie, Turquie, Afrique du Sud, Seychelle, Senegal, Israel, Palestine, Syrie, Jordanie, Botswana, Cameroun, Cap Vert, Comores, Rep.Dom. Congo, Cote d'Ivoire, Egypte, Kenya, Libye, Madagascar, Nigeria, ...
- ZONE 3 : Canada, Etats Unis, Guyana, Haiti, Republique Dominicaine, Jamaique, Argentine, Bresil, Cuba, Mexique, ...
- ZONE 4 : Australie, Japon, Chine, Coree du Nord, Coree du Sud, Inde, Indonesie, Nouvelle Zelande, Taiwan, Thailande, Vietnam, ...

Mes choix :

Mon 1er choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Mon 2ème choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Mon 3ème choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
	Je sélectionne ma zone géographique (F à Zone 4) :	
	J'indique la somme due : (Total)	€

Exemple : je souhaite m'abonner à l'offre GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC (offre 7) et je vis en Belgique (zone 1), ma référence est donc 7zone1 et le montant de l'abonnement est de 175 euros.

Je choisis de régler par :

- ❑ Chèque bancaire ou postal à l'ordre des Éditions Diamond
- ❑ Carte bancaire n° ⌊_⌊_⌊_⌊_ ⌊_⌊_⌊_⌊_ ⌊_⌊_⌊_⌊_ ⌊_⌊_⌊_⌊_
  Expire le : ⌊_⌊_⌊_⌊_
  Cryptogramme visuel : ⌊_⌊_⌊_

Date et signature obligatoire

PKI SUR ANDROID

par Philippe PRADOS [Consultant Senior @ OCTO Technology]

Nous allons étudier les différentes technologies proposées publiquement, ou cachées, permettant d'installer et de sécuriser un certificat numérique client, lors de l'exploitation d'un terminal Android. Plus généralement, comment sécuriser un secret applicatif ?

Afin d'améliorer la sécurité, de plus en plus d'entreprises souhaitent utiliser des certificats numériques pour chaque utilisateur de terminaux Android. Il est alors possible d'ouvrir des connexions HTTPS (SSL v3) avec le serveur, avec une authentification mutuelle forte. Le client est certain du serveur qu'il consomme. Le serveur est certain du client qu'il alimente.

Pour rappel, les certificats numériques sont des fichiers binaires possédant la clef publique d'une bi-clef et d'autres informations. La clef privée permet de prouver que l'utilisateur est bien celui qu'il prétend être.

L'utilisation de certificat numérique a pour objectif d'augmenter la sécurité par rapport à l'utilisation d'un couple utilisateur/mot de passe. En effet, les utilisateurs ont tendance à utiliser le même mot de passe pour des usages différents. Une compromission du mot de passe de l'utilisateur permet généralement au pirate d'obtenir un accès à tous les services qu'il utilise.

L'utilisation de certificat numérique permet de passer du modèle « je connais » (le mot de passe) à « je possède » (un certificat). Souvent, une combinaison des deux est appliquée : je connais le mot de passe qui permet d'avoir accès au certificat que je possède. Dans ce scénario, le mot de passe est validé localement, lors du déchiffrement du certificat.

Cela permet également de ne pas avoir à maintenir une base de données avec les mots de passe des utilisateurs. Par principe, tous les certificats utilisateurs signés par une autorité sont valides. Ainsi, une compromission du serveur ne révèle pas les secrets des utilisateurs. Seuls les numéros des certificats doivent être mémorisés afin de pouvoir les révoquer.

La révocation du certificat client permet de fermer l'accès aux services de l'entreprise.

Les certificats peuvent également être utilisés pour chiffrer des données dans le téléphone. Ainsi, le vol du terminal ne révélera pas les informations sensibles.

Dans l'idéal, les certificats numériques doivent être générés par l'utilisateur avant d'être validés par l'autorité de certification. La génération peut être réalisée en software ou par une puce sécurisée. La signature du certificat proposé par le client ne doit s'effectuer qu'après certaines vérifications. Ainsi, l'utilisateur ne peut pas répudier ses activités.

Si le certificat est généré sur le serveur, il existe des scénarios permettant à l'entreprise d'effectuer des actions au nom de l'utilisateur, à son insu. En effet, à un moment du processus, le certificat avec la clef privée de l'utilisateur a été présent sur le serveur. En mémorisant ces informations, le serveur est capable de se faire passer pour l'utilisateur.

L'utilisation de certificat client est plus ou moins bien intégrée dans les terminaux Android, suivant les versions du système d'exploitation.

Nous allons étudier les différentes technologies de sécurité présentes dans Android, publiques ou cachées, nous permettant d'exploiter une authentification forte, avec le maximum de sécurité. Ces technologies peuvent également être exploitées pour protéger tout secret applicatif.

Nous devons résoudre plusieurs challenges pour utiliser ces technologies dans les meilleures conditions. Nous serons capables d'importer et d'installer un certificat dans le mobile ; le sauver dans le périphérique dans un espace parfaitement sécurisé, résistant au vol du terminal et réduire au maximum le nombre d'applications ayant besoin d'avoir accès aux secrets.

1 Approche traditionnelle

Android est un système d'exploitation qui simule l'exécution simultanée de nombreuses applications. Techniquement, ce n'est pas le cas. Seules les dernières applications utilisées sont encore présentes en mémoire. Les plus anciennes sont tuées après avoir sauvegardé leurs contextes.

Il est possible de demander à l'utilisateur le mot de passe permettant de déchiffrer un certificat numérique et de le garder en mémoire. Mais, à chaque destruction de l'application par l'OS (lors de la réception d'un appel téléphonique, par

Figure 1

exemple), la mémoire est perdue. Lorsque l'application reprend, il faut à nouveau demander le mot de passe de l'utilisateur.

De par la structure de l'OS, il n'est pas concevable de demander un mot de passe dès qu'une application a besoin d'exploiter un certificat numérique.

Il est donc nécessaire de mémoriser le certificat ou le mot de passe de l'utilisateur sans chiffrement, directement dans la mémoire statique du téléphone. Cela l'expose à différentes attaques. Est-ce qu'un pirate peut analyser la mémoire statique du téléphone pour retrouver le certificat ?

Une approche subtile consiste à placer la clef privée ou le mot de passe dans un extra de l'`Intent` de l'activité courante où dans le `onSaveInstanceState()`. Ainsi la clef est mémorisée par le framework Android, dans le processus `system_app`. Il faut faire cela dans toutes les activités. Techniquement, les données peuvent alors être sauvées sur disque par le framework, exposant alors la clef ou le mot de passe.

Cette approche n'est pas satisfaisante au niveau sécurité, au niveau développeur ou au niveau de l'expérience utilisateur.

Pour gérer un certificat numérique, nous devons résoudre plusieurs challenges techniques :

- Comment installer ou générer le certificat sur le terminal ?
- Comment protéger le certificat d'un vol du terminal ou de l'exploitation d'une faille ?
- Comment réduire au maximum l'exposition des secrets ?

Nous allons étudier toutes les solutions qui s'offrent à nous pour cela.

2 Protéger le certificat dans le périphérique

Les certificats doivent être installés dans l'Android. Est-ce que les certificats sont correctement sécurisés ?

2.1 Secure Element (SE)

Un SE est un composant électronique capable de recevoir des mini-applications et de communiquer via des trames binaires. C'est essentiellement un mini ordinateur sur un simple composant, avec CPU, ROM, EEPROM, RAM et des ports I/O. Les dernières générations sont équipées de co-processeurs cryptographiques capables d'implémenter les algorithmes standards comme DES, AES et RSA. Ces composants utilisent des techniques pour résister à différentes attaques physiques, afin de rendre impossible l'extraction de données (Figure 1).

Typiquement, il s'agit de composants compatibles Java Card Runtime Environment (JCRE). Suivant les versions des téléphones, il peut exister plusieurs éléments de sécurité. Un SE peut être présent dans la puce du téléphone, ajouté à l'aide du port SD du téléphone ou directement embarqué dans le périphérique.

Dans ce dernier scénario, le composant est généralement lié au composant NFC. Dans ce cas, il existe trois modes d'utilisation. Soit le composant est désactivé, soit il est directement en écoute des trames NFC venant de l'extérieur du téléphone. Les applications embarquées dans le SE sont alors capables de simuler une carte bancaire ou de transport. Soit le composant est visible des applications comme s'il s'agissait d'une communication avec une carte bancaire externe au téléphone (Figure 2).

Figure 2

Les communications NFC peuvent fonctionner en trois modes : simulation de tag passif, peer-to-peer ou simulation de carte à puce. À ce jour, ce dernier mode n'est pas disponible par soft, sauf à utiliser une version spéciale de Cyanogen.

Les SE servent généralement à mémoriser les secrets comme les clefs privées, les accès VPN, ou les générateurs de mot de passe à usage unique (One Time Password – OTP).

Il existe des applications Javacard pour offrir un environnement permettant de gérer une PKI via un SE. Le projet M.U.S.C.L.E. **[1]** en est un exemple. Il permet de connecter les API SSL avec le SE pour la protection de la clef privée et la génération des clefs symétriques.

Malheureusement, Android ne propose pas d'API pour communiquer avec ces composants. Le projet seek-for-android propose une librairie complémentaire pour Android, permettant d'accéder à ces différents composants. Elle est parfois présente sur certains terminaux **[2]** comme le Sony Xperia S et les Samsung Galaxy S3 et S2 NFC. Pour y accéder, il est nécessaire de lier l'application à la librairie, via un marqueur **<use-library/>**.

```
<uses-library android:name="org.simalliance.openmobileapi" android:required="true" />
```

Cette approche n'étant pas standard, nous décidons de ne pas l'utiliser.

2.1.1 SE dans la puce 3G

Android est découpé en deux parties distinctes. Un processeur est chargé du système d'exploitation. Un autre processeur économe en ressource est chargé des communications radios. Cette partie du logiciel n'est pas publique. Les deux composants communiquent entre eux à l'aide de commandes textuelles, préfixées par **AT**. Cela vient des modems Hayes **[3]** à l'initiative de cette norme, permettant de séparer les données destinées aux modems des données destinées aux réseaux téléphoniques, à l'époque où les modems étaient connectés via des liaisons séries type RS232.

Pour communiquer avec la puce 3G, il faut enrichir le vocabulaire AT du module radio **[4]** conformément aux spécifications 3GPP TS 27.007[5].

- **AT+CSIM** (Generic SIM access)
- **AT+CCHO** (Open Logical Channel)
- **AT+CCHC** (Close Logical Channel)
- **AT+CGLA** (Generic UICC Logical Channel Access)

Sans intégration dès l'origine de ces commandes par le constructeur, il n'est pas possible de communiquer avec le SE présent dans la puce du téléphonique. Les opérateurs font du forcing pour autoriser cela, afin de capter le marché des applications dans les puces SIM. Ils facturent l'installation d'applications dans la puce, voire chaque transaction. Comment tuer un marché naissant ?

Une autre approche consiste à exploiter le Single Wire Protocol **[6]** pour communiquer avec la puce, via la connexion NFC. C'est le cas des Galaxy Nexus et Nexus S, mais désactivé par défaut.

2.1.2 SE intégré

Android 2.3.4 et supérieur propose une API non publique pour manipuler le SE intégré aux téléphones ou tablettes récents. Malheureusement, cette API nécessite un privilège **system**, qui ne peut être accordé qu'au fournisseur du téléphone ou à Google pour la famille Nexus. L'application doit être signée par le signataire de la plate-forme.

Android 4.0.4 (API niveau 15) modifie cela en utilisant à la place une liste blanche de signatures autorisées (**/etc/nfcee_access.xml**). Cette liste est figée sur la plate-forme. Elle peut être mise à jour Over The Air (OTA) par le constructeur. Ce fichier indique la signature et la liste des packages valides.

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
  <signer android:signature="30820...90">
    <package android:name="org.foo.nfc.app">
  </package></signer>
</resources>
```

Avec un accès **root**, il est possible de remonter la partition en mode lecture/écriture pour modifier ce fichier. Seule la permission NFC est alors nécessaire. L'application doit également ajouter une librairie.

```
<uses-library
   android:name="com.android.nfc_extras"
   android:required="true" />
```

Ensuite, une API minimaliste permet de communiquer des trames binaires avec le SE.

```
NfcAdapterExtras adapterExtras =
  NfcAdapterExtras.get(NfcAdapter.getDefaultAdapter(context));
NfcExecutionEnvironment nfceEe = adapterExtras.
getEmbeddedExecutionEnvironment();
nfcEe.open();
byte[] response = nfcEe.transceive(command);
nfcEe.close();
```

Google utilise cela pour Google Wallet. Il s'agit du portefeuille sécurisé de Google, pour y placer toutes les cartes de crédit, données particulièrement sensibles.

Trois applications sont installées sur le SE : l'applet de contrôle du portefeuille, une applet de gestion MIFARE, et bien sûr, l'applet de simulation de carte bancaire, compatible avec les terminaux PayPass **[7]**. La première applet active ou désactive l'applet de simulation de carte suivant la saisie d'un code PIN par l'utilisateur. La deuxième applet sert de conteneur aux offres et tickets numériques (métro de Londres par exemple). La dernière simule une EMV classique comme les cartes bancaires. Cette dernière est active ou non suivant le verrouillage du téléphone. Le NFC est paramétré pour faire transiter les trames vers le SE correspondant.

Ce composant est très intéressant pour pouvoir sauver notre clef privée, ou mieux encore, pour demander la création de clef symétrique lors de l'ouverture d'une communication TLS.

Nous ne pouvons pas utiliser cela, car seul le Trusted Service Manager **[8]** (TSM) du composant est habilité à installer des applications dans le SE. Google envisage d'étendre Google Wallet aux autres applications **[9]**, mais ce n'est pas le cas à ce jour (avril 2013).

Nous avons exploré les techniques hardwares sans succès, car elles ne sont pas encore accessibles aux applications.

2.2 Chiffrement du disque

Depuis la version 3.x d'Android, il est possible de chiffrer l'intégralité du disque du terminal. Lors du *boot* de ce dernier, un mot de passe est demandé. Ce dernier permet alors de rebooter le terminal sur la partition Android, pour déchiffrer l'intégralité des secteurs à la volée. Initialement, le mot de passe de déchiffrement est identique au mot de passe de déverrouillage du téléphone. Ce dernier permet de déchiffrer le mot de passe de chiffrement du disque, présent dans le dernier secteur.

Ainsi, toutes les données présentes sur le téléphone sont protégées. Il n'est plus possible – hors force brute ou récupération de la clef en RAM **[10]** – de récupérer les informations. Cela peut être exigé par le gestionnaire du parc des téléphones.

Cette approche présente néanmoins quelques inconvénients. Il est nécessaire d'avoir un mot de passe alpha-numérique pour débloquer la session de l'utilisateur ; le démarrage du terminal est particulièrement fastidieux, car il faut lancer deux fois de suite un OS Android (la première fois pour pouvoir demander la saisie du mot de passe, la deuxième fois pour l'OS avec déchiffrement). Les utilisateurs préfèrent alors utiliser un mot de passe relativement simple, exposant ainsi le terminal à une attaque par force brute. Les performances sont légèrement dégradées, car il faut chiffrer/déchiffrer tous les accès disques.

Avec un téléphone rooté, il est possible d'utiliser un mot de passe de déchiffrement du disque, différent du mot de passe de déverrouillage du téléphone.

```
$ su -c vdc cryptfs changepw newpass
200 0 0
```

L'application Cryptfs password **[11]** propose une interface utilisateur pour cela, avec un téléphone rooté.

Avec beaucoup de temps, si le mode debug est actif, il est possible d'essayer différents mots de passe pour le déverrouillage du téléphone, en injectant des touches.

```
adb shell input text PASSWORD
adb shell input keyevent 66
```

Pour débloquer un téléphone et le passer éventuellement en mode root, il faut généralement l'éteindre une fois. Pour des raisons de sécurité, cela efface l'intégralité des données. C'est la procédure standard pour débloquer un téléphone de la famille Nexus, via l'utilitaire fastboot.

```
fastboot oem unlock
```

Cet état s'identifie par le petit cadenas qui est ouvert lors du boot du téléphone, sur le premier écran.

Le chiffrement est une protection contre l'extinction du téléphone. C'est généralement nécessaire pour passer un téléphone quelconque à root. Donc, cela protège les données.

Par contre, si le téléphone est déjà rooté, c'est plus facile pour l'attaquant. Un **adb shell** lui permet d'avoir accès à toutes les données déchiffrées. Les dernières versions de l'OS ne permettent pas d'accrocher un **adb** au device, sans l'accord de ce dernier. Il faut débloquer le téléphone pour pouvoir brancher un **adb**, donc il faut avoir l'utilisateur à côté de soi. Ainsi, même en root, il n'est plus possible d'attaquer un téléphone avec le port USB.

Le chiffrement ne protège pas contre une vulnérabilité de l'OS. En effet, toutes les applications ont un accès déchiffré aux données. Tant que le téléphone n'est pas éteint ni verrouillé, il est possible d'installer une application exploitant une faille permettant de devenir root, et d'avoir ainsi accès à toutes les données. Les dernières versions de l'OS ne possèdent pas de vulnérabilité publique permettant d'être root "à chaud".

Nous pouvons sauvegarder nos certificats dans les données des applications, mais rien ne garantit que l'utilisateur chiffre le disque. Nous ne pouvons pas l'imposer. Cette approche est donc à éliminer pour protéger nos données.

2.3 KeyStore

Contrairement à iOS, il n'existe pas officiellement d'API pour protéger les secrets des applications. Chaque application peut sauver des informations dans son espace protégé dans la mémoire statique. Les autres applications ne peuvent pas y accéder. Mais, en cas de vol du téléphone, une analyse de la mémoire de masse, en dehors de l'OS, permet de retrouver tous les secrets.

Il existe pourtant un composant bien caché, permettant de protéger les données. En effet, le système Android lance un service interne codé en C, permettant de protéger les différents certificats numériques. Les applications peuvent communiquer avec ce dernier, via un socket local, pour demander le déblocage du conteneur, y stocker des

Figure 3

Figure 4

secrets ou de les retrouver. Les secrets sont isolés par application.

Ce conteneur est chiffré à l'aide d'un dérivé d'un mot de passe, appliqué suffisamment de fois pour résister à une attaque en force brute.

Le comportement est différent suivant les versions de l'OS.

Pour les versions 2.x, le menu *sécurité* propose à l'utilisateur d'indiquer un mot de passe pour ce conteneur. Il est alors possible d'y stocker des données. C'est ici que se trouvent les certificats numériques permettant une identification Wifi ou VPN. Lorsqu'une application a besoin d'accéder à ces certificats, l'utilisateur doit saisir au moins une fois le mot de passe afin de débloquer le conteneur (Figure 3).

Pour les versions suivantes d'Android, le mot de passe n'est pas spécifique au conteneur. Il s'agit d'un dérivé du verrou du téléphone. Si le téléphone est verrouillé, le conteneur est bloqué. Lorsque l'utilisateur déverrouille le téléphone, le conteneur est ouvert aux applications. Il n'est donc pas possible d'utiliser ce conteneur sans verrou du téléphone.

L'API n'est pas publique, car elle impose de sauvegarder la clef privée dans le conteneur. Google veut se réserver la possibilité de sauvegarder la clef privée dans un composant type SE. Pour ne pas proposer d'API qu'il sera difficile de maintenir dans le temps, Google préfère ne pas exposer ce code.

En attendant, c'est un module très sympathique pour y placer notre clef privée.

Pour communiquer avec ce composant, il faut ouvrir un **`LockSocket`** avec le nom **`keystore`**. Ensuite, il faut envoyer des ordres en respectant un protocole propriétaire. Pour ne pas avoir à tout coder, il est facile d'extraire des sources d'Android, la classe **`android.security.KeyStore`** et de renommer le package. Cela fonctionne parfaitement avec les versions d'Android supérieures à 2.0. Au-delà de la version ICE CREAM SANDWICH, il existe une autre approche officielle pour les certificats clients, même si l'approche **`KeyStore`** fonctionne toujours. Nous pouvons supposer que lorsque la classe **`KeyStore`** sera supprimée, une approche alternative sera disponible.

Cette classe propose plusieurs méthodes intéressantes. Certaines sont accessibles à toutes les applications, d'autres non. Nous pouvons demander l'état du composant (**`state()`**) pour savoir s'il est bloqué ; sécuriser un tableau de byte (**`put()`**) et le récupérer (**`get()`**). Mais, impossible de débloquer le **`KeyStore`** par l'Api.

Il faut utiliser une autre approche consistant à déclencher une activité pour cela. Ainsi l'utilisateur garde le contrôle (Figure 4).

```
if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB)
  context.startActivity(new Intent("android.credentials.UNLOCK"));
else
  context.startActivity(new Intent("com.android.credentials.UNLOCK"));
```

Comme cette activité n'invoque pas **`setResult()`**, il faut gérer un petit automate à état dans votre activité. Avant de demander l'accès à une donnée, il faut vérifier si le conteneur est bloqué. Si c'est le cas, placez votre activité dans l'état d'attente de déblocage et lancez l'activité **`UNLOCK`**. N'oubliez pas de sauver l'état dans la méthode **`onSaveInstanceState()`** et de le récupérer dans **`onCreate()`**.

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  if (savedInstanceState!=null)
  {
    mState=savedInstanceState.getInt(EXTRA_STATE);
  }
}
@Override
protected void onSaveInstanceState(Bundle outState)
{
  super.onSaveInstanceState(outState);
  outState.putInt(EXTRA_STATE, mState);
}
```

En effet, l'activité déclenchée pour déverrouiller le conteneur fait partie d'une autre application. La vôtre peut donc être tuée par l'OS.

Dans le **`onResume()`** de votre activité, si l'état courant est en attente de déblocage et que le conteneur n'est plus bloqué, vous pouvez demander à nouveau

d'avoir accès à la donnée protégée. Sinon, l'utilisateur n'a pas été capable de débloquer le conteneur.

Après trois échecs de saisie du mot de passe du **KeyStore**, le conteneur est effacé.

3 API de certificats numériques sous Android

Android est capable de gérer des conteneurs de certificats clients. L'intégration n'est pas excellente, mais s'améliore de version en version.

Pour les versions 2.x (API version 5+), seuls les certificats permettant d'identifier l'utilisateur pour une connexion WIFI ou pour ouvrir un VPN sont possibles. La procédure d'installation du certificat n'est pas très simple, mais elle fonctionne. Il s'agit de déposer un fichier PKCS12 chiffré dans la racine de la carte SD. Ensuite, le menu Paramètre/Sécurité permet de déclencher l'analyse de ce répertoire. Il faut ensuite sélectionner le certificat à installer. Après saisie d'un mot de passe valide, le certificat est installé sans mot de passe dans le terminal et effacé de la carte SD. Comme le certificat n'est pas lui-même protégé, il doit être placé dans un conteneur lui-même chiffré. C'est le sens du « *mot de passe de stockage des identifiants* » que nous avons vu pour le KeyStore (Figure 5).

Figure 5

Le risque de cette approche est que la carte SD possède une version officiellement effacée, mais accessible. En récupérant ce fichier, il est envisageable d'effectuer une attaque en force brute off-line, pour récupérer la clef privée.

Pour les versions 3 et suivantes d'Android (API level 11+), il est possible d'installer des certificats clients pour une utilisation Web. Il est même possible de sélectionner les autorités de certifications. Lors de la navigation sur un site Web demandant une authentification mutuelle, une boite de dialogue demande le certificat à utiliser pour ouvrir la connexion.

Les certificats clients sont utilisés par le navigateur par défaut. Les navigateurs Opera ou Firefox ne savent pas exploiter ce conteneur pour récupérer les certificats clients à utiliser. Chrome l'exploite à partir de la version beta 27 sortie le 10 avril 2013.

Depuis ICE CREAM SANDWICH (API level 14+), des APIs sont disponibles. Elles permettent de donner l'autorisation aux applications d'avoir accès à certaines clefs privées, sous le contrôle de l'utilisateur.

Il est à noter que l'API **VpnService** permet aux applications d'imposer un VPN privé pour leur utilisation. Les autres applications peuvent continuer à exploiter le réseau, sans le VPN. Le projet ICS-openvpn **[12]** est un exemple d'implémentation. C'est une approche souvent ignorée, et pourtant très pertinente pour séparer les flux pro et perso dans l'utilisation d'un BYOD.

3.1 KeyChain

Depuis l'API niveau 14 d'Android, la classe **KeyChain** permet d'exploiter les certificats clients présents dans le terminal. Le principe est le suivant. Si une application a besoin d'un certificat, elle déclenche un service pour demander à l'utilisateur d'en sélectionner un. L'application est alors enregistrée comme étant habilitée à avoir accès à la clef privée du certificat sélectionné. L'application peut alors la demander pour ouvrir une connexion TLS. Cela est mémorisé dans une base de données pour limiter les accès aux autres certificats. C'est l'utilisateur qui a la responsabilité d'indiquer le ou les certificats qu'une application peut consommer.

En pratique, cela donne :

```
KeyChain.choosePrivateKeyAlias(this,
  new KeyChainCallBack()
  {
    @Override
    public void alias(String alias)
    {
      mAlias=alias;
    }
  },
  new String[] {"RSA"},    // List of acceptable key types. null for any
  null,                    // issuer, null for any
  "internal.example.com",  // host name of server requesting the cert
  443,                     // port of server requesting the cert, -1 if unavailable
  null);                   // alias to preselect, null if unavailable
```

La call-back **alias()** est alors invoquée pour signaler le nom symbolique du certificat que l'application peut exploiter. La plupart des informations de cette méthode ne sont là qu'à titre informatif pour le client.

Puis, deux méthodes permettent de consommer le certificat client.

```
KeyChain.getCertificateChain(this, mAlias);
KeyChain.getPrivateKey(this, mAlias);
```

Ces deux méthodes peuvent être judicieusement exploitées dans les paramètres TLS.

Les certificats sont mémorisés dans le Keystore. Ce dernier est protégé avec la clef de verrouillage de l'écran. Ainsi, il n'est plus nécessaire d'avoir un mot de passe spécifique. Si l'utilisateur débloque l'écran, le Keystore est déverrouillé.

Il existe quelques techniques pour débloquer l'écran, si le mode débuggage est actif. Il est possible d'intervenir directement sur la base de données des paramètres.

```
adb shell
cd /data/data/com.android.providers.settings/databases
sqlite3 settings.db
update system set value=0 where name='lock_pattern_autolock';
update system set value=0 where name='lockscreen.
lockedoutpermanently';
.quit
reboot
```

sqlite3 n'est pas toujours disponible. Il est alors possible de supprimer le fichier du schéma.

```
adb shell rm /data/system/gesture.key
reboot
```

Au reboot, un schéma est demandé, mais il est possible d'indiquer n'importe quoi.

Ces techniques ne fonctionnent pas toujours, car maintenant, il faut être root pour intervenir et avoir le privilège de connecter un **adb**. Néanmoins, avec un émulateur, vous pouvez vous amuser à tester cela.

Quel est l'impact sur la protection du **KeyStore** ? Et bien, dans ce cas, un mot de passe est demandé à l'utilisateur.

Avec un accès à la mémoire via l'interface électronique JTAG, il est possible de brute-forcer le verrou du téléphone **[13]** ou de retrouver en mémoire les clefs privées. Il faut récupérer le fichier **/data/system/password.key** et le sel présent dans la base de données **settings.db** sous la clef **lockscreen.password_salt**. Contre cela, il n'y a rien à faire :-(

3.2 Installation de certificats

Pour exploiter le **KeyChain**, il faut installer les certificats numériques dans le conteneur sécurisé. Nous avons vu qu'il est possible de faire cela en plaçant un fichier PKCS12 dans la racine de la sdcard (ou sa simulation), puis en déclenchant une activité. Cela correspond à l'appui sur la commande « *Installer depuis la carte SD* » du paramétrage d'Android.

```
Intent intent = new Intent("android.credentials.INSTALL");
intent.putExtra(EXTRA_NAME, CERT_NAME); // Controle le nom du
certificat
startActivityForResult(intent, RESULT_CODE);
```

Il faut avouer que cela n'est pas très propre. En effet, rien n'indique qu'il n'y a pas d'autres certificats déjà présents dans le répertoire ; le fichier est effacé dans la carte SD, mais uniquement logiquement. Il est possible de le récupérer pour essayer une attaque en force brute.

Il est possible de faire autrement. **KeyChain** offre différents **EXTRA** pour injecter le contenu d'un certificat à installer.

```
Intent intent =KeyChain.createInstallIntent();
intent.putExtra(KeyChain.EXTRA_NAME, CERT_NAME); // Controle le nom
du certificat
intent.putExtra(KeyChain.EXTRA_PKCS12, out.toByteArray());
startActivityForResult(intent, RESULT_CODE);
```

Cela fonctionne uniquement lorsque la classe **KeyChain** est disponible (API Level 14). Cette approche impose à l'utilisateur de saisir le mot de passe du certificat puis de confirmer son alias.

Pour installer directement une autorité de confiance, il faut en obtenir une version en mémoire, puis la donner dans le paramètre **EXTRA_CERTIFICATE**.

```
Intent intent =KeyChain.createInstallIntent();
intent.putExtra(KeyChain.EXTRA_NAME, CERT_NAME);
intent.putExtra(KeyChain.EXTRA_CERTIFICATE,ca.getEncoded());
startActivityForResult(intent, RESULT_CODE);
```

Après validation de l'alias par l'utilisateur, l'autorité est installée.

Notez que les certificats clients ne sont pas présents dans l'interface de paramétrage des certificats. Seules les autorités sont visibles.

Nous pouvons donc installer un certificat client, puis demander le droit de l'exploiter à l'utilisateur et le présenter dans une connexion TLS avec authentification mutuelle.

4 android-keychain-backport

Nous avons maintenant tous les éléments pour proposer une approche compatible avec les différentes versions d'Android (enfin, supérieures à la version 7). Pourquoi ne pas proposer une librairie de compatibilité qui permet de marier le meilleur des deux mondes ?

Figure 6

Figure 7

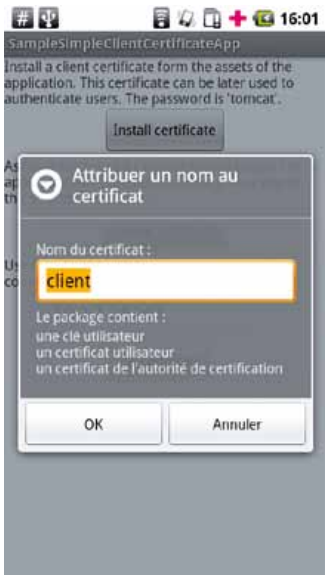


Figure 8

Figure 9

L'idée est de simuler la classe **KeyChain** pour les versions inférieures à 14 d'Android.

Nous utiliserons différents services pour installer un certificat dans le **KeyStore** ou le **KeyChain** suivant les cas. Nous sommes contraints de gérer également le déblocage du **KeyStore** pour les anciennes versions. Nous proposons alors d'ajouter une méthode **isUnlock()** et **unlock(Activity)** à notre classe **KeyChain**.

Comme nous avons besoin d'un **Context**, nous ne pouvons pas proposer strictement la même API que **KeyChain**. Nous ne pouvons pas utiliser de méthodes statiques comme le propose l'interface officielle.

Notre nouvelle classe de compatibilité a alors la forme suivante :

```
public class KeyChain
{
  public KeyChain(Context context) { ... }
  public boolean isUnLocked() { ... }
  public void unlock(Activity context) { ... }
  public Intent createInstallIntent() { ... }
  public void choosePrivateKeyAlias( ... ) { ... }
  public X509Certificate[] getCertificateChain( ... ) { ... }
  public PrivateKey getPrivateKey(Context context,String alias)( ... ) { ... }
}
```

Elle est très similaire à la classe **KeyChain** officielle.

Vous trouverez le code de la librairie Android ici : https://github.com/pprados/android-keychain-backport

Il est également nécessaire de déclarer quelques **Activity** dans le fichier **AndroidManifest.xml**.

```
<!-- You MUST add this three activities in your application. -->
<!-- Activity to install a certificat after enter the password. -->
<activity
  android:name="android.support.v7.security.impl.CertInstaller"
  android:configChanges="orientation|keyboardHidden"
  android:theme="@style/KeyChain_Transparent" />
<!-- Activity to select a certificate to use -->
<activity
  android:name="android.support.v7.security.impl.CertChooser"
  android:configChanges="orientation|keyboardHidden"
  android:theme="@style/KeyChain_Transparent" />
<!-- Activity to unlock the local container. Do nothing. -->
<activity android:name="android.support.v7.security.impl.
UnlockActivity" />
```

En effet, il n'est pas encore possible d'hériter de paramètres décrits dans le **AndroidManifest.xml** d'une librairie Android.

Cette classe simule la classe officielle si elle n'est pas disponible. Sinon, elle délègue simplement les méthodes.

Avant d'utiliser cela, il faut vérifier que le conteneur n'est pas bloqué par un appel à **isUnlocked()**. Si c'est le cas, il faut demander le déblocage du conteneur via **unlock()**. Cela déclenche une nouvelle activité. Au retour, dans **onResume()**, il est possible de reprendre le traitement.

Voici des copies d'écrans d'un exemple d'utilisation avec les différentes boites de dialogues (Figure 6 à 9).

Et voici un extrait du code. Je n'ai laissé que le strict minimum. Le reste est avec les sources.

```
public class MainActivity extends Activity
{
  ...
  @Override
  protected void onCreate(Bundle savedInstanceState)
```

Complétez votre collection d'anciens numéros !

ed-diamond.com

numerique.ed-diamond.com

```
{
  super.onCreate(savedInstanceState);
  if (savedInstanceState!=null)
    mState=savedInstanceState.getInt(EXTRA_STATE);
}

@Override
protected void onResume()
{
  super.onResume();
  // Switch the current state, after unlock the key store, continue the job
  switch (mState)
  {
    case STATE_INSTALL:
      installCertificate(null);
      break;
    case STATE_USE:
      useCertificate(null);
      break;
  }
}
@Override
protected void onSaveInstanceState(Bundle outState)
{
  super.onSaveInstanceState(outState);
  outState.putInt(EXTRA_STATE, mState);
}
public void installCertificate(View view)
{
  if (!mKeyChain.isUnLocked())
  {
    // Key store is locked. Start an activity to unlock it.
    mState=STATE_INSTALL;
    mKeyChain.unlock(this);
  }
  else
  {
    mState=0;
    ...
  }
}
public void chooseCertificate(View view)
{
  mKeyChain.choosePrivateKeyAlias(this,
new KeyChainCallBack()
{
      @Override
      public void alias(final String alias)
      {
        if (alias!=null)
        {
          mAlias=alias;
          // Save last alias
          mPreference.edit().putString(ALIAS_KEY, alias).commit();
        }
      },
  null,null,null,-1,null);
}
public void useCertificate(View view)
{
  if (!mKeyChain.isUnLocked())
  {
    // Key store is locked. Start an activity to unlock it.
    mState=STATE_USE;
    mKeyChain.unlock(this);
    return;
  }
  mState=0;
  ...
 }
}
```

Le code est le même quelque soit les versions d'Android.

Côté serveur, vous pouvez utiliser une instance Tomcat et la paramétrer pour accepter les connexions TLS avec authentification mutuelle. Avec les sources, dans le répertoire **`tomcat_conf`**, vous trouverez les paramètres pour accepter le certificat client présent dans l'application de démonstration.

Il y a quand même quelques petites modifications à l'usage. Avant ICS, les certificats sont installés localement à l'application. Chaque application doit alors installer le certificat dont elle a besoin. Ce dernier ne peut pas être utilisé pour une connexion Web via le navigateur classique.

Pour les versions ICS et suivantes, le certificat est installé globalement pour toutes les applications et pour le navigateur Web standard. Les autres applications peuvent demander à utiliser directement le certificat installé.

5 Utilisation via un AccountAuthenticator

Android propose un cadre de travail pour ajouter un nouveau type de compte, généralement compatible **`OAuth2`**. Ce mécanisme permet à l'utilisateur d'ajouter un compte depuis la console de paramétrage du téléphone. Il est possible d'utiliser ce framework pour gérer les certificats clients (Figure 10).

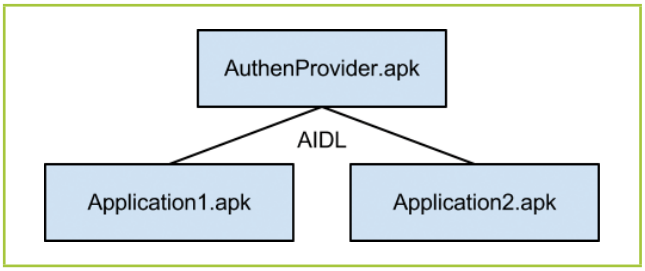


Figure 10

La documentation proposée par Google pour utiliser cela n'est pas très claire ni très juste. Il est possible de s'inspirer de l'exemple **`SampleSyncAdapter`** mais avec précaution.

Pour offrir un nouveau type de compte, il faut proposer un service qui répond à l'intention **`android.accounts.AccountAuthenticator`**. Le service doit indiquer des métadonnées sous forme de fichier XML.

```
<service
   android:exported="true"
   android:name=".CertificateAuthenticationService"
>
  <intent-filter>
    <action android:name="android.accounts.AccountAuthenticator"/>
  </intent-filter>
  <meta-data
    android:name="android.accounts.AccountAuthenticator"
    android:resource="@xml/certificate_authenticator"/>
 </service>
```

Le fichier XML décrit le type de compte, ainsi que les icônes à utiliser pour ce dernier.

```
<account-authenticator xmlns:android="http://schemas.android.com/apk/
res/android"
    android:accountType="fr.prados.android.account.certificate"
    android:icon="@drawable/ic_launcher"
    android:smallIcon="@drawable/ic_launcher"
    android:label="@string/authenticator_certificate_label"
/>
```

Le service est simple à proposer. Il doit juste renvoyer une instance dérivée de **Authenticator** dans la méthode **onBind()**.

```
public class CertificateAuthenticationService extends Service
{
  private Authenticator mAuthenticator;

  @Override
  public void onCreate()
  {
    mAuthenticator = new Authenticator(this,new
CertificateProvider(this));
  }

  @Override
  public IBinder onBind(Intent intent)
  {
    return mAuthenticator.getIBinder();
  }
}
```

L'instance **Authenticator** hérite de **AbstractAccount Authenticator** et répond aux différentes méthodes. Le principe est le suivant. Chacune peut soit retourner des valeurs, soit demander une intervention de l'utilisateur pour pouvoir les retourner. Par exemple, l'utilisateur peut être sollicité pour saisir son nom et son mot de passe afin de pouvoir retourner un *token* de connexion. Les méthodes retournent un **Bundle** avec les valeurs de retours sous différentes clefs, ou bien un **Intent** permettant de démarrer une activité via la clef **KEY_INTENT**. Si une méthode peut retourner une information, elle le fait immédiatement. Sinon, si elle a besoin de demander des précisions à l'utilisateur (mot de passe, validation des privilèges), la méthode construit un **Intent** avec toutes les informations permettant de retourner le résultat lorsque l'utilisateur aura validé son formulaire.

Le framework standard d'Android se charge d'invoquer l'instance **Authenticator**, puis de déclencher ou non l'activité correspondante si un **Intent** est retourné.

L'intent de l'activité doit posséder la clef **KEY_ACCOUNT_AUTHENTICATOR_RESPONSE** avec l'instance **response** présente dans les paramètres des méthodes. Cela permettra à l'activité d'informer du résultat du formulaire.

Pour simplifier l'implémentation, les activités doivent hériter de **AccountAuthenticatorActivity**. Cela permet de bénéficier de la méthode **setAccountAuthenticatorResult()** pour déposer le résultat de l'activité.

Avec ce mécanisme, il est possible de l'exploiter pour installer un certificat client. La difficulté consiste à réussir à transmettre la clef privée et la chaîne des certificats aux applications.

Il n'est pas difficile d'ajouter une confirmation de l'utilisateur lors de l'implémentation de **getAuthToken()**. Comme le fait Google, un écran affiche les applications souhaitant avoir accès au token. Cela est alors mémorisé dans une base de données SQLite pour ne plus demander cela à nouveau. Il faut également réagir à un broadcast **PACKAGE_REMOVED** pour nettoyer la base de données lors de la suppression d'une application.

Côté client, pour utiliser cette API, il faut utiliser la méthode **getAuthToken()** de **AccountManager**. Une instance dérivée de **AccountManagerCallback<Bundle>** permet alors de récupérer les informations. Par ce canal, il est possible de récupérer la clef privée et la chaîne de certificats. Il suffit d'utiliser des clefs spécifiques.

Attention, comme Android garde en cache que le password, il faut que l'application invalide le dernier **authToken** avant de demander à nouveau le token avec les certificats. En effet, comme nous utilisons des clefs complémentaires, les données ne sont pas mémorisées dans le cache.

```
mAccountManager.invalidateAuthToken(mAccountType, mAuthToken);
mAccountManager.getAuthToken(
  mAccount,      // Account retrieved using getAccountsByType()
  mAuthTokenType, // Auth scope
  mOptions,      // Authenticator-specific options
  this,          // Your activity
  mAccountManagerCallback,  // Callback
  mHandler);     // Callback called if an error occurs
```

Pour interdire aux applications ne venant pas de l'entreprise, d'exploiter ce mécanisme pour voler la clef privée, nous devons ajouter une nouvelle permission.

```
<permission
  android:name="fr.prados.USE_CREDENTIEL"
  android:label="@string/permission_use_certificate_label"
  android:description="@string/permission_use_certificate_desc"
  android:protectionLevel="signature"
  android:permissionGroup="android.permission-group.ACCOUNTS"
/>
```

Celle-ci est indispensable aux applications souhaitant se binder sur notre service.

```
<uses-permission android:name="fr.prados.USE_CREDENTIEL"/>
```

Comme la nouvelle permission utilise un **protectionLevel** de type **signature**, seules les applications signées numériquement avec le même certificat numérique peuvent en bénéficier.

Mais, nous rencontrons une difficulté pour vérifier les privilèges de l'appelant. En effet, notre application n'invoque pas directement notre **AccountAuthenticator**. L'application invoque le framework Android qui lui invoque notre **AccountAuthenticator**. Au passage, le numéro du processus client est perdu.

Pour régler cela, Android propose, depuis la version 11, deux clefs permettant de récupérer ces informations lors de l'invocation d'une méthode de l'**AccountAuthenticator**. Mais, comme cela est important pour la sécurité, en regardant les sources de la version Gingerbread, on découvre qu'un backport est disponible. Impossible de connaître la version exacte à partir de laquelle cela est disponible.

Bon, pour régler cela, proposons des alias suivant les versions.

```
public static final String KEY_CALLER_UID =
  (VERSION.SDK_INT>=VERSION_CODES.HONEYCOMB)
    ? AccountManager.KEY_CALLER_UID : "callerUid";
public static final String KEY_CALLER_PID =
  (VERSION.SDK_INT>=VERSION_CODES.HONEYCOMB)
  ? AccountManager.KEY_CALLER_PID : "callerPid";
```

Puis, dans la méthode **getAuth()**, nous pouvons vérifier les privilèges de l'appelant si cela est disponible.

```
if (USE_PERMISSION)
{
  if (options==null)
    return errorDenied();
  int pid=options.getInt(KEY_CALLER_PID);
  int uid=options.getInt(KEY_CALLER_UID);
  if (mContext.checkPermission(PERMISSION, pid, uid)
    ==PackageManager.PERMISSION_DENIED)
    return errorDenied();
}
```

L'implémentation de l'**AccountAuthenticator** utilise bien entendu, le conteneur sécurisé proposé par Android. En effet, il est possible de sauver un mot de passe dans le framework, via la méthode **setPassword()** d'un **AccountManager**, mais ce dernier est alors mémorisé dans une base de données SQLite, accessible uniquement par **root**. La base n'est pas chiffrée. Un téléphone rooté peut alors révéler les secrets.

Cette approche permet de partager l'authentification de l'utilisateur par toutes les applications de l'entreprise. Nous avons proposé cela à un grand opérateur de logistique, pour authentifier tous les livreurs.

6 Utilisation TLS

Maintenant que nous savons comment installer un certificat numérique depuis une application, comment le protéger et comment le récupérer, nous pouvons exploiter cela pour initialiser une connexion TLS. Cela s'effectue via la création de deux classes spécifiques.

```
final KeyManager[] keyManagers=new KeyManager[]
{
  new X509KeyManager()
  {
    ...
    @Override
    public String[] getClientAliases(String keyType, Principal[] issuers)
    {
      return new String[]{mAlias};
    }
    @Override
    public String chooseClientAlias(String[] keyType, Principal[] issuers,
Socket socket)
    {
      return mAlias;
    }
    @Override
    public X509Certificate[] getCertificateChain(String alias)
    {
      return mKeyChain.getCertificateChain(MainActivity.this, mAlias);
    }
    @Override
    public PrivateKey getPrivateKey(String alias)
    {
      return mKeyChain.getPrivateKey(MainActivity.this, mAlias);
    }
  }
};
final X509TrustManager[] X509TrustManager=new X509TrustManager[]
{
  @Override
  public void checkServerTrusted(X509Certificate[] chain, String authType)
    throws CertificateException
  {
    byte[] currentDigest =
      MessageDigest.getInstance("MD5").digest(chain[0].getEncoded());
    if (!Arrays.equals(DIGEST, currentDigest))
      throw new CertificateException("Invalid server certificate");
  }
  ...
};
```

Suivi d'une ouverture de connexion TLS.

```
if (mSocketFactory==null)
{
  final SSLContext sslcontext =
    SSLContext.getInstance(TLS_IMPLEMENTATION_ALGORITHM)
  SecureRandom random=SecureRandom.getInstance(SECURE_RANDOM_
ALGORITHM);
  sslcontext.init(
    keyManagers,
    X509TrustManager,
    random);
```

```
    mSocketFactory=sslcontext.getSocketFactory();
}

// Open socket
SSLSocket socket=(SSLSocket)mSocketFactory.createSocket();
socket.connect(new InetSocketAddress(mHost,mPort),SOCKET_TIMEOUT);
// Write HTTP request
PrintWriter out=new PrintWriter(socket.getOutputStream());
BufferedReader in=new BufferedReader(new InputStreamReader(socket.
getInputStream()));
out.write("GET / HTTP/1.0\n\n");
out.flush();
// Read HTTP response
System.out.println(in.readLine());
socket.close();
```

Notez qu'il est très important de recycler autant que possible le **SocketFactory**. En effet, lors de l'ouverture d'une connexion TLS, une clef symétrique est négociée entre les parties. Cette clef est gardée en cache dans le contexte SSL. Ainsi, lors d'une prochaine connexion, la clef symétrique est proposée au serveur. S'il l'accepte, cela améliore notablement les performances. De plus, initialiser un **SecureRandom** prend un temps certain, afin de garantir l'entropie. Comme l'instance peut être partagée par plusieurs tâches, il est préférable d'utiliser un singleton pour cela.

7 Utilisation HTTPS

Pour utiliser cela en HTTPS, rien de plus simple. Il suffit d'indiquer le **SSLSocketFactory** à utiliser pour les connexions HTTPS et de valider éventuellement tous les hosts.

```
HttpsURLConnection.setDefaultSSLSocketFactory(mSocketFactory);
HttpsURLConnection.setDefaultHostnameVerifier(new HostnameVerifier()
{
  @Override
  public boolean verify(String hostname, SSLSession session)
  {
    return true;
  }
});
URLConnection con=new URL("https://"+mHost+":"+mPort+"/").openConnection();
BufferedReader in=new BufferedReader(new InputStreamReader(con.getInputStream()));
in.readLine();
```

Pour des raisons de sécurité, ce n'est pas bien d'accepter tous les hosts sans vérifier leurs signatures.

8 Comment ne pas exposer la clef privée ?

Toutes ces approches présentent l'inconvénient de donner accès à la clef privée aux applications qui en ont besoin. L'approche avec plusieurs APK permet de limiter cette exposition, mais toutes les applications de l'entreprise ont bien accès à la clef privée. Est-ce vraiment nécessaire ? Il serait sympathique que la clef privée ne soit pas visible par les applications qui ont simplement besoin d'ouvrir un socket authentifié.

8.1 ClientAuth="want"

Côté serveur, il est possible de demander un certificat client, mais de tolérer qu'il ne soit pas fourni par l'utilisateur. Avec le paramètre **clientAuth** à **want** de Tomcat, il est possible d'imaginer un scénario n'exposant pas la clef privée aux applications. L'astuce est la suivante.

Une application qui est la seule à avoir accès à la clef privée ouvre une connexion vers le serveur Web avec une authentification mutuelle. Le serveur constate que le client ne possède pas de cookie. Il exige de vérifier l'identité de l'utilisateur via un certificat client. Si le certificat présenté est valide, le serveur génère un cookie de session et le retourne au client.

L'application est alors en capacité de donner la valeur du cookie aux applications désirant ouvrir une connexion vers le serveur. Si le cookie est signé et possède l'identité de l'utilisateur, cela fonctionne très bien.

Les autres applications récupèrent le cookie et ouvrent une connexion TLS mais sans présenter de certificat client. Le serveur, constatant que le cookie est valide, ignore le fait que le certificat client est absent. Il extrait l'identité de l'utilisateur du cookie.

Ainsi, la clef privée est protégée des vulnérabilités présentes dans les applications de l'entreprise. Cela exige, côté serveur, d'accepter des communications sans certificat client. Cela peut avoir un impact sur la traçabilité des actions des utilisateurs.

Cette approche est très similaire à OAuth. Les applications peuvent exploiter les communications, sans connaître les secrets.

8.2 Approche Ninja

Encore plus fort : une approche permettant d'avoir toujours un certificat client, sans propager le secret de la clef privée.

Les sockets sont ouverts via des handles de fichiers Linux. Android permet d'envoyer un handle de fichier à une autre application. Un **dup()** est alors effectué par le noyau. Il est donc possible d'ouvrir un socket dans une application et l'utiliser dans une autre ! La difficulté réside dans la création d'une instance socket java propre, simplement à partir du handle du socket.

Il faut analyser comment la classe est construite pour reconstruire une instance utilisable, depuis un autre processus.

Nous sommes toujours dans une architecture avec plusieurs APK. L'un possède la clef privée et est en charge d'ouvrir une connexion avec authentification mutuelle. Les autres APK souhaitent continuer la communication, sans avoir accès à la clef privée. Elle ne souhaite pas ouvrir une nouvelle communication, mais continuer avec le même socket.

Côté application qui ouvre le socket, il faut extraire le numéro du handle du socket. Nous avons la méthode cachée **getFileDescriptor$()** pour cela. Elle sert normalement à faire la liaison avec les IO asynchrones. La méthode **getInt$()** du **FileDescriptor** permet alors de récupérer la valeur du handle.

```
Socket socket=new Socket("10.0.2.2",8080);
FileDescriptor fd=(FileDescriptor)socket.getClass()
  .getMethod("getFileDescriptor$").invoke(socket);
int fdi=(Integer)fd.getClass().getMethod("getInt$").invoke(fd);
```

Il faut alors construire un **ParcelFileDescriptor** à partir de ce handle.

```
ParcelFileDescriptor fd=ParcelFileDescriptor.adoptFd(fdi);
```

Ce dernier peut être retourné au processus appelant. La couche AIDL d'Android se charge d'effectuer un **dup()** sur le handle de fichier, lors de la transmission des données entre les processus (voilà à quoi sert de lire le code du framework d'Android ;-).

Pour cela, nous déclarons une interface AIDL.

```
package com.example.sharesocket;
import android.os.ParcelFileDescriptor;

interface ShareSocket
{
  ParcelFileDescriptor openSocket();
}
```

Cette dernière a besoin d'indiquer que **ParcelFileDescriptor** doit être envoyé par valeur. Cela s'effectue en ajoutant un autre fichier AIDL dans le package **android.os** de votre projet.

```
package android.os;
parcelable ParcelFileDescriptor;
```

Et un petit service pour permettre le binding sur l'implémentation de l'AIDL :

```
public class ShareSocketService extends Service
{
  @Override
  public IBinder onBind(Intent intent)
  {
    return new ShareSocketImpl();
  }
}
```

avec une implémentation de l'AIDL comme décrite ci-dessus.

Côté consommateur du socket ouvert par l'autre processus, il faut construire une instance **PlainSocketmpl** avec un **FileDescriptor**, puis le donner au constructeur du socket. Il reste à placer le drapeau **isConnected** à **true** et le socket est utilisable. Le code théorique est celui-ci :

```
int fdi=shareSocket.openSocket().getFd();
FileDescriptor fd=new FileDescriptor();
fd.descriptor=fdi;
Socket socket=new Socket(new java.net.PlainSocketImpl(fd));
socket.isConnected=true;
```

Comme les méthodes ne sont pas toutes accessibles, nous utilisons l'introspection.

```
ParcelFileDescriptor pfd=shareSocket.openSocket();
int fdi=pfd.getFd();
FileDescriptor fd=new FileDescriptor();
Field field;
field=fd.getClass().getDeclaredField("descriptor");
field.setAccessible(true);
field.set(fd, fdi);
Class<?> clPlainSocketImpl=Class.forName("java.net.PlainSocketImpl");
Object psimpl=clPlainSocketImpl.getConstructor(FileDescriptor.class).newInstance(fd);
constructor=Socket.class.getDeclaredConstructor(SocketImpl.class);
constructor.setAccessible(true);
Socket socket=(Socket)constructor.newInstance(psimpl);
field=Socket.class.getDeclaredField("isConnected");
field.setAccessible(true);
field.setBoolean(socket, true);
```

Nous pouvons alors utiliser le socket comme s'il avait été ouvert par notre processus ! (Ninja j'ai dis)

```
PrintWriter out=new PrintWriter(socket.getOutputStream());
BufferedReader in=new BufferedReader(new InputStreamReader(socket.getInputStream()));
out.println("GET / HTTP/1.0\n\n");
out.flush();
String line=in.readLine();
System.out.println(line);
```

Pour les connexions TLS, c'est plus complexe, car il faut de plus, récupérer les clefs symétriques négociées, pour pouvoir continuer à exploiter le flux et reconstruire toute la grappe d'objets pour reprendre une connexion sécurisée.

De plus, rien ne garantit la persistance des méthodes suivant les différentes versions de l'OS. Nous laissons le soin au lecteur de continuer ces travaux.

9 Enrôlement

Nous savons maintenant comment utiliser un certificat numérique, comment le protéger et comment l'installer. Il nous manque à concevoir un mécanisme permettant de créer et télécharger en toute sécurité le certificat signé sur le téléphone. L'approche par carte SD n'est vraiment pas conviviale.

Il faut faire très attention à nos scénarios, car un pirate peut les exploiter pour obtenir un certificat indu. Suivant les exigences, nous devons imaginer des scénarios différents. Par exemple, le même certificat peut être installé sur différents terminaux. Au contraire, un certificat par terminal peut être généré localement, puis signé par le certificat de l'utilisateur. La chaîne de certification permet alors d'identifier

le terminal d'un utilisateur. Il est alors plus facile de révoquer le certificat du terminal, sans révoquer le certificat de l'utilisateur. Cela évite également de sauver sur le terminal, le certificat de l'utilisateur. Il est récupéré en mémoire, juste le temps d'effectuer la signature du certificat du terminal. Une approche plus simple consiste à créer un certificat dans le terminal et à demander à une autorité de certification de le signer.

Verisign a développé pour CISCO, le projet Simple Certificate Enrollment Protocol **[14]** (SCEP). C'est un projet qui est maintenant abandonné, mais il est possible de trouver des implémentations en différents langages. La couche transport est HTTP. Des messages sécurisés sont échangés pour permettre la récupération de certificats client ou de les renouveler. Un serveur SCEP peut être une autorité de certification (CA) ou une autorité d'enrôlement, qui délègue la signature à la CA. Les certificats demandés peuvent être signés immédiatement ou être mis en attente pour une validation humaine.

Pour initier une première demande de certificat, le client doit connaître un mot de passe spécifique, éventuellement à usage unique. Ensuite, il peut exploiter son certificat courant pour le renouveler.

SCEP est proposé par Microsoft depuis 2003. Apple l'utilise **[15]** dans iOS 4 avec un mot de passe dérivé des paramètres du terminal. Comme le mot de passe est alors statique, les serveurs SCEP doivent être paramétrés pour autoriser le recyclage du mot de passe initial. Cela fragilise **[16]** l'architecture globale.

Cette approche présente une faiblesse, car la connaissance de ce mot de passe permet d'obtenir un certificat valide. Nous allons proposer une approche pour combler cette lacune.

Nous allons nous placer dans un cas simple, où nous voulons installer un certificat pour un utilisateur identifié, sur un téléphone identifié avec une puce GSM identifiée. Nous ne voulons pas demander à l'utilisateur de saisir un mot de passe. C'est particulièrement pénible sur un mobile. Seule la saisie de son identifiant sera nécessaire pour obtenir un certificat valide.

Pour valider l'installation d'un certificat sur un terminal, il est nécessaire d'utiliser un deuxième canal de communication. Ce dernier permet de faire transiter une information complémentaire et nécessaire à l'installation. Ce dernier peut être un e-mail, un SMS, un scan de QR Code, etc. Il permet d'identifier l'utilisateur et/ou le terminal. Ainsi, un pirate doit être capable d'attaquer les deux canaux afin d'obtenir un certificat valide.

Le canal complémentaire sélectionné est la connexion 3G. L'utilisateur capable de recevoir un SMS particulier est autorisé à installer un certificat.

Pour associer le certificat au terminal, le code IMEI, unique à chaque téléphone est également nécessaire à la validation du certificat. En ayant connaissance de l'IMEI d'un terminal, il est toutefois possible d'effectuer la procédure d'enrôlement sur un autre terminal. Il faut néanmoins posséder la puce téléphonique correspondante (Figure 11).

Figure 11

Avant l'enrôlement par l'utilisateur, l'administrateur :

- Enregistre ou importe l'association IMEI du terminal avec le numéro de téléphone mobile correspondant dans le serveur SCEP. Cela permet d'identifier le parc de matériel de l'entreprise, habilité à demander un certificat numérique.
- Enregistre l'identité de l'utilisateur avec le couple IMEI, numéro de téléphone.

Puis, l'utilisateur :

- Installe l'application et la démarre ;
- L'application demande l'identifiant de l'utilisateur ;
- L'application interroge le serveur SCEP avec une requête hors protocole contenant le nom de l'utilisateur. Une requête Certificat Enrollement avec un challengePassword vide peut faire l'affaire ;
- Le serveur vérifie la validité de la demande (est-ce que l'utilisateur est connu et habilité à demander un certificat ?) ;
- Le serveur SCEP génère alors un challengePassword aléatoire et l'envoie par SMS à l'utilisateur (si possible, sur un port SMS différent de zéro) ;
- Le serveur SCEP ajoute l'IMEI du téléphone au mot de passe et l'enregistre dans la base de données du SCEP, avec les mots de passe valides ;
- Le client attend le SMS technique avec le mot de passe à usage unique ;
- Le client génère un certificat et formate une demande de signature au serveur SCEP ;

- Le client fourni alors un mot de passe composé du token récupéré par SMS, accolé à l'IMEI du périphérique comme identifiant ;
- Le serveur SCEP vérifie que la demande est valide (validation du mot de passe) ;
- Le serveur signe le certificat du client ;
- Le serveur enregistre le numéro du certificat dans sa base de données pour pouvoir le révoquer ;
- Le serveur retourne le certificat signé à l'application ;
- Une version non chiffrée du certificat est enregistrée dans le téléphone dans un conteneur, autant que possible sécurisé ; si le certificat est installé dans le conteneur officiel d'Android 16, l'utilisateur doit saisir un mot de passe complémentaire. En effet, il n'est pas possible à ce jour d'installer un certificat client non chiffré. Un ticket est ouvert pour cela.
- L'application peut maintenant exploiter le certificat.

Ce scénario présente quelques avantages. Le certificat n'est pas connu par le serveur, car la clef privée reste dans le téléphone ; l'enrôlement est automatique, après la saisie de l'identifiant de l'utilisateur ; le certificat est associé au terminal et non à l'utilisateur ; cela garantit que le certificat est installé via une puce GSM connu de l'entreprise, sur un téléphone identifié.

Néanmoins, l'utilisateur associé à l'IMEI du téléphone ainsi que le numéro de téléphone doit être enregistré avant l'enrôlement. Cela n'est pas toujours possible ; il peut exister des terminaux fantômes, préparés à l'enrôlement, mais jamais enrôlés ; le serveur doit avoir la capacité à envoyer rapidement un SMS ; le vol de la puce permet de récupérer le certificat sur un autre téléphone, en modifiant la valeur de l'IMEI à condition de la connaître ; l'installation d'un cheval de Troie sur le téléphone peut capturer le SMS et l'envoyer vers le pirate. Il peut alors récupérer le certificat. Le niveau de priorité de traitement du SMS peut limiter cette attaque ; cela ne fonctionne qu'avec des terminaux pouvant recevoir des SMS et impose la signature du certificat par le serveur, qui doit donc avoir accès à la clef privée de l'autorité.

Suivant les besoins, d'autres scénarios peuvent être imaginés, utilisant un QRCode présenté sur la console de l'administrateur du parc, une puce NFC, etc.

10 Renouvellement

Tant que le certificat courant est valide, il est facile d'ouvrir une communication authentifiée pour demander un nouveau certificat. Cela peut s'effectuer en tâche de fond, le dernier mois de validité du certificat. Ensuite, deux possibilités : soit on laisse l'ancien certificat finir sa vie tranquillement, pour ne pas interrompre les communications en cours, soit l'ancien certificat est révoqué dans 24 heures. Cette dernière approche est plus lourde, car tous les certificats seront un jour révoqués.

Les requêtes SCEP permettent cela, en exploitant le certificat valide précédant.

11 Certificat Pinning

Les certificats numériques, c'est bien sympathique. Mais finalement, la sécurité dépend des règles réellement appliquées par les différentes autorités. Certaines se sont fait abuser et ont signé des vrais/faux certificats. Avec 650 CA, ce n'est pas étonnant. Certains gouvernements ont utilisé une de leurs CA pour générer des certificats pour Gmail et ainsi, pouvoir récupérer tous les flux en clair.

Pour améliorer cela, Google Chrome 13+ propose une nouvelle approche. L'idée est de maintenir une liste d'autorités ou de certificats valides pour chaque site. L'extension PKPE (Public Key Pinning Extension for HTTP **[17]**) propose d'ajouter un en-tête `Public-Key-Pin` avec le hash des clefs publiques et une durée de vie. Cette information est gardée en cache par le navigateur lors de la première visite. Ainsi, plus tard, si un certificat valide est présenté, ne faisant pas partie de la liste, il est refusé. La première connexion est implicitement valide.

Android 4.2 gère maintenant cela via le `keychain`.

Au niveau applicatif, il est envisageable d'exploiter ces informations pour qualifier les connexions web. La librairie AndroidPinning **[18]** propose une implémentation.

Conclusion

Pour résumer, voici les différentes technologies disponibles et les différentes approches pour les utiliser.

Technologies de sauvegarde des secrets :

- **Sauvegarde dans le contexte de l'application** : Les données ne sont pas chiffrées. Elles sont vulnérables au vol du téléphone.
- **SE dans la carte à puce** : Inaccessible aux applications.
- **SE dans le terminal** : Inaccessible aux applications.
- **Chiffrement du disque** : Non obligatoire. Ne protège pas des vulnérabilités des applications ou du téléphone allumé.
- **KeyStore** : Conteneur sécurisé, mais non officiel. Il peut être modifié dans les prochaines versions d'Android. À ce jour, le meilleur endroit où sauver les secrets.
- **KeyChain** : Gestion officielle de gestion des certificats clients. Non disponible avant l'API 14. Nous proposons une librairie de compatibilité pour les versions comprises entre 7 et 14.

Architecture d'utilisation :

- **Certificat en clair en mémoire** : Exige le mot de passe à chaque rappel de l'application.

- **Certificat client partagé, installé dans le périphérique et disponible pour toutes les applications qui en font la demande** : Possible à partir de l'API 14, après accord de l'utilisateur. Demande le mot de passe du certificat uniquement lors de l'installation. Tous les flux utilisent une authentification mutuelle. Risque d'exploitation de la clef privée sélectionnée par erreur par l'utilisateur, par une application vulnérable ou malveillante.
- **Certificat client partagé par des applications de confiance. Application qui expose la clef privée via une interface AIDL** : Ne demande pas forcément le mot de passe du certificat qui peut être importé dans l'application par tous moyens. Nécessite une application complémentaire pour protéger le certificat. Les autres applications consomment le certificat si elles ont le privilège. Tous les flux utilisent une authentification mutuelle. Risque d'exploitation d'une vulnérabilité des applications habilitées à utiliser le certificat.
- **Certificat client non partagé. Le certificat est optionnel côté serveur (authClient="want"). Ouverture d'un socket avec authentification mutuelle, puis de sockets avec cookies par les autres applications. Approche similaire à OAuth** : Ne demande pas forcément le mot de passe du certificat qui peut être importé dans l'application par tous moyens. Limitation du risque à la seule application qui possède l'accès à la clef privée. Exploitation par des applications de confiance si elles ont le privilège. Une partie des flux utilise une authentification mutuelle.
- **Certificat client non partagé. Une application ouvre un socket avec authentification mutuelle, puis le duplique vers les applications consommatrices** : Limitation du risque à la seule application qui possède l'accès à la clef privée. Exploitation par des applications de confiance si elles ont le privilège. Tous les flux utilisent une authentification mutuelle. Très difficile à implémenter sur tous les modèles de téléphones.

Nous comprenons maintenant pourquoi il est dangereux d'avoir un téléphone rooté, débloqué ou avec le mode debug activé. Des logiciels sont en effet capables de dumper toute la mémoire **[19]**. C'est pour cela que la version 4.2.2 d'Android impose maintenant une association entre le PC de déverminage et le terminal. Il n'est plus possible de brancher le téléphone sur un port USB pour y avoir accès.

Il y a des vulnérabilités avec les versions utilisant les processeurs Exynos 4 équipant les Samsungs **[20]** permettant d'avoir un accès complet à la mémoire, corrigé depuis par les fournisseurs. Une autre vulnérabilité pour ICS et JB a été découverte **[21]**. Il est également possible de découvrir le code de déverminage en détectant les mouvements subtils du téléphone lors de la saisie, par analyse via la caméra et/ou le gyroscope **[22]**.

Nous voici bien équipés pour proposer des applications Intranet à l'ensemble des commerciaux ou autres employés en mobilité. Pour les autres plate-formes comme iOS ou Windows Phone 8, il faudra étudier comment protéger le certificat client correctement.

Si cela vous semble trop complexe ou subtil, je me ferai un plaisir de venir vous aider ;-) ■

Références

[1] http://www.musclecard.com/

[2] http://code.google.com/p/seek-for-android/wiki/Devices

[3] http://fr.wikipedia.org/wiki/Commandes_Hayes

[4] http://code.google.com/p/seek-for-android/wiki/UICCSupport

[5] http://www.3gpp.org/ftp/Specs/html-info/27007.htm

[6] http://en.wikipedia.org/wiki/Single_Wire_Protocol

[7] http://www.mastercard.us/paypass.html#/home/

[8] http://en.wikipedia.org/wiki/Trusted_service_manager

[9] http://www.theverge.com/2012/8/28/3273784/google-to-open-wallet-app-to-third-party-passes-loyalty-cards-and-ids

[10] http://forensics.spreitzenbarth.de/2013/02/14/cracking-androids-full-disk-encryption/

[11] https://play.google.com/store/apps/details?id=org.nick.cryptfs.passwdmanager

[12] https://github.com/kghost/ics-openvpn

[13] http://forensics.spreitzenbarth.de/2012/02/28/cracking-pin-and-password-locks-on-android/

[14] http://tools.ietf.org/html/draft-nourse-scep-23

[15] http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/iPhoneOTAConfiguration/OTASecurity/OTASecurity.html

[16] http://www.css-security.com/wp-content/uploads/2012/05/SCEP-and-Untrusted-Devices.pdf

[17] http://tools.ietf.org/html/draft-ietf-websec-key-pinning-04

[18] https://github.com/moxie0/AndroidPinning/

[19] https://www.google.fr/webhp?q=live%20forensics%20android

[20] http://forum.xda-developers.com/showthread.php?t=2057818

[21] http://forum.xda-developers.com/showpost.php?p=31545627

[22] https://en.wikipedia.org/wiki/Smudge_attack

ERGONOMIE DES SYSTÈMES MOBILES

par Jérémie CHAINE & Philippe PRADOS
[Ergonome et Consultant OCTO Technology]

Le monde du mobile est en plein essor et offre des perspectives enthousiasmantes. Néanmoins, faire sa place dans cet univers hyper concurrentiel n'est pas chose aisée. Pour y arriver, il est indispensable de réfléchir soigneusement à l'expérience utilisateur que l'on va proposer lors de la conception de son application. Concevoir une application « ergonomique », c'est-à-dire une application adaptée à des utilisateurs donnés, pour des besoins donnés, dans un contexte donné, devient un facteur clé de réussite.

Mais le succès passe également par le fait d'être présent sur les différents OS afin de toucher une cible large. Et c'est là que les choses se compliquent. Réaliser un bon produit sur un OS est déjà un défi en soi, mais le porter sur les différents OS devient un véritable challenge. En effet, on pourrait comparer chaque OS à un pays, chacun parlant sa propre langue et ayant sa manière propre d'exprimer les concepts et les idées. Pour apporter une solution adaptée, il est nécessaire de connaître ses différents idiomes pour parler la « langue » connue par les utilisateurs.

L'objet de cet article est de proposer une analyse comparative et factuelle des différentes logiques ergonomiques apportées par les trois principaux acteurs du marché, à savoir : Apple, Google et Microsoft.

Chaque OS s'appuie sur des conventions et des solutions ergonomiques différentes qu'il faut connaître avant de proposer une application. En effet, il est important de respecter chaque utilisateur dans le choix de son OS et donc, de prendre en compte la philosophie proposée par chaque système d'exploitation même si l'on choisit de faire une application « hybride ». L'objectif est bien sûr de proposer une **expérience cohérente** et agréable pour tous.

Figure 1

Ces différences d'approches peuvent s'apparenter à des dialectes qui permettent d'exprimer des idées similaires, mais différemment.

Le fait de porter une application d'un OS sur un autre, en imposant les choix ergonomiques de l'un d'entre eux, est un facteur de dégradation de l'expérience utilisateur. Les retours négatifs seront très vite au rendez-vous sur les stores. Un téléphone Android n'est pas un « iPhone comme les autres » ; un Windows Phone n'est pas une déclinaison de plus d'un Android.

Les différents OS utilisent des composants similaires (les mots), mais organisés différemment (la grammaire). La simple traduction mot-à-mot ne permet pas de respecter une langue.

La variabilité des périphériques ne facilite pas la conception des applications. D'ailleurs, penser une application uniquement pour un device donné n'est plus une solution envisageable. Même chez Apple il y a de nombreuses résolutions et tailles d'écrans à prendre en compte (iPad, iPad mini, iPhone, iPhone 5, résolution normale et retina).

Des règles simples permettent d'aider le concepteur. Par exemple, un minimum de 7mm doit être utilisé pour les zones à toucher **[!]**, avec un espacement de 2 mm entre ces zones. Il s'agit de millimètres et non de pixels puisque la taille des pixels change en fonction de la résolution des écrans. Ce changement induit un changement de taille en pixels des composants, ce qui peut avoir un impact sur le nombre ou l'organisation des éléments à afficher.

Une conception multi terminaux n'est plus faisable en créant un écran au pixel près et en considérant qu'il sera duplicable sur tous les autres appareils. Il y a un fossé entre une distribution des éléments graphiques pour un écran donné et sa déclinaison sur

les autres tailles, et ce en fonction des différentes orientations (portrait, paysage).

Pour cette analyse, nous nous sommes basés sur les recommandations ergonomiques exprimées par les différents éditeurs **[2][3][4]** et sur leurs mises en œuvre sur les applications publiées avec l'OS. Nous avons choisi de bien séparer les faits de nos opinions. Les faits sont des constatations que chacun peut vérifier. Les opinions correspondent à notre analyse et notre positionnement sur les différents items traités.

1 Les différences ergonomiques

Depuis la sortie de l'iPhone en 2007, de nombreux OS pour mobiles ont vu le jour. On peut citer Android, le dernier BlackBerry OS, Firefox OS, Meego, Nokia, Windows Phone, etc.

Chaque OS propose des versions successives faisant évoluer plus ou moins l'ergonomie générale. En s'inspirant les uns des autres, certaines solutions proposées par les différents OS tendent à se ressembler jusqu'à former une norme. Mais dans de nombreuses situations, les grammaires et la logique utilisées sont complètement différentes. Cela ne facilite pas le travail des concepteurs.

Apple, par exemple, s'appuie essentiellement sur le skeuomorphisme, c'est-à-dire une simulation du réel. Autant que possible, les applications copient les matières, les textures, le relief des objets du monde.

Il est fort probable que la version 7 d'iOS remette en cause complètement ce paradigme.

À l'opposé, Windows Phone propose une approche minimaliste, focalisée sur le contenu, en supprimant tout décor superflu comme le contour des boutons. L'ergonomie est plate, sans texture ou pseudo-relief.

Android propose une approche à mi-chemin.

La tendance du moment en design consiste à proposer une approche plate **[5]** et simple. Cela permet de faciliter l'adaptation des sites aux différents contextes d'accès, d'un navigateur sur grand écran à un smartphone, en passant par des tablettes de différentes tailles. Cette approche s'appelle le « Responsive design ».

Une langue maternelle est difficile à oublier, même après de nombreuses années à l'étranger. Il n'est pas rare qu'un bilingue continue à compter dans sa langue maternelle toute sa vie.

Un concepteur de talent doit être conscient qu'il garde souvent un « accent ». En étudiant plus profondément une autre langue, il sera attentif à ses mauvais réflexes.

L'hypothèse que nous défendons ici est qu'il faut concevoir une application différente par OS, même si globalement, les fonctionnalités restent identiques.

Bien entendu, les guides de conception proposés par les éditeurs peuvent ne pas être utilisés. Encore que cette affirmation est relative et variable en fonction des éditeurs. Seul Android permet une liberté totale. Les autres sont plus exigeants pour accepter une application sur leurs stores.

Une application peut être en rupture totale avec certains principes des OS. C'est une posture intéressante, mais qui présente un risque de rejet. En effet si le coût d'apprentissage est trop élevé ou va à contre-courant de la logique du reste des habitudes de l'utilisateur, il y a de fortes chances que le produit ne soit pas ou mal utilisé.

La plupart des applications standard respectent la philosophie documentée de l'éditeur, même s'il existe des contre-exemples. Nous nous sommes appuyés sur les documents officiels de chaque éditeur, ainsi que sur l'analyse des applications présentes en natif dans les systèmes.

Les grammaires sont souvent visibles, c'est-à-dire représentées par un composant graphique que l'utilisateur identifie immédiatement.

Mais il existe également des grammaires invisibles principalement gestuelles, non supportées par un composant graphique. Suivant le niveau de compétence et de connaissance des gestures de l'utilisateur, les fonctions proposées par ces grammaires peuvent être ignorées par les utilisateurs.

Les grammaires invisibles doivent faire l'objet d'une formation de l'utilisateur lors du premier accès ou plus tard. Des vidéos, des animations ou des superpositions sur l'application (overlays), permettent d'informer l'utilisateur des interactions possibles.

Les gestures permettent entre autres d'économiser l'espace disponible pour les informations de l'application et d'optimiser l'efficacité de réalisation d'une action.

Un rapide sondage réalisé sur 130 personnes chez OCTO technology montre que 20 % des utilisateurs de smartphones ignorent les gestures dans la gestion des listes par exemple.

Analyse

Les fonctionnalités importantes doivent toujours utiliser un élément visible ou exploiter une gesture massivement connue, exemple : pincer pour zoomer.

Avant de commencer à étudier les différents OS, il est important de traiter du cas particulier d'Android. En effet, ce système d'exploitation a subi une évolution importante au niveau du design et de l'ergonomie. La grammaire utilisée dans les versions 2 (avant octobre 2011) est très différente

de la grammaire utilisée dans les versions 3 et suivantes. Il est à noter que la version 1 n'est plus utilisée.

Cela ne serait pas un problème si les téléphones avaient migré. Ce n'est pas le cas. À la date de rédaction, la moitié des terminaux du parc mondial utilisent la version 2 du système et ne pourront migrer vers la version suivante **[6]**.

Il existe des librairies de compatibilité (ActionBarSherlock par exemple) permettant de proposer la nouvelle ergonomie sur une ancienne version. Nous préconisons d'utiliser cela dès que possible.

Lorsque les écarts sont importants, nous en ferons mention. Par principe, nous décrirons les approches actuelles à prendre en compte.

Notre étude s'applique à la version 6.1 de iOS, la version 4.2 d'Android et la version 8.0 de Windows Phone.

2 Organisation générale

Nous allons commencer par l'organisation générale de la plupart des applications de chaque OS.

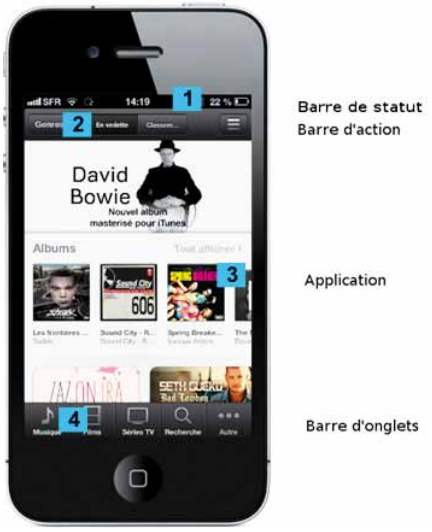


Figure 2

iOS propose, pour une application archétype, une barre de statut en haut avec les informations générales sur l'état des connexions du téléphone, ainsi que l'heure et la batterie. En dessous, une barre de navigation présente les items de navigation et permet de filtrer les informations affichées à l'écran. Elle contient entre autres le bouton retour qui n'existe pas physiquement sur le téléphone contrairement aux Windows Phone et la plupart des Androids. La barre de navigation peut contenir des actions. Puis, un espace important est réservé pour afficher l'application. Le bas de l'écran peut contenir une barre d'onglets qui permet de passer par différentes sous-tâches, vues et modes.

Il est à noter que la zone destinée à l'application possède une résolution variable suivant les modèles de tablettes/téléphones (480 × 320, 960 × 640 et 1136 x 640 pixels).

La résolution utilisable par les applications est limitée à 480 × 320 (iPhone classique) et 568 x 320 (iPhone 5). Il s'agit de multiples de pixels, variables suivant les modèles. Ainsi, les applications sont portables automatiquement pour différents modèles. Seule la finesse des anticrénelages des polices de caractères et des images est alors impactée par l'augmentation de la résolution.

Un bouton physique permet de revenir au bureau, d'activer le multitâche ou d'activer Siri. Suivant la rotation du téléphone ou de la tablette, ce bouton va suivre le positionnement de l'appareil. L'utilisateur doit le chercher en fonction de la manière dont il tient son appareil. Cela est particulièrement notable avec une tablette.

Une étude indique que les utilisateurs allument leur téléphone toutes les six minutes pour savoir s'il y a du nouveau **[7]**.

Pour alerter d'un événement (appels, messages, notifications...) Apple prévoit des signaux sonores ainsi que l'usage de la vibration.

Néanmoins, il est possible d'utiliser le flash de l'appareil photo en guise d'alerte. Ceci dit, le déclenchement du flash à la réception d'un message ou d'un appel peut prêter à confusion et être mal interprété par l'entourage. Vous vous exposez à un regard sévère dans les transports en commun si par malheur vous recevez un message, téléphone en main.

Figure 3

Android propose une organisation avec une barre de statut en haut, puis une barre d'action qui peut être découpée en deux avec une partie en haut et une partie en bas, une zone réservée à l'application et un menu système tout en bas sur fond noir, s'il est nécessaire de remplacer les boutons physiques (suivant les modèles). Les nouveaux téléphones Android affichent désormais des boutons systèmes virtuels alors qu'ils étaient auparavant physiquement présents sur les téléphones.

Le menu système se déplace en fonction du positionnement du périphérique (portrait ou paysage). Son positionnement change s'il s'agit d'un téléphone ou d'une tablette. L'espace vertical sur un téléphone en paysage est limité. Android place donc le menu à droite. Sur la tablette, le menu système reste en bas quel que soit le retournement de la tablette. Il y a plus de place pour l'affichage.

Les boutons systèmes permettent à l'utilisateur de toujours savoir où trouver les actions majeures (retour, accueil, multitâche) : toujours au même endroit pour les tablettes et en bas ou à droite pour les téléphones en fonction de leur position paysage ou portrait.

Tout comme iOS, Android prévoit des alertes, sonores ainsi que l'usage de la vibration. Sur la plupart des modèles, une LED dont la couleur peut être paramétrée en fonction du type d'alerte, est prévue en façade pour signaler à l'utilisateur une notification. Cela lui permet d'avoir une indication que quelque chose se passe, sans avoir à allumer son téléphone toutes les six minutes.

Les anciens modèles Android (d'octobre 2009 à février 2011) proposent une ergonomie différente, sans barre d'action et des boutons physiques pour la navigation ou la recherche. Certains téléphones qui ont migré vers la dernière version du système proposent alors également des boutons physiques à la place des boutons systèmes.

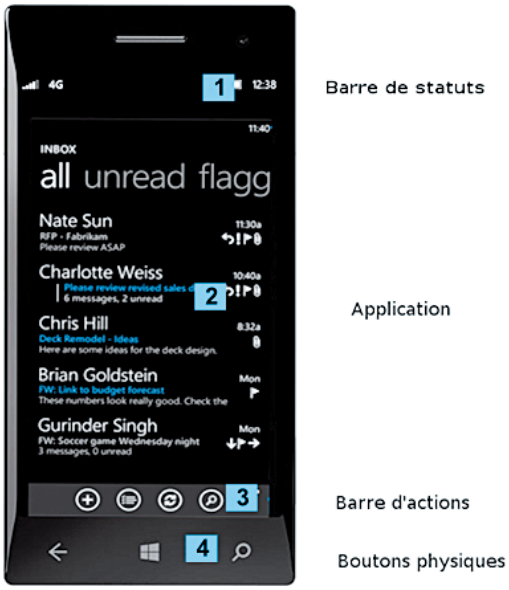


Figure 4

Windows Phone propose une barre de statut en haut de l'écran, un titre, un menu sous forme de sous-titres, un espace pour l'application et une barre d'action en bas. Les terminaux ont tous des boutons physiques.

Une deuxième barre d'actions contextuelle peut être située à droite ou à gauche suivant les cas.

Analyse

L'utilisateur doit rechercher la barre d'action contextuelle.

Pour les alertes, Microsoft prévient ses utilisateurs avec un son et une vibration.

Analyse

Apple choisit de n'utiliser qu'un seul bouton pour réaliser plusieurs actions : revenir au menu, activer Siri et gérer le multitâche. D'autre part, la gestion de l'historique de navigation ne se fait que dans l'application.

Android de son côté, ne présente aucun bouton physique d'interaction pour les nouveaux périphériques, mais opte pour la virtualisation des boutons d'actions les plus courants. Cela permet à l'utilisateur de savoir, sans les chercher, où se trouvent ces actions et de lui présenter au meilleur endroit, au moins pour un droitier.

Pour gérer la version 2 du système, les applications doivent s'adapter pour exploiter les boutons physiques ou proposer des équivalents à l'écran.

Microsoft utilise des boutons physiques pour les actions principales de navigation et de recherche.

Particularité intéressante, Android se distingue sur la question des notifications en proposant sur la plupart des modèles une alerte lumineuse paramétrable.

3 La navigation

Il faut permettre à l'utilisateur de naviguer entre les écrans. Les approches sont différentes suivant les systèmes.

La navigation intègre le parcours des pages et de l'arborescence, les filtres et l'accès au menu principal.

3.1 Arborescence

L'utilisateur doit toujours savoir où il est et où il peut aller. L'affichage doit lui permettre de comprendre comment naviguer entre les différents écrans. Comment revenir en arrière ? Comment retourner à la page d'accueil de l'application ?

iOS recommande de naviguer d'écran en écran via des boutons dans la barre d'action, les boutons ont alors une forme spécifique. Néanmoins, les applications proposent souvent des boutons de navigation directement dans le corps de l'écran.

Pour permettre à l'utilisateur de revenir en arrière, le standard veut que l'on place le bouton retour en haut à gauche dans la barre d'action. Le bouton a une forme qui suggère à l'utilisateur l'action de retour. Le libellé est soit le label « retour », soit le titre de la page précédente.

Figure 5

Si l'utilisateur touche sur un lien qui le mène vers un écran d'une autre application (cartographie, lien internet, image, etc.), il n'a pas de moyen réel, direct de revenir en arrière. Par exemple, un clic sur un lien dans un mail ne permet pas d'y revenir.

Analyse

Ce mode de navigation ne permet pas à l'utilisateur de revenir à une application précédente.

Pour éviter de sortir d'une application, iOS propose de nombreux visualiseurs standards pour différents types de documents (Word, PDF, etc.). Ainsi, les applications peuvent présenter des documents, mais pas les modifier.

Android, inspiré par les navigateurs, a longtemps proposé des boutons physiques pour revenir à l'écran précédant, retourner sur le bureau, afficher le menu contextuel de la page ou effectuer une recherche.

Le bouton retour fonctionne, même si l'utilisateur navigue entre plusieurs applications. Par exemple, après un clic sur un lien dans un mail, le bouton retour permet d'y retourner immédiatement.

Les utilisateurs n'avaient pas toujours conscience qu'ils pouvaient invoquer le menu contextuel des applications (grammaire invisible). Ils ne savaient pas non plus qu'un appui long permet d'avoir accès à des services complémentaires comme la gestion du multitâche.

Figure 6

Depuis la version Honeycomb de février 2011, conçue initialement pour tablette, Android a décidé de supprimer les boutons physiques et de les remplacer par des boutons virtuels dans une barre système, présente sur tous les écrans. Cela permet de tourner et retourner la tablette en ayant ces boutons toujours au même endroit pour l'utilisateur.

Un nouveau bouton (composé de deux rectangles) fait alors son apparition, pour permettre à l'utilisateur de gérer le multitâche. Le bouton de recherche global disparaît. Il est remplacé par un clic sur le bouton home, suivi d'un glissé vers le haut. Les applications doivent alors s'adapter à ces différentes situations.

Figure 7

Si une application est ancienne, conçue pour une version avant ICS et qu'il n'existe pas de bouton physique pour accéder au menu de l'écran, le menu système peut posséder une icône supplémentaire pour remplacer le bouton physique « menu ». Elle est symbolisée par des points de suspension verticaux.

Figure 8

Les applications modernes préfèrent ajouter un menu à la barre d'action (grammaire visible). Cela rend cette fonctionnalité plus accessible.

La barre d'action sert également à la navigation dans l'application. Un chevron avant le titre permet à l'utilisateur de savoir qu'il est dans un sous-écran et qu'il peut remonter d'un niveau. Il lui suffit de cliquer sur le titre de la page.

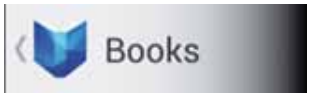


Figure 9

Figure 10

Cette navigation est différente du bouton système « retour ». En effet, « retour » permet de retourner à l'écran précédant, quand l'appui sur le titre permet de remonter d'un niveau en imitant une bonne pratique du web (un clic sur le logo permet de revenir à la page d'accueil). Par exemple, si un utilisateur consulte une image attachée à un mail, le bouton « retour » lui permet de retourner au mail. Si l'utilisateur navigue sur plusieurs écrans, « retour » lui permet de rebrousser chemin.

Suivant les versions, les boutons physiques proposent des fonctions supplémentaires lors d'un appui long. Les boutons systèmes peuvent être touchés deux fois ou glissés vers le haut pour offrir d'autres fonctionnalités.

Analyse

Les deux modes de retours : intra application ou historique type web sont source de confusion pour les utilisateurs. Nous préconisons de supprimer le retour intra application qui est mal compris et qui en plus est dans une zone difficile d'accès pour la grande majorité des utilisateurs.

Windows Phone propose des boutons physiques pour la navigation. L'utilisateur peut à tout moment revenir à l'écran précédant, retourner à la page d'accueil du téléphone ou effectuer une recherche internet. Il peut naviguer d'application en application sans perdre son historique. Il n'y a pas de notion de menu contextuel.

Un chevron à gauche du titre permet de remonter de l'arborescence.

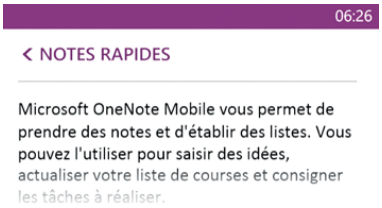


Figure 11

La navigation s'effectue également en touchant certains textes. Cela n'est pas facilement identifiable. Par exemple, dans l'écran suivant, il est possible de toucher « Tous les comptes », mais pas « Retweeté par Kim Dotcom ».

Figure 12

Analyse

C'est une navigation théoriquement visible, mais dans les faits : invisible.

Un appui long sur le bouton retour permet de gérer le multitâche.

Analyse

La navigation par historique est conforme aux usages du web. Elle permet une navigation entre plusieurs applications. Android et Windows Phone proposent une navigation mixte, par historique et par cible.

IOS ne propose pas de navigation par historique entre les applications. Cela a un impact important sur les applications. En effet, quitter l'application ne permet pas d'y revenir facilement. Il faut donc ajouter des composants logiciels pour rester le plus longtemps possible dans l'application. Le système propose alors de nombreux visualiseurs compatibles avec les formats les plus classiques.

Les autres OS peuvent s'appuyer sur des applications qui ne sont pas toujours installées sur le périphérique. Cela nécessite une phase d'installation de ces dernières.

3.2 Suivant/précédent

Le modèle liste / détail est très courant. Une application présente une liste de mails par exemple, puis l'utilisateur les consulte un à un. Lors de la consultation d'un élément d'une liste, il est pratique de proposer une navigation directe à l'élément suivant ou précédent.

iOS propose d'ajouter dans la barre d'action des boutons suivant/précédent représentés par des flèches verticales ou horizontales.

Figure 13

Android propose à l'utilisateur de glisser l'écran sur la droite ou la gauche. Il n'y a pas d'indication que cela est possible (grammaire invisible). Par convention, les applications doivent respecter ce modèle.

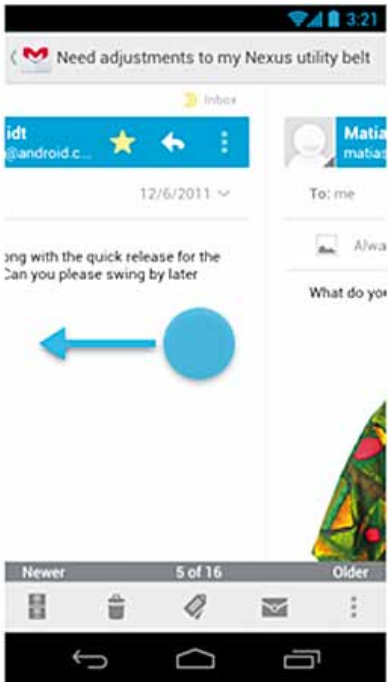


Figure 14

Windows Phone propose d'ajouter des boutons dans la barre d'action avec des flèches horizontales.

Figure 15

Analyse

La navigation directe entre éléments d'une liste est importante. Il est judicieux d'utiliser les approches des différents OS pour cela.

3.3 Les onglets

Les onglets servent à proposer des sous-menus, des vues ou des modes. Cela permet à l'utilisateur de rester dans le même contexte et de naviguer en largeur. Par exemple, un calendrier propose une vue par jour, semaine ou mois. Notez que la barre d'action peut rester visible lorsque les onglets servent de filtre.

iOS propose deux approches. Une barre d'onglets propose des actions, des filtres et un contexte actif. Les choix sont identifiés par un texte et une icône. L'étape en cours est identifiée par le changement de couleur de l'icône. Elle est située en bas de l'écran.

Figure 16

Un nombre limité de boutons est disponible. Si l'application a besoin de plus d'options, le dernier est représenté par des points de suspension. Un appui permet alors d'avoir accès à un menu complémentaire. L'utilisateur peut alors choisir les actions disponibles par défaut à l'écran. Ce choix doit être mémorisé par les applications.

Android propose des onglets qui peuvent être fixes ou mobiles s'ils ne tiennent pas tous à l'écran. Cette zone d'onglets est située en haut de l'écran. L'utilisateur peut glisser la barre des onglets avant d'en sélectionner un. L'onglet actif est signalé par un soulignement de couleur. Les titres et les icônes sont bien délimités et ne peuvent être confondus avec un simple texte.

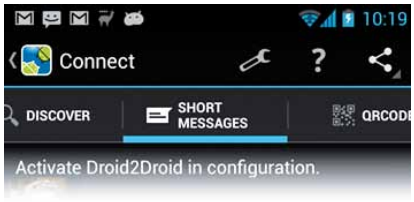


Figure 17

Suivant la largeur de l'écran, les onglets peuvent être présents sous la barre de titre, dans la barre de titre, ou présentés en combo-box dans la barre de titre.

Pour les versions 2 d'Android, il est possible d'utiliser des librairies de compatibilités pour proposer des onglets suivant le même modèle.

Windows Phone propose des onglets via des sous-titres. Cette approche s'appelle « Pivot » dans le monde Microsoft. Le premier utilise une couleur plus vive pour indiquer qu'il est actif. Un glissement permet de passer d'un onglet au suivant et de revenir ainsi au premier. S'il n'est pas visible, il n'est pas possible d'accéder directement à un onglet sans passer par les précédents.

Si plusieurs titres d'onglets sont visibles. Il n'est alors pas évident pour l'utilisateur de connaître exactement le titre de l'onglet suivant. Dans l'exemple suivant, est-ce que le prochain onglet est « jour » ou « jour agenda » ? Les titres des sous-onglets ne sont en effet pas limités à un seul mot et il n'existe pas de délimiteur graphique pour les séparer.

Figure 18

3.4 Les filtres

Les applications proposent souvent de filtrer les éléments affichés (format du calendrier, type de message, etc.)

iOS propose d'utiliser un contrôle de segmentation. Il s'agit de boutons accolés les uns aux autres et présents dans la barre d'action, en haut ou en bas.

Figure 19

Suivant les langues, il peut être difficile de traduire le libellé des boutons pour respecter l'espace disponible.

Android ne propose pas de composant pour cette fonctionnalité. Les applications utilisent généralement des onglets, des combo-boxes ou des actions dans la barre d'action (voir Figure 20).

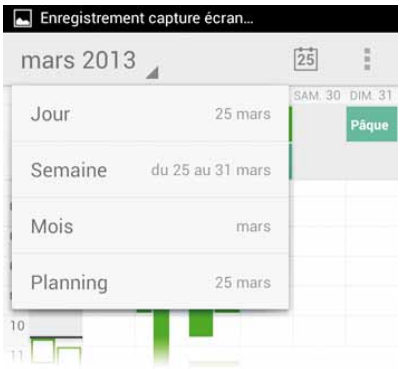


Figure 20

Windows Phone propose d'utiliser les onglets pour les filtres.

Analyse

Les approches sont très différentes et ont un impact sur la conception de l'application et le nombre d'écrans.

Suivant le nombre de filtres à intégrer, l'application peut s'appuyer sur les guidelines ou devra trouver une approche différente.

Par exemple, une application qui utilise beaucoup de filtres pourra utiliser une combobox sur Android et ne pourra pas utiliser le contrôle de segmentation sur iOS. Il faudra trouver dans ce cas une autre solution.

3.5 Menu principal

Pour éviter le retour systématique à l'écran principal de l'application, il est possible de proposer un menu global ou principal, appeler Navigation *Drawer*, disponible depuis pratiquement tous les écrans.

Les versions Web des services de Google utilisent également ce modèle.

iOS ne propose pas de composant pour ce menu global. Néanmoins, on le trouve dans de plus en plus d'applications. Il est accessible via un bouton reconnaissable avec son icône représentant trois barres horizontales.

Analyse

Cette icône n'est pas une norme constructeur, mais une pratique instaurée par une application qui a été plébiscitée jusqu'à devenir une norme.

Figure 21

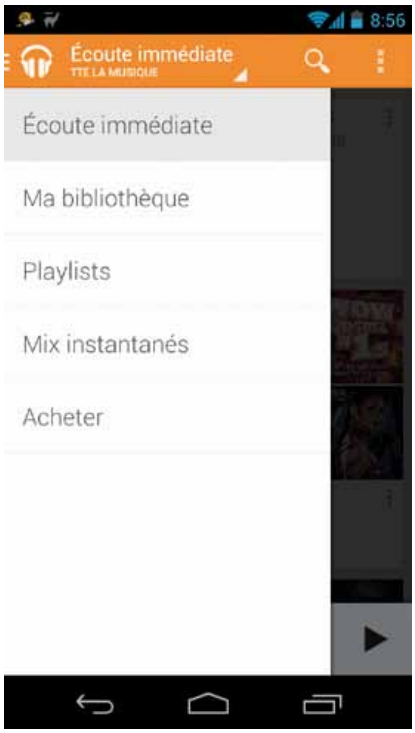


Figure 22

Android propose depuis peu, via Google Play, un Navigation *Drawer*. Les nouvelles versions des applications intègrent peu à peu ce composant (voir Figure 22).

Windows Phone n'utilise pas ce type de composant. L'utilisateur doit remonter à la racine de l'application.

Analyse

La présence d'un menu principal accessible depuis la plupart des écrans est une bonne approche pour optimiser l'utilisation de l'application. La dernière activité utilisée peut être mémorisée pour la proposer au prochain lancement de l'application.

3.6 Les titres

L'utilisateur doit avoir une idée de l'application qu'il est en train d'exécuter et l'endroit où il se situe dans cette application. Il est nécessaire de sacrifier une zone de l'écran pour l'aider à se localiser.

Sur **iOS**, le titre de l'application se trouve dans la barre de navigation lorsqu'elle est présente. Souvent on y trouve le titre de la page en cours. En outre, on retrouve peu souvent d'icônes permettant d'identifier l'application. Après une mise en marche de l'appareil, l'utilisateur retrouve une page d'une application, sans pouvoir l'identifier. Seuls le design et l'univers graphique permettent à l'utilisateur d'identifier l'application dans laquelle il se trouve.

Par exemple, les titres de ces deux applications sont très similaires (voir Figures 23 et 24).

Figure 23

Figure 24

Sur tablette, l'espace disponible permet de présenter des applications visuellement sans ambiguïté. L'espace étant plus grand, les éléments différenciant sont plus nombreux.

Android propose d'indiquer le titre à côté de l'icône de l'application, sur une barre d'action normalisée. Un petit symbole chevron à gauche de l'icône, s'il est présent, signale à l'utilisateur qu'il est dans un sous-menu et qu'il peut revenir d'un niveau de l'arborescence. Le titre de l'application ou le nom de la page est présenté à côté de l'icône. Puis, une collection de boutons d'action est proposée.

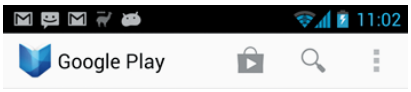


Figure 25

La version 2 d'Android proposait un titre dans une barre grise étroite, équivalente à la barre de statut. Des librairies de portabilité permettent d'utiliser la barre de titre de la nouvelle version de l'OS.

Windows Phone propose plusieurs approches. Il peut exister un titre principal, puis des sous-titres qui correspondent généralement aux titres des onglets ou des filtres. L'onglet actif utilise une couleur plus contrastée que les autres onglets.

Plusieurs combinaisons de titres/sous-titres sont utilisées :

- Le titre principal utilise une grande police de caractère et le sous-titre une police moyenne. Le titre peut être trop grand pour être intégralement visible à l'écran. Il déborde alors sur les vues suivantes, informant l'utilisateur de cette extension. Cela ne laisse que 60 % de l'espace pour l'application .

Figure 26

- Le titre peut utiliser une police moyenne avec un texte en majuscule. Les sous-titres utilisent alors une police plus grande. Le titre et les sous-titres des onglets sont toujours visibles. Seule la zone en dessous est destinée à l'application. Cela laisse 80 % de l'écran pour l'application (voir Figure 27).

Analyse

iOS ne permet pas toujours une identification immédiate de l'application active.

La présence de l'icône de l'application sous Android v3+ permet à l'utilisateur d'identifier immédiatement l'application qu'il utilise, dès l'allumage de l'écran. Il est nécessaire d'utiliser ce modèle d'ergonomie, même si l'application peut être exécutée sur un smartphone utilisant la version 2 de l'OS.

Windows Phone consomme un espace important pour les titres, au détriment de l'espace réservé à l'application. L'application doit intégrer un défilement de cet espace, car l'utilisateur doit très souvent le déplacer. La traduction des titres est délicate, car la taille des mots a un impact important sur l'usage de l'application.

4 Les actions de l'utilisateur

L'utilisateur doit pouvoir déclencher des actions sur l'élément représenté par l'écran ou sur certains éléments de l'écran.

Figure 27

4.1 La barre d'actions

Les écrans sont enrichis de menus, pour proposer des fonctions complémentaires ou pour naviguer d'écran en écran.

Sur **iOS**, les boutons d'actions peuvent se retrouver à tout endroit en haut en bas ou dans le corps de l'écran. Il peut y avoir de un à trois groupes de boutons (gauche, droite et milieu). Comme l'espace est limité, cela peut engendrer des problèmes pour le traducteur : trouver le mot décrivant l'action et entrant dans l'espace.

Figure 28

Chaque icône ou bouton peut déclencher l'ouverture d'un sous-menu, diriger l'utilisateur vers une autre page ou ouvrir une boite de dialogue complémentaire. Il n'existe pas de convention permettant à l'utilisateur de savoir à l'avance qu'une boite de dialogue sera ouverte avant de valider son action. Une icône spécifique indique l'ouverture d'un menu complémentaire (voir Figure 29).

Android propose une barre d'action normalisée pour toutes les applications. Le menu complémentaire associé à l'écran actif est identifié par des points de suspension verticaux (sur les smartphones ne possédant pas de bouton physique « paramètre »). En cliquant dessus, l'utilisateur peut afficher

Figure 29

la liste des actions complémentaires. Si la place le permet, Android affiche les éléments du menu dans la barre d'action par ordre de priorité. **Plus les boutons sont importants, plus ils doivent être à gauche**. En cas de manque de place, seuls les boutons les plus à gauche seront directement visibles les autres seront accessibles via un bouton avec trois points de suspension.

Pour chaque item du menu, il est possible d'afficher uniquement l'icône ou l'icône et son libellé (utilisé sur les tablettes). Pour signaler qu'une action nécessite des paramètres complémentaires, les libellés doivent se terminer par des points de suspension. Ils ne sont jamais tronqués.

Le menu peut être coupé en deux parties, une en haut et une autre en bas. Ainsi, suivant la largeur et la position du téléphone, en portrait ou paysage, la barre d'action s'adapte.

Un appui long sur une icône révèle le libellé associé.

Lorsque l'utilisateur affiche le menu complémentaire identifié par trois points de suspension verticaux, seul le texte est présenté sans les icônes. Suivant l'orientation du terminal, l'utilisateur peut voir une icône ou un texte dans le menu complémentaire.

Le menu complémentaire est présenté sur les téléphones ne possédant pas de boutons physiques. Sinon, seuls les boutons d'actions de l'application sont présents. Les fonctions complémentaires sont disponibles lorsque l'utilisateur clique sur le bouton physique de paramétrage.

Analyse

Ce menu utilise un mode visible dans les derniers téléphones, et un mode invisible sur les anciens téléphones. Cela ne facilite pas la conception des applications.

La barre d'action peut également afficher des onglets pour des sous-tâches. Suivant l'espace disponible, les onglets peuvent être présentés sous le titre et les boutons d'actions,

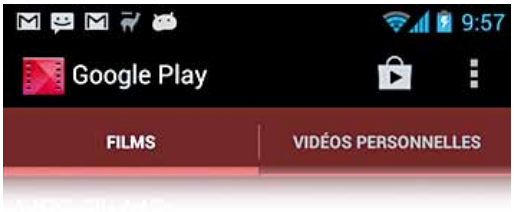


Figure 30

dans une combo-box,

Figure 31

ou dans la barre d'action.

Figure 32

Un champ de saisie peut également y être intégré.

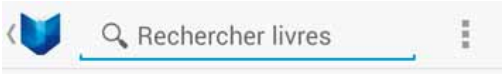


Figure 33

Cette barre peut évoluer suivant le contexte, lors de sélections multiples par exemple.

Les téléphones anciens proposent des boutons physiques. Ces derniers font redondance avec des actions présentes dans la barre d'action.

Par exemple, la recherche peut être déclenchée par l'appui sur le bouton physique correspondant.

Android déconseille de procéder ainsi. Le bouton physique de recherche déclenche maintenant une recherche globale et indépendante de l'application active. Il est nécessaire de laisser dans tous les cas une icône dans la barre d'actions pour offrir une recherche dans l'application.

Le bouton physique de paramétrage est traité automatiquement par la gestion de la barre de tâches. Les points de suspension disparaissent si un bouton physique est présent.

Avec une tablette, la barre d'action consomme l'intégralité de la largeur, limitant en hauteur la surface utile à l'application.

Il est alors intéressant d'exploiter cet espace pour ajouter des commandes, des actions ou des champs de saisie.

Windows Phone propose également une barre d'actions normalisée (Application bar), située à côté des boutons physiques. Elle est constituée d'icônes monochromes, entourées de cercles.

Figure 34

Quatre icônes sont disponibles. Pour les autres actions, des points de suspension sont proposés.

Un appui sur les points de suspension permet d'afficher un libellé avec une police très petite et d'afficher les menus complémentaires.

Figure 35

Lors de la présence d'un menu et d'une saisie, le clavier se positionne au-dessus de la barre d'action, les laissant accessibles (voir Figure 36).

Analyse

iOS propose des actions avec une icône ou un libellé. Suivant les langues, il peut être difficile de les traduire, car cela consomme un espace variable sur l'écran. Trop de boutons peuvent rendre difficile la mise en œuvre d'une traduction.

Android s'adapte automatiquement et présente une grande variabilité suivant les versions et les modèles.

Windows Phone propose un libellé souvent difficile à lire.

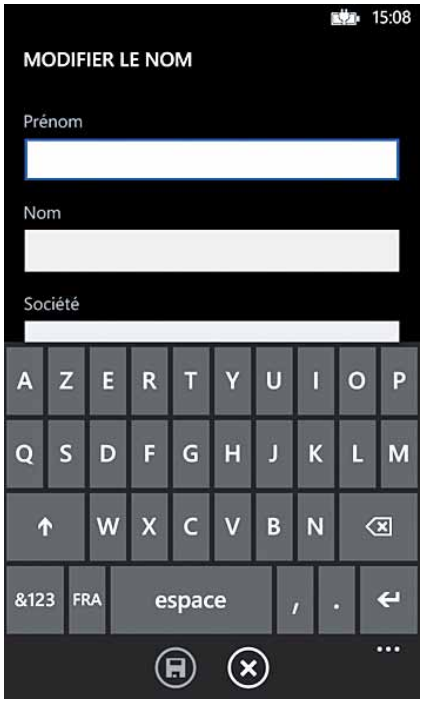


Figure 36

4.2 Menu contextuel d'un objet

Il est souvent nécessaire de proposer des menus contextuels à un objet présenté à l'écran. Cela peut être une image, un élément d'une liste ou tout autre composant graphique.

iOS propose de passer par une page de détail et le signale par un chevron à droite du libellé. Cela mène généralement l'utilisateur vers un nouvel écran.

Figure 37

Si plusieurs types d'actions sont possibles pour un même objet, iOS propose de passer par une boite de dialogue regroupant les différentes actions.

Figure 38

Il existe également un raccourci, en glissant un élément de la liste horizontalement (swipe to action). Un bouton apparaît alors en superposition, permettant par exemple de supprimer un élément.

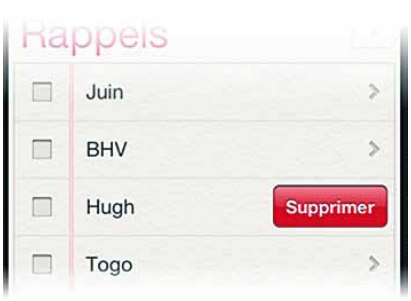


Figure 39

Android propose un appui long pour entrer dans le mode « sélection », et cocher tous les éléments devant subir le même traitement.

Les items doivent indiquer un petit triangle en bas et à droite, pour signaler la présence d'actions complémentaires. Cela ouvre un menu local (voir Figures 40 et 41).

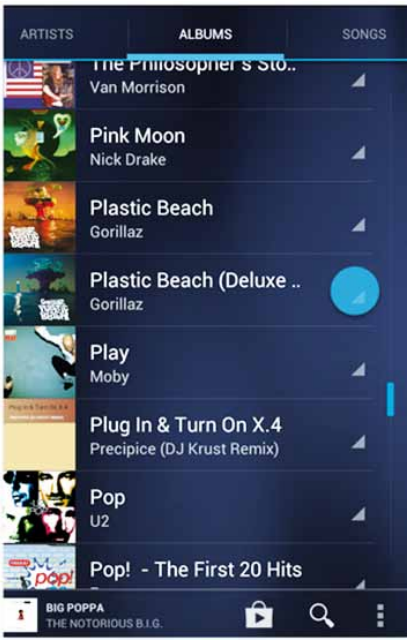


Figure 40

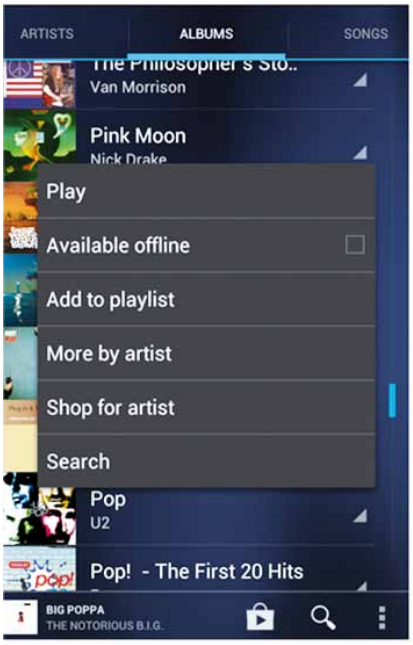


Figure 41

La version précédente proposait d'utiliser un appui long sur l'élément pour déclencher l'ouverture du menu contextuel. Cette approche est maintenant abandonnée.

Parfois, un glissé horizontal sur un élément de la liste permet d'effectuer une action rapide (effacer, archiver...).

Analyse

La gesture (appui long) pour entrer en mode sélection multiple est peu connue et donc son utilisation nécessite de guider l'utilisateur.

Windows Phone propose d'utiliser un appui long sur l'élément pour ouvrir le menu contextuel. Les utilisateurs n'ont pas toujours conscience que cela est possible.

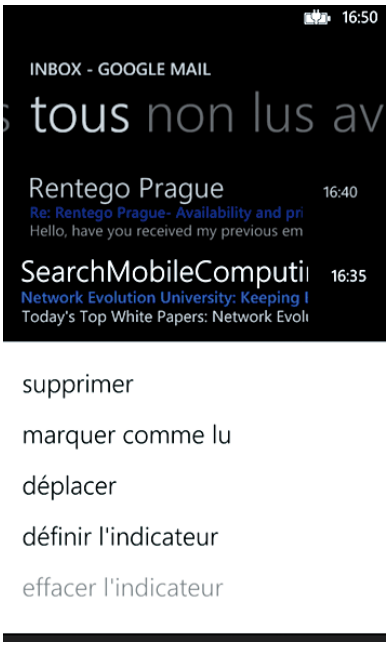


Figure 42

Analyse

Les approches sont très différentes suivant les OS. La navigation, l'ergonomie et le visuel en sont alors fortement modifiés d'un système à un autre.

Android et iOS indiquent clairement à l'utilisateur cette possibilité (grammaire visible).

C'est moins évident pour les utilisateurs de Windows Phone (grammaire invisible).

L'utilisation de gestures comme le glissé d'un élément peut être proposée en complément de l'approche visible. Une formation des utilisateurs est alors à prévoir.

4.3 Les icônes standards

Pour démarrer une application, l'utilisateur doit « cliquer » sur l'icône de l'application. On trouve généralement ces icônes sur des bureaux très différents suivant les OS.

Nous ne nous attardons pas sur les spécificités des bureaux, car cela n'a pas d'impact sur la conception des applications.

Chaque OS propose des conventions pour les différentes icônes.

iOS propose d'utiliser des icônes dans un carré aux coins arrondis.

Figure 43

Les icônes pour la barre d'action ou la barre d'onglets doivent être monochromes.

Figure 44

Android propose d'utiliser des icônes détachées pour les applications. Il est déconseillé d'utiliser un cadre ou un fond.

Figure 45

Les icônes de la barre d'action doivent être monochromes et en silhouette.

Figure 46

Les icônes de la barre de notification sont plus petites et monochromes.

Figure 47

Windows Phone demande des icônes plates, généralement monochromes dans un carré coloré.

Figure 48

La barre d'action utilise des icônes très petites, car ces dernières doivent être entourées d'un cercle.

Figure 49

Les icônes de la barre d'actions doivent être monochromes. Il ne doit pas y avoir d'icône de *Retour*, car tous les terminaux possèdent un bouton physique pour cela. La résolution de 26x26 ne permet pas beaucoup de fioritures.

Analyse

Le peu d'espace disponible rend les icônes parfois difficiles à lire, comme l'icône « Add to favorites ».

L'utilisation de la disquette par Microsoft, qui a une posture anti skeuomorphisme, comme symbole de l'enregistrement est intéressante. Elle montre la difficulté de se détacher de l'allégorie du réel. Cette icône, qui avait du sens pour la génération 1980/1990, ne veut plus rien dire pour les plus jeunes et peut même être confondue avec une sorte de maison. Pourtant, la force de l'habitude et la difficulté d'inventer un symbole abstrait représentant l'enregistrement « pousse » à l'utilisation de ce symbole. À moins que cette icône soit la preuve que le passage du réel à l'abstrait soit aussi une question de temps.

4.4 Barre de statut

Les différents OS proposent des barres de statut avec plus ou moins d'informations techniques et applicatives, ainsi que l'heure. Sauf exception, cette barre est toujours présentée à l'utilisateur.

iOS propose une barre de statut présentant uniquement des icônes systèmes. Elle est toujours à la même place sur le téléphone (partie haute), même après une rotation. Il y a 3 couleurs imposées (gris, noir et noir semi-transparent) sur téléphone. Sur tablette, la barre est toujours noire.

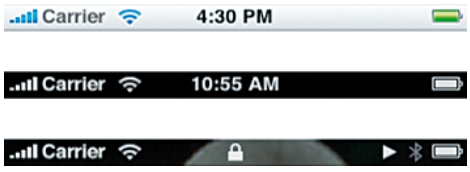


Figure 50

Lors d'un accès réseau, l'application doit demander l'affichage d'une roue d'attente sur cette barre. Elle permet également d'ouvrir le centre de notification en la glissant vers le bas.

Android propose une barre de statut sur fond noir, avec à droite, des informations système comme l'accès aux différents réseaux, et à gauche des icônes gérées par les applications. C'est ainsi qu'une application peut signaler qu'un service est actif, qu'une action est en cours ou qu'un type d'événement est présent (un mail par exemple). Cela sert de notification globale à l'utilisateur. Les icônes peuvent être animées pour signaler un téléchargement d'un fichier par Bluetooth par exemple ou l'envoi de données dans le Cloud.

Figure 51

Certains constructeurs proposent un dégradé foncé sur la barre de statut. Les icônes de notifications doivent en tenir compte et utiliser un fond transparent.

Sur les tablettes, la barre de statut est positionnée toujours en haut. Un glissement vers le bas depuis la gauche de la barre permet de faire apparaître les notifications des applications. Un glissement vers le bas sur la droite de la barre permet de faire apparaître les informations systèmes et donne accès aux paramètres.

Android v2 utilise une barre de statut sur fond blanc. Il est alors nécessaire de prévoir une déclinaison des icônes de notifications pour cette situation.

Windows Phone propose une barre de statut transparente avec uniquement des icônes système. Elle est toujours à la même place sur le téléphone (sur le bord opposé des boutons physiques), même après une rotation. Il n'est pas possible d'ajouter des icônes complémentaires. La barre s'affiche lorsque l'utilisateur la sélectionne et s'efface quelques secondes après.

Figure 52

Les applications peuvent exploiter l'espace sous la barre de statut, ou consommer l'intégralité de l'écran comme c'est le cas sur le bureau avec les tuiles.

Analyse

Android permet au concepteur de prévoir une notification discrète lorsqu'un événement a lieu sur l'application alors qu'elle n'est plus visible. Ceci optimise, du point de vue utilisateur, le multitâche. C'est un avantage ergonomique par rapport aux autres OS qui doit être exploité lors de la conception d'une application.

Dans la deuxième partie, nous continuerons à explorer les différents composants ergonomiques, puis nous synthétiserons toutes nos découvertes dans un tableau. ■

Liens

[1] http://goo.gl/VTGFm

[2] http://goo.gl/8DbYt

[3] http://goo.gl/1cJyM

[4] http://goo.gl/U35Fd

[5] http://goo.gl/juaIA

[6] http://goo.gl/BvZRw

[7] http://goo.gl/ryD3B

ENFIN UNE GESTION DE DÉPENDANCES CORRECTE POUR PHP : COMPOSER !

par Stéphane Mourey - Taohacker

Finies les galères avec PEAR ou PECL, nécessitant d'être administrateur de la machine et d'avoir un accès en ligne de commandes, voici Composer, la solution pour gérer facilement les dépendances de vos projets.

La plupart des développeurs PHP ont entendu parler de PEAR et de PECL, mais ne les ont que rarement utilisés pour des raisons diverses, et plus rarement encore avec bonheur. Le fait que PHP soit également souvent utilisé chez des hébergeurs en environnement mutualisé sans même un accès à la ligne de commandes pourtant nécessaire à ces outils, explique également le peu de succès qu'ils ont eu. Nous ne nous étendrons pas là-dessus, mais nous pouvons nous réjouir de l'émergence de la solution plus adaptée qu'est Composer. Composer vous permet de gérer les dépendances de votre projet dans son propre espace. En ce sens, il s'agit d'un gestionnaire de dépendances et pas d'un gestionnaire de paquets : les paquets sont installés dans un sous-répertoire de votre projet et, selon votre configuration, ne sont accessibles qu'à lui. Du moins est-ce l'idée générale, des options vous permettant de faire des variations sur ce comportement.

1 Installation

Avant d'utiliser Composer, il faudra l'installer. Vous aurez besoin d'une version de PHP supérieure ou égale à 5.3.2, et... d'un accès à la ligne de commandes, mais vous n'aurez pas besoin d'être administrateur, sauf à vouloir l'installer globalement. Pour une installation locale, la commande suivante fera l'affaire :

```
$ curl -sS https://getcomposer.org/installer | php
```

Pour une installation globale, il faudra ajouter cette commande :

```
# mv composer.phar /usr/local/bin/composer
```

Si vous en avez le droit, on ne peut que vous le recommander.

Éventuellement, selon la façon dont sont gérés les paquets que vous voudrez installer, il vous faudra disposer de **git**, **svn** et **hg**, mais cela ne devrait pas être un problème.

1.1 Suis-je en train de me contredire ?

La critique que j'ai rapidement formulée sur PEAR et PECL dans mon introduction leur reprochait notamment la nécessité d'avoir un accès en ligne de commandes aux serveurs pour pouvoir installer les paquets. Et là, je ne fais que parler de ligne de commandes. Et même, je vais continuer sur cette lancée encore un petit moment. Pourtant, je persiste à dire que Composer ne nécessite pas d'accès au shell en production. Alors ? Je me marche sur les pieds ?

Le premier élément de réponse est que PEAR et PECL nécessitent un accès administrateur pour fonctionner, sinon les paquets ne peuvent pas être installés globalement. Pas Composer, ce qui est nettement moins exigeant.

Le second élément de réponse est que le résultat des traitements de Composer est exploitable sans accès à la ligne de commandes. Je m'explique : vous utilisez Composer sur votre machine de développement, tous les fichiers sont installés dans votre projet, le résultat est transmissible au serveur de production sans avoir à exécuter Composer à nouveau. Vous pouvez vous contenter d'un accès FTP, cela ne posera aucun problème de ce point de vue.

En passant, Composer présente du coup un autre avantage sur PEAR : puisque les dépendances sont installées par projet, vous ne rencontrerez pas de problème si vous souhaitez utiliser une version plus récente de telle librairie dans le projet suivant, vous n'aurez pas à vérifier que l'ancien projet supporte bien la nouvelle version, puisqu'il conservera

la sienne. Il faut alors prendre garde à ne pas laisser des projets pourrir avec de vieilles bibliothèques bourrées de failles connues depuis des lustres...

2 Utilisation

2.1 Installer les dépendances

Un projet utilisant Composer se reconnaît à la présence d'un fichier composer.json à sa racine. Dès lors, une seule commande est nécessaire pour installer toutes les dépendances du projet :

```
$ php composer.phar install
```

ou

```
$ composer install
```

selon que vous ayez installé Composer localement ou pas.

Voici la sortie de la commande d'installation avec une seule indication de dépendance PHPExcel, fichier que nous verrons un peu plus loin :

```
Loading composer repositories with package information
Installing dependencies
  - Installing phpexcel/phpexcel (v1.7.7)
    Downloading: 100%

Writing lock file
Generating autoload files
```

À partir de là, vous remarquerez qu'un répertoire vendor a été créé dans le répertoire courant. C'est là que Composer stocke les paquets installés pour votre projet. À l'intérieur, vous trouverez les dossiers **composer** et **phpexcel**, ainsi que le fichier **autoload.php**,dont nous parlerons plus loin.

2.2 Écrire votre fichier composer.json, niveau débutant

Il est sans doute utile de savoir installer les dépendances d'un projet s'appuyant sur Composer, mais il est encore plus utile de savoir écrire son propre fichier **composer.json** afin de gérer les dépendances de son projet. Pour commencer, il n'est pas nécessaire de maîtriser toutes les subtilités de ce fichier pour pouvoir en tirer le bénéfice essentiel : une bonne gestion des dépendances au sein de votre projet. Il y a bien d'autres choses à dire sur ce fichier, mais plutôt avec l'objectif de produire son propre paquet installable par Composer.

Comme son nom l'indique, ce fichier utilise le format JSON (JavaScript Object Notation) et associe donc des clés à des valeurs formant des paires. La clé qui vous sera sans aucun doute la plus utile est **require**, car elle permet d'indiquer les dépendances absolues de votre projet. Ci-dessous, un exemple de composer.json indiquant une dépendance aux paquets Swiftmailer et Psr/log :

```
{
  "require": {
    "swiftmailer/swiftmailer": "v5.0.0",
    "psr/log": "1.0.0"
  }
}
```

require prend pour valeur une liste de paires dont la clé est le nom du paquet et la valeur, son numéro de version. Le nom du paquet se décompose lui-même en nom du fournisseur et nom du projet.

Le numéro de version utilisé peut renvoyer exactement à une version particulière, comme dans notre exemple, mais aussi à un ensemble de versions :

- soit sous la forme d'un intervalle : **>=1.0,<2.0**,
- soit sous la forme d'un astérisque générique qui autorisera toutes les versions vérifiant le motif : **1.***,
- soit encore sous la forme d'un tilde qui exprime l'idée que toutes les versions entre celle indiquée et la prochaine version majeure conviennent : **~1.2** autorise toutes les versions depuis la 1.2 incluse jusqu'à la 2.0 exclue.

require peut être utilisé pour exprimer des dépendances à d'autres éléments que des paquets, comme par exemple la version de PHP, la présence d'une extension ou la version d'une librairie. Ainsi, pour indiquer que votre paquet nécessite une version de PHP supérieure ou égale à 5.4 et la présence de l'extension GD, vous écrirez le fichier suivant :

```
{
  "require" : {
    "php": ">=5.4",
    "ext-gd": "*"
  }
}
```

2.3 Le fichier composer.lock

Celui-ci est apparu lorsque vous avez lancé l'installation de vos dépendances. Ne le retirez pas. Incluez-le dans votre contrôle de version et distribuez-le avec votre projet. En effet, alors que **composer.json** vous a permis une définition souple des versions requises, composer.lock réduit complètement cette souplesse pour lier votre projet aux versions particulières qui ont été installées. Ainsi, vous vous êtes assurés que les différentes instances de votre projet, qu'elles soient maintenues par vous ou par d'autres, utilisent les mêmes versions et que vous n'aurez pas à vous préoccuper de divergences de ce côté-là qui pourraient expliquer un bug qu'on rencontre ici et pas ailleurs.

2.4 Le Packagist

Le dépôt le plus connu de paquets pour Composer est le Packagist (https://packagist.org). Vous pouvez essayer de parcourir les paquets disponibles, mais l'interface ne s'y prête pas réellement. Sachez que les grands frameworks PHP, comme Zend Framework, Symfony ou encore Typo3 alimentent le Packagist. Le mieux est d'utiliser le moteur de recherche pour voir si les paquets auxquels vous pensez y sont disponibles. Ce qu'il contient ne se limite pas à des paquets PHP, mais vous y trouverez également des paquets JavaScript comme JQuery ou CSS comme Twitter Bootstrap.

Lorsque vous aurez trouvé ceux qui vous intéressent, il vous suffira de consulter la page associée pour voir les différentes versions disponibles, les informations liées, telles que le

mainteneur, le site officiel, la licence d'utilisation, etc., et même la valeur à utiliser pour la clé **required** dans votre fichier **composer.json**.

2.5 Autoloading

Un autre intérêt de Composer est qu'il permet d'unifier le fonctionnement du chargement automatique des classes. Les paquets qui le nécessitent contiennent une documentation qui permet à Composer de produire un fichier **vendor/autoload.php**. Ce dernier est particulièrement intéressant : il s'agit d'un autoloader de classes unifié pour tous les paquets qui vous permettra d'en utiliser toutes les classes sans avoir à les charger vous-même, écrire un autoloader adapté, ou encore charger l'autoloader de chacun des paquets.

Il vous suffira d'inclure ce fichier pour que le chargement de classes fonctionne avec tous les paquets installés par Composer :

```
require 'vendor/autoload.php';
```

3 Production

3.1 Écrire le Composer.json de son paquet

3.1.1 Configurer son paquet

Tout projet contenant un fichier **composer.json** est un paquet. Jusqu'ici, nous avons produit des paquets anonymes et donc inaptes à être distribués. Il faut seulement ajouter une ligne dans le fichier **composer.json** pour donner à notre paquet, juste à la ligne suivante l'ouverture de la première accolade, on ajoute :

```
"name" : "distributeur/nom-du-paquet",
```

L'ajout de cette simple ligne permettra à une instance de Composer de l'installer dans **vendor/distributeur/nom-du-paquet**. Naturellement, il nous faudra prendre en compte cette architecture dans notre développement. Il faut également que Composer sache où trouver le paquet, nous allons voir plus loin comment lui enseigner.

Il est possible d'indiquer le chargement automatique de classes. La méthode recommandée est de suivre la recommandation PSR-0 (**https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md**) pour le nommage et la structure de vos classes, répertoires et fichiers. Il vous faudra alors ajouter un champ à votre **composer.json** :

```
  "autoload": {
      "psr-0": {"distributeur": "vendor/"}
```

Dès lors, Composer enregistrera un autoloader pour le namespace **distributeur**.

Si vous n'avez pas suivi la recommandation PSR-0, il est possible de demander à Composer de générer un tableau classmap, qui sera ajouté à l'autoloader. Un tableau classmap est simplement un tableau de paires class/fichier utilisé par l'autoloader pour charger les classes requises. Pour que Composer génère ce tableau et le mette à disposition de l'autoloader, il faut ajouter une directive **autoload** au fichier **composer.json**, contenant une liste **classmap**. Cette liste contiendra tous les répertoires et les fichiers à scanner pour générer le tableau :

```
 "autoload": {
   "classmap": ["repertoire1/","repertoire2
/","fichier1.php","fichier2.inc"]
 }
```

Au moment de l'installation ou d'une mise à jour, Composer va scanner tous les fichiers indiqués, ainsi que tous les fichiers **.inc** ou **.php** présents dans les répertoires et relever l'emplacement de toutes les classes qu'il y trouvera, et produira le tableau correspondant dans le fichier **verdor/composer/autoload_classmap.php**.

Toutefois, si vous avez stocké toutes vos classes dans un seul répertoire, en utilisant ou pas les espaces de noms, vous pouvez simplement écrire une directive indiquant ce répertoire à l'autoloader de la façon suivante :

```
"autoload": {
  "psr-0": { "": "repertoire/" }
}
```

Après toute modification de la configuration de l'autoloading, il vous faudra ré-exécuter **composer** pour qu'il régénère le fichier gérant les chargements automatiques de classes.

3.1.2 Documenter son paquet

Dans la section précédente, nous avons vu comment écrire notre fichier composer.json avec tout juste le nécessaire pour qu'il fonctionne. Mais ce fichier peut être également utilisé pour présenter le paquet dans une interface web destinée à un véritable être humain, comme le Packagist ou Satis (voir plus loin). Il peut donc être utile d'ajouter dans ce fichier des informations complémentaires pour éclairer nos esprits. Pour cela, ce fichier accepte d'autres clés de premier niveau, dont voici les plus importantes :

- **description** : juste une ligne pour expliquer à quoi sert votre paquet ;
- **homepage** : l'URL présentant en détail le paquet ;
- **licence** : la licence adoptée par votre paquet ;
- **authors** : une liste d'éléments pouvant avoir eux-mêmes quatre clés (**name**, **email**, **homepage**, **role**) présentant les auteurs du paquet ;
- **support** : indication concernant les moyens d'obtenir un support, liste de clés pouvant être **email**, **issues**, **forum**, **wiki**, **irc** ;
- **conflict** : permet d'indiquer d'autres paquets qui entrent en conflit avec celui-ci ;
- **suggest** : indique des paquets qui, sans être requis, peuvent utilement compléter celui-ci.

3.2 Distribuer son paquet

La distribution de paquets présente deux aspects : la publication et l'utilisation. L'utilisation se fait par la déclaration de la dépendance dans le fichier **composer.json** comme nous l'avons vu, mais selon la méthode de publication utilisée, cette déclaration peut prendre des formes différentes.

3.2.1 Avec un dépôt VCS

Il est tout à fait possible de distribuer son paquet sans utiliser de dépôt Composer. Il sera toutefois utile d'avoir son propre dépôt de contrôle de version. Il est possible d'utiliser des dépôts Git, Subversion ou Mercurial, à condition d'avoir les clients correspondants installés. Il est également possible de récupérer des paquets depuis GitHub ou ButBucket sans avoir besoin de ces clients.

Dès lors, il suffit d'ajouter quelques lignes à votre **composer.json** pour pouvoir utiliser ce dépôt comme source d'installation de paquets. Il faut y ajouter la directive **repositories** qui est une liste de valeurs. Il est ainsi possible de définir plusieurs dépôts :

```
"repositories" : [
  {
    "type" : "vcs",
    "url" : "http://monserveur/mondepot"
  }
```

3.2.2 Avec le Packagist

Aujourd'hui, distribuer son paquet de façon à ce qu'il soit accessible à tous, cela veut dire le soumettre au Packagist. Cela est d'autant plus intéressant qu'une fois que vos paquets y seront enregistrés, vous n'aurez plus besoin de déclarer le dépôt à utiliser dans votre **composer.json**, vos paquets seront toujours accessibles. En effet, ce dépôt est actif par défaut pour Composer.

Note

Les mordus de sécurité apprécieront de pouvoir désactiver ce dépôt. Ils pourront le faire en ajoutant la directive :

```
"packagist": false
```

à la liste repositories.

Pour ce faire, il va d'abord vous falloir créer un compte sur le Packagist. Une fois cela fait, vous pourrez vous rendre sur la page de soumission à l'adresse https://packagist.org/packages/submit, et y indiquer l'URL publique de votre dépôt VCS que le Packagist va parcourir pour indexer vos paquets.

3.2.3 Avec un dépôt Satis

L'outil dédié à la distribution privée de vos paquets est Satis. Il vous permet en quelque sorte d'avoir votre propre Packagist à moindre coût. En effet, il génère un dépôt Composer statique, ce qui est plus économique qu'une version inutilement dynamique (vous ne distribuez pas plusieurs centaines de paquets par jour quand même ?).

Pour l'installer, utilisez la commande :

```
$ composer create-project composer/satis -stability=dev
```

Une fois cela fait, il vous faut fournir un nouveau fichier de configuration **satis.json** indiquant les dépôts VCS à parcourir pour indexer vos paquets. La syntaxe est identique à celle que nous avons vue jusqu'ici, et voici à quoi ressemble donc ce fichier :

```
{
  "name" : "TaoPHP Repository",
  "homepage" : "http://taophp.info",
  "repositories" : [
    { "type" : "vcs", "url": "http://git.taophp.info/taophp"}
  ],
  "require-all" : true
}
```

Dans cet exemple, tous les paquets seront indexés. Il est possible de limiter les paquets indexés en remplaçant la directive **require-all** par un bloc **require** comme suit :

```
  "require": {
    "taophp/core": "*",
    "taophp/model": "2.0"
  }
```

Ici, le paquet **taophp/core** sera indexé dans toutes ses versions et le paquet **taophp/model** seulement dans sa version 2.0.

Une fois votre fichier de configuration écrit, vous générez votre arborescence statique avec la commande :

```
$ php bin/satis build repertoire_de_destination
```

Vous obtiendrez alors dans **repertoire_de_destination** un site web statique présentant vos différents paquets en utilisant les informations que vous aurez bien voulu mettre dans les fichiers **composer.json**.

Conclusion

Avec Composer, PHP se dote enfin d'un outil sérieux pour gérer les dépendances de vos projets, que celles-ci soient internes ou publiques. Finis les téléchargements laborieux à chaque nouveau projet qui limitaient tant la possibilité de conjuguer les différents outils : Composer pourra rassembler toutes les briques pour vous. Voilà qui devrait vous faire gagner en souplesse. ■

TRAVAILLER EFFICACEMENT SUR DU CODE LEGACY

par Mathieu GANDIN [Développeur senior et Consultant chez OCTO Technology. Aime bien la Debian sur Raspberry Pi. Guitariste aussi..]

Travailler sur du code existant n'a jamais été une chose simple, surtout si ce code n'a pas été testé unitairement et ressemble plus à un vieux plat de spaghettis froids qu'à la dernière application à la mode. Nous parlons alors de legacy. Mais rassurez-vous, il existe des techniques pour reprendre ce code et pouvoir le changer à un rythme soutenable.

Concrètement, le code legacy c'est quoi ?

- Du code que vous n'avez pas écrit ;
- Du code sans test ;
- Du vieux code ;
- Du code écrit dans n'importe quel langage existant ;
- Du code qui n'est plus supporté ;
- Du code avec de vieilles technos dedans ;
- Du code de la version précédente.

Ça vous semble familier ? C'est le genre de code sur lequel nous nous retrouvons souvent à développer quand nous avons trop laissé grossir la dette technique.

Nous parlons de dette technique quand nous choisissons, temporairement, de baisser le niveau de qualité de notre code pour produire plus rapidement de la fonctionnalité. S'endetter techniquement de temps en temps ne représente, en général, pas un problème si nous faisons ensuite l'effort de refactorer notre code.

Ça devient embêtant quand nous passons notre temps à nous endetter en permanence et quand nous oublions de reprendre notre applicatif. À son optimum, la dette technique nous fait perdre en efficacité et en productivité de développement, car notre application est devenue difficilement maintenable.

Alors comment allons-nous nous y prendre ? Essayons sur un exemple simple avant d'aller plus loin.

Considérons l'application suivante, qui a pour objectif d'exposer une API REST (au format XML). Elle propose un service qui restitue le résultat annuel d'une organisation. Voici son architecture (figure 1).

Nous souhaitons apporter deux évolutions sur cette API :

- Afficher l'année et le turnover pour chaque résultat ;
- Exposer un nouveau service REST, cette fois-ci au format JSON, afin de développer un nouveau front-end en HTML5, avec *Twitter Bootstrap* et *Backbone.js*, pour le rendre plus attractif que l'actuel qui est en production.

Le code à refactorer se trouve ici sur Github : github/octomga/indianajones

Nous commençons par afficher l'année et le turnover dans l'API existante. Pour prendre en compte cette évolution et pour améliorer le code, nous allons développer un test unitaire sur le point d'entrée de notre composant (comme nous conseille **Michael Feathers** dans son livre « *Working Effectively With Legacy Code* ».)

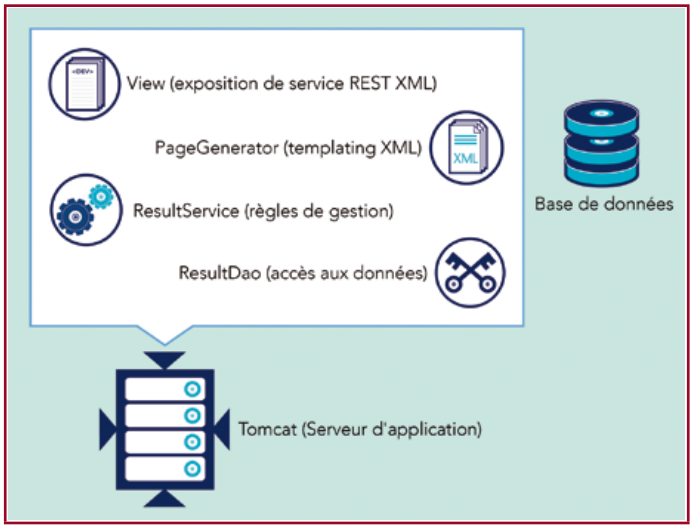


Figure 1

Ici, il s'agit de s'attaquer à la classe **PageGenerator** qui traverse toute notre application pour générer le code XML à afficher.

Nous faisons ensuite apparaître dans le test les évolutions que nous souhaitons apporter à notre code (afficher l'année et le turnover).

```
@Test public void should_generate_xml_report_integration() {
  String expectedXml = "<report>\n" +
  "\t<title>Quaterly Report</title>\n" +
  "\t<frame>2012 - 2012</frame>\n" +
  "\t<results>\n" +
  "\t\t<result>\n" +
  "\t\t\t<lob>UX</lob>\n" +
  "\t\t\t<manager>Bob</manager>\n" +
  "\t\t\t<net>40.0</net>\n" +
  "\t\t\t<alertNet>Alert : Net Profit too low</alertNet>\n" +
  "\t\t\t<operatingExpense>80.0</operatingExpense>\n" +
  "\t\t\t<alertExpense>Alert : Too much notes</alertExpense>\n" +
  "\t\t\t<year>2012</year>\n" +
  "\t\t\t<turnover>11</turnover>\n" +
  "\t\t</result>\n" +
  "\t\t<result>\n" +
  "\t\t\t<lob>Media</lob>\n" +
  "\t\t\t<manager>Bob</manager>\n" +
  "\t\t\t<net>80.0</net>\n" +
  "\t\t\t<operatingExpense>40.0</operatingExpense>\n" +
  "\t\t\t<year>2012</year>\n" +
  "\t\t\t<turnover>9</turnover>\n" +
  "\t\t</result>\n" +
  ... // Plein d'autres résultats
  "\t</results>\n" +
  "</report>";
  PageGenerator pageGenerator = new PageGenerator();
  String xml = pageGenerator.generate();
  Assert.assertEquals(expectedXml,xml);
}
```

Ce premier test - qui est avant tout un test d'intégration plus qu'un test unitaire - nous permet d'implémenter le code à rajouter dans les classes **PageGenerator**, **Result** et **ResultDao** pour remonter l'année et le turnover.

```
pageXml += "\t\t\t<year>" + result.getYear() + "</year>\n";
pageXml += "\t\t\t<turnover>" + result.getTurnover() + "</turnover>\n";
```

Nous relançons les tests, ils sont tous verts. Nous déployons l'application et la fonctionnalité marche. Il est temps de refactorer le code pour rendre ça plus propre.

Nous allons rendre plus modulaire la classe **PageGenerator**. Les principes SOLID de programmation orientée-objet sont de bons moyens pour découpler ce code. Dans notre cas, nous allons utiliser l'injection de dépendance, notamment avec **ResultService**.

Nous allons ensuite utiliser un framework de *mock* permettant de simuler le comportement de **ResultService**, ceci permettra de rendre le test sur la classe **PageGenerator** indépendant de la base de données (pour cela, nous allons utiliser le **framework Mockito**). Voici la nouvelle méthode de test :

```
@Test public void should_generate_xml() {
   String expectedXml = "<report>\n" +
   "\t<title>Quaterly Report</title>\n" +
   "\t<frame>2013 - 2013</frame>\n" +
   "\t<results>\n" +
   "\t\t<result>\n" +
   "\t\t\t<lob>Media</lob>\n" +
   "\t\t\t<manager>Mathieu</manager>\n" +
   "\t\t\t<net>40.0</net>\n" +
   "\t\t\t<operatingExpense>40.0</operatingExpense>\n" +
   "\t\t\t<year>2013</year>\n" +
   "\t\t\t<turnover>9</turnover>\n" +
   "\t\t</result>\n" +
   "\t</results>\n" +
   "</report>";

   BDDMockito.given(resultService.getResults()).
      willReturn(ResultFixture.generateResults());

   BDDMockito.given(resultService.getFrame()).willReturn("2013 - 2013");
   PageGenerator pageGenerator = new PageGenerator(resultService);
   String xml = pageGenerator.generate();
}
```

Il est temps de faire un nouveau refactoring. Nous sortons l'initialisation du **ResultService** hardcodée dans la méthode **generate()** vers un premier constructeur. Ce constructeur reste là pour ne pas impacter d'un coup tous les vieux composants qui utilisent la classe **PageGenerator**.

Pour indiquer que ce constructeur ne doit plus être utilisé et qu'il va falloir prendre le nouveau, nous l'annotons avec **@Deprecated**. Nous rajoutons notre nouveau code avec le second constructeur permettant de faire de l'injection de dépendance.

```
private ResultService resultService;

public PageGenerator(ResultService resultService) {
  this.resultService = resultService;
}

@Deprecated
public PageGenerator() {
  resultService = new ResultService();
}

public String generate() {
  List<Result> results = resultService.getResults();
...
```

Ensuite, il est temps de nettoyer la méthode **generate()** qui est bien trop grosse.

```
public String generate() {
  List<Result> results = resultService.getResults();
  String pageXml = new String();
  pageXml += "<report>\n";
  pageXml += "\t<title>";
  pageXml += "Quaterly Report";
  pageXml += "</title>\n";
  pageXml += "\t<frame>";
  pageXml += resultService.getFrame();
  pageXml += "</frame>\n";
  if (results.size() == 0) {
    pageXml += "\t<message>" + resultService.getMessage() + "</message>\n";
  } else {
    pageXml += "\t<results>\n";
    for (Result result : results) {
      pageXml += "\t\t<result>\n";
      pageXml += "\t\t\t<lob>" + result.getDepartement() + "</lob>\n";
      pageXml += "\t\t\t<manager>" + result.getManager() + "</manager>\n";
      pageXml += "\t\t\t<net>" + (result.getNetProfit() / 100) + "</net>\n";
      if(result.getUnderKpiMessage()) {
        pageXml += "\t\t\t<alertNet>Alert : Net Profit too low</alertNet>\n";
      }
```

```
      pageXml += "\t\t\t<operatingExpense>" + (result.getOperatingExpense() /
100)
  + "</operatingExpense>\n";
      if (result.getTooMuchExpenseMessage()) {
        pageXml += "\t\t\t<alertExpense>Alert : Too much notes</alertExpense>\n";
      }
      pageXml += "\t\t\t<year>" + result.getYear() + "</year>\n";
      pageXml += "\t\t\t<turnover>" + result.getTurnover() + "</turnover>\n";
      pageXml += "\t\t</result>\n";
    }
    pageXml += "\t</results>\n";
  }
  pageXml += "</report>";
  return pageXml;
}
```

Pour cela, nous appliquons le pattern « *Extract Method* » présenté par **Kent Beck** dans le livre « *Refactoring* » écrit par **Martin Fowler**. Cette pratique nous permet de découper une grosse méthode en plusieurs qui sont plus petites et plus faciles à maintenir. Le nommage de ses nouvelles méthodes permet aussi d'améliorer la lisibilité et l'intention du code.

Enfin, ce sont des petits refactorings qui ne coûtent pas grand-chose – ils sont automatisables avec un bon éditeur de code – et ils permettent de préparer des remaniements de code plus importants. Pour résumer, ce sont les petits refactorings qui préparent les plus grands.

```
public String generate() {
  List<Result> results = resultService.getResults();
  String pageXml = new String();
  pageXml = generateReportHeader(pageXml);
  if (results.size() == 0) {
    pageXml = generateNoResultMessage(pageXml);
  } else {
    pageXml = generateResultsHeader(pageXml);
    for (Result result : results) {
      pageXml = generateHeader(pageXml, result);
    }
    pageXml = generateResultsFooter(pageXml);
  }
  pageXml = generateFooter(pageXml);
  return pageXml;
}
```

Nous redéployons l'application, tout fonctionne. Bravo ! Les nouvelles fonctionnalités sont là et le code est dans un meilleur état que quand nous l'avons trouvé. Une partie de la dette technique vient d'être remboursée.

Maintenant il est temps d'implémenter la seconde fonctionnalité, exposer le même service en JSON pour que nous puissions développer une jolie interface avec notre nouvelle API.

Nous allons prendre cette fonctionnalité comme une belle opportunité pour rafraîchir notre *legacy* et créer une sorte de « bulle » de code propre qui va cohabiter avec le reste. Progressivement, au cours de l'évolution de l'application, nous pourrons y remanier de l'ancien code vers le nouveau, comme une sorte de vase communiquant.

Nous disposons donc d'un contrôleur qui expose les mêmes résultats au format JSON, il a été implémenté avec **Spring MVC** et **JAXON**. Pour le moment, le code renvoie des résultats codés en dur, il faut maintenant utiliser le composant **`ResultService`**.

Nous allons changer le code en utilisant l'injection de dépendance entre la classe **`ResultService`** et **`ResultController`**. Nous en profitons aussi pour reprendre nos pratiques de *Mocks* afin d'implémenter notre test et nous abstraire d'une partie de l'environnement.

```
@RunWith(MockitoJUnitRunner.class)
public class ResultControllerTest {
  @Mock ResultService resultService;

@Test public void should_generate_json() {
   BDDMockito.given(resultService.getResults()).
    willReturn(ResultFixture.generateResults());

   ResultController resultController = new ResultController();
   resultController.setResultService(resultService);
   List<Result> results = resultController.generateJson();
   Assert.assertEquals(ResultFixture.generateResults(),results);
  }
}
```

Le contrôleur reste simple, il se contente juste d'exposer un service REST au format JSON et délègue les traitements à la classe **`ResultService`**.

```
@Controller
@RequestMapping("/results")
public class ResultController {
  private ResultService resultService;
  @Autowired
  public ResultController (ResultService resultService) {
    this.resultService = resultService;
  }

  @RequestMapping(method = RequestMethod.GET)
  public @ResponseBody List<Result> generateJson() {
    return resultService.getResults();
  }

  public void setResultService(ResultService resultService) {
    this.resultService = resultService;
  }
}
```

Nous redéployons. L'application fonctionne avec notre jeu de données, mais il manque encore les messages d'alerte.

Voici ce que nous obtenons :

```
[{"year":2012,"manager":"Bob","departement":"UX","netProfit":4000.0,"
operatingExpense":8000.0,"underKpiMessage":true,"tooMuchExpenseMessag
e":true,"turnover":11]
```

Et voici le résultat que nous attendons.

```
[{"year":2012,"manager":"Bob","other":true,"departement":"UX","netP
rofit":4000.0,"operatingExpense":8000.0,"underKpi":true,"hasTooMuchE
xpense":true,"turnover":11,"tooMuchExpenseMessage":"Alert : Too much
notes","underKpiMessage":"Alert : Net Profit too low"}]
```

Pour y arriver, il va falloir détricoter les classes **`PageGenerator`** et **`ResultService`**, car si nous relisons le code nous nous rendons compte du problème suivant : **`ResultService`** calcule les seuils d'alerte de chaque résultat et positionne les booléens **`underKpiMessage`** et **`tooMuchExpenseMessage`** de la classe **`Result`** à **`true`**. **`PageGenerator`** affiche ensuite les messages d'alerte si ces booléens sont à **`true`**.

Bref, c'est le plat de spaghettis. Si nous gardons le code dans cet état, nous sommes partis pour faire de la duplication.

Pour nous en sortir, nous allons déplacer la méthode **generateResult()** de **PageGenerator** vers **Result**. Cette classe étant utilisée par **PageGenerator** et **ResultController**, nous avons la possibilité de faire un peu de réutilisation de code. Mais ça n'est pas encore suffisant, il y a encore des choses à faire.

```
public class Result {
...
  public String generateResult() {
    String pageXml = "\t\t<result>\n";
    pageXml += "\t\t\t<lob>" + getDepartement() + "</lob>\n";
    pageXml += "\t\t\t<manager>" + getManager() + "</manager>\n";
    pageXml += "\t\t\t<net>" + (getNetProfit() / 100) + "</net>\n";
    if(getUnderKpiMessage()) {
      pageXml += "\t\t\t<alertNet>Alert : Net Profit too low</alertNet>\n";
    }
    pageXml += "\t\t\t<operatingExpense>" + (getOperatingExpense() / 100)
      + "</operatingExpense>\n";
    if (getTooMuchExpenseMessage()) {
      pageXml += "\t\t\t<alertExpense>Alert : Too much notes</alertExpense>\n";
     }
     pageXml += "\t\t\t<year>" + getYear() + "</year>\n";
     pageXml += "\t\t\t<turnover>" + getTurnover() + "</turnover>\n";
     pageXml += "\t\t</result>\n";
     return pageXml;
  }
}
```

Maintenant, nous allons nous attaquer à la classe **ResultService** en déplaçant du code de traitement vers la classe **Result**.

Mais nous avons un premier problème, **ResultService** est hardcodé avec **ResultDao** qui ne propose qu'une méthode de classe **static**, ce qui ne simplifie pas la modularité du code.

```
public class ResultService {
  private String frame;

  public List<Result> getResults() {
    List<Result> resultList = ResultDao.query();
...
```

Les méthodes de classe **static** sont enquiquinantes et un bon moyen de s'en débarrasser consiste à les déprécier et ensuite à appliquer le pattern « *Extract Method* » pour déplacer toute son implémentation vers une méthode d'instance de classe (que nous appelons **findAll()** ce qui est plus explicite).

```
public class ResultDao {
  @Deprecated
  public static List<Result> query() {
    ResultDao resultDao = new ResultDao();
    return resultDao.findAll();
  }
```

Le code continue de fonctionner comme avant, et maintenant nous pouvons refactorer **ResultService** pour le découpler du **ResultDao**.

Pour cela, nous devons utiliser un jeu de données qui nous permet de faire apparaître les messages d'alerte.

Nous disposons de la classe **ResultFixture** qui contient des jeux de données pour nos tests. Nous allons lui rajouter la méthode **generateResultsWithAlertMessage()** avec les données adéquates pour la suite.

```
public static List<Result> generateResultsWithAlertMessage() {
  Result result = new Result();
  result.setNetProfit(40.0);
  result.setOperatingExpense(400000.0);
  result.setManager("Mathieu");
  result.setDepartement("Media");
  result.setTurnover(9);
  result.setYear(2013);
  List<Result> results = new ArrayList<Result>();
  results.add(result);
  return results;
}
```

Nous bouchonnons ensuite la classe **ResultDao** avec notre framework de Mock et nous utilisons l'injection de dépendance pour découpler les classes **ResultService** et **ResultDao**.

```
@RunWith(MockitoJUnitRunner.class)
public class ResultServiceTest {
  @Mock ResultDao resultDao;

  @Test public void should_compute_result_from_db() {
    BDDMockito.given(resultDao.findAll()).
      willReturn(ResultFixture.generateResultsWithAlertMessage());

    ResultService resultService = new ResultService(resultDao);
    List<Result> results = resultService.getResults();
    Assert.assertEquals(ResultFixture.generateResultsWithAlertMessag
e(),results);
  }
}
```

Maintenant que le code est couvert par un test, nous disposons d'un harnais de sécurité pour refactorer en profondeur cette classe. Dans la classe **ResultService**, nous commençons par remplacer l'appel de la méthode de classe **query()** de **ResultDao** par un appel sur la méthode d'instance **findAll()**.

Comme pour la classe **PageGenerator**, nous gardons un vieux constructeur pour ne pas impacter d'un coup tous les vieux composants qui utilisent la classe **ResultService**. Pour indiquer que ce constructeur ne doit plus être utilisé et qu'il va falloir prendre le nouveau, on l'annote avec **@Deprecated**.

Nous rajoutons notre second constructeur permettant d'implémenter l'injection de dépendance.

```
public class ResultService {

  private ResultDao resultDao;
  ...

  @Autowired
  public ResultService (ResultDao resultDao) {
    this. resultDao = resultDao;
  }

  @Deprecated
  public ResultService () {
    this. resultDao = new ResultDao();
  }

  public List<Result> getResults() {
    List<Result> resultList = resultDao.findAll();
...
```

Ensuite, nous constatons de la duplication de code. Pour chaque département, nous vérifions les dépenses et les gains de chaque résultat, nous pouvons refactorer cette règle de gestion dans une seule méthode dédiée (encore une fois, nous appliquons le pattern « *extract method* ».)

```
public List<Result> getResults() {
  List<Result> resultList = resultDao.findAll();
  List<Result> results = new ArrayList<Result>();
  if(resultList.size() != 0) {
    int maxYear = 0;
    int minYear = 9999;
    for (int i = 0; i < resultList.size(); i++) {
      Result result = resultList.get(i);
      if (result.getDepartement() != "Media" && result.
getDepartement() != "Bank" && result.getDepartement() != "Indus") {
        handleProfit(result,5000);
        handleExpense(result,5000);
        results.add(result);
    } else {
...
}

public void handleExpense(Result result,int maxExpense) {
  if(result.getOperatingExpense() >= maxExpense) {
    result.setTooMuchExpenseMessage(true);
  }
 }

public void handleProfit(Result result,int minProfit) {
  if(result.getNetProfit() < minProfit) {
    result.setUnderKpiMessage(true);
  }
}
```

Ce refactoring nous permet d'identifier que ces deux méthodes n'ont pas besoin d'être dans **ResultService**. Nous les déplaçons vers la classe **Result** qui semble la plus appropriée pour porter ce code.

Nous constatons aussi que les messages d'alerte que nous souhaitons réutiliser dans le JSON sont coincés au milieu d'une chaîne de caractères, il est temps de les déplacer vers une constante.

```
public String generateResult() {
  String pageXml = "\t\t<result>\n";
...
  if(getUnderKpiMessage()) {
    pageXml += "\t\t\t<alertNet>" + ALERT_NET_PROFIT_TOO_LOW + "</alertNet>\n";
  }
  pageXml += "\t\t\t<operatingExpense>" + (getOperatingExpense() / 100)
    + "</operatingExpense>\n";
  if (getTooMuchExpenseMessage()) {
    pageXml += "\t\t\t<alertExpense>" + ALERT_TOO_MUCH_NOTES + "</
alertExpense>\n";
  }
...
  return pageXml;
}
```

Maintenant, pour plus de lisibilité, nous allons renommer les deux booléens **underKpiMessage** et **tooMuchExpenseMessage** par **isUnderKpi** et **hasTooMuchExpense** qui sont plus explicites.

Enfin, nous pouvons rajouter nos deux chaînes de caractères qui seront ensuite utilisées pour afficher les messages d'alerte.

```
public class Result {
  public static final String ALERT_NET_PROFIT_TOO_LOW = "Alert : Net Profit too
low";
  public static final String ALERT_TOO_MUCH_NOTES = "Alert : Too much notes";
...
  private boolean isUnderKpi;
  private boolean hasTooMuchExpense;
...
  private String underKpiMessage;
  private String tooMuchExpenseMessage;
```

Il faut maintenant que ces messages d'alerte soient positionnés dans les méthodes **handleExpense()** et **handleProfit()**.

```
public void handleExpense(int maxExpense) {
  if(getOperatingExpense() >= maxExpense) {
    setHasTooMuchExpense(true);
    setTooMuchExpenseMessage(ALERT_TOO_MUCH_NOTES);
  }
}

public void handleProfit(int minProfit) {
  if(getNetProfit() < minProfit) {
    setUnderKpi(true);
    setUnderKpiMessage(ALERT_NET_PROFIT_TOO_LOW);
  }
}
```

C'est mieux, nous repassons tous les tests, ils sont verts. Nous redéployons l'application et tout fonctionne.

Bravo ! Nous avons encore remboursé une partie de notre dette technique, et nous avons pu faire un changement plus important dans le code grâce à l'ensemble de nos refactorings.

Il reste encore plein de choses à faire : par exemple une arborescence de classes pour gérer les résultats par département (**MediaResult**, **BankResult**, **IndusResult** qui hérite de **Result**) et puis le **ResultDao** mériterait d'être revu (il ouvre une connexion à la base de données lors de chaque requête), mais maintenant vous savez comment faire, non ?

Travailler sur du code *legacy* n'est pas une fatalité, nous pouvons toujours rembourser la dette technique en commençant à écrire un premier harnais de tests unitaires et en refactorant ce code pour le rendre plus modulaire.

Pour cela, il vous faudra :

- Identifier les endroits à changer dans votre code (c'est facile, il suffit de prendre les demandes d'évolutions fonctionnelles de l'application) ;
- Écrire un test sur le composant de plus haut niveau (généralement, il s'agit d'écrire un test d'intégration sur la couche de service de plus haut niveau de votre application) ;
- Implémenter l'évolution demandée ;
- Refactorer votre code pour le rendre plus modulaire ;
- Recommencer.

Avec un peu de méthode et de pratique, vous voilà prêt à reprendre du code *legacy*. ■

DES LOGICIELS

LINAGORA

FRANÇAIS ET LIBRES

CHOISIR LINAGORA C'EST :

■ CHOISIR DES LOGICIELS LIBRES QUI **AUGMENTERONT LA PERFORMANCE** DE VOS ÉQUIPES ET DE VOTRE SI. LINAGORA ÉDITE ELLE-MÊME PLUSIEURS LOGICIELS D'ENTREPRISE, CONÇUS POUR FACILITER ET SÉCURISER LES ÉCHANGES COLLABORATIFS. **OBM**, SOLUTION UNIFIÉE DE **MESSAGERIE, ANNUAIRE, AGENDA ET GESTION DE TEMPS, LINSHARE** POUR UN **PARTAGE DE DOCUMENTS** FACILE ET CONTRÔLÉ, **LINID** POUR UNE **AUTHENTIFICATION** ET DES **ÉCHANGES SÉCURISÉS,** ET **PETALS** POUR FLEXIBILISER VOTRE SYSTÈME D'INFORMATION ;

■ BÉNÉFICIER D'UNE **EXPERTISE TECHNIQUE UNIQUE** : DEPUIS SA CRÉATION, LINAGORA **S'IMPLIQUE DANS LES COMMUNAUTÉS** DU LOGICIEL LIBRE ET **BÂTIT** AINSI UNE **EXPERTISE RICHE ET DIVERSIFIÉE**. AUJOURD'HUI, NOTRE OFFRE DE **SUPPORT OSSA** COUVRE PLUS DE **400 LOGICIELS LIBRES**. VOTRE PROJET COMBINE DIX LOGICIELS LIBRES À CONFIGURER, SUPPORTER, MODIFIER ? **LINAGORA EST LÀ POUR VOUS APPORTER UNE RÉPONSE SIMPLE, RAPIDE ET FIABLE** ;

■ FAIRE LE MÊME CHOIX QUE DE GRANDS CLIENTS QUI ONT OPTÉ POUR DES **SOLUTIONS INDUSTRIELLEMENT ÉPROUVÉES** : LEADER FRANÇAIS DE L'ÉDITION DE LOGICIELS LIBRES ET DE SERVICES ASSOCIÉS, **LINAGORA MET EN ŒUVRE, DEPUIS PLUS DE DIX ANS, LES STRATÉGIES « OPEN SOURCE » DE GRANDES ADMINISTRATIONS ET ENTREPRISES FRANÇAISES** : GRANDS MINISTÈRES, GENDARMERIE NATIONALE, ASSEMBLÉE NATIONALE, RÉGION ÎLE-DE-FRANCE, BOUYGUES TELECOM, AIR FRANCE, AREVA... ;

■ S'ASSURER UNE **PÉRENNITÉ TECHNOLOGIQUE : LINAGORA EST LE PREMIER INVESTISSEUR PRIVÉ EN FRANCE DANS LA R&D OPEN SOURCE**. GRÂCE À SES ÉQUIPES IMPLIQUÉES DANS D'IMPORTANTS PROJETS DE RECHERCHE FRANÇAIS ET EUROPÉENS, LINAGORA EST À LA POINTE DE L'INNOVATION ET EN FAIT BÉNÉFICIER SES CLIENTS ;

■ NOS LOGICIELS, NOTAMMENT GRÂCE AUX « INVESTISSEMENTS D'AVENIR », BÉNÉFICIENT D'**UNE ROAP MAP TECHNOLOGIQUE AMBITIEUSE** ET S'INSCRIVENT DANS UNE VISION LONG TERME QUI **PÉRENNISE VOTRE INVESTISSEMENT** ;

■ SOUTENIR **UN ESPRIT AUTHENTIQUEMENT LIBRE** : L'ENSEMBLE DE NOS CORRECTIONS SONT REVERSÉES AUX COMMUNAUTÉS CONCERNÉES. DE MÊME, NOS PROPRES LOGICIELS SONT INTÉGRALEMENT FOURNIS SOUS LICENCE LIBRE. **AVEC NOUS, PAS DE FRAIS CACHÉS OU DE SURPRISES SUR LES FONCTIONNALITÉS : UNE SEULE VERSION, 100 % FONCTIONNELLE, 100 % LIBRE** ;

■ **ŒUVRER EN FAVEUR DE LA SOUVERAINETÉ ET DU PATRIOTISME NUMÉRIQUES EN ADOPTANT DES LOGICIELS DÉVELOPPÉS EN FRANCE PAR UNE ENTREPRISE FRANÇAISE.**

CHOISIR LINAGORA, C'EST COMBINER LE MEILLEUR DU LOGICIEL LIBRE AVEC LE MEILLEUR DES PARTENAIRES !

FAITES LE CHOIX DE VRAIS LOGICIELS LIBRES & GRATUITS

OBM.org — COMMUNICATION & COLLABORATION

LinShare — PARTAGE DE FICHIERS ET COFFRE-FORT ÉLECTRONIQUE

LinID — GESTION ET FÉDÉRATION DES IDENTITÉS, SSO

petals — ESB ET SOA

www.LINAGORA.com