N°191 mars 2016

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

PYTHON / NPYSCREEN

Créez simplement des interfaces textuelles p.58

GOOGLE CARDBOARD / JAVASCRIPT

RÉALITÉ VIRTUELLE

CRÉEZ VOS PROPRES MONDES AVEC A-FRAME ! p.06

PHOTOGRAMMÉTRIE / MICMAC

Étudiez le traitement d'images acquises par microdrone p.48

CRYPTOGRAPHIE / SRP

Sécurisez vos authentifications Web par mot de passe p.72

NGINX / PHP

Découvrez Docker Compose pour orchestrer vos conteneurs Docker p.30

MYSQL / CLUSTER

Automatisez la bascule maître-esclave avec mysqlfailover p.24

PYTHON / UNITTEST

Validez vos programmes à l'aide des tests unitaires p.36

ET AUSSI : Bibliothèques statiques/dynamiques - Wildfly Swarm - Bottle, un module Python pour le Web

L 19275 - 191 - F: 7,90 € - RD

France MÉTRO. : 7,90 € – CH : 13 CHF – BEL/LUX/PORT.CONT : 8,90 € – DOM/TOM : 8,50 € – CAN : 14 $ CAD

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

sales@ikoula.com
01 84 01 02 66
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 58
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz

GNU/Linux Magazine France
est édité par Les Éditions Diamond

10, Place de la Cathédrale - 68000 Colmar - France
Tél. : 03 67 10 00 20 – **Fax :** 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Responsable service Infographie : Kathrin Scali
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 – v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne

Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04

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

Périodicité : Mensuel
Prix de vente : 7,90 €

Visuel Google Cardboard sur la couverture :
Auteur : othree. Photo distribuée sous licence CC-by-2.0 :
https://commons.wikimedia.org/wiki/File:Assembled_Google_Cardboard_VR_mount.jpg

SUIVEZ-NOUS SUR :

https://www.facebook.com/editionsdiamond

@gnulinuxmag

LES ABONNEMENTS ET LES ANCIENS NUMÉROS SONT DISPONIBLES !

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

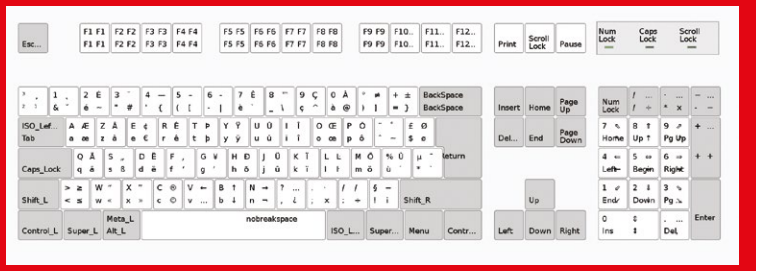
Codes sources sur https://github.com/glmf

ÉDITORIAL

Connaissez-vous la Délégation générale à la langue française et aux langues de France (la DGLFLF de son petit nom) ? Non ? Alors pour votre plus grand bonheur, vous allez apprendre qu'il s'agit d'un service directement rattaché au ministre chargé de la culture et qu'il est composé d'une trentaine de personnes. Son objectif est, je cite, « d'animer et de coordonner la politique linguistique du Gouvernement et d'orienter son évolution dans un sens favorable au maintien de la cohésion sociale et à la prise en compte de la diversité de notre société ». Si j'ai décidé de vous parler de cette instance ce mois-ci, c'est suite à la publication de l'article « Vers une norme française pour les claviers informatiques », paru sur le site du Ministère de la Culture [1]. En résumé, ce texte nous explique qu'il est « presque impossible d'écrire en français correctement avec un clavier commercialisé en France ». Ainsi on ne peut pas (ou très difficilement si je me réfère à l'atténuation impliquée par l'usage du mot « presque ») utiliser :

- des caractères majuscules accentués tels que « É » ou « Ç » ;
- des guillemets français doubles en forme de chevrons c'est-à-dire « ... » en lieu et place des guillemets droits " ... " ;
- les ligatures avec « æ » (« e » dans « a ») et « œ » (« e » dans « o ») et bien entendu leurs variantes en caractères majuscules « Æ » et « Œ ».

Mais par quel miracle ai-je pu donc écrire tous les caractères qui précèdent ? Peut-être simplement en ayant correctement configuré mon clavier ? Je ne m'étais jusqu'ici jamais trop posé la question et j'ai donc découvert qu'il s'agit de la configuration « Français Français (variante) » :



Les différents caractères sont accessibles soit en appuyant simplement sur la touche, soit en la combinant avec la touche de verrouillage majuscule, avec <AltGr> ou avec <AltGr> + <Shift>. On peut donc en conclure que les éminents membres de cette institution n'ont pas configuré leur clavier correctement et que plutôt que de suivre une formation en informatique ou de s'informer auprès d'une personne compétente, ils ont préféré lancer un nouveau projet normatif auprès de l'AFNOR [2].

Plus grave, il paraît indispensable à la DGLFLF que ce nouveau clavier permette une utilisation aisée des différentes langues présentes sur notre territoire (langues régionales et étrangères), car on doit pouvoir recourir à plusieurs langues au sein d'un même document. Je me suis donc renseigné sur les langues régionales et étrangères présentes sur notre territoire en me basant sur les informations données par la DGLFLF :

- langues régionales [3] : alsacien, basque, breton, catalan, corse, flamand occidental, franco-provençal, langues d'oil, langue d'oc, parlers liguriens ;
- langues des Outre-mer [4] : créoles (guadeloupéen, martiniquais, guyanais, réunionnais), kanak, tahitien, etc. (une cinquantaine de langues au total) ;

suite page 4...

- langues non-territoriales **[5]** : arabe dialectal, arménien occidental, berbère, judéo-espagnol, romani et yiddish.

On peut donc commencer à imaginer à quoi pourrait ressembler la touche A (en se limitant quand même dans le nombre d'alphabets).

Pour accéder sur une même touche aux différents alphabets, il faudra alors utiliser des combinaisons de touches... Quel sera donc l'intérêt par rapport à ce que l'on peut faire actuellement ? Et que dire des symboles mathématiques ? Ils ne constituent pas une langue, mais il est fréquent de devoir utiliser des caractères du grec ancien. Il faudrait donc ajouter également cet alphabet et un ensemble assez conséquent de symboles. Pour la DGLFLF, les caractères « utiles » seraient « ≤ », « ≥ » et « ‰ ». Je propose donc de supprimer aussi les caractères « inutiles » tels que le *pipe* « | » (d'ailleurs sur un clavier standard de Mac, il faut un minimum de trois mains pour réaliser la combinaison de touches permettant d'accéder à ce caractère).

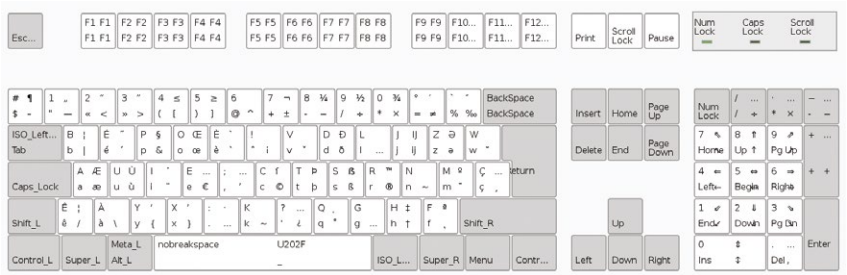
L'article **[1]** résume en fait la section II « Vers une norme française pour les claviers informatiques ? » du rapport au parlement sur l'emploi de la langue française **[6]**. Dans ce rapport, on est rassuré d'apprendre que sur un « système Mac » les utilisateurs « chevronnés » seront capables d'accentuer une lettre majuscule à l'aide de la touche de verrouillage majuscule. Ouf ! Donc dans la vraie vie, il y a les utilisateurs de Windows qui ne peuvent pas écrire correctement en français et les utilisateurs de Mac chevronnés qui y parviendront. GNU/Linux ça vous dit quelque chose ? Le noyau de Mac OS X n'est-il pas basé sur UNIX ? Au lieu de se dire que le problème vient du système d'exploitation ou de la méconnaissance des utilisateurs, lançons un vaste projet de création d'une nouvelle norme, c'est bien plus efficace !

Il faut noter que l'AFNOR se base apparemment sur ce rapport et non sur la déclaration de l'article **[1]** puisque son projet de normalisation se borne aux langues d'écriture latine. Je vous rappelle d'ailleurs à ce propos qu'il existe déjà une disposition de clavier adaptée à notre langue : le Bépo.

Par contre, dans le contexte d'une utilisation multilingue, l'usage de cette disposition ne sera plus adapté. Les mots-clés des langages sont tirés de l'anglais et il serait donc plus intéressant d'utiliser un clavier Dvorak, car ces dispositions ont été pensées aussi pour minimiser la fatigue musculaire. Cela revient donc à modifier perpétuellement la disposition des touches ce qui, vous en conviendrez, ne serait guère efficace en terme de vitesse de frappe. Sans compter les problèmes inhérents à un changement de clavier (pour aider un collègue, pour les enseignants ou formateurs passant de machine en machine, etc.). Avez-vous remarqué le nombre de fautes de frappe que l'on fait lorsque l'on passe d'un clavier Azerty à un autre et que les touches sont décalées de quelques millimètres ? Imaginez donc avec des lettres se trouvant sur des touches « mouvantes » dont la disposition est complètement modifiée d'un clavier à l'autre.

Pour finir, en résumant tout cela à l'extrême, on en déduit qu'il est difficile d'écrire correctement en français sous Windows. Pour rappel, le ministère de l'Éducation nationale a signé un partenariat avec Microsoft et c'est bien ce ministère qui veille aux programmes appliqués dans les écoles, collèges et lycées, lesquels programmes contiennent également un apprentissage de la langue française... cherchez l'erreur ! Quant à nous, tant que nos claviers nous le permettent, nous pourrons continuer à administrer des serveurs, développer des applications et tester les techniques proposées dans les pages qui suivent. Je vous souhaite une bonne lecture...

Il y a deux mois nous nous insurgions de la signature du partenariat entre le ministère de l'Éducation nationale et Microsoft et nous apprenons donc maintenant que ce partenariat qui revient à « vendre » nos enfants à Microsoft et à ses suites logicielles ne leur permettra pas d'écrire en français alors que des solutions libres et fonctionnelles existent ! Dernièrement, les amendements déposés pour donner la priorité au logiciel libre dans le secteur public se sont transformés en un « encouragement » à l'utilisation des logiciels libres et des formats ouverts lors du développement, de l'achat ou de l'utilisation d'un système informatique malgré les actions de l'APRIL et le soutien de certains députés. Autant dire que bien qu'il existe des solutions pérennes et libres que vous, lecteurs de *GNU/Linux Magazine*, utilisez tous les jours, nous allons continuer à payer des délégations inadaptées qui imagineront d'autres normes aussi inutiles qu'elles sont incompétentes. Je vous souhaite tout de même une bonne lecture... ■



Références

[1] DGLFLFL, « *Vers une norme française pour les claviers informatiques* » : http://www.culturecommunication.gouv.fr/Politiques-ministerielles/Langue-francaise-et-langues-de-France/Politiques-de-la-langue/Langues-et-numerique/Les-technologies-de-la-langue-et-la-normalisation/Vers-une-norme-francaise-pour-les-claviers-informatiques

[2] AFNOR, « *Projet de nouveau modèle de clavier informatique* » : http://www.afnor.org/liste-des-actualites/actualites/2015/novembre-2015/respect-de-l-ecriture-francaise-vers-un-nouveau-modele-de-clavier-informatique

[3] DGLFLFL, « *Langues régionales* » : http://www.culturecommunication.gouv.fr/Politiques-ministerielles/Langue-francaise-et-langues-de-France/Politiques-de-la-langue/Langues-de-France/Langues-regionales

[4] DGLFLFL, « *Langues des Outre-mer* » : http://www.culturecommunication.gouv.fr/Politiques-ministerielles/Langue-francaise-et-langues-de-France/Politiques-de-la-langue/Langues-de-France/Langues-des-Outre-mer

[5] DGLFLFL, « *Langues non-territoriales* » : http://www.culturecommunication.gouv.fr/Politiques-ministerielles/Langue-francaise-et-langues-de-France/Politiques-de-la-langue/Langues-de-France/Langues-non-territoriales

[6] « *Rapport au parlement sur l'emploi de la langue française* », 2015 (la section II débute en p. 51) : http://www.culturecommunication.gouv.fr/content/download/129266/1410822/version/4/file/Rapport%20au%20parlement%202015.pdf

Tristan Colombo

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°191

CONSTRUIRE UN AUTRE MONDE AVEC DES TECHNOLOGIES WEB : A-FRAME

Tristan Colombo

Pour s'immerger dans une réalité virtuelle, il faut du matériel... mais ce matériel ne coûte plus très cher puisque l'écran utilisé est celui de votre smartphone. Pourquoi donc ne pas franchir le pas et tenter une petite expérience de réalité virtuelle ? D'autant plus que le framework A-Frame simplifie grandement le développement !

Mots-clés : Réalité virtuelle, Google Cardboard, Javascript, A-Frame, WebGL

Résumé

Le projet Google Cardboard a définit des normes à respecter pour développer des applications de réalité virtuelle sous Android et utiliser des visionneuses 3D compatibles avec le projet. Le *framework* A-Frame respecte ces normes et vous permet de développer rapidement des scènes 3D dans lesquelles vous pourrez vous immerger.

Pour pouvoir « voir » en trois dimensions une scène projetée sur un écran il n'y a pas de miracle : il faut un équipement particulier. Ce matériel, ce sont des lunettes ou plutôt une visionneuse 3D. Grâce à la possibilité d'utiliser nos smartphones en tant qu'écran de ces lunettes, ces visionneuses se sont considérablement démocratisées. Et le summum a été atteint lorsque Google a lancé son projet Google Cardboard en publiant des plans permettant de fabriquer soi-même ses propres visionneuses en carton. Bien entendu, ne vous attendez pas à rivaliser avec des Oculus Rift à 600 dollars mais l'expérience sera somme toute acceptable.

Une fois équipé, nous pourrons utiliser le *framework* **A-Frame** pour créer notre propre monde de réalité virtuelle. En l'occurrence il ne s'agira que d'un arbre, mais rien ne vous empêchera par la suite de planter une forêt...

1 Le choix de la visionneuse 3D

Toutes les manipulations décrites dans cet article peuvent être réalisées et utilisées sur un écran d'ordinateur... mais pour une véritable expérience de réalité virtuelle il est bien entendu obligatoire de disposer d'une visionneuse. Comme dit en introduction, le projet Google Cardboard permet à tout un chacun de se procurer ou de fabriquer

une visionneuse 3D dont les spécifications ont été précisément définies **[1]**. Sur le site du projet, on peut avoir accès à une liste de *Google Cardboard* **[2]**. Seul problème : tous les fournisseurs se trouvent aux États-Unis et les frais de port sont donc assez élevés ! Je me suis donc rabattu sur la visionneuse *BriztechVR Google Cardboard V2* **[3]** disponible dans une grande enseigne de vente en ligne. Dans votre choix, faites surtout attention au fait que la visionneuse respecte bien les spécifications Google Cardboard V2 (au même tarif on peut trouver des visionneuses en plastique au look bien plus alléchant mais qui ne sont compatibles qu'avec un petit nombre de smartphones). Vous pouvez bien entendu préférer acheter simplement les lentilles et confectionner votre visionneuse avec des matériaux de récupération **[4]**... pour les besoins de l'article, j'avais besoin de quelque chose qui fonctionne et qui me prenne le moins de temps possible et je me suis donc orienté vers un achat sous forme de pack. Si vous pouvez privilégier la méthode DIY n'hésitez pas !

Même si vous choisissez la méthode des fainéants comme moi, une fois le pack reçut il faut assembler les pièces, en tout cas, c'est ce que je croyais...

2 Test de la visionneuse 3D

Les instructions permettant de construire la visionneuse sont assez simples. Je vous expose ici mon retour d'expérience ce qui me permettra en même temps de vous présenter cet objet.

2.1 Assemblage

Contrairement à ce que l'on peut voir sur le site du projet en se basant sur l'ensemble des pièces nécessaires si l'on construit soi-même entièrement la visionneuse, il n'y a que peu de manipulations à effectuer. La visionneuse arrive sous la forme d'une boîte en carton protégée par un emballage cellophane (voir figure 1). Il faut retirer le cellophane (étonnant, non ?), une protection en carton et déplier l'engin. Au passage vous pourrez noter la présence d'un bouton tactile qui permettra de cliquer au centre de l'écran (voir figure 2).

Une fois les bonnes applications installées vous n'aurez plus qu'à positionner votre smartphone sur la visionneuse, rabattre un volet de carton et le fixer à l'aide du scratch positionné sur le haut du boîtier.

Soyons clair, 20 euros pour quatre bouts de carton, deux lentilles en plastique, deux morceaux de bande scratch, un

Fig. 1 : Visionneuse 3D non montée.

élastique (pour tenir le smartphone) et un morceau de tissu tactile... c'est un peu cher ! D'autant plus que l'absence d'un bandeau pour accrocher l'appareil à votre tête est assez pénible (un peu comme porter des lunettes sans branches). Pour relativiser un peu ces critiques, je savais dès la commande ce que j'allais recevoir et je visais surtout un gain de temps au montage. Sur ce point le contrat est rempli, en deux minutes l'appareil est prêt à être utilisé (oui deux minutes : il a fallu que je fasse des photos et j'ai perdu un peu de temps...).

La visionneuse étant prête, nous allons pouvoir la tester.

Fig. 2 : Bouton tactile

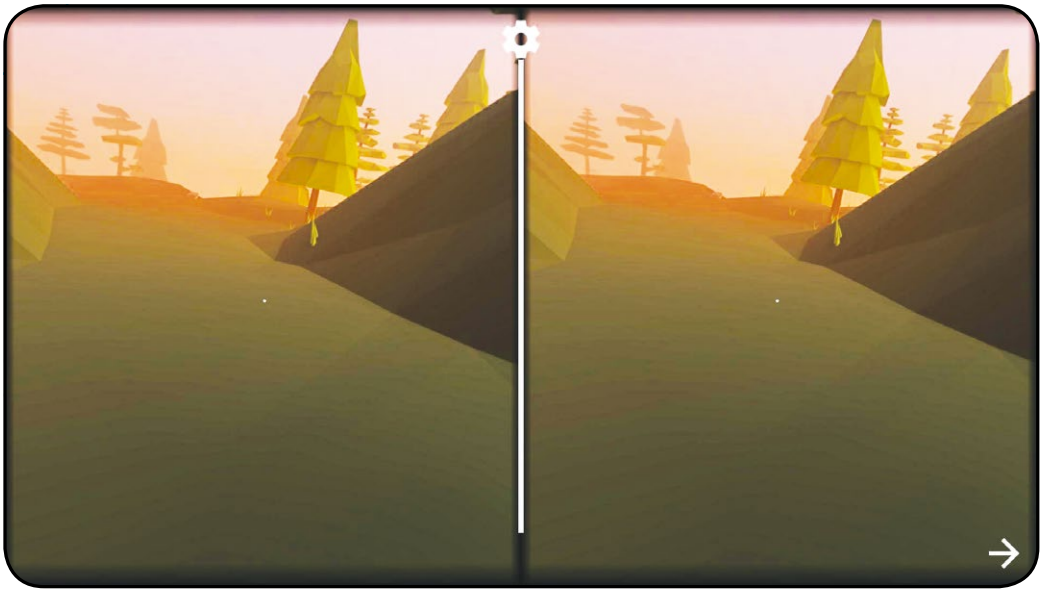


Fig. 3 : L'environnement hébergeant le menu (sur cette capture « à plat » il y a deux fois la même image décalée de quelques degrés puisque vous n'utilisez pas de visionneuse pour la regarder...).

Elle possède également de petites démonstrations accessibles depuis un menu se trouvant dans un environnement forestier comme en figure 3. De ces démonstrations, celle qui au final est la plus plaisante est... ce fameux menu ! On regrette d'ailleurs de ne pas pouvoir évoluer plus longtemps dans cet environnement. Mais heureusement, cela sera réalisable à l'aide de l'application suivante.

3.2 Cardboard Design Lab

Cette application explique comment est conçu un environnement de réalité virtuelle. Deux menus sont disponibles :

- ***Foundation*** pour les bases de l'interface et des perceptions de l'utilisateur,
- et ***Immersion*** qui sera plus consacré au « gameplay », à l'expérience d'immersion dans un nouveau monde.

On retrouvera l'environnement forestier du menu de l'application Cardboard. Il est assez agréable de se balader dans cette forêt, même si la définition graphique n'est pas très poussée. Optimiste après cette expérience j'ai voulu tester une dernière application (Figure 3).

3.3 Star Wars

Même si j'ai pour habitude de ne pas installer les applications qui demandent d'accéder à ma position alors qu'elles n'en ont pas besoin, j'ai voulu tester Star Wars. On a accès à quelques mini scènes en 3D qui mettent très, très longtemps à charger. Au premier lancement j'ai eu la désagréable surprise d'avoir un décalage trop important entre les images gauche et droite : on voit deux images décalées et ça fait assez mal aux yeux ! En relançant l'application, le problème a disparu.

Même si ce sont des scènes de l'univers de Star Wars que l'on peut visionner avec cette application, je reste sur ma bonne impression de Cardboard Design Lab et je vous encourage surtout à tester cette application-là. Maintenant tout cela est bien intéressant mais nous sommes des développeurs, et ce qui nous intéresse c'est de pouvoir tester la visionneuse sur nos propres codes !

4 À nous de jouer avec A-Frame

A-Frame est un projet de la fondation **Mozilla** (plus précisément **MozVR**, la branche réalité virtuelle des équipes de recherche). Il s'agit d'un *framework* Web permettant d'accéder aux fonctionnalités de WebGL directement à partir de tags et permettre ainsi un développement plus simple et rapide d'une expérience de réalité virtuelle au sein d'une application Web.

4.1 Installation

Tapez simplement en tant qu'administrateur la commande suivante :

```
# npm install aframe
```

Vous obtiendrez un répertoire **`node_modules/aframe`** contenant principalement des exemples (répertoire **`examples`**) et le *framework* (**`dist/aframe.js`**). Faites en sorte que ce

fichier puisse être lu par vos applications ou copiez-le simplement dans vos projets.

4.2 Premiers pas

Tout élément de réalité virtuelle que vous allez créer doit tenir dans une scène. La balise utilisée sera **<a-scene>**. Dans cette scène on peut placer différents objets prédéfinis comme un cube **<a-cube>**, une sphère **<a-sphere>** ou un cylindre **<a-cylindre>**. Ces balises admettent bien entendu des attributs pour les définir précisément. Parmi les attributs importants et communs à ces trois objets, on trouve :

- **color** : la couleur de l'objet (par défaut **gray** soit grise) ;
- **opacity** : l'opacité (par défaut **1** soit complètement opaque) ;
- **translate** : la translation suivant **x**, **y** et **z** à effectuer sur l'objet (par défaut l'objet apparaît en **(0, 0, 0)**).

Pour le cube, on peut définir sa hauteur (**height**), sa largeur (**width**) et sa profondeur (**depth**), pour la sphère ce sera son rayon (**radius**) et pour le cylindre son rayon (**radius**) et sa hauteur (**height**).

Voyons cela sur des exemples.

4.2.1 Un cube

Nous allons créer un petit cube marron qui servira de base à la création d'un arbre. Pour ceux que ça n'aurait pas encore choqués, un « cube » étant défini suivant sa hauteur, sa largeur et sa profondeur, si vous allouez des valeurs différentes à ces attributs vous obtiendrez un parallélépipède rectangle dont le cube est un cas particulier. Nous ferons donc en sorte que notre cube reste un cube :

```
01: <!doctype html>
02: <html lang="fr">
03:   <head>
04:     <meta charset="utf-8" />
05:     <title>Un cube</title>
06:     <script src="aframe.js"></script>
07:   </head>
08:
09:   <body>
10:     <a-scene>
11:       <a-cube color="maroon" height="0.3" depth="0.3"
width="0.3" translate="1 1 1"></a-cube>
12:     </a-scene>
13:   </body>
14: </html>
```

Nous obtenons un petit cube marron décalé de **(1, 1, 1)** par rapport à l'origine. Le résultat est visible en figure 4.

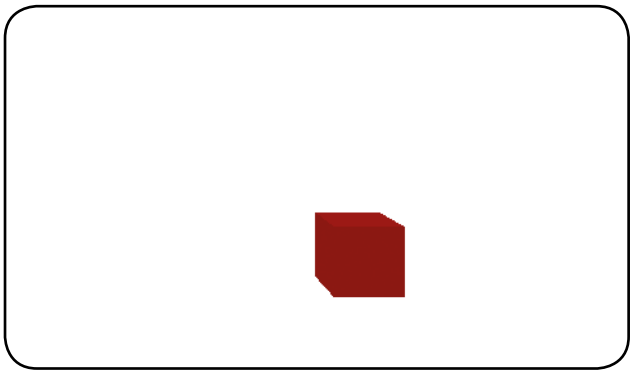


Fig. 4 : Un simple cube.

4.2.2 Plusieurs cubes

Pour le tronc de notre arbre, plutôt que d'utiliser un seul « cube » en le déformant en pavé, nous allons associer plusieurs petits cubes sur lesquels nous appliquerons de légères rotations, ce qui permettra de briser l'aspect rectiligne :

```
...
09:   <body>
10:     <a-scene>
11:       <a-cube color="maroon" height="0.3" depth="0.3"
width="0.3" translate="1 1 1"></a-cube>
12:       <a-cube color="maroon" height="0.3" depth="0.3"
width="0.3" translate="1 1.3 1" rotation="0 1 0"></a-cube>
13:       <a-cube color="maroon" height="0.3" depth="0.3"
width="0.3" translate="1 1.6 1" rotation="0 -1 0"></a-cube>
14:       <a-cube color="maroon" height="0.3" depth="0.3"
width="0.3" translate="1 1.9 1" rotation="0 1 0"></a-cube>
15:     </a-scene>
16:   </body>
17: </html>
```

Les lignes 11 à 14 définissent des cubes utilisés pour concevoir un élément plus important. On peut les lier grâce à la balise **<a-entity>** :

```
...
11:       <a-entity id="tronc">
12:         <a-cube color="maroon" height="0.3" depth="0.3"
width="0.3" translate="1 1 1"></a-cube>
...
16:       </a-entity>
17:     </a-scene>
18:   </body>
19: </html>
```

Une fois les éléments groupés sous forme d'entité, vous pouvez leur appliquer une translation ou une rotation directement depuis la balise **<a-entity>** (ce qui évite de copier/coller les attributs sur les différents éléments).

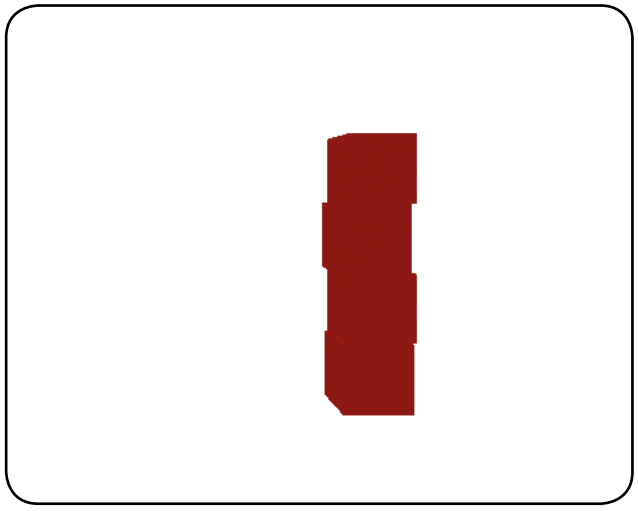


Fig. 5 : Un tronc d'arbre composé de cubes.

La figure 5 montre la progression de la création de notre arbre.

4.2.3 Une sphère

Nous utiliserons une simple sphère verte pour représenter les branches et les feuilles :

```
...
17:       <a-sphere color="green" radius="0.8" translate="1 2.8
1"></a-sphere>
18:     </a-scene>
19:   </body>
20: </html>
```

Bien entendu, comme nous l'avons fait pour le tronc, il est préférable d'organiser correctement les éléments :

```
...
11:       <a-entity id="arbre">
12:         <a-entity id="tronc">
13:           <a-cube color="maroon" height="0.3" depth="0.3"
width="0.3" translate="1 1 1"></a-cube>
...
17:         </a-entity>
18:         <a-entity id="feuillage">
19:           <a-sphere color="green" radius="0.8" translate="1
2.8 1"></a-sphere>
20:         </a-entity>
21:       </a-entity>
22:     </a-scene>
23:   </body>
24: </html>
```

Nous obtenons un arbre, certes peu évolué (voir figure 6), mais un arbre que nous pouvons admirer depuis notre visionneuse 3D.

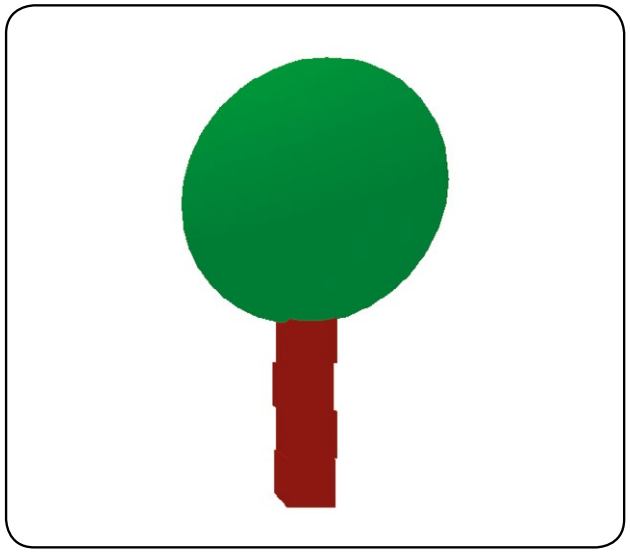


Fig. 6 : Un arbre composé de cubes et d'une sphère.

4.3 Interaction

Ajoutons un peu de vie à notre arbre : si l'utilisateur le touche, il va réveiller des oiseaux qui vont se mettre à gazouiller. Pour cela nous allons utiliser l'attribut **sound** dans la balise **<a-entity>**. Mais il y a un autre élément à ajouter : le curseur qui va permettre d'interagir avec l'environnement. Ce curseur se trouve au centre de la fenêtre et il faut définir une nouvelle caméra pour le rendre visible (balise **<a-camera>**). Là où il faudra être attentif, c'est que la caméra créée par défaut se trouve en **(0, 1.8, 4)** et que la création d'une nouvelle caméra se fait automatiquement en **(0, 0, 0)**... Donc si nous voulons pouvoir continuer à avoir la même vue sur la scène il va falloir positionner la caméra au même endroit que la caméra par défaut :

```
...
10:     <a-scene>
11:       <a-camera cursor-visible="true" cursor-color="blue"
position="0 1.8 4"></a-camera>
12:       <a-entity id="arbre" sound="src: birds.mp3; on: click">
...
22:       </a-entity>
23:     </a-scene>
24:   </body>
25: </html>
```

Lorsque vous placerez le curseur sur l'arbre et que vous cliquerez (ou plutôt que vous appuierez sur le bouton du cardboard), le son **birds.mp3** sera joué. Le curseur apparaît comme un petit cercle que j'ai coloré en bleu (par défaut il est blanc). Notre scène est désormais semblable à celle de la figure 7.

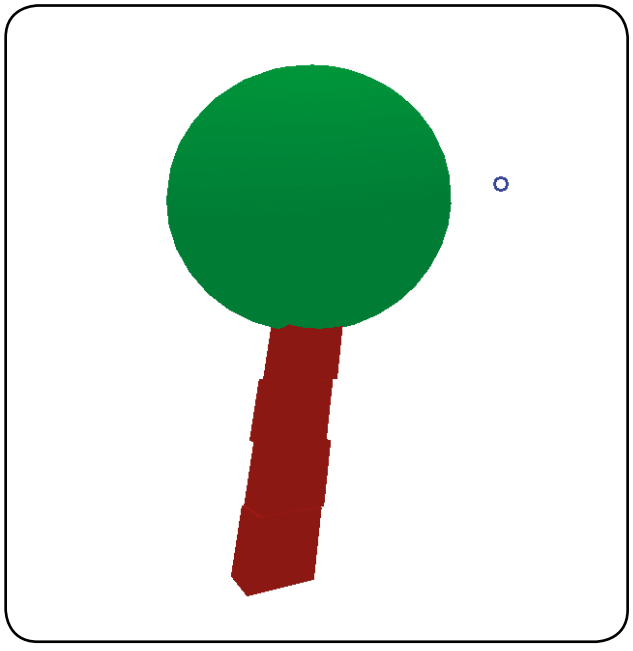


Fig. 7 : Arbre accompagné du curseur.

4.4 Interactions plus générales

Nous avons utilisé une balise particulière pour intercepter un clic de l'utilisateur et jouer un son mais il est bien entendu possible d'intercepter n'importe quel type d'événement pour réaliser n'importe quelle action : c'est du Javascript « classique ». Par exemple, on peut changer la couleur du feuillage sur un clic utilisateur :

```
...
10:     <a-scene>
11:       <a-camera cursor-visible="true" cursor-color="blue"
position="0 1.8 4"></a-camera>
12:       <a-entity id="arbre">
...
19:         <a-entity id="feuillage">
20:           <a-sphere id="sphere" color="green" radius="0.8"
translate="1 2.8 1"></a-sphere>
21:         </a-entity>
22:       </a-entity>
23:     </a-scene>
24:
25:     <script>
26:       document.querySelector('#feuillage').
addEventListener('click', function () {
27:         document.querySelector('#sphère').
setAttribute('color', 'red');
28:       });
29:     </script>
30:   </body>
31: </html>
```

Si **feuillage** avait été plus complexe le clic aurait été validé sur n'importe quel élément de l'entité. Dans le cas présent nous aurions pu nous satisfaire d'intercepter l'événement sur **sphère** ce qui aurait économisé un appel à **querySelector()**. De plus, le bloc de script en fin de document n'est présent qu'à titre de test, il aurait normalement fallu créer un nouveau fichier avec exécution à la fin du chargement de la page.

4.5 Un peu de vie...

Pour donner un peu de vie à la page, il est possible de définir des animations à l'aide de la balise... **<a-animation>** bien entendu ! Plusieurs attributs seront à définir :

- **attribute** : le nom de l'attribut à animer ;
- **dur** : la durée d'animation en millisecondes ;
- **to** : valeur de l'attribut choisi en fin d'animation ;
- **repeat** : nombre de fois où l'animation doit être répétée ou **indefinite** pour une boucle infinie ;
- **easing** : fonction d'accélération de l'animation au cours du temps (**linear**, **ease-in**, **ease-out**, etc **[5]**).

Voici un exemple où notre arbre va tourner autour du centre de la page en accélérant :

```
...
10:     <a-scene>
11:       <a-camera cursor-visible="true" cursor-color="blue"
position="0 1.8 4"></a-camera>
12:       <a-entity id="arbre">
13:         <a-animation attribute="rotation"
14:                      to="0 360 0"
15:                      dur="5000"
16:                      easing="ease-in"></a-animation>
...
26:       </a-entity>
27:     </a-scene>
28:   </body>
29: </html>
```

Si l'on avait voulu que l'arbre se déplace on aurait utilisé une animation du type :

```
<a-animation attribute="position"
  to="3 1 1"
  dur="10000"
  easing="linear"></a-animation>
```

Et maintenant, la question que tout le monde se pose : comment créer des animations complexes ? Cela se fait très simplement en associant des balises **<a-animation>**.

Par exemple en associant les deux animations précédentes on obtient un déplacement qui est la composée des deux animations :

```
<a-animation attribute="position"
  to="3 1 1"
  dur="5000"
  easing="linear"></a-animation>
<a-animation attribute="rotation"
  to="0 640 0"
  dur="5000"
  easing="ease-in"></a-animation>
```

4.6 ... et de couleurs !

Appliquons maintenant des textures sur nos objets pour plus de crédibilité. Nous allons remplacer notre grosse sphère verte par une sphère sur laquelle une image de feuillage sera plaquée. Cette image sera contenu dans un fichier **texture.png** :

```
...
10:     <a-assets>
11:       <img id="feuilles" src="texture.png">
12:     </a-assets>
13:     <a-scene>
...
22:         <a-entity id="feuillage" geometry="primitive: sphere;
radius: 0.8; translate: 1 2.8 1" material="src: #feuilles">
23:         </a-entity>
24:       </a-entity>
25:     </a-scene>
26:   </body>
27: </html>
```

Comme vous l'aurez remarqué, la texture est définie dans la balise **<a-assets>** des lignes 10 à 12. Ceci n'est nullement obligatoire mais permet d'y faire référence depuis l'attribut **material** de **<a-entity>** en ligne 22. Nous aurions pu nous passer de **<a-assets>** en employant **material="src: url(texture.png)"** en ligne 22, mais cette syntaxe est moins élégante et moins pratique si l'on souhaite réemployer une même texture. L'utilisation de **<a-sphere>** a été remplacée grâce à l'attribut **geometry** de **<a-entity>** qui définit une sphère ayant les mêmes caractéristiques que la précédente (une texture ne peut pas s'appliquer sur **<a-sphere>**, **<a-cube>**, etc). La figure 8 montre le rendu obtenu.

Conclusion

Nous sommes bien loin d'un monde complet avec des arbres, des rivières et des nuages mais les bases sont là et sont très simplement accessibles. D'autres éléments tels que

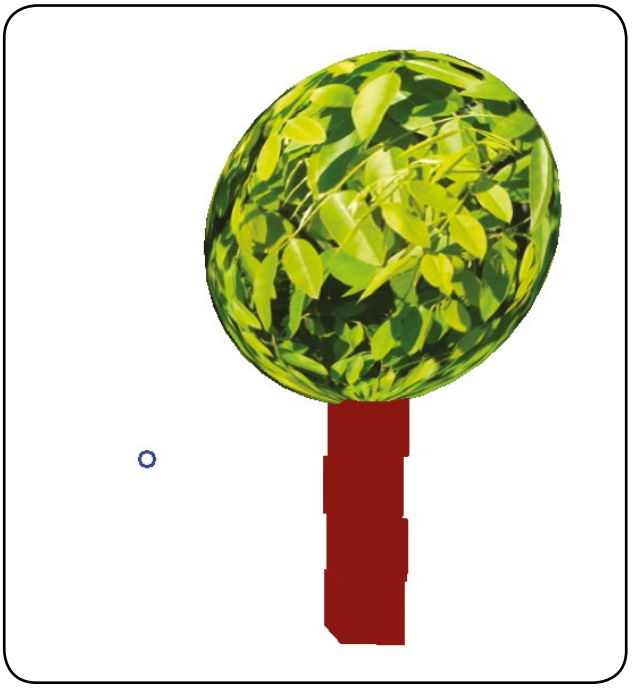


Fig. 8: Arbre avec texture de feuilles.

le brouillard ou des vidéos projetées sur un écran ou une sphère sont disponibles. A-Frame n'en est qu'à ses débuts puisqu'il s'agit de la version 0.1.0 (en tout cas au moment où ces lignes sont écrites) mais ce *framework* entrouvre les portes de la réalité virtuelle sur le Web pour tous les développeurs, même ceux qui ne connaissent pas WebGL. Vous pouvez donc commencer à rêver d'un autre monde où la Terre ne serait pas forcément ronde... ■

Références

[1] Spécifications des Google Cardboard : https://www.google.com/intl/fr_ALL/get/cardboard/manufacturers/

[2] Liste des Google Cardboard : https://www.google.com/intl/fr_ALL/get/cardboard/get-cardboard/

[3] BriztechVR Google Cardboard V2 : https://vr.briztech.co.uk/

[4] Instructions pour fabriquer des Google Cardboard (DIY) : https://www.google.com/get/cardboard/downloads/wwgc_manufacturers_kit_v2.0.zip

[5] Les différentes fonctions d'accélération : https://aframe.io/docs/core/animation.html#Easing

DISPONIBLE DÈS LE 11 MARS

GNU/LINUX MAGAZINE HORS-SÉRIE N°83 !

LE GUIDE POUR
APPRENDRE LES PRINCIPES CLÉS DU LANGAGE C++ !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

DONNE UN POISSON À UN CHAT ...

The Cheshire Cat [Everyone here is mad]

... il aura à manger un jour. Apprends-lui à pêcher, il aura à manger pour tous les jours. Et si pour ce premier papier dans les billets d'humeur, nous parlions d'enseignement, d'apprentissage et d'attente de recruteur ?

Mots-clés : Entreprise, Recrutement, Formation, Apprentissage, Étudiant

Résumé

Les entreprises se plaignent de ne pas trouver de candidats leur convenant. Les étudiants se plaignent de ne pas apprendre ce qui pourrait leur servir à leur arrivée dans la vie active. Les écoles se plaignent que les entreprises ne sont jamais satisfaites et ne comprennent pas les contraintes de la mise en place d'une formation digne de ce nom. En bref, tout le monde se plaint. Et si l'on essayait de comprendre pourquoi ?

Il n'y a pas assez de développeurs, d'administrateurs système, de gens compétents, de chefs de projets, de « ninja » ou de développeurs multicasquettes. Les entreprises n'arrêtent pas de se lamenter et demandent à cor et à cri qu'on forme les gens qui leur manquent et qu'on les forme correctement parce que là, pour l'instant, cela ne va pas du tout, ils sont tout mal formés les étudiants. On pourrait presque penser, en écoutant certains recruteurs, qu'ils sont plus déformés que formés et qu'une fois en entreprise, il faut tout leur désapprendre pour leur réapprendre les choses utiles. Et si, ayant eu successivement les trois casquettes (étudiants, recruteur, professeur) au cours des 15 dernières années, je vous disais ce que j'en pense ? Attention ça va vous défriser les moustaches.

1 Le temps de l'entreprise et le temps de la formation

L'entreprise veut pouvoir recruter des profils efficaces tout de suite. Dans notre société de l'instantané, du prêt à consommer et aussitôt jeté, être efficace tout de suite, c'est important. Et peu importe la jeunesse des technologies que l'on souhaite mettre en place, on veut des profils compétents, seniors, experts. Les deux meilleurs (ou pire) exemples que je peux vous citer correspondent à deux offres de postes que j'ai vu rapidement passer dans les six derniers mois. La première offre ciblait un expert **Node.JS** avec une expérience d'au moins 5 ans et si possible plus (bon, Node.JS a été créé en 2009, donc cela semble possible). Quant à la seconde, elle concernait un profil DevOps avec une expérience Docker d'au minimum 5 ans (difficile quand on sait que **Docker** existe depuis 2013). Et je suis quasiment sûr que si nous demandions pourquoi elles n'ont eu aucun candidat validant leurs critères aux talentueuses personnes qui ont rédigé ces deux annonces, elles répondraient que c'est la faute des écoles qui ne forment pas convenablement les jeunes.

À cette volée de critiques, les écoles répondent majoritairement qu'un parcours d'enseignement ne se fait pas ou ne se modifie pas en une heure ou même en une semaine. Que l'enseignement n'est pas là pour apprendre les modes mais pour bâtir un socle de savoirs sur lequel les étudiants pourront s'appuyer pendant de nombreuses années, voire toute leur carrière. Et après tout, les écoles n'ont pas tort. Imaginons qu'elles changent de programme comme de chemises... et mettons-nous dans la peau d'un(e) gentil(le) étudiant(e) qui commence son école en 2013. On lui offre des cours à la pointe : des cours d'**Angular JS** V1 ! Retrouvons cette sympathique personne, diplômée fin 2016, qui arrive sur le marché du travail très fier de son CV avec en ligne principale « expert Angular JS 1 ». Vous me rétorquerez que l'on trouve encore des offres d'emploi en Cobol alors qu'en 2016, il y aura toujours du boulot pour quelqu'un connaissant l'Angular JS 1. C'est vrai. Mais moi, si j'avais passé un temps non négligeable à me former sur un *framework* en fin de vie au moment de mon arrivée sur le marché du travail, je peux vous dire que j'aurais sérieusement envie de griffer des gens.

2 Apprendre à faire ou apprendre à apprendre

L'un des credo des écoles est donc de donner une base solide et pérenne. Le but d'un enseignement ce n'est pas seulement d'apprendre à faire, c'est aussi et même surtout apprendre à apprendre, apprendre à comprendre. Cela me semble bien en phase avec le discours qui court aujourd'hui sur le fait que tout au long de notre vie professionnelle, nous allons changer de métier plusieurs fois. Autant alors tout de suite ne pas se limiter à apprendre à faire, mais autant avoir tout de suite les méthodes, les savoirs qui nous aideront à changer de métier, à rebondir rapidement, non ? Dans cette optique, on penserait, et il me semble avec raison qu'il faut apprendre un minimum de notions théoriques. Si l'on parle d'une formation centrée sur le développement, cela passe par des notions d'algorithme, de conception de systèmes d'exploitation, de théorie des langages, de complexité (et donc de mathématiques). Si l'on considère une formation plus administrateur réseau et système, on va être sur des notions d'architecture des ordinateurs, des systèmes d'exploitations, de cours théoriques sur les protocoles réseaux ou de théories du signal (tiens là aussi des mathématiques).

Bien entendu ce n'est pas à priori très amusant. Et l'intérêt n'est pas aussi évident de suivre un cours sur la théorie des compilateurs comparé à passer des jours entiers dans une « piscine » pour pisser du code qu'on ne comprend qu'à moitié. Mais cela a l'avantage de vous doter d'un savoir moins sujet à péremption Eet, personnellement, j'aime me dire que ce que j'ai appris en cours a peu de chance de finir à la poubelle en me laissant les mains vides, un peu comme un poisson qu'on aurait laissé trop longtemps au frigo et que l'on se retrouve obligé de jeter, alors qu'il ne restait plus que cela dans le frigo.

Quant à ce qui est d'apprendre à comprendre, il me semble que ce n'est pas ce à quoi nous pousse la société en général (ici attention, je ne fais pas de « les choses étaient mieux avant » ; non, il me semble que les choses étaient déjà dans un état bien décrépies lors de ma propre scolarité, même si cela remonte à loin). Pour illustrer cela, je vais vous citer un de mes anciens étudiants qui m'a si bien demandé quand je présentais les différents types de listes chaînées en introduction d'un cours d'algo : « Mais à quoi cela va nous servir Monsieur ? On a le type liste, il suffit de s'en servir et ça marche. On s'en fout de comment ça marche tant que ça marche ». Remarque imparable. N'ayant sur l'instant pas de réponse rapide à lui donner, je l'ai alors, en pensée, assommé sous un déluge de craies. À la place, j'ai débattu de la question pendant un bon quart d'heure avec l'ensemble des étudiants pour en arriver à la conclusion que la bonne moitié ne voyait strictement pas l'intérêt de comprendre comment les choses marchent.

3 Inadéquation dans les attentes

Assez relativement souvent, ce que l'on va attendre de quelqu'un dans un contexte professionnel, c'est qu'il fasse quelque chose qui marche. Et qu'il le fasse rapidement. Le plus rapidement possible. Vous êtes un(e) ninja ? Vous êtes capable de coder 18 heures par jour pendant des mois ? Alors vous allez devenir le roi du monde. Ce qu'il faut c'est juste livrer vite. Si « ça marche » cela suffit. Valider que cela marche avec des tests, ce n'est pas rentable, pas utile. Et puis s'il reste des petits bugs, ce n'est pas vraiment un problème, tant que le client ne s'en rend pas compte. Lors d'une discussion avec divers « chefs de projets », il m'est même arrivé d'entendre que quelques petits bugs, ce n'était pas si mal. Après tout, s'ils étaient découverts après la fin de la période de garantie, cela faisait toujours ça en contrat de TMA (Tierce Maintenance Applicative) de pris.

Pas vraiment ce qu'en tant que développeur(euse), on espère de sa vie professionnelle. Enfin, pas vraiment ce que moi, j'espérerais avoir.

De même, quand une entreprise cherche un(e) développeur(euse), elle cherche vraiment un(e) développeur(euse), et pas quelqu'un qui va coder pendant trois ou quatre ans et puis passer sur un poste de Chef de Projet. À l'inverse, bien souvent, le discours présent dans les écoles est que si vous perdez toujours votre temps à développer à 35 ans, c'est que vous avez raté votre vie et que le seul et unique Graal qui vaut la peine, c'est devenir Chef de Projet. Dans cette optique, quelle désillusion quand à 37 ans on vous annonce que vous allez être placé sur une mission pour débugguer du Fortran dans une application industrielle qui date du siècle dernier...

Conclusion

Là comme cela, j'ai l'air d'un vieux matou aigri qui perd ses poils et ses dents. Les entreprises chercheraient des profils ultra-qualifiés et toujours à la pointe qu'elles pourraient tout de même payer moins que rien. Quant aux étudiants(es), ce ne seraient que des gens inconséquents et sans profondeur qui suivraient des cursus sans vraiment chercher à comprendre ce qu'ils apprennent.

J'ai bien l'impression que cette vision est malheureusement très souvent vraie. Mais courage, parce que si vous êtes une entreprise, il existe un certain nombre de profils de personnes qui vous combleront, plus que ce que vous ne pouviez l'imaginer. Et si vous êtes un(e) développeur(euse), il existe aussi un certain nombre d'entreprises avec qui il fait bon signer un CDI : des entreprises qui exigeront des tests unitaires, de la documentation, du vrai travail d'équipe et vous laisseront évaluer vous-mêmes le temps nécessaire pour réaliser les tâches que l'on vous affecte.

Une dernière chose, un dernier conseil, plus spécialement destiné aux entreprises. Si vous voulez trouver de bons profils, pensez au télétravail. Et si vous êtes une entreprise parisienne, embauchez une Bordelaise en télétravail ne veut pas dire lui donner un salaire de Parisienne ... Vous allez voir, les bons profils seront plus faciles à trouver ! ■

CRÉEZ VOS PROPRES BIBLIOTHÈQUES

Sami Ben Youssef [Ingénieur développement Linux embarqué]

Peut-on concevoir un projet volumineux sans planifier la création de bibliothèques ? Comment créer et exploiter sa propre bibliothèque ? Quel avantage aurons-nous d'y placer un code source ? Comment choisir entre bibliothèque dynamique ou statique ? On va tenter de répondre à toutes ces questions dans ce qui suit.

Mots-clés : Bibliothèque, Statique, Dynamique, Partagée, Compilation, Édition de liens

Résumé

On essaye de mettre en relief l'importance de l'utilisation des bibliothèques statiques et dynamiques, leurs avantages, leurs inconvénients et comment choisir le type de bibliothèque le plus approprié à une solution donnée. Ultérieurement, on consacre une partie, orientée surtout vers la pratique, dans laquelle on présente la réalisation de deux bibliothèques et on conçoit deux programmes en C qui ont recours à ces dernières.

1 Besoin de bibliothèques

Compiler un projet codé en langage C est considéré comme une tâche qui prend énormément de temps si le code est volumineux : on se trouve parfois dans la nécessité de recompiler la totalité du projet lors d'un ajout d'une toute petite instruction. Le temps qu'on investit à la recompilation des parties de code inchangées est considéré comme une perte.

La réalisation d'un projet passe toujours par la récupération de quelques parties du code réalisé dans des projets antérieurs ; si le projet est codé en bloc, le développeur doit fouiller le code entier pour trouver les fonctionnalités requises, copier et coller tout en s'assurant qu'il a récupéré l'ensemble des fonctions appelées par la fonction initiale, ceci paraît un peu délicat, gênant et coûteux en termes de temps pour le développeur qui assure cette tâche.

Les intervenants dans le développement comme les architectes, les développeurs et tous ceux qui souffrent des difficultés du codage en bloc sentent le besoin d'un changement qui facilite leurs vies à savoir un mécanisme (ou un système) qui leur évite de parcourir des millions de lignes de codes, et qui leur offre non seulement un code réutilisable immédiatement, mais également réduit le temps de compilation et leur propose aussi plusieurs autres avantages qu'on explicitera par la suite.

2 Avantages et inconvénients

Une bibliothèque est un fichier reconnu par l'éditeur de liens qui a ces propres caractéristiques, un format bien défini et des règles de nommage à suivre ; ce fameux fichier peut être composé par un ou plusieurs fichiers *sources* « **.c** » et fichiers *headers* « **.h** » qui sont passés par la première étape de compilation en fichiers *objets* « **.o** » puis la deuxième étape qui consiste à créer la bibliothèque proprement dite à partir de ces fichiers *objets* ; cette étape peut être réalisée de deux manières afin de générer deux types de bibliothèques différents : les bibliothèques statiques

reconnues par l'extension « `.a` » et les bibliothèques dynamiques, qu'on appelle aussi des bibliothèques partagées et sont reconnues par l'extension « `.so` ».

Les fonctions proposées par la bibliothèque (initialement codées dans les fichiers « `.c` ») ne doivent impérativement pas contenir de fonction `main()`. On tient tout de même à mentionner qu'elles peuvent être appelées depuis un ou plusieurs programmes. Chaque programme peut profiter des services des bibliothèques statiques et dynamiques à la fois.

Dans le but d'exploiter les fonctions d'une bibliothèque statique, on doit indiquer cela au niveau de la phase d'édition des liens. L'éditeur des liens inclut la totalité des fonctions du fichier « `.o` » concerné de la bibliothèque statique dans notre fichier exécutable pour avoir en fin de compte un fichier complet et autonome pouvant tourner indépendamment de l'environnement où il sera exécuté. En contrepartie, l'utilisation des bibliothèques statiques présente quelques inconvénients. On peut citer à titre d'exemple la taille volumineuse de notre exécutable ; ce dernier renferme le contenu du programme principal et de tous les fichiers objet « `.o` » concernés des bibliothèques statiques dont il exploite les fonctions. On tient à attirer votre attention sur le fait que chaque évolution apportée au code source de la bibliothèque, aussi mineure soit-elle nous inflige la pénible tâche de recompiler tout notre code afin d'intégrer ces modifications dans notre fichier autonome et complet.

Comme on l'a déjà mentionné, une bibliothèque statique peut être appelée par un ou plusieurs programmes, ce qui provoque une duplication du code puisque chaque programme intègre la totalité des fonctions dans son fichier exécutable.

Pareillement aux bibliothèques statiques, nous sommes appelés à présenter l'ensemble des bibliothèques dynamiques dont nous avons exploité les fonctions lors de l'édition des liens ; mais cette fois ce n'est pas pour inclure la totalité des fichiers « `.o` » des bibliothèques concernées dans notre exécutable, c'est juste pour permettre à l'éditeur des liens de résoudre les références, en d'autres termes pour lui indiquer dans quelle bibliothèque partagée la fonction appelée est définie. Ainsi on remarque un gain considérable au niveau de la taille du fichier exécutable qui devient plus léger puisqu'il ne renferme pas le code de la bibliothèque. Cela dit, on se trouve étroitement lié à l'environnement dans lequel le programme sera exécuté et on doit s'assurer que toutes les bibliothèques partagées dont on a besoin sont bien installées pour éviter les messages d'erreurs qui nous gâchent le plaisir de voir notre programme tourner convenablement.

La maintenance de ce type de bibliothèque présente plus de souplesse et plus de facilité que celle des bibliothèques statiques, surtout lorsque l'on modifie le corps des fonctions de la bibliothèque. Dans ce cas on peut générer une nouvelle version et remplacer l'ancienne pour faire profiter les fichiers exécutables faisant appel à cette bibliothèque des évolutions apportées et ce, non seulement sans perturber leurs fonctionnements mais aussi sans les obliger à recompiler leurs codes à nouveau. Le seul et unique cas où les exécutables doivent recompiler leurs codes pour s'aligner sur les évolutions de la bibliothèque est lorsque l'on touche aux signatures des fonctions ; ce genre de modifications est considéré comme majeur. Cela dit, un exécutable est autorisé à ignorer la nouvelle version de la bibliothèque et peut rester « branché » sur l'ancienne version de la bibliothèque qui reste toujours disponible et offre les mêmes services. Le fait qu'une bibliothèque dynamique ne soit chargée qu'une seule fois en mémoire, même si elle est utilisée simultanément par plusieurs exécutables, représente un point fort qui peut plaire énormément aux personnes qui développent sur des plateformes avec une mémoire réduite à savoir spécialement le domaine de l'embarqué.

Pour résumer cette partie, le choix entre bibliothèque dynamique ou bibliothèque statique n'est pas évident et reste intimement lié à la nature du projet et aux contraintes de ce dernier comme la taille de la mémoire, la taille de l'exécutable, l'environnement dans lequel il va tourner, la maintenance des bibliothèques et bien d'autres métriques spécifiques à chaque projet.

3 Compilation

Démarrez vos machines, on va passer aux choses sérieuses ! Avant tout, on va préparer notre environnement de travail : créons un répertoire `workspace` qui va contenir la liste des répertoires suivants :

- `src` : contiens les fichiers « `.c` » nécessaires à la création de la bibliothèque ;
- `include` : contient les fichiers *headers* « `.h` » de la bibliothèque ;
- `lib` : les bibliothèques que l'on va générer seront placées dans ce dossier ;
- `programme` : ce répertoire, dont on aura besoin ultérieurement, va contenir le fichier « `.c` » du programme principal ainsi que le fichier exécutable.

Ouvrez vos terminaux et tapez les commandes suivantes :

```
$ mkdir -p workspace/src workspace/include workspace/lib
workspace/programme
```

Une fois l'arborescence préparée, on se positionne dans le répertoire **workspace/src** pour créer le fichier **leCode.c** qui va contenir le code suivant :

```
#include "leCode.h"

int ma_variable=3;

void ma_fonction() {
  printf("Bienvenue dans ma_fonction\n");
}

int mon_addition(int i,int j) {
  printf("Bienvenue dans mon_addition\n");
  return (i+j);
}

int ma_soustraction(int i,int j) {
  printf("Bienvenue dans ma_soustraction\n");
  return (i-j);
}
```

Ensuite on crée le fichier **leCode.h** qui va être placé dans **workspace/include**. Voici le code contenu dans ce fichier :

```
#include <stdio.h>

int ma_variable;

void ma_fonction();

int mon_addition(int i,int j);

int ma_soustraction(int i,int j);
```

Le code qu'on vient d'insérer dans le fichier **workspace/src/leCode.c** contient l'ensemble des éléments que va proposer la bibliothèque à tout programme voulant profiter des services de cette dernière. En parcourant le code, on peut voir que les exécutables peuvent faire appel aux trois fonctions : **ma_fonction()**, **mon_addition()** et **ma_soustraction()** et une variable de type entier intitulée **ma_variable** ayant une valeur initiale égale à **3** et qui peut être modifiée dans le programme principal.

Enfin, on passe à la première étape de compilation qui consiste à générer les fichiers objets « **.o** » ; dans le cas présent, on ne va générer qu'un seul fichier objet **leCode.o** puisque l'on n'a qu'un seul fichier source « **.c** » **leCode.c**.

Voici la toute puissante commande **gcc** qui va nous permettre de concrétiser cela, on explicitera juste après les options employées :

```
$ gcc -c src/leCode.c -I include/ -o src/leCode.o
```

L'option **-c** ordonne à la commande **gcc** de s'arrêter à la phase de compilation qui consiste à générer les fichiers objet « **.o** » et de ne pas passer à la phase d'édition de liens.

L'option **-I** avec l'argument **include** indique le chemin relatif des fichiers *hearders* « **.h** ».

L'option **-o src/leCode.o** spécifie au compilateur l'intitulé du fichier objet et le chemin relatif de ce dernier.

Une fois la commande ci-dessus exécutée avec succès, on remarque qu'il y a un fichier de plus sous **src** appelé **leCode.o**, allons voir le contenu de ce fichier objet moyennant la commande **nm -s** :

```
$ nm -s src/leCode.o
00000000 T ma_fonction
00000030 T ma_soustraction
00000000 D ma_variable
00000014 T mon_addition
         U puts
```

On est rassuré : le fichier objet contient bien tous les éléments que l'on souhaite proposer au programme. On peut attaquer la dernière étape, à savoir la génération de la bibliothèque. Il ne nous reste plus qu'une commande et l'on aura enfin notre bibliothèque statique qui pourra être appelée par tout programme voulant bénéficier de ces services :

```
$ ar -cr lib/libma_lib_static.a src/leCode.o
```

ar est la commande qui a conçu la bibliothèque **libma_lib_static.a** sous **workspace/lib** avec les options **-c** pour créer l'archive et l'option **-r** pour insérer les fichiers objets « **.o** » dans l'archive, ou remplacer les anciens fichiers par les nouveaux dans le cas où ils existaient déjà. Pour vérifier le bon déroulement de la commande qu'on a lancée, rendez-vous dans **workspace/lib** où se trouve notre bibliothèque **libma_lib_static.a**.

```
$ nm -s lib/libma_lib_static.a
Archive index:
ma_variable in leCode.o
ma_fonction in leCode.o
mon_addition in leCode.o
ma_soustraction in leCode.o
leCode.o:
```

```
00000000 T ma_fonction
00000030 T ma_soustraction
00000000 D ma_variable
00000014 T mon_addition
         U puts
```

Cette commande nous affiche deux parties principales. La première partie représente la table de symboles de la bibliothèque. En fait, elle nous présente l'ensemble des éléments offerts à tout programme utilisant cette bibliothèque pour épargner au développeur l'effort de fouiller dans les fichiers sources pour dégager la liste des éléments qu'il peut exploiter. La deuxième partie est celle dans laquelle on trouve le plus de détails concernant les éléments de la bibliothèque. Ces derniers sont regroupés par fichiers objets : je vous laisse le soin d'approfondir la signification des valeurs et du type associé à chaque élément. Actuellement, la bibliothèque statique est prête et archivée dans **workspace/lib** ; on va donc se lancer dans la création de la bibliothèque dynamique. Vu que la première partie consistant à générer les fichiers « **.o** » est déjà réalisée lors de la préparation de la bibliothèque statique, désormais on va profiter de l'existence du fichier **leCode.o** dans **workspace/src** et passer directement à la deuxième étape, celle de la génération de la bibliothèque dynamique à travers la commande suivante :

```
$ gcc -shared -o lib/libma_lib_dynamic.so src/leCode.o
```

Contrairement à la génération d'une bibliothèque statique réalisée moyennant l'outil **ar**, la bibliothèque dynamique est générée avec la commande **gcc** suivie de l'option **–shared** ; pour le reste des arguments, on spécifie juste le chemin relatif ainsi que l'intitulé de la bibliothèque pour l'option **-o** présentée avec la liste des fichiers « **.o** ». On tient à préciser que dans le cas présent la liste ne contient qu'un seul fichier : **leCode.o**.

À présent on vous invite à jeter un coup d'œil sur le contenu de notre bibliothèque dynamique ; cette fois, c'est la commande **objdump** qu'on va lancer. En effet, on a récupéré une partie de la table de symboles affichée par cette commande : la partie contenant les trois fonctions et la variable.

```
$ objdump -t lib/libma_lib_dynamic.so

lib/libma_lib_dynamic.so:     file format elf32-i386

SYMBOL TABLE:
00000114 l    d  .note.gnu.build-id 00000000              .note.
gnu.build-id
...
00002018 g     O .data  00000004              ma_variable
```

```
00000000  w      *UND*  00000000              _ITM_
deregisterTMCloneTable
0000201c g       .data  00000000              _edata
0000060c g     F .fini  00000000              _fini
...
000005bb g     F .text  00000014              ma_fonction
0000201c g       .bss 00000000               __bss_start
000005cf g     F .text  0000001c              mon_addition
00000000  w      *UND*  00000000              _Jv_RegisterClasses
00000000  w      *UND*  00000000              _ITM_
registerTMCloneTable
000005eb g     F .text  0000001e              ma_soustraction
00000438 g     F .init  00000000              _init
```

4 Utilisation des bibliothèques

Les deux bibliothèques **libma_lib_static.a** et **libma_lib_dynamic.so** qu'on a générées précédemment sont désormais prêtes à l'emploi. On va changer de casquette et prendre la place d'un développeur qui a consulté le contenu de la bibliothèque statique **libma_lib_static.a**, trouvé les éléments dont il a besoin pour son programme et décidé d'intégrer la bibliothèque dans son exécutable. Voici le code du programme principal :

```
#include "leCode.h"

int main(void) {
  ma_fonction();
  printf("Ma_variable = %i\n", ma_variable);
  printf("Ma_variable + 2 = %i\n", mon_addition(ma_variable, 2));
  return 0;
}
```

Le programme ci-dessus inclut le fichier **leCode.h** de la bibliothèque et fait appel aux fonctions suivantes qui figurent toutes dans la bibliothèque : **ma_fonction()**, **mon_addition()** et la variable **ma_variable**. Pour voir son programme tourner convenablement et éviter tous les messages d'erreurs ennuyeux, le développeur doit inclure la bibliothèque lors de l'édition des liens de la manière qui suit :

```
$ gcc programme/prog.c -lma_lib_static -L lib/ -I include/ -o
programme/prog_lib_static
```

Grâce à cette commande on a lancé la compilation et l'édition des liens du programme principal **prog.c** se trouvant dans **workspace/programme** ; remarquez que contrairement à la génération des bibliothèques, ici on n'a pas inséré l'option **-c** dans notre commande sinon elle nous obligerait

à arrêter **gcc** à la phase de compilation. À la fin de cette commande, si tout se passe bien, on obtiendra un fichier exécutable **prog_lib_static** archivé dans **workspace/programme**. L'intitulé **prog_lib_static** et le chemin **workspace/programme** ne sont pas choisis par hasard. En effet ils sont le résultat de l'utilisation de l'option **-o** suivi de l'argument **programme/prog_lib_static**. On a également utilisé l'option **-I** pour préciser le chemin des includes et l'option **-L** pour indiquer l'emplacement des bibliothèques ; c'est dans cet emplacement qu'on trouve la bibliothèque **libma_lib_static.a** qu'on a collé à l'option **–l**. Le nom exact de la bibliothèque est **libma_lib_static.a**, mais lors de l'édition des liens, on enlève toujours le préfixe « **lib** » et le suffixe « **.a** » avant de la proposer comme argument de l'option **-l**. Avant de voir tourner l'exécutable sur la console, nous allons lancer la commande **objdump** pour voir le contenu de ce dernier :

```
$ objdump -t programme/prog_lib_static

programme/prog_lib_static:     file format elf32-i386

SYMBOL TABLE:
08048154 l    d  .interp 00000000              .interp
08048168 l    d  .note.ABI-tag 00000000        .note.ABI-tag
...
08048560 g     F .text  00000002              __libc_csu_fini
0804a024 g     O .data  00000004              ma_variable
00000000  w      *UND*  00000000              _ITM_
deregisterTMCloneTable
...
08048578 g     O .rodata  00000004              _fp_hw
0804849c g     F .text  00000014              ma_fonction
0804a028 g       .bss 00000000              __bss_start
0804844d g     F .text  0000004f              main
080484b0 g     F .text  0000001c              mon_addition
...
00000000  w      *UND*  00000000              _ITM_
registerTMCloneTable
080484cc g     F .text  0000001e              ma_soustraction
080482d4 g     F .init  00000000              _init
```

La récupération de la partie du résultat de l'exécution de la commande **objdump** (affichée ci-dessus), montre que tous les éléments de la bibliothèque statique figurent dans le fichier exécutable, y compris la fonction **ma_soustraction()**. On voit aussi que le type des éléments est défini dans cette table des symboles : **.text** pour les trois fonctions (**ma_fonction()**, **mon_addition()** et **ma_soustraction()**) et **.data** pour la variable **ma_variable**. Après avoir vérifié que tout est bien placé, on va lancer notre exécutable et voir le résultat du travail élaboré depuis les pages précédentes :

```
$ ./programme/prog_lib_static
Bienvenue dans ma_fonction
Ma_variable = 3
Bienvenue dans mon_addition
Ma_variable + 2 = 5
```

On garde toujours la casquette du développeur cherchant la bibliothèque qui lui convient, mais cette fois il opte pour la bibliothèque dynamique **libma_lib_dynamic.so** qu'il va appeler lors de l'édition des liens. La commande **gcc** à lancer est presque la même que celle de la bibliothèque statique, sauf que l'option **-l** sera suivie de l'intitulé de la bibliothèque dynamique :

```
$ gcc programme/prog.c -lma_lib_dynamic -I include/ -L lib/ -o
programme/prog_lib_dynamic
```

prog_lib_dynamic est le nom de notre exécutable qui se trouve lui aussi dans **workspace/programme**. Dans le cas présent, spécifier le chemin relatif ainsi que le nom de la bibliothèque dynamique ne servira pas à intégrer la définition des éléments requis dans l'exécutable, mais il sera utile pour résoudre les références ; en d'autres termes pour indiquer l'emplacement de définition des éléments appelés dans l'exécutable et qui ne sont pas définis dans ce dernier. Pour confirmer ce qui a été dit, jetons un coup d'œil sur le contenu du fichier exécutable qu'on vient de générer moyennant la commande **objdump**.

```
$ objdump -t programme/prog_lib_dynamic
programme/prog_lib_dynamic:     file format elf32-i386
SYMBOL TABLE:
08048154 l    d  .interp 00000000              .interp
08048168 l    d  .note.ABI-tag 00000000        .note.ABI-tag
...
080486a0 g     F .text  00000002              __libc_csu_fini
0804a028 g     O .bss 00000004              ma_variable
00000000  w      *UND*  00000000              _ITM_
deregisterTMCloneTable
...
080486b8 g     O .rodata  00000004              _fp_hw
00000000       F *UND*  00000000              ma_fonction
0804a028 g       .bss 00000000              __bss_start
080485dd g     F .text  0000004f              main
00000000       F *UND*  00000000              mon_addition
00000000  w      *UND*  00000000              _Jv_RegisterClasses
0804a028 g     O .data  00000000              .hidden __TMC_END__
```

La partie extraite de la table des symboles de notre exécutable prouve que ce dernier fait référence aux trois éléments de la bibliothèque dont il a besoin : **mon_addition()**, **ma_fonction()** et **ma_variable**. Par opposition à l'emploi de la bibliothèque statique, le fichier exécutable actuel ne fait pas référence à la fonction **ma_soustraction()** puisqu'il n'en a pas besoin. Je tiens à attirer votre attention sur l'expression « faire référence » pour mettre en évidence la différence entre l'emploi des bibliothèques statiques et des bibliothèques dynamiques. La comparaison des tables de symboles nous aide à mettre tout au clair : la variable **ma_variable** est placée dans la section **.data** lors de

l'utilisation de la bibliothèque statique, cette section contient les données initialisées ; lorsqu'on utilise la bibliothèque dynamique la variable **ma_variable** est placée dans la section **.bss** qui contient les variables non initialisées. Les fonctions **ma_fonction()** et **mon_addition()** ont le symbole ***UND*** signifiant qu'elles ne sont pas définies dans l'exécutable lors de l'utilisation des bibliothèques dynamiques alors que pour l'utilisation des bibliothèques statiques, ces deux fonctions ainsi que la fonction **ma_soustraction()** se trouvent dans la section **.text** qui contient le code.

Retournons à notre exécutable qui est normalement prêt à l'emploi. Il faut bien signaler que le fait de le lancer tout de suite provoquera un message d'erreur nous indiquant qu'il n'arrive pas à charger la bibliothèque partagée. Ceci est tout à fait compréhensible puisque l'éditeur des liens dynamiques n'a pas l'habitude d'aller chercher ses bibliothèques dans notre dossier **workspace/lib** ; on a donc intérêt d'ajouter le dossier indiqué dans la liste des répertoires où l'éditeur de lien dynamique doit chercher la bibliothèque à travers la commande suivante :

```
$ export LD_LIBRARY_PATH=./lib/
```

Désormais on peut dire que tout est dans l'ordre ; le lancement de la commande ci-dessous fera tourner le programme principal en utilisant la bibliothèque partagée et affichera sur la console le même résultat que l'exécution du programme faisant appel à la bibliothèque statique :

```
$ ./programme/prog_lib_dynamic
```

Conclusion

Pour couronner le tout, on peut dire qu'on a d'abord mis l'accent sur l'utilité des bibliothèques dans un programme c, ensuite on a clarifié les différences entre bibliothèque statique et bibliothèque dynamique, puis on a réalisé ces deux dernières et finalement on les a utilisées dans nos programmes principaux.

On tient tout de même à préciser que cet article ne cerne pas la totalité des aspects de l'utilisation des bibliothèques surtout dynamiques, on peut citer à titre d'exemple le chargement d'une bibliothèque lors de l'exécution, la gestion des versions d'une bibliothèque et bien d'autres aspects. ■

DES GIGA-OCTETS AUX GIBI-OCTETS

Tristan Colombo

Nous utilisons tous les jours des préfixes pour désigner des capacités mémoire mais il faut savoir que beaucoup de personnes n'utilisent pas les bonnes unités pour définir ces quantités. Je vous propose donc de faire un point rapide sur la norme en vigueur.

Mots-clés : Capacité, Mémoire, Préfixe, Binaire, Norme

Résumé

Lorsqu'une normalisation intervient après des années de mauvaises pratiques, il est difficile de faire accepter les nouveaux usages. C'est le cas avec les unités de dénombrement de capacités mémoire qui ont été mal employées et qui, malgré une normalisation de pratiquement vingt ans, sont toujours utilisées de manière erronée.

Lorsque l'on doit exprimer une quantité de mémoire en informatique, on utilise des multiples de l'octet qui est composé de 8 bits. En anglais on parle de *byte* : il s'agissait à l'origine d'une unité qui pouvait contenir un nombre de bits qui pouvait être différent de 8 (un « multiplet ») mais qui représente maintenant la plupart du temps 8 bits (un `byte` en Java contient 8 bits, le type `bytes` en python est une liste d'octets, etc.).

Pour rappel, un octet étant un ensemble de 8 bits, on peut y stocker un entier dont la valeur est comprise entre `0` (`00000000b`) et `255` (`11111111b` soit $2^0 + 2^1 + ... + 2^7 = 1 + 2 + ... + 128 = 255$).

D'après le système international, si nous voulons exprimer une quantité de `1000` octets, nous pourrons utiliser le préfixe **kilo** dont la fonction est de multiplier par `1000` (10^3) une valeur. Nous savons bien qu' `1km` vaut `1000` mètres, qu' `1kg` vaut `1000` grammes, etc. Donc, pourquoi penser qu'il puisse en être autrement avec les octets ? `1ko` vaut `1000` octets ! Malheureusement, en informatique les capacités mémoires sont calculées en utilisant des puissances de deux ou plus précisément des multiples de 2^{10}, c'est-à-dire la quantité d'informations qui peut être stockée sur `1024` octets, etc. Ainsi pour nous « `1ko` » correspond en fait à 1×2^{10} octets. Bien entendu 2^{10} ne vaut pas `1000` mais `1024` ce qui génère une erreur de `24` bits. On ne va quand même pas se formaliser pour quelques bits ! Et pourtant...

On commence à voir sur le marché des disques durs de `10To`. Sur une telle quantité de mémoire, l'erreur qui paraissait minime précédemment va prendre beaucoup plus d'ampleur. Ainsi, on peut s'attendre en achetant un tel disque à disposer d'une capacité de stockage de `(10 x 1024)Go` ou `(10 x 1024 x 1024)Mo` soit `(10 x 1024 x 1024 x 1024)ko` c'est-à-dire $10 \times 1024^4 = 10 \times 2^{40}$ octets. Or, d'après le système international, votre disque dur ne pourra stocker en réalité « que » 10^{13} octets. La différence est de l'ordre de 10^{12} octets soit `10%` de la capacité du disque dur !

Le besoin de normalisation est évident et vous ne le saviez peut-être pas mais une nouvelle norme a bien été adoptée... en 1999 ! C'est la Commission Electrotechnique Internationale (IEC pour *International Electrotechnical Commission*) qui a publié cette norme sous le nom CEI 60027-2 **[1]**. Elle a reçu l'appui du Bureau International des Poids et Mesures (BIPM) **[2]**, de

Unités de calcul d'une capacité mémoire

Usage correct			Mésusage		Bilan
Symbole	Nom complet	Valeur en octets	Symbole	Valeur en octets	Taux d'erreur
Kio	Kibi-octet	2^{10} = 1024	ko	10^3 = 1000	2 %
Mio	Mébi-octet	2^{20} = 1 048 576	Mo	10^6 = 1 000 000	5 %
Gio	Gibi-octet	2^{30}	Go	10^9	7 %
Tio	Tébi-Octet	2^{40}	To	10^{12}	10 %
Pio	Pébi-octet	2^{50}	Po	10^{15}	13 %

l'*Institute of Electrical and Electronics Engineers* (IEEE) et du *National Institute of Standards and Technology* (NIST). Le suffixe « bi » est employé pour indiquer une puissance de 2 : « kibi » au lieu de « kilo », « mébi » au lieu de « méga », etc. Le tableau suivant résume ces unités : tableau ci-dessus.

Deux remarques sur ce tableau :

- Le calcul du taux d'erreur se fait en divisant le résultat de la différence entre les deux valeurs (usage correct - mésusage) par la valeur de l'usage correct. Par exemple, pour le **Kio** le calcul est : **(1024 - 1000) / 1024 = 24 / 1000 = 2,4 %** ;
- Le symbole du kibi-octet est **Kio** avec un « K » majuscule car le symbole associé au préfixe binaire est « Ki » et non « ki ».

Revenons à notre disque dur maintenant que nous disposons des bonnes unités. Nous allons pouvoir énoncer plus clairement la différence de capacité : ce disque vendu pour **10To** peut ne contenir que **9Tio**... Le fabricant ne peut pas être tenu pour responsable de cet état de fait : il annonce bien sûr son produit **10To** donc **10 x** 10^{12} octets. Avec un peu de chance, il n'utilise pas la bonne unité et vous aurez un disque de **10Tio**, sinon ce sera **9Tio**. La seule manière d'être certain de ce que vous allez acheter est de trouver un fabricant qui utilise les préfixes binaires.

Conclusion

Il n'y a pas grand-chose à changer à nos habitudes pour respecter de manière stricte les normes du système international. En effet, nous utilisons rarement le nom complet des unités et préférons employer leur symbole. Remplacer un « k » minuscule par un « K » majuscule et insérer un « i » devant le « o » de « octets », ce n'est quand même pas très compliqué... ■

Références

[1] Wikipedia, *« IEC 60027 »* : https://en.wikipedia.org/wiki/IEC_60027

[2] Bureau International des Poids et Mesures, *« The International System of Units (SI) »*, 2008, p. 121 : http://www.bipm.org/utils/common/pdf/si_brochure_8_en.pdf

CLUSTER MYSQL 5.6 ET BASCULE AUTOMATIQUE AVEC MYSQLFAILOVER

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

Vous cherchez un cluster MySQL maître-esclave qui bascule automatiquement si le maître vient à faillir ? Ne cherchez plus, depuis la version 5.6, MySQL propose l'utilitaire mysqlfailover qui automatise tout ce processus. Voici le « how-to » de mise en place de ce type de cluster.

apt-get mysql mysqlfailover haute-disponibilité Debian Jessie

L'OBJECTIF

Afin que les données de votre base de données MySQL soient hautement disponibles, un choix classique consiste à installer un cluster de deux machines. La première assure le service en recevant les requêtes. La seconde quant à elle se tient prête à prendre la main en cas de défaillance du premier nœud. On parle de cluster maître-esclave car à un moment `T` un seul nœud rend le service quand l'autre attend.

MySQL comprend différents mécanismes de réplication des données mais il aura fallu attendre la version 5.6 pour voir apparaître le programme `mysqlfailover` permettant de déléguer à un outil la bascule de notre cluster maître-esclave. Nous nous proposons ici d'illustrer la mise en place d'un cluster MySQL 5.6 en mode maître-esclave et d'expliquer l'utilisation de la commande `mysqlfailover`.

LES OUTILS

- Un système GNU/Linux **Debian** en version **Jessie** ;
- gestionnaire de paquets Debian `apt-get` ;
- `vim` (paquet `vim`) ou tout autre éditeur de code.

PHASE 1

Installation de MySQL

Un mot rapide sur la topologie voulue pour notre cluster :

- machine maître : `jessie1` `192.168.1.10` ;
- machine esclave : `jessie2` `192.168.1.11`.

Nous installons MySQL 5.6 sur deux serveurs installés en Debian stable Jessie. MySQL 5.6 n'est pas présent par défaut dans Debian Jessie donc nous choisissons d'utiliser les paquets Debian mis à disposition par le projet MySQL.

Nous commençons par installer la clé des dépôts Apt pour MySQL sur les deux serveurs :

```
# apt-key adv --keyserver pgp.mit.edu --recv-keys 8C718D3B5072E1F5
```

Afin de télécharger les paquets du dépôt Apt pour MySQL, nous ajoutons le fichier **/etc/apt/sources.list.d/mysql.list** avec le contenu suivant sur les deux serveurs :

```
deb http://repo.mysql.com/apt/debian/ jessie mysql-5.6
deb-src http://repo.mysql.com/apt/debian/ jessie mysql-5.6
```

Puis nous installons les paquets Debian du serveur, du client et des utilitaires MySQL sur les deux serveurs :

```
# apt-get install mysql-server mysql-client mysql-utilities
```

PHASE 2

Configuration du cluster MySQL

Sur **jessie1**, nous créons l'utilisateur suivant qui sera dédié à la réplication. Ne le créez pas sur **jessie2**, car dès que la réplication sera mise en place, elle essaierait de le créer également sur le nœud esclave et vu que l'enregistrement existerait déjà, la réplication ne s'initialiserait pas correctement :

```
root@jessie1# mysql -uroot -p
mysql> create user 'repl-user'@'192.168.1.10' identified by 'S3cr3tP4sS';
mysql> create user 'repl-user'@'jessie1' identified by 'S3cr3tP4sS';
mysql> grant all on *.* to 'repl-user'@'192.168.1.10' with grant option;
mysql> grant all on *.* to 'repl-user'@'jessie1' with grant option;
mysql> create user 'repl-user'@'192.168.1.11' identified by 'S3cr3tP4sS';
mysql> create user 'repl-user'@'jessie2' identified by 'S3cr3tP4sS';
mysql> grant all on *.* to 'repl-user'@'192.168.1.11' with grant option;
mysql> grant all on *.* to 'repl-user'@'jessie2' with grant option;
```

Nous modifions maintenant le fichier **/etc/mysql/my.cnf** sur **jessie1** (le maître) afin d'activer la réplication par fichier binaire en rajoutant les lignes suivantes :

```
log-bin = mysql-bin
server-id = 1
binlog_format = MIXED
log-slave-updates = true
enforce-gtid-consistency = true
gtid-mode=ON
master-info-repository=TABLE
report-host = jessie1
```

Puis nous redémarrons MySQL sur le nœud **jessie1** :

```
root@jessie1# systemctl restart mysql
```

Nous modifions maintenant le fichier **/etc/mysql/my.cnf** sur **jessie2** (l'esclave) afin d'activer la réplication par fichier binaire en rajoutant les lignes suivantes :

```
log-bin = mysql-bin
server-id = 2
binlog_format = MIXED
log-slave-updates = true
enforce-gtid-consistency = true
gtid-mode=ON
master-info-repository=TABLE
report-host = jessie2
```

Puis nous redémarrons MySQL sur le nœud **jessie2** :

```
root@jessie2# systemctl restart mysql
```

PHASE 3

Démarrage de la réplication MySQL

Nous sommes maintenant en position d'activer la réplication sur notre cluster. Pour cela il suffit de passer la commande suivante sur notre nœud esclave **jessie2** afin de lui donner l'ordre de se soumettre à son maître :

```
root@jessie2# mysql -uroot -p
mysql> CHANGE MASTER TO MASTER_
USER='repl-user', MASTER_
PASSWORD='S3cr3tP4sS', MASTER_
HOST='192.168.1.10';
Query OK, 0 rows affected, 2 warnings
(0.03 sec)
```

Cette commande indique qui est le maître et comment s'y connecter. Nous démarrons maintenant la réplication en activant l'esclave **jessie2** :

```
mysql> start slave;
Query OK, 0 rows affected (0.01 sec)
```

Faisons maintenant un point de la situation avec la commande suivante :

```
mysql> show slave status\G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 192.168.1.10
                  Master_User: repl-user
                  Master_Port: 3306
        ...
          Retrieved_Gtid_Set: 0a907902-62b9-11e5-9148-
080027b5abd5:1-2
           Executed_Gtid_Set: 0a907902-62b9-11e5-9148-
080027b5abd5:1-2
              Auto_Position: 0
1 row in set (0.00 sec)
```

Les trois premières lignes sont les plus intéressantes, à savoir l'adresse IP du maître, l'utilisateur avec lequel nous y accédons et l'état de notre esclave qui est ici en attente des données du maître. Il est temps de tester la réplication au niveau du maître **jessie1** en créant une base de données test avec une table client. Puis nous ajouterons du contenu à cette table et vérifierons qu'il est bien répliqué sur le nœud esclave **jessie2** :

```
mysql> create database test;
Query OK, 1 row affected (0.00 sec)
mysql> use test
mysql> create table client (id int primary key, name text not
null);
Query OK, 0 rows affected (0.01 sec)
mysql> insert into client values (1, 'brasserie tsoin tsoin');
Query OK, 1 row affected (0.00 sec)
mysql> insert into client values (2, 'librairie au bon bouquin');
Query OK, 1 row affected (0.01 sec)
mysql> select * from client;
+----+------------------------+
| id | name                   |
+----+------------------------+
|  1 | brasserie tsoin tsoin  |
|  2 | librairie au bon bouquin |
+----+------------------------+
2 rows in set (0.00 sec)
```

La base de données test et la table client ont bien été créées. Nous vérifions maintenant l'existence de la base de données et de la table sur l'esclave **jessie2** :

```
mysql> use test
mysql> select * from client;
+----+------------------------+
| id | name                   |
+----+------------------------+
|  1 | brasserie tsoin tsoin  |
|  2 | librairie au bon bouquin |
+----+------------------------+
2 rows in set (0.00 sec)
```

La base de données test et la table client ont bien été répliquées sur le nœud esclave **jessie2**. La réplication est donc en place entre les deux nœuds et le maître alimente l'esclave en données à chaque modification de la base de données.

PHASE 4

Mise en place de la bascule automatique

Afin d'assurer la bascule automatique du service vers le nœud esclave, nous allons maintenant mettre en place **mysqlfailover**. Nous commençons par vérifier l'état du cluster à l'aide du paramètre **check** de la commande **mysqlfailover** :

```
# mysqlfailover --master=repl-user:S3cr3tP4sS@jessie1
--slaves=repl-user:S3cr3tP4sS@jessie2 check
Replication Health Status
+----------+-------+---------+--------+-----------+---------+
| host     | port  | role    | state  | gtid_mode | health  |
+----------+-------+---------+--------+-----------+---------+
| jessie1  | 3306  | MASTER  | UP     | ON        | OK      |
| jessie2  | 3306  | SLAVE   | UP     | ON        | OK      |
+----------+-------+---------+--------+-----------+---------+
```

Les deux nœuds apparaissent bien dans la sortie de la commande. Cette dernière est rafraîchie régulièrement. Vous pouvez quitter avec la touche **<Q>**.

Nous sommes maintenant prêts à armer la bascule automatique du cluster. Pour cela nous utiliserons également la commande **mysqlfailover**, mais avec l'argument **auto** en lieu et place de **check** utilisé précédemment :

```
# mysqlfailover --master=repl-user:S3cr3tP4sS@jessie1
--slaves=repl-user:S3cr3tP4sS@jessie2 auto --log=/var/log/
mysql/mysqlfailover.log --daemon=start --pidfile /var/run/
mysqlfailover.pid
NOTE: Log file '/var/log/mysql/mysqlfailover.log' does not exist.
Will be created.
Starting failover daemon...
```

Nous avons également utilisé les options **--log=/var/log/mysql/mysqlfailover.log** pour définir un fichier journal que nous pourrons consulter afin de suivre l'état de la bascule automatique, et **--pidfile /var/run/mysqlfailover.pid** pour le démon **mysqlfailover**.

Si vous souhaitez stopper le démon, il suffit de passer la valeur **stop** à l'option **--daemon** :

```
# mysqlfailover --master=repl-user:S3cr3tP4sS@jessie1 --slaves=repl-
user:S3cr3tP4sS@jessie2 auto --log=/var/log/mysql/mysqlfailover.log
--daemon=stop --pidfile /var/run/mysqlfailover.pid
Stopping failover daemon...
```

PROFESSIONNELS DES TICE, COLLECTIVITÉS, ÉCOLES D'INGÉNIEURS, UNIVERSITÉS, R & D, ENSEIGNANTS, ...

LINUX MAGAZINE / FRANCE

PROFESSIONNELS !

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS ...

...EN VOUS CONNECTANT À L'ESPACE DÉDIÉ AUX PROFESSIONNELS SUR :

www.ed-diamond.com

PDF COLLECTIFS PRO

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

Prix TTC en Euros / France Métropolitaine

PROFESSIONNELS :
N'HÉSITEZ PAS À NOUS CONTACTER POUR UN DEVIS PERSONNALISÉ PAR E-MAIL :
abopro@ed-diamond.com
OU PAR TÉLÉPHONE :
03 67 10 00 20

ACCÈS COLLECTIFS BASE DOCUMENTAIRE PRO

OFFRE	ABONNEMENT	1 - 5 connexion(s) Réf	1 - 5 connexion(s) Tarif TTC	6 - 10 connexions Réf	6 - 10 connexions Tarif TTC	11 - 25 connexions Réf	11 - 25 connexions Tarif TTC
PROLM+3	GLMF + HS	☐ PRO LM+3/5	267,-	☐ PRO LM+3/10	534,-	☐ PRO LM+3/25	1068,-
PROA+3	GLMF + HS + LP + HS	☐ PRO A+3/5	297,-	☐ PRO A+3/10	594,-	☐ PRO A+3/25	1188,-
PROH+3	GLMF + HS + LP + HS + MISC + HS + OS	☐ PRO H+3/5	447,-	☐ PRO H+3/10	894,-	☐ PRO H+3/25	1788,-

Prix TTC en Euros / France Métropolitaine

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

SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE CI-DESSUS ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

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

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

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

LINUX MAGAZINE / FRANCE

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

Le démon **mysqlfailover** est maintenant chargé sur l'esclave. Nous allons donc tester la bascule en arrêtant le démon mysql sur le nœud maître :

```
# systemctl stop mysql
```

Nous constatons les entrées suivantes écrites par le démon **mysqlfailover** dans le fichier journal **/var/log/mysql/mysqlfailover.log** :

```
2015-09-24 16:37:05 PM INFO host: jessie1, port: 3306, role: MASTER, state: UP, gtid_mode: ON, health: OK
2015-09-24 16:37:05 PM INFO host: jessie2, port: 3306, role: SLAVE, state: UP, gtid_mode: ON, health: OK
2015-09-24 16:37:20 PM INFO Master may be down. Waiting for 3 seconds.
2015-09-24 16:37:23 PM INFO Cannot reconnect to master.
2015-09-24 16:37:26 PM CRITICAL Master is confirmed to be down or unreachable.
2015-09-24 16:37:26 PM INFO Failover starting in 'auto' mode...
2015-09-24 16:37:26 PM INFO Candidate slave jessie2:3306 will become the new master.
2015-09-24 16:37:26 PM INFO Checking slaves status (before failover).
2015-09-24 16:37:26 PM INFO Preparing candidate for failover.
2015-09-24 16:37:26 PM INFO Creating replication user if it does not exist.
2015-09-24 16:37:26 PM INFO Stopping slaves.
2015-09-24 16:37:26 PM INFO Performing STOP on all slaves.
2015-09-24 16:37:26 PM INFO Switching slaves to new master.
2015-09-24 16:37:26 PM INFO Disconnecting new master as slave.
2015-09-24 16:37:26 PM INFO Starting slaves.
2015-09-24 16:37:26 PM INFO Performing START on all slaves.
2015-09-24 16:37:26 PM INFO Checking slaves for errors.
2015-09-24 16:37:26 PM INFO Failover complete.
2015-09-24 16:37:31 PM INFO Unregistering existing instances from slaves.
2015-09-24 16:37:31 PM INFO Registering instance on new master jessie2:3306.
2015-09-24 16:37:31 PM INFO Master Information
2015-09-24 16:37:31 PM INFO Binary Log File: mysql-bin.000003, Position: 191, Binlog_Do_DB: N/A, Binlog_
Ignore_DB: N/A
2015-09-24 16:37:31 PM INFO GTID Executed Set: 0a907902-62b9-11e5-9148-080027b5abd5:1-17
2015-09-24 16:37:31 PM INFO Getting health for master: jessie2:3306.
2015-09-24 16:37:31 PM INFO Health Status:
2015-09-24 16:37:31 PM INFO host: jessie2, port: 3306, role: MASTER, state: UP, gtid_mode: ON, health: OK
```

Nous constatons que le démon **mysqlfailover** a successivement détecté que le maître **jessie1** est devenu injoignable, qu'il a attendu 3 secondes avant de réessayer d'y accéder, puis qu'il a effectué la bascule vers le nœud esclave.

L'étape suivante consiste à vérifier le statut du nœud restant dans le cluster. Pour cela nous passons les commandes suivantes :

```
# mysql -uroot -p
mysql> show master status;
+----------------+----------+-------------+-----------------+-----------------------------------+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set                 |
+----------------+----------+-------------+-----------------+-----------------------------------+
| mysql-bin.000003 |      191 |             |                 | 0a907902-62b9-11e5-9148-
080027b5abd5:1-17 |
+----------------+----------+-------------+-----------------+-----------------------------------+
1 row in set (0.00 sec)
mysql> show slave status\G;
Empty set (0.00 sec)
```

La première commande nous fournit des informations sur le statut du maître actuel et la seconde vérifie le statut des esclaves du cluster, qui est logiquement vide.

Tentons maintenant d'ajouter une nouvelle ligne dans notre table client de la base de données test sur le nœud **jessie2** ayant récupéré le service :

```
mysql> use test
mysql> insert into client values (4,
'restaurant chez robert');
Query OK, 1 row affected (0.01 sec)
mysql> select * from client;
+---+-----------------------+
| id | name                  |
+---+-----------------------+
|  1 | brasserie tsoin tsoin    |
|  2 | librairie au bon bouquin |
|  3 | boucherie mignon cochon  |
|  4 | restaurant chez robert   |
+---+-----------------------+
4 rows in set (0.00 sec)
```

L'ajout d'une ligne à la table client s'est déroulé sans problème. Le service MySQL s'exécute donc correctement sur l'ex-nœud esclave après avoir basculé depuis le nœud maître.

PHASE 5

Ajout d'un esclave et réarmer le cluster

Avant d'ajouter un nouvel esclave au cluster, à savoir l'ancien maître **jessie1** redevenu disponible, nous allons arrêter le démon de bascule automatique :

```
# mysqlfailover --daemon stop --pidfile
/var/run/mysqlfailover.pid
Stopping failover daemon...
```

Puis nous redémarrons maintenant le MySQL arrêté sur l'ancien nœud maître **jessie1** :

```
# systemctl start mysql
```

On déclare **jessie1** en nouveau nœud esclave du maître actuelle **jessie2** :

```
mysql> CHANGE MASTER TO MASTER_USER='repl-
user', MASTER_PASSWORD='S3cr3tP4sS',
MASTER_HOST='192.168.1.11';
Query OK, 0 rows affected, 2 warnings
(0.02 sec)
mysql> start slave;
Query OK, 0 rows affected (0.01 sec)
mysql> show slave status\G;
```

Vérifions maintenant que le nouvel esclave voit bien des données répliquées depuis le nœud maître actuel **jessie2** :

```
mysql> insert into client values (5, 'poissonnerie la manche');
```

Vous devez voir apparaître immédiatement la ligne sur le nouveau nœud esclave **jessie1** :

```
mysql> select * from client;
+----+------------------------+
| id | name                   |
+----+------------------------+
|  1 | brasserie tsoin tsoin  |
|  2 | librairie au bon bouquin |
|  3 | boucherie mignon cochon |
|  4 | restaurant chez robert |
|  5 | poissonnerie la manche |
+----+------------------------+
5 rows in set (0.00 sec)
```

La dernière étape de cet *how-to* consiste à réarmer le démon **mysqlfailover** afin d'activer la bascule automatique en cas de perte du maître. Nous lançons la commande suivante sur l'actuel nœud esclave **jessie1** :

```
# mysqlfailover --master=repl-user:S3cr3tP4sS@jessie2
 --slaves=repl-user:S3cr3tP4sS@jessie1 auto --log=/var/log/
mysql/mysqlfailover.log --daemon=start --pidfile /var/run/
mysqlfailover.pid
```

Votre cluster est maintenant armé et prêt à basculer si le nœud maître devient inaccessible.

RÉSULTAT

À l'issue de ce *how-to* nous avons un cluster MySQL en version 5.6 dont la bascule est automatisée à l'aide de **mysqlfailover**. Une fois le cluster armé, la bascule est automatique et intervient dès que l'absence du nœud maître est confirmée.

On peut imaginer de nombreux cas d'utilisation de ce type de clusters, d'un site internet à fort trafic souhaitant renforcer son infrastructure contre les pannes à une application lourde nécessitant constamment l'accès à une base de données. Il s'agit d'une solution complètement intégrée au projet MySQL et relativement simple à mettre en place qui s'avère robuste à l'utilisation. Donc que des bonnes raisons de lui confier nos données ! ■

ORCHESTREZ VOS CONTENEURS DOCKER AVEC DOCKER COMPOSE

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

Gérer un conteneur Docker ne présente pas de difficultés particulières. La tâche est autrement plus complexe lorsqu'on commence à gérer plusieurs conteneurs Docker qui interagissent les uns avec les autres. Heureusement une solution existe : Docker Compose.

Mots-clés : Docker compose, Conteneur, Nginx, PHP, Debian

Résumé

Il est rare qu'un seul conteneur Docker suffise pour fournir le service nécessaire. Plus communément deux ou trois services interagissent de concert. La difficulté représentée par l'ordonnancement de ces services s'accroît lorsqu'ils sont chacun dans un conteneur Docker, la gestion de l'image associée et la gestion de ce conteneur s'imposant aux administrateurs du service. La solution proposée ici se nomme Docker Compose.

Docker Compose représente une solution élégante et simple à prendre en main que nous allons présenter dans cet article pour faire fonctionner ensemble notre serveur Web **Nginx** et **PHP5-fpm**, démon auquel Nginx va déléguer l'interprétation du code PHP.

1 Installer Docker et Docker Compose

Nous commençons par installer Docker. Le système d'exploitation hôte de nos conteneurs sera une Debian stable « Jessie ». Sous Debian Jessie, nous allons nous servir du dépôt officiel **Debian Backports** qui contient le paquet officiel Debian `docker.io`. En effet, il a été décidé de placer ce paquet dans un dépôt évoluant plus rapidement que les dépôts dédiés à la distribution stable car Docker demeure un programme qui évolue rapidement et qu'une bonne partie du public souhaite mettre à jour très régulièrement.

Nous commençons par vérifier la présence du dépôt Backports dans notre fichier `/etc/apt/sources.list` :

```
deb http://ftp.fr.debian.org/debian/ jessie-backports main contrib non-free
deb-src http://ftp.fr.debian.org/debian/ jessie-backports main contrib non-free
```

Puis nous procédons à l'installation des paquets `docker.io` et `docker-compose` :

```
# apt-get update && apt-get install docker.io docker-compose
```

Nous voilà prêts à écrire le nécessaire pour obtenir les images de nos futurs conteneurs.

2 Répertoire contenant les fichiers PHP

À la racine de notre projet, nous créons un répertoire contenant les fichiers relatifs au site Web que nous souhaitons servir. Voici l'arborescence utilisée :

```
# tree www.test.com
www.test.com/
ÐÐÐ index.php

Ø directories, 1 file
```

Nous créons donc un répertoire portant le nom de domaine que nous allons utiliser pour nos tests, ici www.test.com. Ce répertoire ne contiendra qu'un seul fichier nommé **index.php** dont le contenu sera :

```
<?php
phpinfo();
?>
```

Il s'agit ici d'un simple appel à la fonction **phpinfo()** qui fournit de nombreux détails sur le PHP installé sur la machine, ce qui sera amplement suffisant dans le cadre de notre exemple d'utilisation de Docker Compose.

3 Nginx

Nginx est le serveur Web qui monte depuis quelques années. Dans le cadre de son utilisation sous forme de conteneur, sa simplicité à se configurer ainsi que ses performances en font un choix pertinent dans la plupart des cas d'utilisation.

À la racine de notre projet, nous créons donc l'arborescente suivante :

```
# tree nginx/
nginx/
ÐÐÐ Dockerfile
ÐÐÐ nginx-data
    ÐÐÐ confs
        ÐÐÐ www.test.com

2 directories, 2 files
```

Notre répertoire **nginx** contient le fichier **Dockerfile** permettant la construction de l'image Docker mais aussi un répertoire **nginx-data** qui contiendra les données associées à Nginx, à savoir dans le dossier **confs** la configuration d'un hôte virtuel pour notre domaine www.test.com.

3.1 Écrire le Dockerfile de Nginx

Rappelons rapidement qu'un **Dockerfile** est le fichier qui contient toutes les informations nécessaires à la création d'une image Docker, à partir de laquelle on instancie par la suite les conteneurs Docker eux-mêmes.

Nous éditons le fichier **nginx/Dockerfile** pour y écrire le contenu suivant :

```
# nginx for Debian Jessie
FROM debian:jessie
MAINTAINER carl.chenet@mytux.fr
RUN apt-get update && apt-get install -y nginx-full && \
    rm -rf /var/lib/apt/lists/* && \
    echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \
    chown -R www-data:www-data /var/lib/nginx && \
    chown -R www-data:root /var/log/nginx && \
    rm /etc/nginx/sites-enabled/default && \
    mkdir -p /var/www/www.test.com

# Copy nginx default configuration
COPY nginx-data/confs/* /etc/nginx/sites-enabled/

# Need a volume for the logs
VOLUME ["/var/log/nginx","/var/www/www.test.com"]

# Define working directory.
WORKDIR /etc/nginx
CMD ["nginx"]
EXPOSE 8Ø
EXPOSE 443
```

Ce fichier mérite quelques explications :

- le mot-clé **FROM** définit la base de l'image Docker sur laquelle nous allons construire notre propre image. Il s'agit ici de l'image officielle de la distribution Linux Debian Jessie ;
- le mot-clé **MAINTAINER** indique le mainteneur du **Dockerfile** ;
- le mot-clé **RUN** quant à lui propose de lancer une commande à la création de l'image ;
- **COPY** indique de copier des fichiers de notre système de fichiers local dans l'image ;
- **VOLUME** indique des répertoires du système de fichiers de l'hôte qui seront montés dans le conteneur, permettant ainsi de faire apparaître des fichiers présents sur l'hôte dans le conteneur ;
- **WORKDIR** indique le positionnement du répertoire de travail lors de la création de l'image ;
- le mot-clé **CMD** va nous permettre de lancer une commande à la fin de la création du conteneur ;

- **EXPOSE** va nous permettre de rendre accessible à l'hôte des ports ouverts par notre conteneur.

3.2 Configuration de Nginx

Il nous manque maintenant la définition de l'hôte virtuel Nginx permettant de définir les caractéristiques de notre application PHP vue par Nginx. Voici le fichier **nginx/nginx-data/confs/www.test.com** :

```
server {
    listen 80;

    server_name www.test.com;
    location / {
        root /var/www/www.test.com;
        index index.php;
        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log error;
        try_files $uri $uri/ /index.php =404;
    }

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000

    location ~ \.php$ {
        fastcgi_pass php5fpmengine:9000;
        fastcgi_index index.php;
        fastcgi_param  QUERY_STRING      $query_string;
        fastcgi_param  REQUEST_METHOD    $request_method;
        fastcgi_param  CONTENT_TYPE      $content_type;
        fastcgi_param  CONTENT_LENGTH    $content_length;

        fastcgi_param  SCRIPT_NAME       $fastcgi_script_name;
        fastcgi_param  SCRIPT_FILENAME   /var/www/www.test.com$fastcgi_
script_name;
        fastcgi_param  REQUEST_URI       $request_uri;
        fastcgi_param  DOCUMENT_URI      $document_uri;
        fastcgi_param  DOCUMENT_ROOT     $document_root;
        fastcgi_param  SERVER_PROTOCOL   $server_protocol;

        fastcgi_param  GATEWAY_INTERFACE CGI/1.1;
        fastcgi_param  SERVER_SOFTWARE   nginx;

        fastcgi_param  REMOTE_ADDR       $remote_addr;
        fastcgi_param  REMOTE_PORT       $remote_port;
        fastcgi_param  SERVER_ADDR       $server_addr;
        fastcgi_param  SERVER_PORT       $server_port;
        fastcgi_param  SERVER_NAME       $server_name;
    }
}
```

Décrivons cette configuration :

- la ligne **root /var/www/www.test.com;** indique la source des données pour Nginx qui correspond à l'un des deux volumes définis dans le **Dockerfile** de Nginx. Ces fichiers seront donc directement hébergés sur l'hôte accueillant les conteneurs Docker de notre projet ;
- le mot-clé **index** définit le nom du fichier qui servira d'index dans notre répertoire **root** ;
- **access_log** et **error_log** prennent des chemins qui correspondent à des fichiers contenus dans notre volume **/var/log/nginx**, ce qui permet de retrouver nos fichiers journaux au niveau de notre hôte et donc d'étudier ces fichiers journaux en cas de problème ;
- pour le mot-clé **fastcgi_pass**, nous utilisons l'alias **php5fpmengine** qui grâce à Docker Compose pointera sur l'adresse IP du conteneur.

Avec le contenu du répertoire **nginx/** nous sommes maintenant prêts à créer l'image Docker nous permettant d'instancier un conteneur qui lancera l'application Nginx servant le site Web répondant au domaine www.test.com.

4 PHP5-fpm

PHP5-fpm est un démon qui va recevoir les requêtes fast-cgi depuis Nginx et les distribue à un groupe de processus traitant les différentes requêtes afin d'interpréter les fichiers PHP interrogés. Nous allons donc écrire une image Docker permettant à ce démon de travailler conjointement avec Nginx pour qu'il soit en mesure d'interpréter le fichier PHP qui constitue notre site www.test.com.

À la racine de notre projet, nous créons l'arborescence suivante :

```
php5-fpm/
ÐÐÐ confs
Ð   ÐÐÐ php5-fpm.conf
Ð   ÐÐÐ www.conf
ÐÐÐ Dockerfile

1 directory, 3 file
```

Le fichier **php5-fpm/confs/Dockerfile** contiendra les commandes nécessaires à la création de notre image Docker. La configuration proprement dite du démon **PHP5-fpm** sera contenue dans les fichiers **php5-fpm/confs/php5-fpm.conf** et **php5-fpm/confs/www.conf**.

4.1 Écrire le Dockerfile pour PHP5-fpm

Notre dossier **php5-fpm** contient un **Dockerfile** qui va permettre de générer notre image Docker mais aussi un dossier **confs** contenant deux fichiers de configuration. Voici le fichier **Dockerfile** :

```
# php5-fpm for Debian Jessie
FROM debian:jessie
MAINTAINER carl.chenet@mytux.fr
```

```
RUN apt-get update && apt-get install -y php5-fpm && mkdir -p
/var/log/php5-fpm
COPY confs/php5-fpm.conf /etc/php5/fpm/php-fpm.conf
COPY confs/www.conf /etc/php5/fpm/pool.d/www.conf
VOLUME ["/var/log/php5-fpm/", "/var/www/www.test.com"]
CMD ["php5-fpm", "-F"]
EXPOSE 9000
```

Ce **Dockerfile** est relativement simple :

- le mot-clé **RUN** installe le paquet **php5-fpm** et crée le répertoire des fichiers journaux que nous souhaitons utiliser ;
- les deux **COPY** indiquent de copier nos deux fichiers de configuration au bon endroit dans le système de fichiers du conteneur ;
- la directive **VOLUME** indique quant à elle que les conteneurs basés sur l'image **php5-fpm** se serviront de deux volumes, l'un pour les fichiers journaux de php5-fpm et l'autre pour accéder au code PHP à exécuter ;
- nous définissons grâce à **CMD** la commande **php5-fpm** ;
- finalement avec **EXPOSE** nous indiquons que le conteneur devra présenter le port **9000** qui nous servira à recevoir les requêtes cgi transmises par Nginx.

4.2 Configurations pour PHP5-fpm

Le fichier **nginx/nginx-data/confs/php5-fpm.conf** n'est pour sa part que le fichier d'origine accessible lors de l'installation du paquet **php5-fpm** sur Debian Jessie, à l'exception de la modification de la ligne suivante :

```
error_log = /var/log/php5-fpm/php5-fpm.log
```

Nous souhaitons en effet enregistrer vers le fichier **/var/log/php5-fpm/php5-fpm.log** le journal des erreurs de php5-fpm. Vous pourriez également être intéressé par décommenter la ligne suivante lors de la mise en place de votre installation afin d'obtenir des informations plus précises que celles communiquées par le journal une fois en production :

```
log_level = debug
```

Ce paramètre est bien évidemment à éviter en environnement de production à cause du remplissage du fichier journal qu'il va générer si le nombre de requêtes est important.

Nous voilà prêts à générer l'image de notre conteneur qui exécutera le code PHP de notre domaine www.test.com.

5 Orchestrer le tout avec Docker Compose

Docker Compose permet la construction d'images, le lancement des conteneurs associés ainsi que leur mise en relation. Et tout cela à travers un point d'entrée unique constitué d'un fichier yaml. Nous commençons par construire l'arborescence suivante :

```
# tree docker-compose/
docker-compose/
ÐÐÐ nginx-php-fpm
    ÐÐÐ docker-compose.yml

1 directory, 1 file
```

Notre répertoire **docker-compose** contiendra un sous-répertoire **nginx-php-fpm** indiquant la suite de services gérés par le **docker-compose.yml** contenue dans ledit répertoire. Ce fichier **docker-compose.yml** est la configuration de notre orchestration pour notre architecture en conteneurs Nginx-PHP5-fpm.

Nous écrivons notre fichier **docker-compose/nginx-php-fpm/docker-compose.yml** avec le contenu suivant :

```
nginxengine:
  build: ../../nginx
  ports:
   - "80:80"
   - "443:443"
  volumes:
   - /var/www/www.test.com:/var/www/www.test.com
   - /var/log/nginx:/var/log/nginx
  links:
   - php5fpmengine
php5fpmengine:
  build: ../../php5-fpm
  ports:
   - "9000:9000"
  volumes:
   - /home/chaica/tests/articles/www.test.com:/var/www/www.test.com
   - /var/log/php5-fpm:/var/log/php5-fpm
```

Détaillons un peu. Nous divisons le fichier **docker-compose.yml** en deux parties. Chacune est appelée à être un service. Le premier, nommé **nginxengine**, est dédié à la gestion de Nginx. Le second, nommé **php5fpmengine**, est consacré à php5-fpm.

- Le mot-clé **build** permet d'indiquer l'emplacement du répertoire de construction de l'image où réside notre **Dockerfile** ;

- le mot-clé **ports** est suivi par une association de deux numéros de ports, celui de gauche définissant le numéro de port qui doit apparaître sur la machine hôte et celui de droite le numéro de port correspondant du service sur le conteneur ;
- avec le paramètre **volumes**, nous effectuons une correspondance entre un répertoire sur notre machine hôte (le chemin de gauche) et le chemin dans le conteneur où apparaîtra le contenu dudit répertoire ;
- enfin le mot-clé **links** lie un conteneur à un autre service. Nous lions ici le service **nginxengine** au service **php5fpmengine**.

Chacun de nos deux services **nginxengine** et **php5fpmengine** construit son image à partir du répertoire indiqué par **build** fait le lien entre le port du conteneur et le port de l'hôte avec **ports** et finalement fait le lien entre des répertoires du conteneur et de l'hôte avec **volumes**.

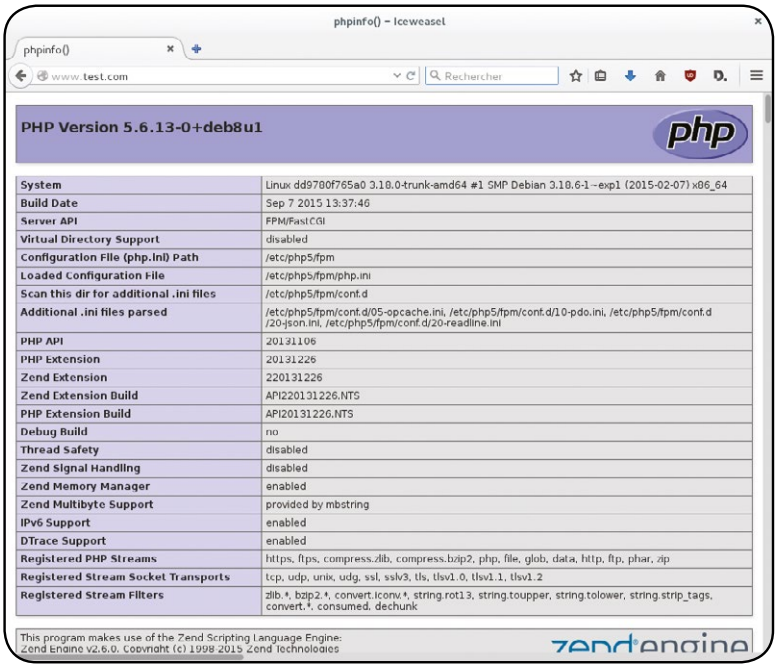


Fig. 1: Affichage du phpinfo()

Nous allons maintenant invoquer la commande **docker-compose**, qui en tant qu'orchestrateur de nos différents conteneurs se chargera de déployer notre infrastructure :

```
# docker-compose up
Building php5fpmengine...
Step 0 : FROM debian:jessie
jessie: Pulling from debian
843e2bded498: Extracting [==================>] 51.36 MB/51.36 MB
8c00acfb0175: Download complete
Successfully built cfd430a77b45
........
Creating nginxphpfpm_nginxengine_1...
Attaching to nginxphpfpm_php5fpmengine_1, nginxphpfpm_nginxengine_1
```

Nous pouvons donc constater que nos conteneurs tournent désormais. Si nous souhaitons reléguer l'exécution de la commande **docker-compose** en tâche de fond, il est possible de conjointement utiliser le paramètre **up** avec l'option **-d** pour détacher qui va exécuter les conteneurs en tâche de fond.

Nous rajoutons maintenant la ligne suivante dans notre fichier **/etc/hosts** afin de pouvoir utiliser le nom de domaine www.test.com pour nos tests :

```
127.0.0.1   localhost www.test.com
```

Ainsi nous sommes assurés que nos requêtes vers le nom de domaine www.test.com sont bien renvoyées sur notre interface locale.

Puis nous entrons le nom de domaine www.test.com dans notre navigateur Web afin de vérifier les bonnes créations des images et exécutions des conteneurs par **docker-compose**. Le résultat de l'exécution de **phpinfo()** s'affiche bien dans notre navigateur comme le montre la figure 1.

Lors de vos tests, vous aurez sans nul doute besoin de supprimer les conteneurs avant de les recréer. Il suffit pour cela de passer la commande suivante :

```
# docker-compose rm -f
```

Conclusion

Au cours de cet article, nous avons détaillé la création de deux services interconnectés et gérés par Docker Compose. Fini les lignes de commande à rallonge pour lancer vos conteneurs, **docker-compose** s'occupe de tout !

À travers une syntaxe en YAML simple et efficace, la description par service permet de construire l'image associée à un service puis de lancer le conteneur en gérant ses ports, ses volumes et les liens qu'il aura avec d'autres services. Docker Compose s'avère vite indispensable, surtout quand le nombre de conteneurs interconnectés augmente. ■

Maker Faire® Paris

Présenté par Leroy Merlin

30 avril & 1er mai 2016

Paris expo porte de Versailles - Foire de Paris

APPEL AUX MAKERS !

Nous recherchons

Bricoleurs ★ Ingénieurs

Amateurs ⚓ **Hackers** ⚓ Geeks

Chercheurs ★ Inventeurs

Entrepreneurs ◊ **Rêveurs**

Inscription sur

makerfaireparis.com


MISE EN PLACE DE TESTS UNITAIRES EN PYTHON AVEC UNITTEST

Tristan Colombo

Personne n'est à l'abri d'une erreur : erreur d'inattention, modification d'un programme sous pression, mésusage d'une fonction, etc. Si l'on ne peut pas échapper aux erreurs, il existe un moyen de prévenir leurs apparitions : les tests unitaires.

erreurs
Tests unitaires fixtures
unittest Python

L'OBJECTIF

Nous allons ajouter des tests unitaires au programme **Roman.py** publié dans l'article *« Ils sont fous ces romains ! »* de GNU/Linux Magazine n°188. Ceci nous permettra de détecter automatiquement l'erreur qui était présente...

LES OUTILS

Seulement Python3.x et sa bibliothèque standard de modules.

PHASE 1 Comprendre les tests unitaires

Les tests unitaires permettent de vérifier (de tester) des éléments particuliers d'un programme. Par exemple, si un objet contient de nombreuses méthodes, les tests unitaires permettront de vérifier le fonctionnement de chacune des méthodes et de s'assurer ainsi que le comportement de l'objet est bien celui souhaité. De plus, la mise en place de tests unitaires permet de s'assurer que la correction de bugs n'entraînera pas des régressions au niveau du code. En effet, en schématisant, si l'on a un programme possédant trois ensembles de code **A**, **B** et **C** où **B** utilise **C** et **A** utilise **B**, le fait de corriger une erreur sur **C** peut entraîner une erreur sur **B** puis sur **A** (puisque vous aviez fait en sorte que **A** fonctionne malgré l'erreur de **C**...).

En Python, il existe de nombreux moyens de mettre en place les tests unitaires. J'ai choisi ici **unittest** pour son compromis entre simplicité et efficacité. De plus, ce module est disponible nativement avec Python et l'on peut retrouver sa structure en C, Perl, Java et Smalltalk car il est basé sur le modèle de *framework* Xunit imaginé par Kent Beck et Erich Gamma (oui, le même que celui qui a écrit *« Design Patterns : Elements of Reusable Object-Oriented Software »*).

PHASE 2 Le module unittest

Vous pouvez utiliser unittest immédiatement puisqu'il fait partie de la bibliothèque standard de modules Python. Avant de l'appliquer à notre programme, il faut exécuter quelques tests pour se familiariser avec l'outil. J'aurai besoin

ici d'un petit exemple et je prendrai le code suivant, stocké dans **Operation.py** :

```
01: class Operation:
02:
03:     def __init__(self, valueA, valueB):
04:         self.__a = valueA
05:         self.__b = valueB
06:
07:     def add(self):
08:         return self.__a + self.__b
09:
10:     def div(self):
11:         return self.__a / self.__b
12:
13: if __name__ == '__main__':
14:     op = Operation(4, 2)
15:     print('Valeurs : 4 et 2')
16:     print('Addition :', op.add())
17:     print('Division :', op.div())
```

Il n'y a rien de compliqué dans ce programme où la classe **Operation** stocke deux attributs **__a** et **__b** et permet d'obtenir le résultat de leur addition par la méthode **add()** (lignes 7 et 8) ou le résultat de leur division par la méthode **div()** (lignes 10 et 11). Le programme principal des lignes 13 à 17 nous permet de nous assurer que tout fonctionne correctement. On peut d'ailleurs le vérifier à l'exécution :

```
$ python3 Operation.py
Valeurs : 4 et 2
Addition : 6
Division : 2.0
```

Écrivons maintenant des tests nous permettant de nous assurer que la méthode **add()** fonctionne correctement. Une méthode de développement, le *Test-Driven Developement* ou TDD, préconise une écriture complète du jeu de tests avant de se lancer dans l'écriture du code. C'est une bonne méthode pour produire du code de qualité mais sans devenir un intégriste du test unitaire, il est aussi possible d'écrire des tests a posteriori uniquement sur des parties particulièrement sensibles du code. C'est ce que nous faisons ici.

Pour stocker les tests, nous allons créer un fichier **Operation.test.py** :

```
01: import unittest
02: from Operation import Operation
03:
04: class Operationtest(unittest.TestCase):
05:     def test_simple_add(self):
06:         self.obj = Operation(2, 3)
07:         self.assertEqual(self.obj.add(), 2 + 3)
08:
09: if __name__ == '__main__':
10:     unittest.main()
```

Nous importons bien entendu **unittest** en ligne 1 mais également **Operation** en ligne 2 puisqu'il s'agit de la classe que nous souhaitons tester. Il faut ensuite créer une classe héritant de **unittest.TestCase** (ligne 4) dans laquelle les méthodes correspondront à nos tests. J'ai défini ici un seul test appelé **test_simple_add** (il est important de donner des noms compréhensibles aux tests pour pouvoir les retrouver rapidement lorsqu'ils sont signalés comme ayant échoué). Ce test crée un objet **Operation** avec les valeurs **2** et **3** (ligne 6) et s'assure que le résultat de l'appel de la méthode **add()** soit bien **2 + 3**. Le programme principal des lignes 9 et 10 permet de lancer les tests à l'exécution comme on peut le voir dans la console suivante :

```
$ python3 Operation.test.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

PHASE 3 Les valeurs de retour des tests

Il y a trois valeurs de retour pour un test :

- **OK** : le test s'est déroulé correctement ;
- **F** (*Fail* ou échoué) : le test a échoué. Son nom et la cause de l'échec sont affichés. Exemple en modifiant la ligne 7 de **Operation.test.py** :

```
07:         self.assertEqual(self.obj.add(), 23)
```

En lançant les tests on obtient :

```
$ python3 Operation.test.py
F
======================================================================
FAIL: test_simple_add (__main__.Operationtest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "Operation.test.py", line 7, in test_simple_add
    self.assertEqual(self.obj.add(), 23)
AssertionError: 5 != 23


----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
```

- **E** (*Error* ou erreur) : une erreur est présente dans le code. Exemple en modifiant la ligne 6 de **Operation.test.py** :

```
06:         self.obj = Operation(2, 3, 4)
```

On obtient alors :

```
$ python3 Operation.test.py
E
======================================================================
ERROR: test_simple_add (__main__.Operationtest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "Operation.test.py", line 6, in test_simple_add
    self.obj = Operation(2, 3, 4)
TypeError: __init__() takes 3 positional arguments but 4 were given

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (errors=1)
```

PHASE 4 Les différents tests de unittest

Nous avons utilisé la méthode **assertEqual()** pour vérifier l'égalité entre la valeur renvoyée par la méthode **add()** et la valeur attendue. Il existe de nombreuses autres méthodes (voir https://docs.python.org/3/library/unittest.html#unittest.TestCase.debug). En voici un descriptif rapide sous forme de tableau ci-dessous.

PHASE 5 Ajouter des tests

Dans notre fichier **Operation.test.py** nous n'avons qu'un seul test mais nous pouvons en ajouter bien d'autres. Par exemple, testons si l'addition des nombres négatifs fonctionne :

```
01: import unittest
02: from Operation import Operation
03:
04: class Operationtest(unittest.TestCase):
05:     def test_simple_add(self):
06:         self.obj = Operation(2, 3)
07:         self.assertEqual(self.obj.add(), 2 + 3)
08:
09:     def test_negatif_add(self):
10:         self.obj = Operation(-2, -3)
11:         self.assertEqual(self.obj.add(), -2 + -3)
12:
13: if __name__ == '__main__':
14:     unittest.main()
```

Comme vous pouvez le constater, il faut créer une nouvelle instance d'**Operation** pour chaque nouveau test... Dans le cas d'un objet complexe ou si nous avions utilisé

Méthode	Description
assertEqual(a, b)	Teste l'égalité (**==**) entre les valeurs **a** et **b**.
assertNotEqual(a, b)	Vérifie que **a** et **b** sont différents (**!=**).
assertTrue(a)	Vérifie que **a** est vrai (**True**).
assertFalse(a)	Teste si **a** est faux (**False**).
assertIs(a, b)	Vérifie que **a** et **b** sont équivalents (**is**).
assertIsNot(a, b)	Vérifie que **a** et **b** ne sont pas équivalents (**is not**).
assertIsNone(a)	Teste si **a** vaut **None**.
assertIsNotNone(a)	Teste si **a** ne vaut pas **None**.
assertIn(a, b)	Vérifie que **a** est dans **b** (**a in b**)
assertNotIn(a, b)	Vérifie que **a** n'est pas dans **b** (**a not in b**)
assertIsInstance(a, b)	Teste si **a** est une instance de **b** (**isinstance(a, b)**)
assertNotIsInstance(a, b)	Teste si **a** n'est pas une instance de **b** (**not isinstance(a, b)**)
AssertRaises(exception) **AssertWarns(warning)**	Vérifie qu'un traitement déclenche bien l'exception ou l'avertissement passé en paramètre. Ces méthodes acceptent des paramètres supplémentaires comme arguments et une version avec validation du message par expression régulière est disponible via **AssertRaisesRegex()** ou **AssertWarnsRegex()**.

une instance possédant les mêmes valeurs, il pourrait être intéressant de factoriser le code. Les méthodes **setUp()** et **tearDown()** sont là pour ça. Elles sont appelées respectivement avant et après l'exécution de chaque test (même si le test échoue). Voici un exemple montrant leur fonctionnement :

```
01: import unittest
02: from Operation import Operation
03:
04: class Operationtest(unittest.TestCase):
05:     def setUp(self):
06:         print('Lancement setUp()')
07:         self.fixture = Operation(2, 3)
08:
09:     def tearDown(self):
10:         print('Lancement tearDown()')
11:
12:     def test_simple_add(self):
13:         self.obj = Operation(2, 3)
14:         self.assertEqual(self.obj.add(), 2 + 3)
15:
16:     def test_negatif_add(self):
17:         self.obj = Operation(-2, -3)
18:         self.assertEqual(self.obj.add(), 2 + -3)
19:
20: if __name__ == '__main__':
21:     unittest.main()
```

La méthode **setUp()** des lignes 5 à 7 affiche un message signalant son exécution et crée un attribut **fixture** qui est une instance d'**Operation**. La méthode **tearDown()** des lignes 9 et 10 affiche seulement un message indiquant qu'elle a bien été exécutée. Enfin, en ligne 18 j'ai ajouté une erreur. Voyons le comportement à l'exécution :

```
$ python3 Operation.test.py
Lancement setUp()
Lancement tearDown()
FLancement setUp()
Lancement tearDown()
.
======================================================================
FAIL: test_negatif_add (__main__.Operationtest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "Operation.test.py", line 18, in test_negatif_add
    self.assertEqual(self.obj.add(), 2 + -3)
AssertionError: -5 != -1

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

On constate que les méthodes **setUp()** et **tearDown()** sont bien appelées à chaque fois et même en cas d'erreur. De plus, même si un test échoue les autres tests (ici un seul) sont exécutés.

Note

La division par 0 va lever une exception. Il faudra alors créer un test en utilisant **assertRaises(ZeroDivisionError)**.

PHASE 6 Exécution de l'ensemble des tests du projet

Dans un programme il peut y avoir de nombreuses classes et donc de nombreux fichiers de test. On ne va certainement pas les appeler un à un ! Il est possible de lancer automatiquement les tests qui seront découverts (mode *Test Discovery*) dans le répertoire courant et les sous-répertoires.

La seule contrainte est de nommer les fichiers de tests en commençant par **test**. Si nous tentons un lancement des tests sans modification, aucun test ne sera découvert :

```
$ python3 -m unittest
----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
```

En changeant le nom du fichier **Operation.test.py** en **test_Operation.py**, cette fois-ci tous les tests seront vus :

```
$ python3 -m unittest
...
Ran 2 tests in 0.001s

FAILED (failures=1)
```

PHASE 7 Architecture des tests

Il est plus lisible et maintenable de séparer le code source du code des tests. Pour cela, nous pouvons placer le fichier **test_Operation.py** dans un répertoire **tests**. Pour que celui-ci soit accessible pour la découverte des tests, n'oubliez pas d'y ajouter un fichier vide **__init__.py**. Notre arborescence est donc maintenant la suivante :

```
$ tree
.
ÐÐÐ Operation.py
ÐÐÐ tests
    ÐÐÐ __init__.py
    ÐÐÐ test_Operation.py
```

Si vous préférez isoler complètement votre code source, vous pouvez adopter une structure similaire à la suivante :

```
$ tree
.
ÐÐÐ src
Ð   ÐÐÐ __init__.py
Ð   ÐÐÐ Operation.py
ÐÐÐ tests
    ÐÐÐ __init__.py
    ÐÐÐ test_Operation.py
```

Vous devrez alors modifier l'import du module **Operation** dans **test_Operation.py** :

```
02: from src.Operation import Operation
```

La seule contrainte est d'exécuter la commande **python3 -m unittest** depuis le répertoire racine du projet.

PHASE 8 Application à Roman.py

Reprenons le code de **Roman.py** qui permet de convertir un nombre décimal en son écriture romaine. Si vous n'avez pas lu GNU/Linux Magazine n°188, voici le programme :

```
01: #!/usr/bin/python3
02:
03: class Roman:
04:     units = ['', 'I', 'II', 'III', 'IV', 'V', 'VI', 'VII',
'VIII', 'IX', 'X']
05:     decades = ['', 'X', 'XX', 'XXX', 'XL', 'L', 'LX', 'LXX',
'LXXX', 'XC']
06:     hundreds = ['', 'C', 'CC', 'CCC', 'CD', 'D', 'DC', 'DCC',
'DCCC', 'CM']
07:     dicos = [units, decades, hundreds]
08:
09:
10:     def __init__(self, decimal_number):
11:         self.__decimal = decimal_number
12:         self.__roman = self.__convert()
13:
14:
15:     def __getDecimal(self):
16:         return self.__decimal
17:     decimal = property(__getDecimal)
18:
19:
20:     def __getRoman(self):
21:         return self.__roman
22:     roman = property(__getRoman)
23:
24:
25:     def __partition(self):
26:         def calc_partition(n):
27:             while len(n) > 3:
28:                 n = n[:-3]
29:             return int(n)
30:
31:         strDecimal = str(self.__decimal)
32:
33:         return list(map(calc_partition,
34:             [strDecimal[i:] for i in range(-3,
-len(strDecimal)-4, -3)]))
35:
36:
37:     def __translateGroup(self, group, rank):
38:         if rank == 1 and group <=4 and group >= 1:
39:             return (group * 'M', 0)
40:
41:         group = str('{:03d}'.format(group))
42:
```

```
43:         symbol = ''
44:         for i in range(3):
45:                 symbol += Roman.dicos[2-i][int(group[i])]
46:
47:         return (symbol, rank)
48:
49:
50:     def __convert(self):
51:         result = ''
52:
53:         for i, p  in enumerate(self.__partition()):
54:             conv = self.__translateGroup(p, i)
55:             if conv[1] == 0:
56:                 result = conv[0] + result
57:             elif conv[0] != '':
58:                 result = '(' + conv[0] + ')(' + str(conv[1])
+ ')' + result
59:
60:         return result
61:
62:
63:     def __str__(self):
64:         return self.__roman
65:
66:
67: if __name__ == '__main__':
68:     d = Roman(14000022)
69:     print('{} => {}'.format(d.decimal, d.roman))
```

Nous voyons que la conversion est effectuée par la méthode **__convert()** faisant appel à **__partition()** et à **__translateGroup()**. C'est donc pour ces méthodes que nous allons écrire les tests dans un fichier **test_Roman.py** dans un répertoire tests (n'oubliez pas d'ajouter le fichier **__init__.py**).

Vous vous demandez sans doute comment nous allons pouvoir accéder à ces méthodes puisqu'elles sont privées... N'oubliez pas que la notion de visibilité privée en Python est une convention : la méthode **__privee()** de la classe **MaClasse** sera accessible de l'extérieur par **_MaClasse__privee()** sur une instance de **MaClasse**.

Le test de __partition()

La méthode partition prend l'entier stocké dans **self.__decimal** et retourne une liste de centaines, centaines de milliers, etc. Donc pour **12** on doit obtenir **[12]**, pour **123** on doit obtenir **[123]**, pour **1234** on doit obtenir **[234, 1]**, pour **123 456** on doit obtenir **[456, 123]**, pour **1 234 567** on doit obtenir **[567, 234, 1]** etc.

```
01: import unittest
02: from Roman import Roman
03:
04: class Romantest(unittest.TestCase):
05:     def test_partition_12(self):
06:         self.number = Roman(12)
07:         self.assertEqual(self.number._Roman__partition(),
[12])
08:
09:     def test_partition_123(self):
10:         self.number = Roman(123)
11:         self.assertEqual(self.number._Roman__partition(),
[123])
12:
13:     def test_partition_1234(self):
14:         self.number = Roman(1234)
15:         self.assertEqual(self.number._Roman__partition(),
[234, 1])
16:
17:     def test_partition_123456(self):
18:         self.number = Roman(123456)
19:         self.assertEqual(self.number._Roman__partition(),
[456, 123])
20:
21:     def test_partition_1234567(self):
22:         self.number = Roman(1234567)
23:         self.assertEqual(self.number._Roman__partition(),
[567, 234, 1])
24:
25: if __name__ == '__main__':
26:     unittest.main()
```

Et en exécutant les tests on s'aperçoit immédiatement qu'il y a un problème :

```
$ python3 -m unittest
.F.
======================================================================
FAIL: test_partition_123 (tests.test_Roman.Romantest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "tests/test_Roman.py", line 11, in test_partition_123
    self.assertEqual(self.number._Roman__partition(), [123])
AssertionError: Lists differ: [123, 123] != [123]

First list contains 1 additional elements.
First extra element 1:
123

- [123, 123]
+ [123]

======================================================================
FAIL: test_partition_123456 (tests.test_Roman.Romantest)
----------------------------------------------------------------------
Traceback (most recent call last):
 File "tests/test_Roman.py", line 19, in test_partition_123456
    self.assertEqual(self.number._Roman__partition(), [456, 123])
AssertionError: Lists differ: [456, 123, 123] != [456, 123]

First list contains 1 additional elements.
```

```
First extra element 2:
123

- [456, 123, 123]
?       -----

+ [456, 123]

----------------------------------------------------------------------
Ran 5 tests in 0.001s

FAILED (failures=2)
```

Le partitionnement des entiers dont la « taille » est un multiple de **3** ne fonctionne pas (on convertit l'entier en chaîne de caractères en ligne 31 de **__partition()**). En recherchant un peu on s'aperçoit que le problème vient de la borne de fin du **range()** de la ligne 34 :

```
34:             [strDecimal[i:] for i in range(-3,
-len(strDecimal)-3, -3)]))
```

Après correction, tous les tests sont validés. Nous assurons ici que la méthode **__partition()** fonctionne conformément à ce que nous avions défini. Dans le cas où une nouvelle erreur serait détectée à l'usage, il faudra ajouter le test provoquant cette erreur et résoudre le problème. Ainsi il n'y aura pas de régression possible.

Le test de __translateGroup()

La méthode **__translateGroup()** prend en paramètre un entier compris entre **0** et **999** et sa « position » dans l'écriture de l'entier à traduire. Pour **4 122** nous appellerons ainsi **__translateGroup(122, 0)** et **__translateGroup(4, 1)**. En retour la méthode produit un tuple composé d'une chaîne de caractères (la traduction en numération romaine) et le nombre de fois où ces caractères doivent être surlignés (la multiplication par un facteur **1000**). Pour tester nous pourrons employer les tuples suivants : **(122, 0)** qui doit retourner **('CXXII', 0)**, **(4, 1)** qui doit renvoyer **('MMMM', 0)**, **(5, 1)** qui doit renvoyer **('V', 1)**, et **(34, 2)** qui doit renvoyer **('XXXIV', 2)**. Voici le code des tests :

```
01: import unittest
02: from Roman import Roman
03:
04: class Romantest(unittest.TestCase):
05:     def setUp(self):
06:         self.roman = Roman(1)
...
29:     def test_translateGroup_122(self):
30:         self.assertEqual(self.roman._Roman__
translateGroup(122, 0), ('CXXII', 0))
31:
32:     def test_translateGroup_4(self):
33:         self.assertEqual(self.roman._Roman__translateGroup(4,
1), ('MMMM', 0))
34:
35:     def test_translateGroup_5(self):
36:         self.assertEqual(self.roman._Roman__translateGroup(5,
1), ('V', 1))
37:
38:     def test_translateGroup_34(self):
39:         self.assertEqual(self.roman._Roman__translateGroup(34,
2), ('XXXIV', 2))
...
```

Ici j'ai utilisé la méthode **setUp()** pour ne pas avoir à recréer une instance pour chaque test.

Les tests sont validés sans problème.

Le test de __convert()

La méthode **__convert()** fournit simplement le résultat de la conversion en numération romaine où les facteurs multiplicatifs par mille seront signalés par des parenthèses. Par exemple **63925** deviendra **(LXIII)(1)CMXXV**. Nous utiliserons cet exemple dans nos tests ainsi que les valeurs **123** et **14 000 022** dont la conversion est respectivement **CXXIII** et **(XIV)(2)XXII** :

```
...
42:     def test_convert_123(self):
43:         self.number = Roman(123)
44:         self.assertEqual(self.number._Roman__convert(), 'CXXIII')
45:
46:     def test_convert_63925(self):
47:         self.number = Roman(63925)
48:         self.assertEqual(self.number._Roman__convert(), '(LXIII)
(1)CMXXV')
49:
50:     def test_convert_14000022(self):
51:         self.number = Roman(14000022)
52:         self.assertEqual(self.number._Roman__convert(), '(XIV)
(2)XXII')
...
```

Là encore les tests sont validés.

LE RÉSULTAT

Avec un jeu de douze tests, nous avons renforcé la fiabilité de notre programme. Il est bien sûr possible d'aller plus loin dans les tests unitaires avec, par exemple, **tox**. Mais ça, ce sera pour une autre fois... ■

VOICI TOUTES LES OFFRES COUPLÉES AVEC GNU/LINUX MAGAZINE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

SUPPORT

Prix en Euros / France Métropolitaine

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

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

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

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

MODIFIEZ UN TABLEAU HTML AVEC GREASEMONKEY

Tristan Colombo

S'ils sont bien conçus, les tableaux permettent d'obtenir des données de manière ordonnée. Dans le cas contraire, il faut trouver un moyen de faire ressortir les informations. C'est ce que nous allons faire grâce à Greasemonkey.

tableau
jquery Greasemonkey html
css javascript

L'OBJECTIF

Considérons que nous utilisons un tableau présent sur une page html et que nous n'avons pas accès au code source côté serveur (cela est souvent le cas pour les affichages de type « tableau de bord » dans des logiciels de suivi). Il est parfois illusoire de faire une demande de modification au service compétent et il faut se débrouiller pour bidouiller les outils dont nous disposons. Ici nous voulons mettre en relief les lignes d'un tableau dont une cellule est vide. J'utiliserai comme exemple la page http://trac.edgewall.org/demo-1.0/report/1 (voir figure 1) et je vais afficher en mauve les lignes dont la cellule `Propriétaire` est vide.

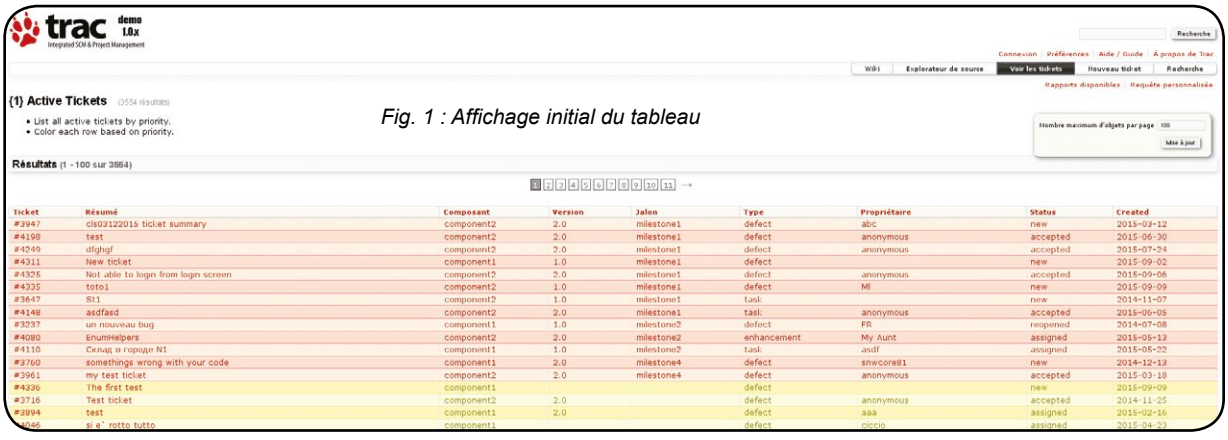


Fig. 1 : Affichage initial du tableau

LES OUTILS

- le navigateur **Mozilla Firefox** ;
- l'extension **Greasemonkey** (paquet `xul-ext-greasemonkey` sous Debian) ;
- un éditeur de code.

PHASE 1 Création du fichier de script

Pour associer un nouveau script à Greasemonkey il y a deux possibilités :

- cliquez sur l'icône en forme de tête de singe puis sélectionnez ***Nouveau Script...*** ;
- ou alors, dans le menu général de Firefox, cliquez sur ***Outils > Greasemonkey > Nouveau Script...***.

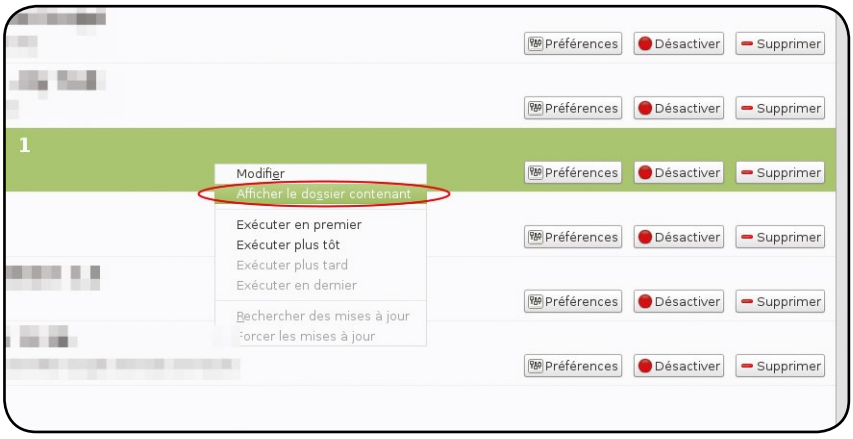


Fig. 3 : Ouverture du répertoire contenant le script

Une nouvelle fenêtre apparaît dans laquelle il faut spécifier le nom du script, donner un espace de nom (identifiant unique utilisé par Greasemonkey), éventuellement une description et l'adresse de la page sur laquelle ce script devra être exécuté. La figure 2 montre les données utilisées pour la création de notre script.

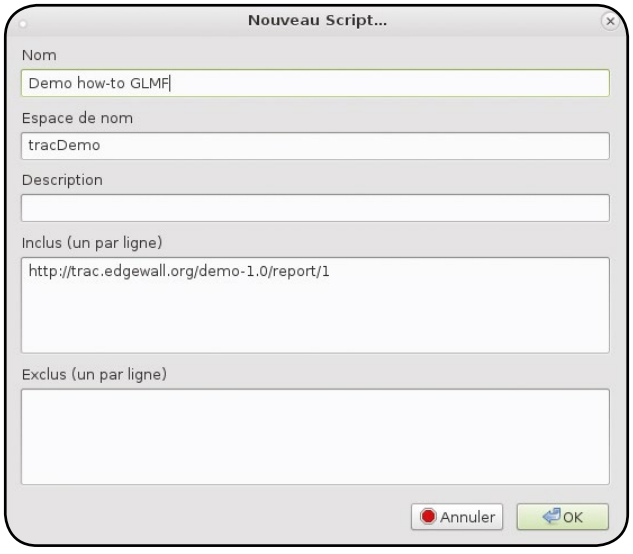


Fig. 2 : Création d'un nouveau script

PHASE 2 Modification du script et premier essai

La question qui se pose maintenant est : « Mais où se trouve ce $!?#§ de fichier ? ». Pour y accéder il faut passer par ***Gérer les scripts*** (toujours en cliquant sur la tête de singe ou en passant par ***Outils > Greasemonkey***). Sur la nouvelle page qui apparaît vous pourrez voir votre script et il faudra alors effectuer un clic droit sur ce dernier et cliquer sur ***Afficher le dossier contenant*** (voir figure 3, ci-dessus).

Vous pouvez alors éditer le fichier **Demo_how-to_GLMF.user.js** qui contient déjà quelques lignes :

```
01: // ==UserScript==
02: // @name         Demo how-to GLMF
03: // @namespace    tracDemo
04: // @include      http://trac.edgewall.org/demo-1.0/report/1
05: // @version      1
06: // @grant        none
07: // ==/UserScript==
08:
09: alert('Greasemonkey hack!');
```

Les lignes 1 à 7 ont été générées automatiquement d'après les informations que nous avons données lors de la création du script. Il est inutile de rentrer dans le détail des tags en **@**, ceux-ci sont suffisamment clairs de par leur nom.

Si vous rechargez la page http://trac.edgewall.org/demo-1.0/report/1, vous verrez apparaître une fenêtre d'alerte avec le message « Greasemonkey hack! ». Donc tout fonctionne correctement.

PHASE 3 Chargement de jQuery

Pour nous simplifier la tâche nous allons utiliser **jQuery**. Pour cela nous allons ajouter une directive **@require** en tête de fichier pour indiquer à Greasemonkey où récupérer le fichier js. Pour tester si cela a fonctionné, nous allons colorer toutes les lignes du tableau en rouge :

```
01: // ==UserScript==
02: // @name         Demo how-to GLMF
03: // @namespace    tracDemo
```

```
04: // @include     http://trac.edgewall.org/demo-1.0/report/1
05: // @version     1
06: // @require  https://ajax.googleapis.com/ajax/libs/
jquery/2.1.4/jquery.min.js
07: // @grant       none
08: // ==/UserScript==
09:
10: $('table.tickets tr').css('background', 'red');
```

En ligne 10 nous sélectionnons les lignes (**tr**) des éléments **table** ayant pour classe **tickets** (**table.tickets**). Sur ces éléments nous modifions le style **background** en lui attribuant la valeur **red**.

Les lignes deviennent toutes rouges : jQuery a été correctement téléchargé.

PHASE 4 Détection des lignes et changement de couleur de fond

Sur la page sur laquelle nous travaillons, la cellule que nous voulons étudier a pour classe **owner**. De plus les classes de couleur des lignes sont **color1-even**, **color2-even**, **color3-even**, **color1-odd**, **color2-odd**, et **color3-odd**. Nous pourrions grouper les actions pour ces classes en détectant uniquement la fin du nom d'une classe **-even** ou **-odd** à l'aide d'une expression régulière mais, au vu de l'implémentation de la méthode **hasClass()**, ce serait moins efficace.

```
01: // ==UserScript==
02: // @name        Demo how-to GLMF
03: // @namespace   tracDemo
04: // @include     http://trac.edgewall.org/demo-1.0/report/1
05: // @version     1
06: // @require  https://ajax.googleapis.com/ajax/libs/
jquery/2.1.4/jquery.min.js
07: // @grant       none
08: // ==/UserScript==
09:
10: $('table tr').each(function () {
11:     var cell = $.trim($(this).find("td.owner").text());
12:
13:     if (cell.length == 0)
14:     {
15:         if ($(this).hasClass('color1-even') || $(this).
hasClass('color2-even') || $(this).hasClass('color3-even'))
16:         {
17:             $(this).css("background", "#ddd7de");
18:         }
19:         else if ($(this).hasClass('color1-odd') || $(this).
hasClass('color2-odd') || $(this).hasClass('color3-odd'))
20:         {
21:             $(this).css("background", "#ddc2df");
22:         }
23:     }
24: });
```

En ligne 10 nous parcourons toutes les lignes (**tr**) de la table et pour chacune d'elles :

- en ligne 11 nous extrayons le contenu du tag **td** de classe **owner** ;
- si le contenu est vide (test de la ligne 13), si la ligne possède un nom de classe se terminant par **-even** (ligne 15), nous changeons le style de son **background** en **#ddd7de** (ligne 17), sinon, si son nom de classe se termine par **-odd** (ligne 19), alors la couleur du fond sera **#ddc2df** (ligne 21).

LE RÉSULTAT

Au final, nous obtenons bien un tableau dans lequel les lignes qui nous intéressaient particulièrement apparaissent avec une nouvelle couleur de fond (voir figure 4). ■

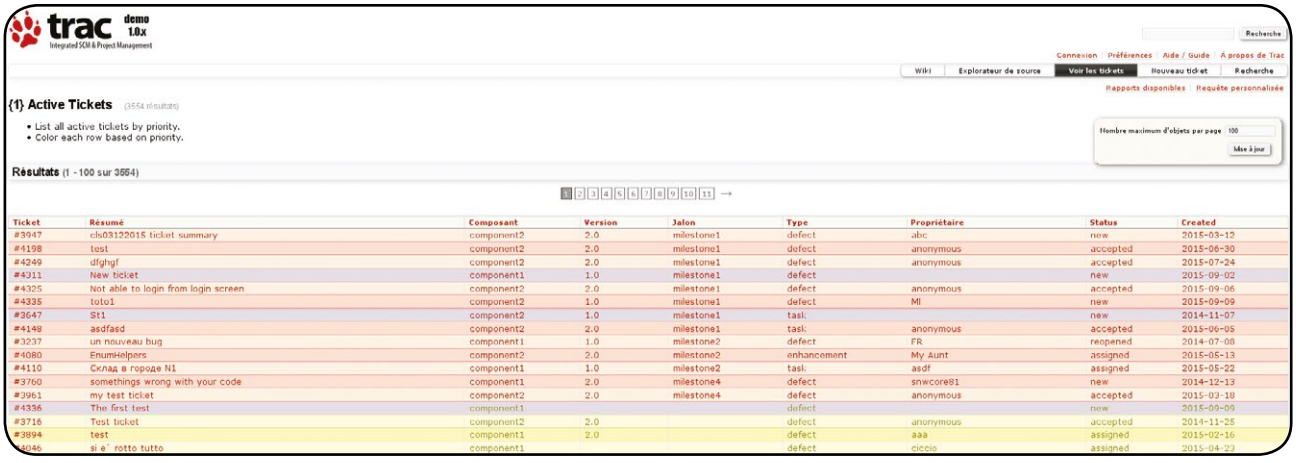


Fig. 4 : Tableau final dont les lignes contenant une cellule vide dans la colonne « Propriétaire » sont affichées en mauve

UTILISATION DE MICMAC
POUR LA GÉNÉRATION DE MODÈLE NUMÉRIQUE D'ÉLÉVATION PAR TRAITEMENT D'IMAGES ACQUISES PAR MICRODRONE

Jean-Michel Friedt [Institut FEMTO-ST, dpt. temps-fréquence, Besançon]

Éric Bernard [Laboratoire ThéMA, Besançon]

Florian Tolle [Laboratoire ThéMA, Besançon]

Dans le contexte de l'étude d'un glacier en région arctique, un microdrone commercialement disponible est utilisé pour acquérir des images en vue azimutale couvrant la moraine. Des modèles d'élévation numériques et orthoimages sont générés par assemblage et correction géométrique des effets de topographie par le logiciel Micmac. Les produits de ces traitements sont analysés pour estimer la résolution et l'exactitude de la mesure. Un glissement de terrain est observé entre deux prises de vues séparées d'une semaine.

Mots-clés : Micmac, QGis, Drone, Modèle numérique d'élévation, Orthoimage

Résumé

Dans un contexte de convergence de technologies alliant centrale inertielle précise, batterie légère, moteurs électriques compacts, capteur optique de haute résolution et puissance de calcul infinie, les microdrones sont devenus réalité. Nous proposons d'exploiter les photographies aériennes pour une analyse quantitative de modèle numérique d'élévation.

Nous avions présenté dans Open Silicium **[1]** le logiciel **Micmac [2]** développé principalement par le laboratoire MATIS associé à l'Institut Géographique National (IGN). Dans cet article, nous nous proposons d'exploiter Micmac dans un contexte pratique de cartographie de la moraine d'un glacier en région arctique (Spitsberg, Norvège). Le contexte peut sembler trivial mais a son importance : il s'agit d'une région sans végétation (donc peu de variation du terrain entre deux prises de vues tel que pourraient l'induire des mouvements de végétation soumis au vent), dynamique avec une érosion par la circulation d'eau, et libre de la majorité des dangers de vol connu en milieu urbain (mais nécessitant néanmoins une autorisation de vol). Nous nous sommes intéressés à l'observation de cette région par microdrone commercialement disponible pour un coût inférieur à 2500 euros. En effet, alors que la surface d'un glacier se cartographie par quelques points de mesure judicieusement distribués et entre lesquels l'interpolation est raisonnable, une moraine présente une topographie variable sur laquelle l'interpolation entre des points de mesure distants n'est pas valable.

D'un autre côté, le travail en région quelque peu isolée impose de nouvelles contraintes **[3]** : autonomie des batteries aux basses températures et transport du matériel à pied, qui nous ont induit à évaluer la pertinence de ce mode de mesure par rapport à des approches de référence basées sur le LiDAR (*Light Detection and Ranging* **[4]** – version optique du RADAR) aéroporté ou terrestre, avec un coût dépassant ce qui est accessible par l'amateur. Ce compte rendu n'a pas vocation à considérer et interpréter les aspects d'évolution de la morphologie du terrain observé, mais de proposer un flux de traitement des données acquises et évaluer la résolution et l'exactitude des modèles numériques d'élévation (MNE) établis pour diverses dates séparées d'un intervalle de temps allant jusqu'à une semaine.

1 Conditions de prises de vues et résolution attendue

Une étude préliminaire des conditions de prises de vues est de rigueur. Elle considère la résolution attendue des structures au sol, le taux de recouvrement nécessaire à une analyse photogrammétrique, et la vitesse horizontale du vecteur de vol au cours de la prise de vue pour couvrir la zone envisagée. Nous avons acquis et exploité un microdrone commercial *DJI Phantom3 Professional* : ce modèle, récent à la date d'acquisition des données en fin-septembre 2015, ne possédait pas l'option de planification de la trajectoire à l'avance, et tout le pilotage s'est fait à la radiocommande avec vol à vue et suivi sur la télé-mesure de la position GPS retransmise en temps réel ainsi que des images observées par le drone (voir figure 1). Cette option, a priori un gadget sans intérêt, s'est révélée primordiale pour garantir le recouvrement latéral des trajectoires successives. La consigne de Micmac sur les prises de vues des photographies successives est d'obtenir un recouvrement de 80%. Avec une ouverture angulaire de 94° selon la largeur de la photographie, une élévation de vol de 100 m induit un pixel de 4 cm de large, sous le décimètre visé mais surtout avec une élévation de vol sous les 120 m autorisés (voir figure 2, droite). Les 2250 pixels de hauteur d'image couvrent donc une longueur au sol de 56 m. Avec une vitesse horizontale maximale de 10 m/s (que nous utiliserons pour couvrir une distance maximale lors de chaque vol), une prise de vue chaque seconde vérifie la consigne de recouvrement. Le logiciel de commande de DJI ne permet pas une prise de vue aussi rapprochée : nous déclenchons à la main la prise de vue avec un intervalle aussi régulier que possible et proche du hertz, avec un recouvrement vérifié lors du vol sur le retour vidéo.

Un point, qui pourra paraître trivial au lecteur mais qui nous a interpellés au cours du premier vol : un drone asservit son vol sur une élévation constante par rapport au point de décollage. En région présentant une forte variation de topographie – telle que la langue d'un glacier – les 100 m séparant le drone du sol au décollage se réduisent rapidement lorsque la distance de vol dépasse les 500 m. Un défilement rapide des images du terrain alors que la vitesse horizontale est maintenue constante sera considéré comme un indicateur de proximité du sol nécessitant de reconsidérer la trajectoire avant de rentrer en collision avec le versant de montagne ou de glacier survolé. Nous sommes ainsi arrivés à une élévation de moins de 20 m au-dessus de la glace (d'après le traitement issu de Micmac) avant de nous interroger sur la cause de ce défilement excessivement rapide !

Fig. 1 : À gauche : conditions expérimentales - le DJI Phantom3 répond aux exigences de facilité de transport, de déploiement rapide, d'autonomie et de couverture du champ de vision de la caméra. Il s'est avéré idéal pour cette campagne de mesures.

À droite : canyon dans la moraine du glacier étudié, qui sera au cœur de l'étude car proposant des structures identifiables susceptibles de varier à court terme. La langue du glacier est visible en bas à gauche de cette photographie.

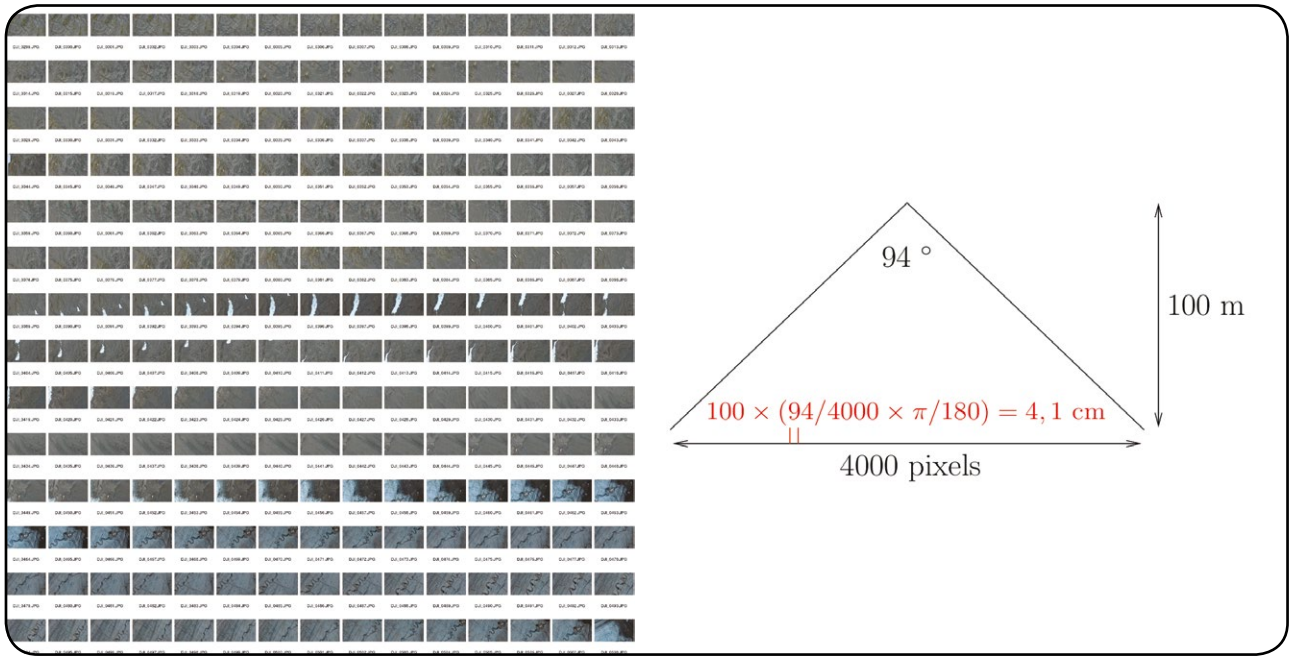


Fig. 2 : À gauche : planche contact d'une série de photographies prises par drone en vue azimutale, en vue de leur traitement pour générer un MNE haute résolution et une image orthorectifiée de la zone analysée.

À droite : estimation de la résolution au sol compte tenu de l'ouverture angulaire, du nombre de pixels dans la largeur de l'image, et l'élévation de vol.

2 Géoréférencement des sites de prises de vues

Ayant acquis les images, nous désirons les traiter pour en extraire une image unique orthorectifiée (compensant les déformations induites par la topographie du sol) et le produit associé qu'est le MNE. Chaque image acquise par le DJI Phantom3 est munie de sa position GPS avec une latitude et longitude en degrés, minute et seconde d'angle, ainsi qu'une élévation relative au point de décollage. Ces informations seront exploitées deux fois au cours du traitement : lors de l'appariement des photographies adjacentes, et lors du géoréférencement dans un référentiel absolu du modèle d'élévation.

L'extraction dans un format exploitable de ces informations est donc un pré-requis à tout traitement ultérieur :

1. extraire les 3 informations de position de chaque photographie ;
2. convertir les degrés-minute-seconde en degrés décimaux ;
3. convertir le géoréférencement en coordonnées sphériques (WGS84 du GPS) en coordonnées dans un référentiel plan approximant localement l'orientation de la sphère formant la surface de la Terre (UTM33N pour le Spitsberg) ;
4. tronquer ces coordonnées pour s'affranchir, lors des calculs ultérieurs, des erreurs d'arrondis lors du calcul sur les flottants.

Ces opérations utilisent trois outils en plus du shell bash : **exiftool** extrait les informations des en-têtes des photographies, **GNU/Octave [5]** effectue les calculs sur les coordonnées, et **QGis [6]** effectue le changement de projection.

```
$ for i in *.JPG; do echo  $i ; done > noms
$ for i in *.JPG; do echo -n $i ; exiftool $i | grep atitu | grep -v R; done > latitude
$ for i in *.JPG; do echo -n $i ; exiftool $i | grep ngitu | grep -v R; done > longitude
$ for i in *.JPG; do echo -n $i ; exiftool $i | grep Altit | grep -v R; done > altitude
$ cat latitude  | cut -c 56-100 | cut -d\  -f1,3,4 | sed "s/'//g" | sed 's/"//g' >
latitude.txt
$ cat longitude | cut -c 56-100 | cut -d\  -f1,3,4 | sed "s/'//g" | sed 's/"//g' >
longitude.txt
$ cat altitude | cut -c 56- | cut -d\  -f1 > altitude.txt
```

Les trois fichiers latitude, longitude et altitude contiennent les informations de position de chaque photographie nom. L'argument 56 est susceptible de devoir être ajusté selon l'arborescence de l'utilisateur et les noms de fichiers manipulés.

Ces trois fichiers alimentent les commandes GNU/Octave suivantes pour générer des coordonnées en degrés décimaux :

```
load latitude.txt
l=latitude(:,1)+latitude(:,2)/60+latitude(:,3)/3600;
save -text latitude.dec l
load longitude.txt
l=longitude(:,1)+longitude(:,2)/60+longitude(:,3)/3600;
save -text longitude.dec l
```

Les deux fichiers d'extension **.dec** sont édités avec notre éditeur de texte favori (vi) pour en éliminer l'en-tête, et finalement toutes ces informations sont combinées dans un unique fichier par :

```
$ paste  latitude.dec longitude.dec altitude.txt noms > for_qgis.txt
```

La conversion de ce jeu de coordonnées en degrés décimaux dans un référentiel en coordonnées cartésiennes s'effectue sous QGis. En effet, la conversion n'est pas triviale, le passage de la sphère de WGS84 aux plans tangents locaux UTM (*Universal Transverse Mercator*) tenant compte des déformations de la Terre par rapport à la sphère. Sous QGis, un fichier comportant 4 colonnes (longitude, latitude, altitude, nom du fichier) est chargé au moyen de l'icône en forme de virgule sur la gauche de l'interface graphique. Nous pouvons faciliter l'identification de la signification de chaque colonne en ajoutant, dans le fichier texte à charger, trois labels **"Y X Z N"** signifiant latitude, longitude, altitude et nom (les identifiants **X** et **Y** sont reconnus par QGis, les autres seront simplement reproduits dans le fichier de sortie). Une fois le fichier chargé, cliquer par le bouton de droite dans l'entrée sur la liste des structures de données chargées, ***Save As*** et choisir le mode de projection cartésien approprié (dans notre cas au nord de la Norvège, CRS WGS84-UTM33N (EPSG : 32633) – valeur qui doit être ajustée pour chaque région géographique, par exemple UTM31N pour la majorité de la France). Finalement, nous sauvons dans un fichier dont les colonnes sont séparées par des virgules (CSV, nommé **p.csv**) et ne conservons que les colonnes utiles par :

```
$ cat p.csv | cut -d, -f1-2,5,6 | sed 's/,/ /g' | sed 's/^43//g'
| sed 's/ 875/ /g' > p.fin
```

Dans cette ligne de commande, les champs 1, 2, 5, 6 sont la longitude et la latitude projetées, l'altitude et le nom de fichier. Cette information sera transmise à Micmac en ajoutant en en-tête de ce fichier, toujours avec son éditeur de texte favori, la séquence **#F=X Y Z N** qui signifie que le caractère de commentaire est un **#** et indique l'ordre des données (**N** est compris comme nom de fichier).

À l'issue de ces manipulations préliminaires, nous avons un fichier associant coordonnées en référentiel cartésien avec chaque photographie. Il s'agit d'une information cruciale à toute la séquence de traitement qui va suivre.

3 Assemblage des images individuelles

Un vol de 20 minutes génère environ $N=1000$ photos. Une approche naïve d'appariement en testant tous les couples possibles nécessite $C_N^2=\frac{N!}{2!\cdot(N-2)!}=N\cdot(N-1)/2$ essais, un nombre qui devient rapidement prohibitif, avec par exemple 499500 couples si $N=1000$. Micmac offre la possibilité d'exploiter la position de prise de vue de chaque photographie pour évaluer les couples susceptibles de présenter des points homologues sur les prises de vue. En pratique, cette option a permis de limiter, pour 614 images traitées, à 10413 couples au lieu de 188191.

Cette étape de traitement réduit considérablement le nombre de comparaisons et permet d'aboutir rapidement à une liste de points homologues entre paires d'images :

```
$ mm3d OriConvert OriTxtInFile p.fin jmfgps MTD1=1
NameCple=FileImagesNeighbour.xml CalcV=1 \
     ImC=DJI_0115.JPG NbImC=25
```

La ligne précédente informe Micmac de la création du répertoire d'orientation **Ori-jmfgps** (qui nous servira à la fin du traitement pour convertir, par homothétie et translation, le nuage de points généré par Micmac dans un référentiel arbitraire vers le référentiel absolu géographique) et d'un fichier de couples **FileImagesNeighbour.xml** en s'inspirant (**OriTxtInFile**) du fichier de configuration **p.fin**.

Par ailleurs, une image de référence est sélectionnée – ici **DJI_0115.JPG** – qui servira avec ses voisines à estimer les propriétés optiques de l'appareil de prise de vue. Le choix de cette image devrait se faire en respectant autant que faire se peut la condition d'un coin pour caractériser les propriétés optiques des lentilles – pour autant qu'un coin puisse se trouver sur le terrain observé. Une vallée entre deux collines semble convenir. La variable **PATC** qui est générée à l'issue de cette étape contient la liste des photographies pour cet étalonnage.

Validation des couples d'images sélectionnés par Micmac

Nous n'avons pas de raison a priori de faire confiance à l'algorithme d'appariement des images par Micmac. Pour nous convaincre de sa validité, nous désirons tracer une droite entre chaque couple d'images sélectionnées. Nous avons d'une part un fichier contenant la position de chaque photographie, et d'autre part un fichier contenant les couples. Comment générer un fichier expliquant à QGis de relier ces ensembles de points ? Une solution porte sur le langage WKT **[7]** compris par QGis, qui permet de définir des opérations entre éléments géoréférencés. Dans notre cas, la commande sera le tracé de droite par **LINESTRING(X1 Y1,X2 Y2)**. Ainsi, partant du fichier de couples **FileImagesNeighbour.xml**, nous allons remplacer le nom de chaque photographie par ses coordonnées :

```
$ cat FileImagesNeighbour.xml | sed 's/<Cple>//g' | sed 's/\/Cple>//g' | grep -v
S | grep -v x | sed 's/ //g' > imgpairs
```

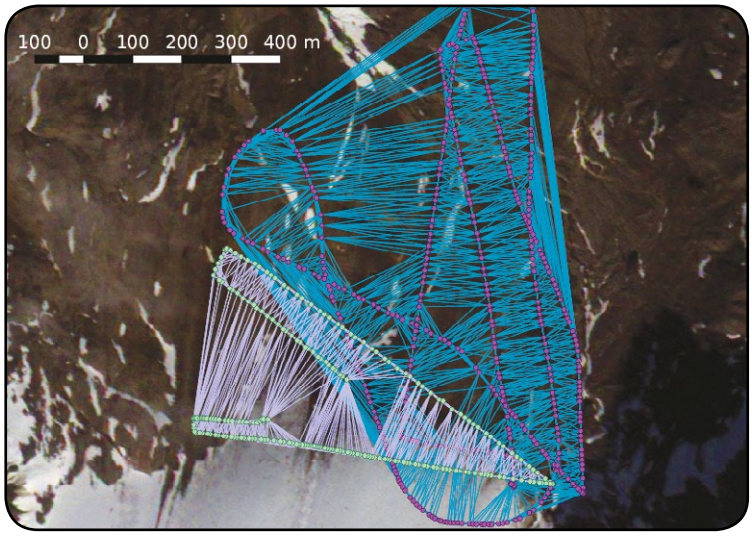
Ceci génère le fichier qui sera passé comme argument d'un script awk (nommé **s.awk**) sensé rétablir le biais de position que nous avons inséré :

```
{ print "sed 's/"$4"/43"$1" "875$2"/g' | \\" } par cat p.fin | awk -f s.awk > p.sh
```

Dans le script **p.sh**, nous remplaçons la première ligne par **cat imgpairs | ** et nous retirons le dernier caractère **** pour obtenir un script qui génère le fichier wkt :

```
$ sh ./p.sh | sed 's/^/LINESTRING(/g' | sed 's/$/)/g' | sed 's/ /,/2' > couples.wkt
```

Ce fichier est chargé dans QGis au moyen de l'icône en forme de virgule (résultat ci-dessous).



Les couples sélectionnés par Micmac semblent donc corrects, nous voilà rassurés.

Nous complétons la définition de **PATC** par une variable **P** qui pointe vers toutes les photographies stockées dans le répertoire courant, en supposant que nous ayons déjà fait le ménage dans les prises de vues et n'avons conservé que celles utiles, en vue azimutale exclusivement : **P=".*JPG"**.

La recherche de points homologues se conclut par :

```
$ mm3d Tapioca File "FileImagesNeighbour.xml" -1
```

Cette commande doit être suivie de l'étalonnage des propriétés de la lentille de l'appareil de prise de vue (modifier le modèle de lentille vers un *fish eye* par exemple si une GoPro ou un ancien appareil DJI est utilisé) :

```
$ mm3d Tapas RadialStd "$PATC" Out=Cal
```

Ce point est critique car fortement dépendant de la photographie de référence sélectionnée : il faut parfois plusieurs étapes pour que le modèle converge (résidu **< 1** pixel) et ne diverge pas (résidu **NaN**).

Finalement cet étalonnage est appliqué à toutes les photographies pour établir la position de la caméra lors de chaque prise de vue.

```
$ mm3d Tapas AutoCal "$P" InCal=Cal Out=Init
```

Nous utilisons ici le répertoire d'orientation pour convertir le référentiel arbitraire dans lequel Micmac effectue ses calculs vers le référentiel géographique : **CenterBascule "$P" Init jmfgps tmp** avant de générer le nuage de points grossier qui permet de valider le positionnement des prises de vue et la cohérence du nuage de points (voir figure 3) **mm3d AperiCloud "$P" tmp**.

Il est envisageable, après le passage au référentiel géographique par **CenterBascule**, de créer une nouvelle orientation pondérant les informations

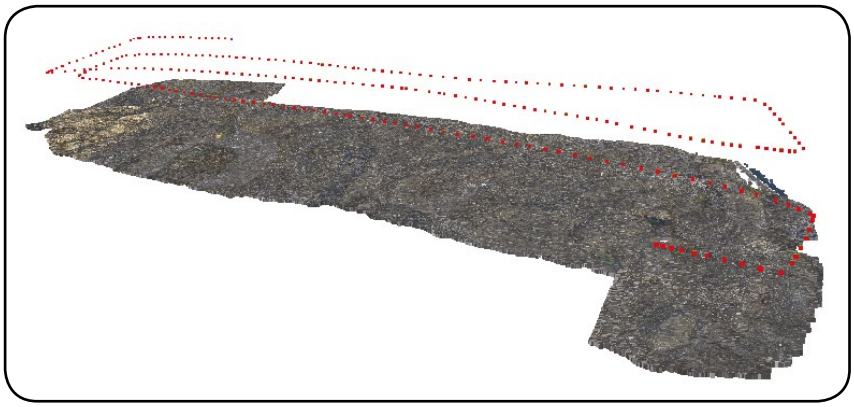


Fig. 3 : Nuage de points grossiers, incluant la position identifiée par traitement photogrammétrique, de la position de la caméra au moment de chaque prise de vue. Les points rouges sont la position de la caméra, le nuage grossier gris est la surface reconstruite à partir des images.

issues du GPS (avec leur imprécision inhérente) et de la photogrammétrie au moyen de **Campari** :

```
$ Campari "$P" tmp tmp-Campari EmGPS=[jmfgps,2] FocFree=1 PPFree=1
```

Dans ce cas, le nuage de point grossier et la suite des opérations s'effectuent sur le répertoire d'orientation **tmp-Campari** au lieu de **tmp**.

4 Produits issus du traitement des images

4.1 Image orthorectifiée

Ayant validé la cohérence du nuage de points (**meshlab** ou **cloudcompare**), nous calculons le nuage de points dense par :

```
$ mm3d Malt Ortho "$P" tmp "DirMEC=Resultats" UseTA=1 ZoomF=2
ZoomI=32 Purge=true AffineLast=false
```

Avant d'en déduire les photographies orthorectifiées (compensées des effets de la topographie et assemblant les photographies adjacentes) :

```
$ mm3d Tawny Ortho-Resultats/ RadiomEgal=0
```

Et finalement le nuage de points dense est drapé de l'image orthorectifiée :

```
$ Nuage2Ply Resultats/NuageImProf_STD-MALT_Etape_7.xml
Attr="Ortho-Resultats/Ortho-Eg-Test-Redr.tif"
```

Les images ainsi produites sont immenses – tellement grandes qu'elles dépassent la taille d'une image TIF et occupent plusieurs giga-octets pour le flux de traitement qui nous intéresse. Bien entendu, avec un pixel de 4 cm, une couverture d'une zone de 800 m de long nécessite 20000 pixels ! Il est donc courant d'obtenir des images de 20000 x 30000 pixels d'1,2 GB en l'absence de compression. Nous convertissons donc ces images au format TIF vers du PNG – plus efficace au détriment d'un algorithme de compression à perte – et assemblons le résultat. Toutes ces opérations se font évidemment par ImageMagick, qui peut cependant rencontrer des problèmes si **/tmp** est trop petit (par exemple si c'est un ramfs de taille modeste). Nous exportons le répertoire temporaire vers notre maison **export MAGICK_TMPDIR=$HOME** avant de lancer conversions et assemblages **convert Ortho-Eg-Test-Redr_Tile_0_0.tif Ortho-Eg-Test-Redr_Tile_0_0.png** dans **Ortho-Resultats** avant d'assembler par :

```
$ montage -tile 1x2 -geometry +0+0 Ortho-Eg-Test-Redr_Tile_0_0.png \
   Ortho-Eg-Test-Redr_Tile_0_1.png out.png
```

Les images restent immenses en nombre de pixels, mais au moins exploitables en terme de taille de stockage : **geeqie** arrive parfois à les afficher sans se figer.

Les images obtenues sont associées à un fichier de positionnement qui informe de la position du coin en haut à gauche (aussi connu comme le nord-ouest – deux dernières lignes du fichier) et de la taille du pixel : le fichier d'extension **tfw** est renommé **pngw** pour l'image que nous venons d'assembler. La position doit cependant être incrémentée du biais de position que nous avions éliminé lors de la génération du fichier de position initial : nous avions retiré le préfixe 43 aux longitudes et 875 aux latitudes, que nous prenons soin de réinsérer ici. Dans notre cas, un fichier typique de position ressemble à :

```
0.045
0
0
-0.045
438839.62
8760127.52
```

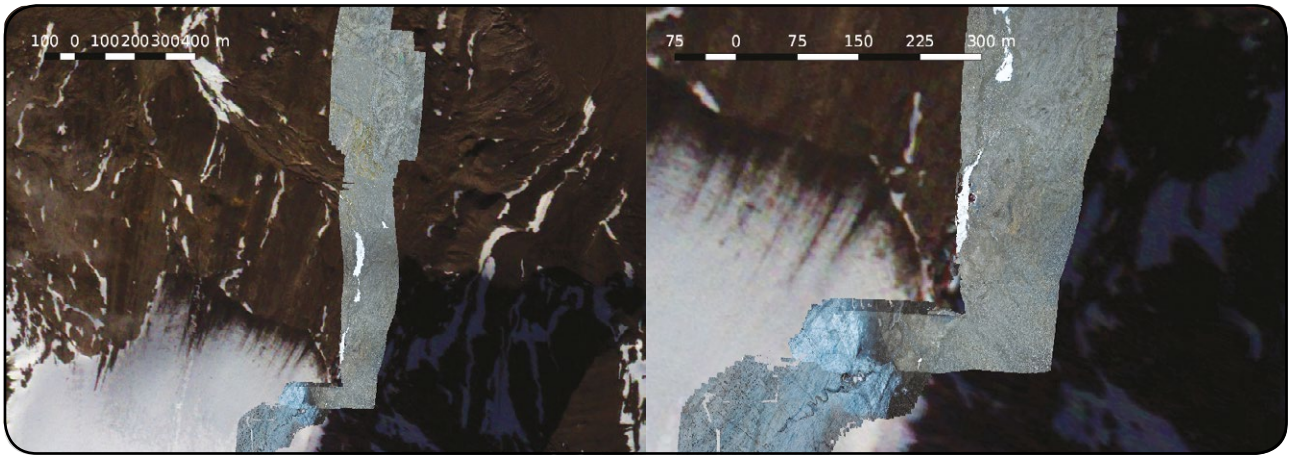


Fig. 4 : Image orthorectifiée sur un fond d'image satellitaire. Noter que le fond opaque de l'image orthorectifiée a été éliminé en sélectionnant Transparency > Add et le triplet 0 0 0 comme couleur à considérer comme transparente dans les attributs de cette entrée dans QGis. Noter le gain en résolution par rapport à l'image satellite de fond considérée comme référence.

Noter en lignes 1 et 4 la taille du pixel (4,5 cm, en accord avec nos attentes), et que la latitude étant de l'ordre de 10^5 m, l'ajout de 8750000 se traduit par les premiers chiffres devenant 876.

À première vue, les images orthorectifiées ainsi acquises semblent convenablement se superposer à une image satellite de référence (voir figure 4). Cependant, les sources libres d'images satellites (par exemple Landsat) présentent une taille de pixel considérablement plus importante que ce que nous visons – typiquement 30 m/côté – et cette cohérence ne fait que valider grossièrement notre flux de traitement, sans préjuger de l'exactitude du résultat avec la résolution décimétrique visée.

Nous allons donc nous efforcer ci-dessous d'estimer quantitativement la reproductibilité de la mesure, en particulier en comparant des jeux de données acquis à plusieurs jours d'intervalle.

4.2 Modèles numériques d'élévation (MNE)

Le problème du positionnement du MNE, accessible dans `Resultats/Z_Num7_DeZoom2_STD-MALT*`, est le même, avec un fichier geoTiff à modifier pour y insérer le biais de position que nous avions introduit. Par ailleurs, un fichier XML associé nous informe du biais en élévation et le

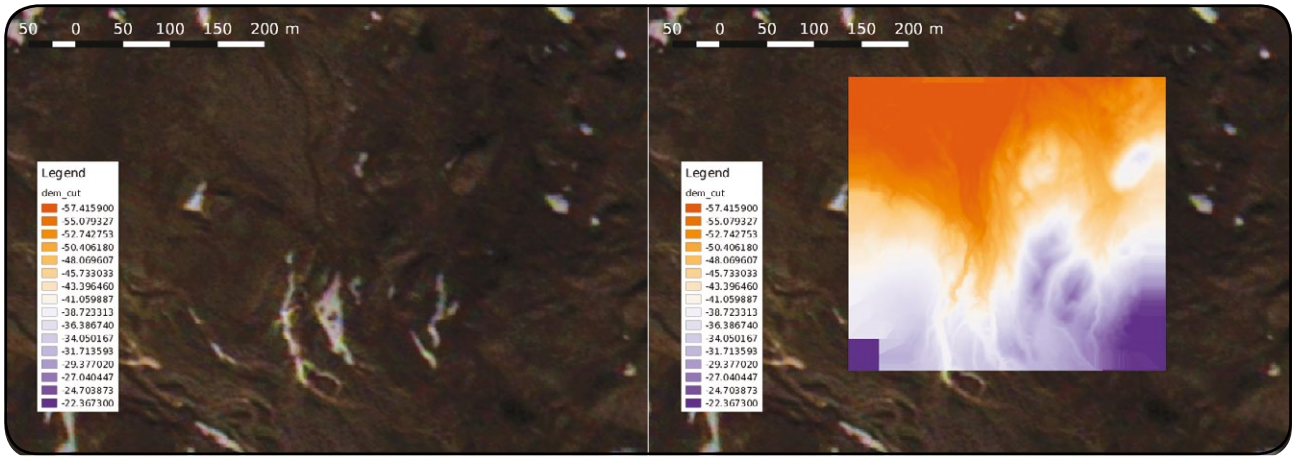


Fig. 5 : À gauche : image satellitaire de la moraine.

À Droite : MNE positionné uniquement par la position GPS du coin supérieur gauche, sans recalage manuel. La cohérence des structures visibles est particulièrement convaincante aux limites entre image et MNE de la figure de droite.

facteur d'homothétie entre le fichier TIF et les dimensions réelles :

```
<OrigineAlti>9.675</OrigineAlti>
<ResolutionAlti>0.09</ResolutionAlti>
```

Lors du chargement de l'image dans QGis (icône de chargement des images matricielles – raster), le MNE est convenablement positionné, mais les altitudes sont fausses.

Nous effectuons l'homothétie et la translation en altitude au moyen de ***Raster > Raster Calculator*** et en créant une nouvelle couche (*Output layer*) au format GeoTIFF qui contient la transformée affine du jeu de données d'entrée (voir figure 5).

Sur la figure 5 , nous proposons à gauche une vue satellitaire d'une région de la moraine présentant une topographie identifiable depuis l'espace, et à droite le modèle numérique de terrain, reproduisant fidèlement le réseau hydrographique. Ce résultat est le produit de nos traitements, dont il nous reste à établir la résolution et l'exactitude.

5 Comparaisons de mesures effectuées à plusieurs jours d'intervalle

Bien que la résolution de la constellation GPS soit de l'ordre du mètre dans une configuration donnée de satellites, l'exactitude n'est que de la dizaine de mètres et induit un décalage inacceptable entre jeux de données acquis à plusieurs heures ou jours d'intervalle. Un recalage des orthophotos et des MNEs est donc nécessaire pour une comparaison entre jeux de données acquis à plusieurs jours ou mois d'intervalle (lorsque la configuration des satellites et des conditions ionosphériques ont significativement évolué). Dans notre cas, nous avons omis de placer des points de contrôle au sol géoréférencés au cours des prises de vues, compte tenu des contraintes de délai de déploiement du drone sur site, mais nous allons recaler a posteriori les jeux de données sur des structures remarquables au sol. Ainsi, même si nos jeux de données ne sont pas positionnés dans un référentiel géographique absolu, ils seront au moins cohérents entre eux pour permettre une inter-comparaison (voir figure 6).

Deux approches sont disponibles pour ce recalage. La première consiste à exploiter le greffon **`rasmover`** de QGis : bien que graphique et trivial d'emploi (il suffit de faire glisser la flèche indiquant le vecteur de déplacement de l'image), cette approche souffre de l'absence d'information quant à la valeur numérique de la translation (de combien de mètres l'image a été ainsi déplacée). Nous préférons donc directement manipuler l'en-tête **`world`** des images pour y ajouter les quantités nécessaires au recalage. Ainsi, nous contrôlons la cohérence de ce vecteur de translation et évitons tout risque de manipulation aberrante que permettrait une interface graphique.

Après une manipulation fine de la position des orthophotos, et donc des MNEs associés, une comparaison de deux jeux de données pris à 1 semaine d'intervalle est proposée

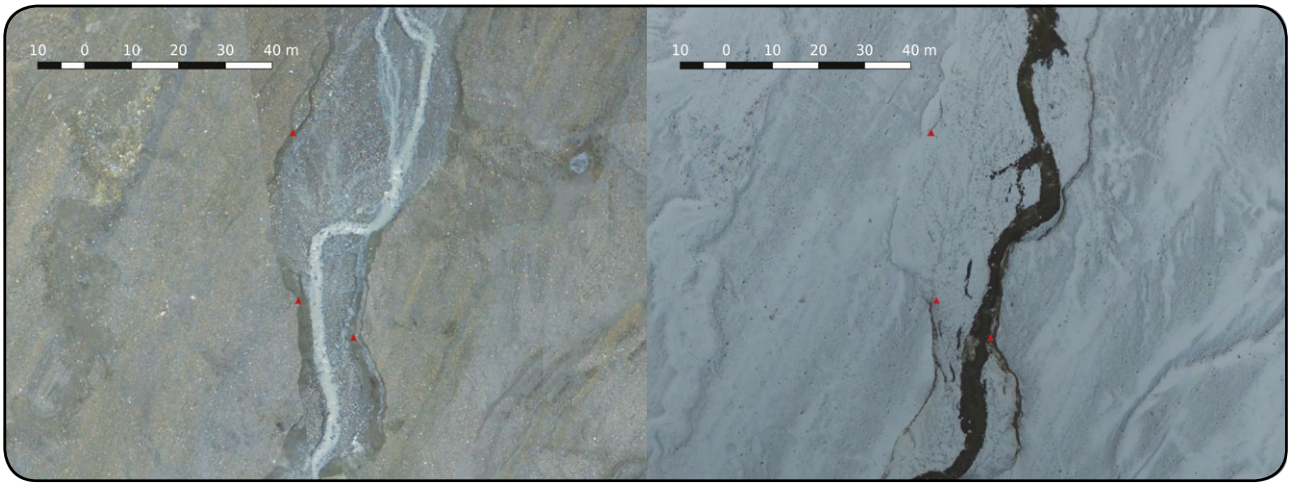


Fig. 6 : Recalage de deux images orthorectifiées prises à 1 semaine d'intervalle. Les points rouges sont des points de référence (shapefile contenant des points) permettant de repérer les points remarquables supposés fixes entre les deux photographies.

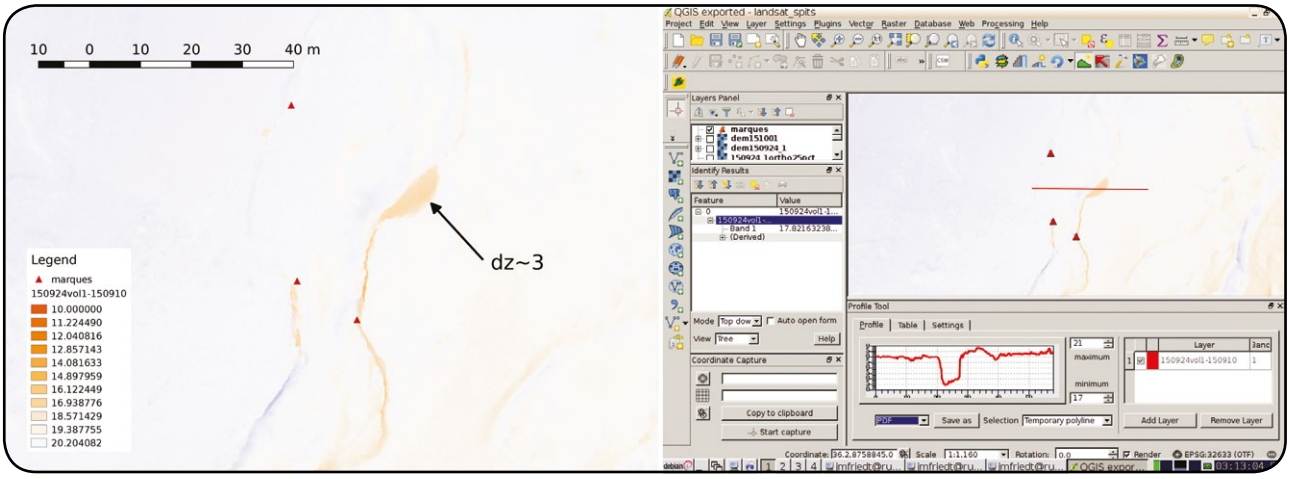


Fig. 7 : Analyse de l'écart de deux MNE pris à 1 semaine d'intervalle. À gauche : noter le glissement de terrain de 3 m de profondeur, cohérent avec la profondeur du canyon adjacent, sur la carte de la différence des MNEs.

À droite : capture d'écran de QGis avec l'outil Terrain Profile actif, la ligne rouge indiquant la section considérée.

sur la figure 7. Nous y constatons que le canyon sur lequel nous focalisons notre attention ne présente que peu de variation puisque les différences d'élévation sont proches d'une valeur constante (non nulle car nous n'avons pas retranché la différence d'élévation de décollage du drone aux deux dates) : les tons clairs sont distribués dans les 70 cm du bas de la palette. Un léger écart résiduel entre les positions des deux MNEs est visible par les versants du canyon qui ne se superposent pas exactement, avec son versant ouest en bleu et le versant est en rouge. Cependant, une zone rouge sombre est nettement visible (indiquée par $dz \sim 3$), attribuée à un glissement de terrain sur une épaisseur de 3 m (quatrième tranche de couleur de la palette à partir du bas), une épaisseur cohérente avec la profondeur du canyon. Cette analyse est confirmée par l'analyse le long d'un transect par le greffon **`Terrain Profile`** (voir figure 7, droite).

Conclusion

Nous avons proposé une séquence d'acquisitions et de traitements de données pour générer un modèle numérique d'élévation de résolution décimétrique. Nous avons comparé l'exactitude des données acquises et ayant conclu à la nécessité d'un recalage manuel en l'absence de points de contrôle au sol géoréférencés, avons observé un glissement de terrain sur un intervalle de temps d'une semaine.

Cette approche, développée initialement sur un intervalle de temps d'environ trois mois après l'acquisition du jeu de données, nécessite désormais une journée de traitements sur un ordinateur quad-cœur Xeon cadencé à 2,13 GHz (8 nœuds de calcul, 32 GB RAM) par vol (en supposant 700 photographies exploitables acquises sur la durée de vol de 20 minutes). Nul doute que les présentations proposées lors du « Colloque Photogrammétrie Numérique et Perception 3D : les Nouvelles Conquêtes » par la Société Française de Photogrammétrie et de Télédétection (http://www.spft.fr **[8]**) entre les 15 et 17 mars apporteront des pistes d'améliorations à cette séquence de traitements.

En attendant, les données sont disponibles sur http://jmfriedt.free.fr/dji_micmac et, pour un second exemple plus proche de nous portant sur le parc de l'Observatoire de Besançon (mettant en évidence les difficultés liées à la végétation), sur http://jmfriedt.free.fr/dji. ■

Remerciements

Cette activité prospective d'exploitation de microdrone pour la cartographie et la génération de MNE est soutenue financièrement par la Région Franche-Comté. Les mesures en Arctique sont soutenues financièrement par l'Agence Nationale pour la Recherche (projet PRISM) et par l'Institut Paul-Émile Victor (projet GRAAL). J.-P. Culas (CM-Drones, Besançon) fournit le soutien technique à la maintenance du drone. Les contributeurs au forum Micmac () ont patiemment répondu aux questions et fourni les solutions ayant permis de faire fonctionner cette séquence de traitements : leur contribution est aussi importante que celle des auteurs du logiciel Micmac lui-même.

Références

[1] FRIEDT J.-M., *« Reconstruction de structures tridimensionnelles par photographies : le logiciel MicMac »*, OpenSilicium n°12, Sept-Oct-Nov 2014.

[2] PIERROT-DESEILLIGNY M., *« MicMac, a free open source solution for photogrammetry »*, RMLL 2015 (vidéo disponible sur https://2015.rmll.info/micmac-une-solution-libre-de-photogrammetrie?lang=en).

[3] NOLAN M., LARSEN C. et STURM M., *« Mapping snow depth from manned aircraft on landscape scales at centimeter resolution using structure-from-motion photogrammetry »*, The Cryosphere, 9, 1445–1463, 2015, disponible sur http://www.the-cryosphere.net/9/1445/2015.

[4] LIN Y., HYYPPÄ J. et JAAKKOLA A., *« Mini-UAV-borne LIDAR for fine-scale mapping »*, IEEE Geoscience and Remote Sensing Letters, 8 (3), 426–430, 2011.

[5] GNU/Octave est une version open source de Matlab proposant une compatibilité syntaxique excellente pour les applications qui nous intéresseront ici

[6] Quantum GIS est un outil open source de Gestion d'Informations Spatialisées. Outre sa capacité de conversion entre les modes de projection, il offre un certain nombre de fonctions de traitements sur données vectorielles et matricielles (bitmap), ainsi que l'analyse statistique de données géoréférencées. Nous nous en servirons pour projeter les points GPS sur un plan tangent à la région d'intérêt, puis insérer nos images dans un fond de carte servant de référence et estimer l'exactitude de nos calculs.

[7] http://gis.stackexchange.com/questions/11519/draw-lines-between-two-points-in-qgis

[8] http://www.sfpt.fr/2015/12/colloque-photogrammetrie-numerique-et-perception-3d-les-nouvelles-conquetes/#more-305

INTERFACES UTILISATEUR EN PYTHON : LE MODE TEXTE

Tristan Colombo

À mi-chemin entre la ligne de commande et l'interface graphique, l'interface textuelle permet de proposer à l'utilisateur un affichage plus convivial que le mode CLI sans pour autant être aussi abouti qu'une interface graphique. En Python il existe quelques modules permettant d'accélérer le développement de ce type d'interfaces.

Mots-clés : Interface textuelle, Npyscreen, Python, Formulaire

Résumé

Toute application se devrait de proposer plusieurs types d'interfaces : une interface en ligne de commandes ou CLI, une interface textuelle et une interface graphique. Ce mois-ci, nous allons tester le module npyscreen permettant de créer simplement des interfaces textuelles. Nous l'emploierons ensuite pour compléter un projet proposant à l'utilisateur de sélectionner le type d'interface qu'il souhaite utiliser.

Sous GNU/Linux vous avez forcément été confronté à un moment ou à un autre à une interface textuelle, l'ancêtre des interfaces graphiques. Mais oui, lorsque vous installez le système, que vous installez ou reconfigurez un paquet ou tout simplement au démarrage lorsque vous accédez au menu de Grub, vous devez utiliser une interface en mode texte. La figure 1 montre une telle interface pour la reconfiguration de PHPMyAdmin. On peut voir une présentation sous forme de pseudo fenêtre, des cases à cocher (notées `[ ]`) et des « boutons » `<Ok>` et `<Annuler>`. Ces différents éléments sont accessibles en appuyant sur les flèches ou sur la touche **<Tab>**, fonctionnement conservé dans les interfaces graphiques correctement conçues.

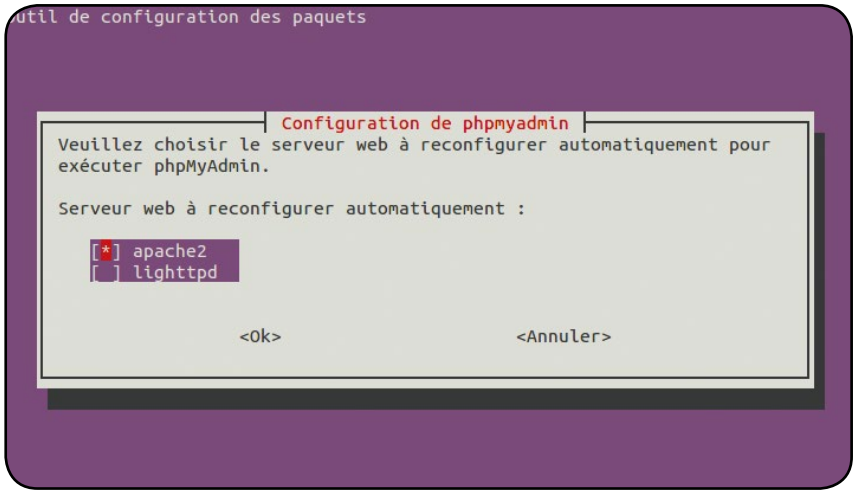


Fig. 1 : Exemple d'interface textuelle : reconfiguration du paquet PHPMyAdmin.

Bien que précédant chronologiquement, les interfaces graphiques, les interfaces textuelles n'en restent pas moins utilisées et fort pratiques : pas de fioritures et utilisation du

clavier en font des outils puissants et, pour certains, un peu moins repoussants que la ligne de commandes. Dans cet article nous allons voir comment développer de telles interfaces en Python à l'aide d'un module nommé **npyscreen**.

1 Le module npyscreen

Il existe essentiellement trois modules Python qui viennent ajouter une surcouche à **curses** pour une utilisation plus simple : **npyscreen [1]**, **textland [2]** et **Urwid [3]**. J'ai choisi d'utiliser le premier pour sa documentation plus claire (ou plutôt moins obscure que pour les autres) et sa simplicité d'utilisation dans les cas les plus courants.

Nous travaillerons dans un environnement virtuel Python 3.4, ce qui nous permettra d'installer le module **npyscreen** à l'aide de **pip** sans « polluer » le système :

```
$ mkvirtualenv contacts --python=/usr/bin/python3.4
$ workon contacts
(contacts) $ pip install npyscreen
```

Si vous le souhaitez, vous pouvez récupérer le code source sur : https://pypi.python.org/pypi/npyscreen/. Vous aurez alors à votre disposition une dizaine d'exemples utilisant le module.

Pour commencer, nous allons voir comment créer une fenêtre. Nous aborderons ensuite des objets graphiques (*widgets*) permettant de composer un formulaire.

1.1 Une fenêtre d'interface textuelle

npyscreen a été créée pour permettre le développement d'interfaces textuelles simplement. Le but d'une telle interface est de communiquer avec l'utilisateur : l'objet principal de npyscreen est donc le formulaire **Form**. Même pour créer une fenêtre « vide », vous devrez créer un formulaire qui contiendra un titre et un bouton de soumission « Ok » :

```
01: import npyscreen
02:
03: class Window(npyscreen.NPSApp):
04:     def main(self):
05:         F  = npyscreen.Form(name = "GNU/Linux Magazine")
06:
07:         F.edit()
08:
09: if __name__ == "__main__":
10:     App = Window()
11:     App.run()
```

La structure employée ici est une structure classique du développement des interfaces graphiques : on crée une instance d'un objet contenant la définition de l'interface (ici **Window**), puis on lance une boucle événementielle infinie qui va analyser les saisies ou les événements de l'utilisateur (ligne 11). La définition de notre fenêtre est donnée dans les lignes 3 à 7 avec une classe héritant de **NPSApp** et définissant dans la méthode **main()** un formulaire (ligne 5) rendu éditable en ligne 7. Notez qu'en supprimant la ligne 7, vous n'obtiendrez aucun affichage, le formulaire étant immédiatement effacé puisque l'utilisateur ne peut pas intervenir. L'exécution de ce code affichera un formulaire vide contenant un bouton ***Ok*** comme on peut le voir sur la figure 2.

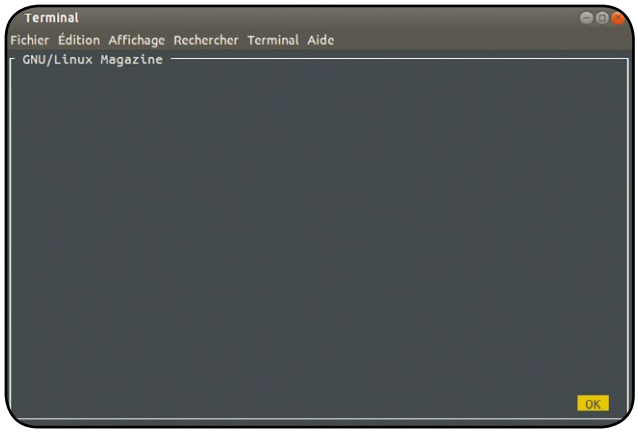


Fig. 2 : Un formulaire contenant un titre et un bouton Ok.

Pour une dizaine de lignes, c'est déjà bien. Voyons maintenant comment changer les couleurs.

1.2 Utilisation des thèmes

npyscreen utilise un gestionnaire de thèmes pour définir les couleurs qui seront associées aux différents types de widget. La méthode **setTheme()** permet de choisir un thème parmi les six prédéfinis : **DefaultTheme** (celui-ci sera utilisé par défaut si aucun thème n'est spécifié), **ElegantTheme**, **ColorfulTheme**, **BlackOnWhiteTheme**, **TransparentThemeDarkText** et **TransparentThemeLightText**. Pour les tester, vous pouvez ajouter avant la ligne 5 la ligne suivante :

```
05:         npyscreen.setTheme(npyscreen.Themes.ColorfulTheme)
```

Cette fois-ci, la bordure et le texte apparaîtront en jaune alors que le bouton Ok sera sur fond vert. Modifiez le nom du thème pour trouver celui qui vous convient.

Mais si aucun ne vous plaît, que faire ? Vous pourrez alors créer votre propre thème en créant une classe héritant de **ThemeManager**. Les couleurs sont énumérées dans un dictionnaire **default_colors** où la clé indique la classe de widget (**DEFAULT**, **LABEL**, **WARNING**, etc) et la valeur est une couleur définie à partir de **curses** dans le tuple **colors_to_define** sous la forme « couleur du texte sur couleur de fond » (par exemple **WHITE_BLACK** pour texte blanc sur fond noir). Vous n'aurez à redéfinir le tuple **colors_to_define** que si vous souhaitez utiliser des couleurs « exotiques » du type bleu sur rouge. Créons un thème utilisant les couleurs déjà définies. Ce thème se trouvera dans le fichier **BeautifulTheme.py** :

```
01: import npyscreen
02:
03: class BeautifulTheme(npyscreen.ThemeManager):
04:     default_colors = {
05:         'DEFAULT'     : 'GREEN_BLACK',
06:         'FORMDEFAULT' : 'GREEN_BLACK',
07:         'NO_EDIT'     : 'BLACK_WHITE',
08:         'STANDOUT'    : 'CYAN_BLACK',
09:         'CURSOR'      : 'WHITE_BLACK',
10:         'LABEL'       : 'GREEN_BLACK',
11:         'LABELBOLD'   : 'WHITE_BLACK',
12:         'CONTROL'     : 'BLUE_BLACK',
13:         'IMPORTANT'   : 'RED_BLACK',
14:         'SAFE'        : 'GREEN_BLACK',
15:         'WARNING'     : 'YELLOW_BLACK',
16:         'DANGER'      : 'RED_BLACK',
17:         'CRITICAL'    : 'BLACK_RED',
18:         'GOOD'        : 'BLUE_BLACK',
19:         'GOODHL'      : 'GREEN_BLACK',
20:         'VERYGOOD'    : 'BLACK_GREEN',
21:         'CAUTION'     : 'YELLOW_BLACK',
22:         'CAUTIONHL'   : 'BLACK_BLUE',
23:     }
```

Pour utiliser ce thème dans l'exemple précédent il faudra l'importer et modifier l'appel à **setTheme()** :

```
...
02:  from BeautifulTheme import BeautifulTheme
...
06:         npyscreen.setTheme(BeautifulTheme)
...
```

Imaginons maintenant que vous souhaitiez utiliser du texte rouge sur fond bleu... **RED_BLUE** n'a pas été défini et vous obtiendrez un message d'erreur du type :

```
KeyError: 'RED_BLUE'
```

Il faut donc reprendre le fichier **BeautifulTheme.py** et y ajouter la définition de nouvelles couleurs en utilisant le module **curses** :

```
01: import npyscreen
02: import curses
03:
04: class BeautifulTheme(npyscreen.ThemeManager):
05:     _colors_to_define = (
06:         ('RED_BLUE', curses.COLOR_RED, curses.COLOR_BLUE),
07:         ('BLUE_YELLOW', curses.COLOR_BLUE, curses.COLOR_YELLOW),
08:     )
09:
10:     default_colors = {
11:         'DEFAULT'     : 'RED_BLUE',
12:         'FORMDEFAULT' : 'RED_BLUE',
13:         'NO_EDIT'     : 'BLUE_YELLOW',
...
25:         'GOODHL'      : 'BLUE_YELLOW',
26:         'VERYGOOD'    : 'RED_BLUE',
27:         'CAUTION'     : 'BLUE_YELLOW',
28:         'CAUTIONHL'   : 'BLUE_YELLOW',
29:     }
30:
31:     def __init__(self, *args, **kwargs):
32:         curses.use_default_colors()
33:         super(BeautifulTheme, self).__init__(*args, **kwargs)
```

1.3 Un formulaire basique

Dans un formulaire, il est important de pouvoir récupérer la valeur saisie par l'utilisateur. Pour cela nous allons ajouter des *widgets* au formulaire à l'aide de la méthode **add()**. Pour l'instant, nous emploierons deux types de *widgets* : **FixedText** qui affiche seulement un texte (sans interaction possible) et **TitleText** qui affiche un message et récupère la valeur saisie par l'utilisateur (valeur accessible via l'attribut **value**).

Voici un exemple où, après la validation du formulaire par l'utilisateur, nous afficherons un petit message rappelant la chaîne de caractères qui a été saisie :

```
01: import npyscreen
02: from BeautifulTheme import BeautifulTheme
03:
04: class Window(npyscreen.NPSApp):
05:     def main(self):
06:         npyscreen.setTheme(BeautifulTheme)
07:         Form = npyscreen.Form(name="GNU/Linux Magazine")
08:         text = Form.add(npyscreen.FixedText, value="Exemple
de formulaire minimal")
09:         user_text = Form.add(npyscreen.TitleText,
name="Saisir texte:")
10:
11:         Form.edit()
12:
13:         npyscreen.notify_wait("Valeur saisie : " + user_text.
value, title="Vérification")
14:
15: if __name__ == "__main__":
16:     App = Window()
17:     App.run()
```

Les champs ajoutés se trouvent en lignes 8 et 9. Le paramètre **value** permet de définir la valeur d'un champ et il est obligatoire pour le *widget* **FixedText** (sinon rien n'apparaît à l'écran) alors que pour **TitleText** il permet le cas échéant de donner une valeur par défaut. Notez que si la chaîne passée en paramètre dans **name** est trop longue, la saisie utilisateur se fera sur la ligne suivante (ce qui ne sera pas forcément très esthétique mais assure l'alignement des champs). En ligne 13 la fonction **notify_wait()** affiche un message pendant quelques secondes à l'écran (voir figure 3).

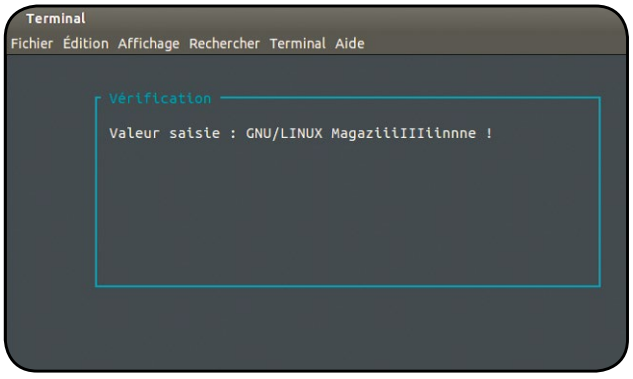


Fig. 3 : Affichage d'un message sous forme de notification.

1.4 Les boutons radio

Le *widget* permettant l'ajout de boutons radio (choix d'une option dans une liste) s'appelle **TitleSelectOne**. Il nécessite forcément plus de paramètres que pour une simple entrée de texte :

```
...
06:         radio_values = ["Option 1", "Option 2", "Option 3"]
07:         radio = Form.add(npyscreen.TitleSelectOne, max_
height=len(radio_values),
08:                          value = [0], name="Choix:", values =
radio_values, scroll_exit=True)
...
```

J'ai choisi de présenter les différents choix possibles dans une liste **radio_values** (ligne 6). Ceci permet d'indiquer simplement les options de choix possibles (paramètre **values** en ligne 8) et la taille d'affichage souhaitée (paramètre **max_height** en ligne 7). En choisissant une taille inférieure au nombre d'options, l'utilisateur verra apparaître un choix **- more -** permettant de *scroller* entre les différents choix possibles. En ligne 7 on peut voir encore deux paramètres : **value** permet d'indiquer le choix par défaut - le bouton qui sera marqué dès l'affichage et le paramètre **scroll_exit** qui, s'il vaut **True**, permet de sortir du choix boutons radio à l'aide des flèches (dans le cas contraire il

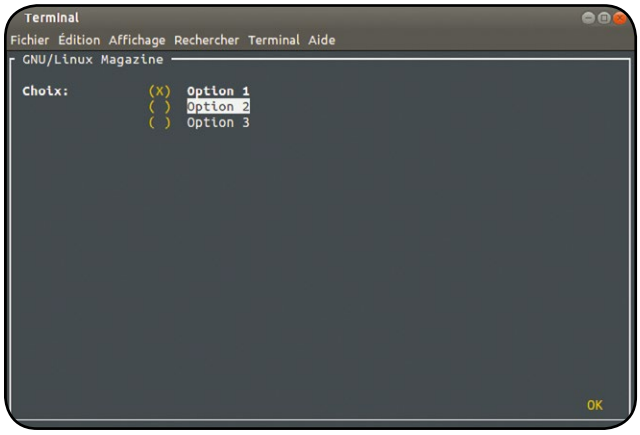


Fig. 4 : Utilisation de boutons radio.

faudra appuyer sur la touche **<Tab>**). La figure 4 montre comment npyscreen affiche des boutons radio.

Pour récupérer le choix de l'utilisateur, il faut utiliser la méthode **get_selected_objects()** qui renvoie une liste. Bien sûr comme un seul choix est ici possible, nous prendrons toujours le premier et unique élément :

```
...
12:         npyscreen.notify_wait("Valeur saisie : " + radio.
get_selected_objects()[0],
13:
title="Vérification")
...
```

Une méthode qui renvoie une liste pour un seul élément peut paraître étrange... mais en fait c'est cette méthode qui sera également utilisée pour les cases à cocher.

1.5 Les cases à cocher

Le fonctionnement des cases à cocher, permettant plusieurs choix dans une liste, est exactement le même que pour les boutons radio à la différence prêt que l'on utilise un *widget* **TitleMultiSelect** :

```
...
06:         cbox_values = ["Option 1", "Option 2", "Option 3"]
07:         cbox = Form.add(npyscreen.TitleMultiSelect, max_
height=len(cbox_values),
08:                                      value=[1], name="Choix
:", values = cbox_values, scroll_exit=True)
...
12:         npyscreen.notify_wait("Valeur saisie : " + str(cbox.
get_selected_objects()),
13:
title="Vérification")
...
```

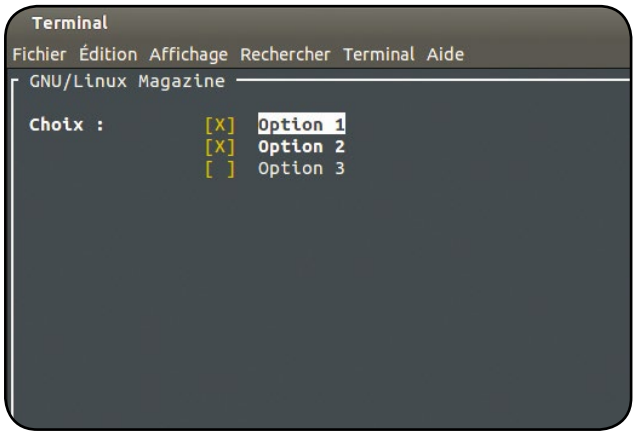


Fig. 5 : Utilisation de cases à cocher.

Pour pouvoir différencier rapidement les cases à cocher des boutons radio, leur affichage est différent : on utilise des crochets (voir figure 5) alors que pour les cases à cocher il s'agit de parenthèses (voir figure 4).

1.6 L'auto-complétion

Lorsqu'en introduction je disais que la documentation de npyscreen était moins obscure que pour les autres modules du même type, c'était vrai pour tous les *widgets* simples. En abordant l'auto-complétion, la documentation nous annonce : *« See the Filename and TitleFilename classes for examples »*. Ce qui, dans une traduction non littérale pourrait s'écrire : Débrouillez-vous (pour rester poli).

Le principe de l'auto-complétion est assez simple : on va étendre un champ de texte **TitleText** en lui indiquant comment gérer l'auto-complétion qui sera activée par un appui sur la touche **<Tab>**. Pour cela il faut créer deux classes : une classe qui va indiquer la liste des choix possibles et une classe qui utilisera la classe précédente pour étendre **TitleText**. La seconde classe étant minuscule, nous placerons ces deux classes dans un même fichier **MyTitleAutocomplete.py** :

```
01: import npyscreen
02:
03: class MyTitleAutocompleter(npyscreen.Autocomplete):
04:     possibilities = ['Abba', 'Benabar', 'Cranberries']
05:
06:     def auto_complete(self, input):
07:         choices = []
08:
09:         for word in MyTitleAutocompleter.possibilities:
10:             if word.startswith(self.value):
11:                 choices.append(word)
12:
13:         self.value = choices[self.get_choice(choices)]
14:
15: class MyTitleAutocomplete(npyscreen.TitleText):
16:     _entry_type = MyTitleAutocompleter
```

La classe **MyTitleAutocompleter**, qui gère l'auto-complétion, hérite de **Autocomplete** et doit définir une méthode **auto_complete()** (lignes 6 à 13). On commence par créer une liste des choix possibles (ligne 4) sous forme d'attribut de classe. Dans la méthode **auto_complete()** on crée une liste vide (ligne 7) qui contiendra les choix qui commencent par les lettres saisies par l'utilisateur. Le remplissage de cette liste se fait dans les lignes 9 à 11 à l'aide de l'attribut **self.value** qui contient la saisie (partielle) de l'utilisateur. En ligne 13, on place dans **self.value** le choix de l'utilisateur qui aura été fait dans une liste affichée grâce à la méthode **get_choice()** (qui prend en paramètre une liste de choix et renvoie l'indice de l'élément sélectionné). Enfin, la définition de la classe **MyTitleAutocomplete** des lignes 15 et 16 est très simple puisqu'elle hérite de **TitleText** et qu'il faut seulement indiquer que l'attribut statique **_entry_type** est un pointeur vers la classe **MyTitleAutocompleter** définie précédemment.

L'utilisation de ce nouveau *widget* est ensuite classique :

```
01: import npyscreen
02: from MyTitleAutocomplete import MyTitleAutocomplete
...
07:         text = Form.add(MyTitleAutocomplete, name = "Texte :")
...
08:
11:         npyscreen.notify_wait("Valeur saisie : " + text.value,
title="Vérification")
...
```

Au lancement, lorsque l'utilisateur appuie sur la touche **<Tab>** une liste de choix complétant les lettres qu'il a déjà saisies apparaîtra (voir figure 6).

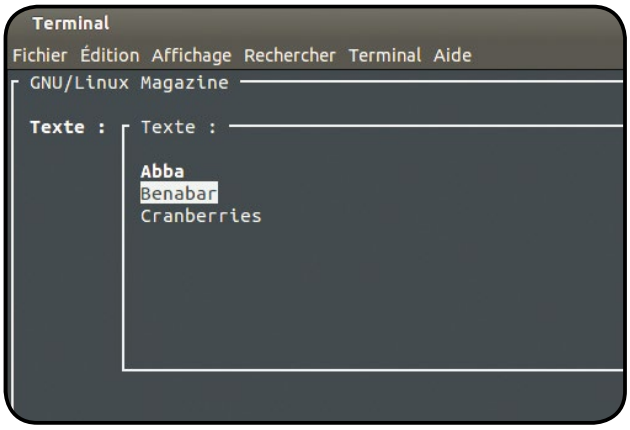


Fig. 6 : Un champ de texte avec auto-complétion en cours d'utilisation

2 Une interface en mode texte dans un projet

Nous avons vu comment utiliser quelques-uns des *widgets* principaux de npyscreen mais seulement au sein de petits exemples. Je vous propose de voir maintenant son utilisation sur un véritable projet. Pour cela, je vais repartir du projet présenté dans un article précédent **[4]** où nous avions créé un petit annuaire dans lequel nous stockions les nom, prénom et adresse mail des contacts et où nous pouvions envoyer un mail à un contact en donnant uniquement son prénom et son nom. Ce mail contenait un petit message et un fichier pdf qui était généré automatiquement et qui indiquait le temps :

```
Bonjour <Prénom> <Nom>,

Ceci est un message généré automatiquement.
Aujourd'hui il <fait beau>/<pleut> !
```

La fin de la dernière phrase était déterminée par l'utilisateur lors de l'appel du programme.

Le code de ce projet est disponible sur notre dépôt Github associé au n°190. L'intérêt de ce projet était de proposer trois types d'interfaces à l'utilisateur : la ligne de commandes, une interface textuelle et une interface graphique.

C'est le fichier **`contacts.py`** qui permet de lancer la fonction associée au type d'interface choisi. Pour le mode ligne de commandes, il fallait transmettre le prénom et le nom d'un contact sous forme d'une chaîne de caractères **`args.name`** et les variables de connexion à la base SQLite3 **`args.base`** et **`args.cursor`** :

```
01: def cli_mode(args):
02:     person = Contact(args.name, (args.base, args.cursor))
03:     forename = person.forename
04:     name = person.name
05:     mail = person.mail
06:
07:     if mail == None:
08:         print('Nouveau contact')
09:         mail = input('Veuillez saisir le mail: ')
10:         person.mail = mail
11:
12:     person.generateDoc(args.sun)
13:     person.sendMail()
14:
15:     print('Le mail a été envoyé à', forename, name)
```

Nous devons repartir de ce code pour obtenir le même traitement mais avec une interface textuelle. À la rigueur, en dehors du fait de savoir que c'est la fonction **`text_mode()`** qui sera exécutée pour ce type d'interface, il n'y a rien d'autre à savoir. Pour notre interface nous aurons besoin :

- d'un champ de texte avec auto-complétion pour la saisie/recherche d'un contact par son prénom et son nom ;
- d'un champ de texte « mail » qui sera complété automatiquement si l'utilisateur est connu ;
- de cases à cocher permettant d'indiquer le temps.

La difficulté va résider ici dans la communication entre le champ contenant le prénom et le nom de l'utilisateur avec le champ de mail. Le comportement de l'interface textuelle sera défini dans une classe **`TextUI`** (fichier **`TextUI.py`**). Au niveau du fichier **`contacts.py`** les modifications seront donc mineures :

```
...
17: from TextUI import TextUI
...
36: def text_mode(args):
37:     app = TextUI()
38:     app.run()
```

On retrouve dans les lignes 36 à 38 le code qui correspondait au programme principal de nos exemples.

Nous allons maintenant commencer par implémenter une première version de la classe **`TextUI`** sans mettre en place l'auto-complétion.

2.1 Interface textuelle non fonctionnelle et sans auto-complétion

Pour voir vers quoi nous nous dirigeons, il est important d'avancer pas à pas. La première étape est l'obtention de l'interface textuelle sous forme de coquille vide. On veut seulement afficher les champs dont nous aurons besoin :

```
01: """
02:     Interface textuelle pour envoi de mails
...
12: """
13:
14: import npyscreen
15:
16: class TextUI(npyscreen.NPSApp):
17:     weather_values = ["Soleil", "Pluie"]
18:
19:     def __init__(self, db_args, *args, **kwargs):
20:         self.db_args = db_args
21:         super().__init__(*args, **kwargs)
22:
23:     def main(self):
```

```
24:         form = npyscreen.Form(name="Agenda")
25:         contact = form.add(npyscreen.TitleText, name =
"Contact:")
26:         mail = form.add(npyscreen.TitleText, name = "Mail:")
27:         weather = form.add(npyscreen.TitleSelectOne,
28:                 max_height=len(TextUI.weather_values),
29:                 value = [0], name="Temps:", values = TextUI.
weather_values, scroll_exit=True)
30:
31:         form.edit()
32:
33:         npyscreen.notify_wait("Contact saisi : " + contact.
value, title="Vérification")
```

Nous n'utilisons que des éléments vus précédemment. Notez l'ajout des choix possibles pour les boutons radio sous forme d'attribut statique en ligne 17 (si on veut ajouter d'autres choix par la suite ce sera plus simple à modifier) et la réécriture du constructeur dans les lignes 19 à 21 de manière à conserver sous forme d'attributs les variables de connexion à la base de données. Cette interface ne fonctionne pas mais permet d'afficher l'écran présenté en figure 7.

Fig. 7 : Interface textuelle pour le projet d'agenda.

2.2 Interface textuelle avec auto-complétion

Ajoutons maintenant les fonctionnalités permettant d'enregistrer un nouveau contact et d'envoyer le mail.

Lorsqu'un contact existe, il faut pouvoir le proposer dans l'auto-complétion en fonction des premières lettres de son nom ou de son prénom. Le problème qui va se poser est la transmission des variables de connexion d'objet en objet sachant qu'il y a de nombreux héritages dans npyscreen. Du coup nous n'avons pas d'autre solution que d'externaliser les fonctions de connexion dans un fichier **DB.py** et de les utiliser dans la fonction d'auto-complétion.

```
...
14: import sqlite3
15:
16: def create_db():
17:     base = sqlite3.connect('base.db')
18:     cursor = base.cursor()
19:
20:     try:
21:         cursor.execute("""create table Contact
22:                         (idContact integer primary key,
23:                         name text,
24:                         forename text,
25:                         mail text)""")
26:         base.commit()
27:     except sqlite3.OperationalError:
28:         pass
29:
30:     return (base, cursor)
31:
32:
33: def close_db(base, cursor):
34:     cursor.close()
35:     base.close()
```

Ces fonctions seront ensuite employées dans la classe **TextUI** (fichier **TextUI.py**) de manière à ouvrir la base et à renvoyer la liste des noms valides pour l'auto-complétion. Cette liste sera générée par la classe **Contact** :

```
...
056:     @staticmethod
057:     def setDB(db_args):
058:         if Contact.base is None:
059:             Contact.base = db_args[0]
060:         if Contact.cursor is None:
061:             Contact.cursor = db_args[1]
062:
...
095:     @staticmethod
096:     def complete(value):
097:         value = Contact.capitalizeName(value) + "%"
098:         result = Contact.cursor.execute("""select forename,
name from Contact
099:                                 where forename like ?
100:                                 or name like ?""", (value,
value))
101:         contacts = []
102:         for row in result.fetchall():
103:             contacts.append(row[0] + " " + row[1])
104:
105:         if len(contacts) == 0:
106:             contacts = ["-- Aucun --"]
107:
108:         return contacts
...
```

Les deux méthodes définies dans ce code sont des méthodes statiques (elles peuvent donc être appelées sans qu'aucune instance de **Contact** n'ait été créée). Pour la complétion, nous construisons une requête qui recherche dans la base un contact dont le prénom ou le nom commence par les lettres tapées par l'utilisateur et transmises dans le paramètre **value** (lignes 98 à 100). Notez qu'en

ligne 97 nous transformons la casse des lettres de la saisie en accord avec le stockage dans la base (voir la méthode **capitalizeName()**) et que nous ajoutons à la fin le caractère **%**, synonyme de joker. Nous construisons ensuite une liste **contacts** avec l'ensemble des contacts trouvés (lignes 101 à 103) et si aucun contact n'est renvoyé, nous insérons dans la liste la chaîne de caractères **-- Aucun --**.

Dans le fichier **TextUI.py** nous allons ajouter les classes **ContactAutoCompleter** et **ContactAutocomplete** permettant de mettre en place l'auto-complétion :

```
...
14: import npyscreen
15: import DB
16: from Contact import Contact
17:
18: class ContactAutocompleter(npyscreen.Autocomplete):
19:     def auto_complete(self, input):
20:         db_args = DB.create_db()
21:         Contact.setDB(db_args)
22:
23:         choices = Contact.complete(self.value)
24:
25:         self.value = choices[self.get_choice(choices)]
26:
27:
28:
29: class ContactAutocomplete(npyscreen.TitleText):
30:     _entry_type = ContactAutocompleter
31:
32:
33: class TextUI(npyscreen.NPSApp):
34:     weather_values = ["Soleil", "Pluie"]
35:
36:     def main(self, *args, **kwargs):
37:         form = npyscreen.Form(name="Agenda")
38:         contact = form.add(ContactAutocomplete, name="Contact:")
...
```

Une fois que la méthode **complete()** est écrite dans la classe **Contact**, il n'y a plus beaucoup de travail à effectuer pour créer les classes d'auto-complétions : on ouvre la base (lignes 20 et 21), on cherche la liste des contacts (ligne 23) et on affiche cette liste pour que l'utilisateur puisse choisir une entrée (ligne 25). Il faut bien penser à changer le type de *widget* en ligne 38 pour utiliser l'auto-complétion !

npyscreen est en cours de développement : la fonction permettant de détecter la prise de focus d'un champ n'existe pas encore et la liaison entre plusieurs champs est complexe. Nous ne pouvons donc pas remplir automatiquement (et simplement) le champ mail d'un contact saisi dans le champ précédent, ce qui rend l'interface peu pratique (si ce n'est inutilisable dans certains cas). La solution la plus simple serait de créer deux formulaires : le premier demandant un nom d'utilisateur et le second un mail puis la météo, le nom d'utilisateur étant alors connu avant la création du champ mail. Cette solution permettrait de réaliser une interface fonctionnelle mais ne présente aucun intérêt technique puisqu'il s'agit pratiquement d'une répétition de ce que nous avons déjà vu.

Avant de finir, notez tout de même qu'il est possible d'opter pour une structure plus orientée objet que celle que nous avons utilisée en créant une classe dédiée au formulaire :

```
class MyForm(npyscreen.Form):
    def create(self):
        self.contact = self.add(ContactAutocomplete, name="Contact:")
        self.mail = self.add(MailAutocomplete, name="Mail:")
        ...
```

Cette classe sera ensuite utilisée dans la méthode **main()** pour créer ledit formulaire :

```
        form = MyForm(name="Agenda")
```

Conclusion

Il est toujours intéressant de gagner du temps grâce à des modules bien construits et bien documentés... mais pour ce qui est des interfaces textuelles qui sont peu utilisées, il faut reconnaître que le choix est limité. Pour une utilisation basique, npyscreen répondra tout à fait à la demande et n'hésitez surtout pas à l'employer... mais comme nous avons pu le voir en tentant un développement plus poussé, le gain de temps du départ sera neutralisé par la difficulté de communication et de mise à jour des *widgets*. Cette difficulté n'est bien sûr pas insurmontable mais il faudra soit sacrifier une partie de l'ergonomie de l'interface, soit devoir naviguer dans le code du projet, éventuellement devoir le modifier, et aboutir à une architecture bancale. Peut-être est-il alors plus bénéfique d'utiliser directement **curses** ? Vous passerez bien sûr du temps sur ce module mais vous pourrez réaliser exactement ce que vous souhaitez ! ■

Références

[1] Le projet npyscreen : http://www.npcole.com/npyscreen/

[2] Le projet textland : https://github.com/zyga/textland

[3] Site officiel de Urwid : http://urwid.org/index.html

[4] COLOMBO T., *« Interfaces utilisateurs en Python : le mode CLI »*, GNU/Linux Magazine n°190, février 2016, p. 56 à 63.

WILDFLY SWARM

Romain Pelisse [Sustain Developer @Red Hat] Relecture d'Aurélie Garnier

Si les microservices ont le vent en poupe, ils semblent parfois incompatibles avec le modèle proposé par les serveurs d'applications JEE [1] tel que Widlfy [2]. Heureusement, Wildfly Swarm [3] est là pour vous permettre d'allier le meilleur des deux mondes sans difficulté.

Mots-clés : Java, JEE, MicroServices, Wildfly

Résumé

Pré requis : Java/JEE, Maven, Wildfly

L'objet de cet article est d'expliquer comment développer des microservices tout en utilisant l'infrastructure technique du serveur JEE Wildfly [2], à l'aide de Wildfly Swarm [3].

L'histoire de l'informatique est faite d'une constante oscillation, optant un jour pour une stratégie, pour reculer à toute vitesse le lendemain. En effet, les décisions technologiques semblent parfois comme assises sur une balançoire où les bonnes pratiques d'hier forment déjà les anti-patterns de demain.

Les illustrations de ce phénomène sont très nombreuses mais le plus facile à identifier est certainement le balancement entre un client « léger », puis « lourd », puis finalement « Web » (donc « léger »), puis « riche » (et en effet de nouveau « lourd »). C'est important à prendre en compte dans le cadre de cet article, car les microservices ne sont finalement aujourd'hui que la réponse à un développement, souvent jugé trop lourd et complexe, basé sur le standard JEE et sur ses implémentations.

Comme nous l'a appris mille fois l'histoire de l'informatique, une légère méfiance est de rigueur. Certes, il peut paraître plus simple de développer un microservice comme une application indépendante, embarquant sa propre pile logicielle, pour répondre à ses besoins, plutôt que de déployer son application sur un serveur **JEE**. Mais que va-t-il se passer quand le besoin de gestion de transaction ou de disposition d'un cache local va apparaître ?

Les anti JEE primaire répondront en cœur qu'il suffit d'ajouter toutes les librairies appropriées à la pile logicielle. Mais ces librairies ne sont-elles pas déjà justement placées au cœur d'un serveur JEE ? Est-ce vraiment pertinent, et plus sûr au final, de refaire tout un lourd travail d'intégration ? Surtout si votre compagnie est cliente de Red Hat et dispose de supports pour JBoss EAP - car ce travail est déjà fait par Red Hat, ou même simplement par la communauté **Wildfly**.

Note

Notez néanmoins que l'utilisation de Wildfly Swarm conjointement à JBoss EAP n'est pas, à ce jour, supportée par Red Hat.

Ne serait-il pas plus pertinent de réutiliser autant que possible le serveur existant pour construire son microservice ? Surtout que ce dernier, depuis sa version 9, a été justement construit pour être aisément composé des seuls services dont vous avez besoin **[4]**. Et c'est justement tout le propos du projet **Wildfly Swarm** : mettre à disposition cette modularité pour la conception de microservices comme nous allons le décrire et le démontrer dans cet article.

1 Mon microservice « ReST » avec Swarm

1.1 Un service « ReST » en Java avec JAX-RS

Avant de regarder en détail Wildfly Swarm et sa mise en place, voici une petite piqûre de rappel sur la mise en place à l'aide d'un standard issu de JEE, **JAX-RS [5]**, d'un service « ReST » en Java. Nous allons donc rapidement voir comment mettre en place un très simple service qui nous servira d'exemple concret d'utilisation de Wildfly Swarm.

On notera de suite que si les technologies Java, et spécialement celles issues de JEE, ont souvent privilégié une approche, disons, quelque peu complexe, voire cryptique, cette spécification échappe drastiquement à la règle et propose, au contraire, une API d'une grande clarté, et elle est donc très facile à prendre en main et à mettre en place. En effet, il suffit de quelques annotations sur les méthodes d'une classe pour que le *container* d'exécution (dans notre cas un serveur d'application JEE tel que Wildfly) se charge d'exposer ces dernières sous forme de service « ReST » :

```
package org.belaran;

import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;

@Path("/")
public class MyRestEndpoint {

    private String message = "hello";

    @GET
    @Path("/test")
    @Produces("text/plain")
    public Response test() {
     return  Response.ok(message).build();
    }

    @PUT
    @Path("/add")
    @Consumes("text/plain")
    @Produces("text/plain")
    public Response push(String message) {
        return Response.ok(setAndReturn(message)).build();
    }

    private String setAndReturn(String message) {
     this.message = message;
     return this.message;
    }
}
```

Voilà, nous avons maintenant un service « ReST » minimal, implémenté à l'aide d'un standard, et notre objectif va donc être d'utiliser **Swarm** pour construire un microservice, que nous démarrerons comme une simple application Java, mais embarquant toutes les briques nécessaires issues de Wildfly.

1.2 Mise en place du fichier de construction Maven

Comme l'objectif est de réaliser un « microservice » sous forme d'une application Java indépendante (*standalone*), nous allons donc devoir compiler et construire notre application. Pour ce faire, le projet Swarm s'intègre donc naturellement avec **Maven** – le standard de facto, du moins pour le moment, de l'industrie Java :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.belaran</groupId>
  <artifactId>simple-swarm</artifactId>
  <version>1.0-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>org.wildfly.swarm</groupId>
      <artifactId>wildfly-swarm-jaxrs</artifactId>
      <version>1.0.0.Alpha4</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.wildfly.swarm</groupId>
        <artifactId>wildfly-swarm-plugin</artifactId>
        <configuration>
          <mainClass>org.belaran.App</mainClass>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>package</goal>
```

```
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

Ce fichier **pom.xml** ne contient que le strict minimum pour faire fonctionner Swarm :

- l'ajout d'une dépendance vers **wildfly-swarm-jaxrs** qui indique quel jeu de dépendances nous sera nécessaire (ici, on indique qu'on doit ensuite déployer un service « ReST » et donc embarquer les dépendances liées à l'implémentation de JAX-RS fournies par Wildfly) ;
- la configuration du plugin Maven **wildfly-swarm-plugin** dont le principal élément est le nom de la classe principale, dans notre cas **org.belaran.App**, dont nous allons discuter du contenu ci-après.

1.3 La classe principale

Si le fichier **pom.xml** nous a déjà surpris par sa simplicité et par l'absence de configuration complexe, la classe principale de notre futur « microservice » va, elle aussi, se réduire au strict minimum :

```
package org.belaran;

import org.wildfly.swarm.container.Container;

public class App {

    public static void main(String[] args) throws Exception {
        new Container().container.start();
    }
}
```

Comme on peut le voir, notre méthode **main()** se limite à instancier un objet **Container**, représentant notre conteneur d'exécution – soit le serveur JEE Wildfly, et à la démarrer. Un seul import et une seule instance masquant avec beaucoup d'élégance la complexité sous-jacente !

Faisons un rapide test pour voir si déjà ceci fonctionne sans problèmes. Swarm s'intégrant pleinement avec Maven, on peut donc exécuter notre classe à l'aide de ce dernier :

```
$ mvn wildfly-swarm:run
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the
effective model for org.belaran:simple-swarm:jar:1.0-SNAPSHOT
[WARNING] 'build.plugins.plugin.version' for org.wildfly.
swarm:wildfly-swarm-plugin is missing. @ line 31, column 13
[WARNING]
[WARNING] It is highly recommended to fix these problems because
they threaten the stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer
support building such malformed projects.
[WARNING]
Downloading: https://repo.maven.apache.org/maven2/org/wildfly/
swarm/wildfly-swarm-plugin/1.0.0.Alpha4/wildfly-swarm-plugin-
1.0.0.Alpha4.pom
Downloaded: https://repo.maven.apache.org/maven2/org/wildfly/
swarm/wildfly-swarm-plugin/1.0.0.Alpha4/wildfly-swarm-plugin-
1.0.0.Alpha4.pom (0 B at 0.0 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/wildfly/
swarm/wildfly-swarm-parent/1.0.0.Alpha4/wildfly-swarm-parent-
1.0.0.Alpha4.pom
Downloaded: https://repo.maven.apache.org/maven2/org/wildfly/
swarm/wildfly-swarm-parent/1.0.0.Alpha4/wildfly-swarm-parent-
1.0.0.Alpha4.pom (0 B at 0.0 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/wildfly/
swarm/wildfly-swarm-plugin/1.0.0.Alpha4/wildfly-swarm-plugin-
1.0.0.Alpha4.jar
Downloaded: https://repo.maven.apache.org/maven2/org/wildfly/
swarm/wildfly-swarm-plugin/1.0.0.Alpha4/wildfly-swarm-plugin-
1.0.0.Alpha4.jar (0 B at 0.0 KB/sec)
[INFO]
[INFO] ------------------------------------------------------
[INFO] Building simple-swarm 1.0-SNAPSHOT
...
[INFO]
[INFO] --- wildfly-swarm-plugin:1.0.0.Alpha4:run (default-cli) @
simple-swarm ---
tmpDir: /tmp
08:18:40,492 INFO  [org.jboss.msc] (main) JBoss MSC version
1.2.6.Final
08:18:40,589 INFO  [org.jboss.as] (MSC service thread 1-7)
WFLYSRV0049: WildFly Core 2.0.0.Beta1 "Kenny" starting
2015-12-15 08:18:41,193 INFO  [org.wildfly.extension.io]
(ServerService Thread Pool -- 11) WFLYIO001: Worker 'default'
has auto-configured to 8 core threads with 64 task threads based
on your 4 available processors
2015-12-15 08:18:41,207 INFO  [org.jboss.as.security]
(ServerService Thread Pool -- 14) WFLYSEC0002: Activating
Security Subsystem
2015-12-15 08:18:41,210 INFO  [org.jboss.as.security]
(MSC service thread 1-2) WFLYSEC0001: Current PicketBox
version=4.9.2.Final
2015-12-15 08:18:41,212 INFO  [org.jboss.as.naming]
(ServerService Thread Pool -- 13) WFLYNAM0001: Activating Naming
Subsystem
2015-12-15 08:18:41,223 INFO  [org.wildfly.extension.undertow]
(MSC service thread 1-5) WFLYUT0003: Undertow 1.3.0.Beta6
starting
2015-12-15 08:18:41,224 INFO  [org.wildfly.extension.undertow]
```

```
(ServerService Thread Pool -- 10) WFLYUT0003: Undertow
1.3.0.Beta6 starting
2015-12-15 08:18:41,232 INFO  [org.jboss.as.naming] (MSC service
thread 1-7) WFLYNAM0003: Starting Naming Service
2015-12-15 08:18:41,257 INFO  [org.xnio] (MSC service thread
1-6) XNIO version 3.3.1.Final
2015-12-15 08:18:41,289 INFO  [org.xnio.nio] (MSC service thread
1-6) XNIO NIO Implementation Version 3.3.1.Final
2015-12-15 08:18:41,330 INFO  [org.wildfly.extension.undertow]
(MSC service thread 1-4) WFLYUT0012: Started server default-
server.
2015-12-15 08:18:41,389 INFO  [org.wildfly.extension.undertow]
(MSC service thread 1-4) WFLYUT0006: Undertow HTTP listener
default listening on /0:0:0:0:0:0:0:0:8080
2015-12-15 08:18:41,516 INFO  [org.jboss.as] (Controller Boot
Thread) WFLYSRV0025: WildFly Core 2.0.0.Beta1 "Kenny" started
in 1070ms - Started 65 of 71 services (13 services are lazy,
passive or on-demand)
```

1.4 Déploiement du service « ReST »

Tout ceci fonctionne bien, mais pour le moment, notre méthode **`main()`** n'est qu'un raccourci pour démarrer une instance minimale du serveur Wildfly. On est encore loin d'avoir déployé et démarré notre service « ReST » conçu plus haut. C'est ce que nous allons faire maintenant.

Pour ceci, nous allons utiliser l'API **Shrinkwrap [6]**. Déjà évoquée lors d'un précédent article sur le *framework* **Arquillian**, cette API permet de construire, de manière programmatique, des archives Java. Dans notre cas, puisque nous déployons un service ReST, nous allons construire une archive de type WAR (*Web Application aRchive*).

```
JAXRSArchive deployment = ShrinkWrap.create( JAXRSArchive.class ,
MyRestEndpoint.class.getSimpleName() +  ".war");
deployment.addResource(MyRestEndpoint.class);
deployment.addAllDependencies();
```

Le code est, là encore, relativement transparent : on crée le squelette d'une archive WAR à l'aide de ShrinkWrap auquel on ajoute notre précédente classe **`MyRestEndpoint`**. Pour faire bonne mesure, on demande aussi à ShrinkWrap d'ajouter toutes les dépendances nécessaires à notre archive.

Une fois ceci fait, il ne reste plus qu'à modifier légèrement le démarrage du conteneur pour déployer notre archive :

```
new Container().deploy(deployment).start();
```

Reconstruisons notre projet et démarrons à nouveau notre microservice :

```
...
2015-12-15 10:57:49,905 INFO  [org.jboss.as] (Controller Boot Thread)
WFLYSRV0025: WildFly Core 2.0.0.Beta1 "Kenny" started in 1111ms -
Started 65 of 71 services (13 services are lazy, passive or on-demand)
2015-12-15 10:57:50,378 INFO  [org.jboss.as.server.deployment]
(MSC service thread 1-7) WFLYSRV0027: Starting deployment of
"MyRestEndpoint.war" (runtime-name: "MyRestEndpoint.war")
2015-12-15 10:57:50,850 INFO  [org.wildfly.extension.undertow] (MSC
service thread 1-7) WFLYUT0018: Host default-host starting
2015-12-15 10:57:51,089 INFO  [org.jboss.resteasy.spi.
ResteasyDeployment] (ServerService Thread Pool -- 6) Deploying
javax.ws.rs.core.Application: class org.wildfly.swarm.generated.
WildFlySwarmDefaultJAXRSApplication
2015-12-15 10:57:51,120 INFO  [org.wildfly.extension.undertow]
(ServerService Thread Pool -- 6) WFLYUT0021: Registered web context: /
2015-12-15 10:57:51,231 INFO  [org.jboss.as.server] (main)
WFLYSRV0010: Deployed "MyRestEndpoint.war" (runtime-name :
"MyRestEndpoint.war")
```

Comme l'illustre la dernière ligne, notre archive **`MyRestEndpoint.war`** a bien été déployée avec succès. Reste à vérifier si le service fonctionne comme attendu. Ce que l'on peut faire aisément en accédant à l'url **`/check`** :

```
$ curl http://127.0.0.1:8080/check
check
```

Notre service a répondu sans problèmes ! Vérifions maintenant que nous pouvons aussi lui passer des paramètres, à l'aide de la méthode **`/add`** :

```
$ curl http://127.0.0.1:8080/add -d 'Hello' -H "Content-Type:
text/plain" -X PUT
Hello
```

2 Aller un peu plus loin – moniteur transactionnel

Une des grandes différences entre le monde JEE et le monde ReST et NoSQL, est l'importance placée sur la notion de transaction. En effet, au début de la norme JEE, un des objectifs clairement avoués était la gestion des transactions au sein du conteneur et l'assurance par là que n'importe quelle opération aurait un comportement transactionnel clair et défini. Dans le monde ReST, sans état, et NoSQL, les transactions sont loin d'être au centre des problématiques discutées.

C'est pour ceci qu'il semble très pertinent d'illustrer maintenant comment nous pouvons modifier notre petit service ReST pour le rendre transactionnel.

2.1 Modification du fichier de construction

En tant que serveur JEE, Widlfly dispose bien sûr d'un moniteur transactionnel, il suffit donc d'ajouter la dépendance appropriée pour intégrer ce dernier à la pile logicielle de notre microservice :

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>wildfly-swarm-transactions</artifactId>
  <version>${version.wildfly-swarm}</version>
</dependency>
```

Et voilà, c'est tout ce qui est nécessaire comme changement !!!

2.2 Modification du service

Maintenant si un moniteur transactionnel est bien déployé au sein de notre service, reste encore à lui donner les moyens d'agir. Nous allons donc lui indiquer, par l'intermédiaire de simples annotations quelles méthodes effectuent les opérations de *commit* ou de *rollback* :

```
    @GET
    @Produces("text/plain")
    public String init() throws Exception {
        return "Active";
    }

    @Path("/commit")
    @GET
    @Produces("text/plain")
    public Response beginCommit() throws Exception {
        UserTransaction txn = (UserTransaction) new
InitialContext().lookup("java:comp/UserTransaction");
        String value = "Transaction ";

        try {
            txn.begin();

            value += "begun ok";

            try {
                txn.commit();

                value += " and committed ok";
            } catch (final Throwable ex) {
                value += " but failed to commit";
            }
        } catch (final Throwable ex) {
            value += "failed to begin: " + ex.toString();
        }

        return Response.ok(value).build();
    }

    @Path("/rollback")
    @GET
    @Produces("text/plain")
    public Response beginRollback() throws Exception {
        UserTransaction txn = (UserTransaction) new
InitialContext().lookup("java:comp/UserTransaction");
        String value = "Transaction ";

        try {
            txn.begin();

            value += "begun ok";

            try {
                txn.rollback();

                value += " and rolled back ok";
            } catch (final Throwable ex) {
                value += " but failed to rollback " +
ex.toString();
            }
        } catch (final Throwable ex) {
            value += "failed to begin: " + ex.toString();
        }

        return Response.ok(value).build();
    }
```

Maintenant, vérifions que le support transactionnel a bien été activé :

```
$ curl  http://127.0.0.1:8080/
Active
```

Tentons de réaliser une transaction :

```
$ curl http://127.0.0.1:8080/commit
Transaction begun ok and committed ok
```

Et enfin, testons le *rollback* !

```
$ curl http://127.0.0.1:8080/rollback
Transaction begun ok and rolled back ok
```

Conclusion

Comme annoncé au début de cet article, nous avons donc fait un rapide tour des possibilités offertes par Wildfly Swarm et, espérons-le, démontré qu'avec ce dernier il est

extrêmement simple de construire sa propre version de Wildfly, embarqué au sein d'une simple application Java indépendante, mais bénéficiant toujours de nombreux services disponibles dans le serveur JEE. Nul besoin de reconstruire manuellement son propre « serveur d'application » par l'ajout de dizaines de dépendances, sans compter celles ajoutées de manière transitive, dont il faudra par la suite maintenir non seulement la cohérence, mais aussi garantir le bon fonctionnement en parallèle. La communauté Wildlfy, et les équipes associées chez Red Hat, font déjà ce lourd travail pour vous, pourquoi le refaire vous-même ?

Notons aussi que tout ceci n'est bien évidemment qu'une mise en bouche ; vous pouvez faire beaucoup plus à l'aide de Swarm car pratiquement tous les services à votre disposition dans une instance de Wildfly sont à portée de main : couche d'abstraction d'accès à la persistance (JPA), cache applicatif distribué (**Infinispan**), *clustering*, etc. Et l'ajout de tous ces services n'est non seulement guère plus compliqué que les cas pratiques illustrés plus haut, mais en plus il est bien documenté sur le site de Wildfly Swarm, et illustré par des projets-exemples **[7]** !

Il ne vous reste donc plus qu'à explorer par vous-même et développer, en quelques minutes, votre propre pile logicielle, adaptée à vos besoins, mais bénéficiant toujours d'un travail d'intégration de la communauté Wildfly ! ■

Références

[1] JEE : https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition/

[2] Wildfly : http://wildfly.org/

[3] Wildfly Swarm : http://wildfly.org/swarm/

[4] Wildfly Core : https://github.com/wildfly/wildfly-core

[5] JAX-RS : https://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services

[6] ShrinkWrap : http://arquillian.org/guides/shrinkwrap_introduction/

[7] Swarm Examples : https://github.com/wildfly-swarm/wildfly-swarm-examples/

IMPLÉMENTEZ SRP EN JAVASCRIPT ET PHP !

Stéphane Mourey [Mousse sur le Seeraiwer]

Depuis le scandale PRISM, beaucoup ont ressenti le besoin d'accroître la confidentialité de leurs données. Mais comment faire s'il n'est pas possible d'accorder sa confiance aux serveurs hébergeant vos données et/ou votre application ? La réponse peut se révéler très complexe. Mais aujourd'hui, une chose est certaine : en ce qui concerne les applications Web, la mise en œuvre de la cryptographie en Javascript est un élément de la solution.

Mots-clés : Cryptographie, SRP, mot de passe, espionnage, AJAX, sécurité

Résumé

Première étape pour concevoir une application résistante à l'espionnage de vos serveurs, nous examinons ici une implémentation du protocole SRP, permettant au serveur de rester ignorant des mots de passe des utilisateurs. Cette implémentation est destinée à être utilisée par une application Web, les aspects clients sont écrits en JavaScript, les aspects serveur en PHP.

La cryptographie est soumise à de nombreuses critiques : en particulier, elle permettrait à des organisations illégales de communiquer sans que le contenu de leurs communications puisse être surveillé par les autorités ; ou encore, elle permettrait des échanges de contenus illégaux en toute impunité. Ces critiques ont toujours existé, mais les nécessités de l'économie numérique ont amené les gouvernements à se montrer plus compréhensifs : d'une arme de guerre dans les années 80, la cryptographie s'est aujourd'hui largement répandue dans notre quotidien en même temps qu'Internet. Impossible d'imaginer effectuer un achat en ligne sans recourir à https.

Car la confiance en l'économie numérique a un prix. Il faut en effet que les gouvernements acceptent de ne plus pouvoir observer facilement le contenu de ces communications et de se contenter d'observer à quels moments, à quelles fréquences et entre qui elles ont lieu, bref de ce que l'on appelle les métadonnées. Ce n'était pourtant pas rien, car, bien que présentées comme inoffensives, elles permettent de tirer bien des conclusions intéressantes... Mais au fait, pourquoi s'embarrasser d'interceptions de communication et d'analyse des métadonnées lorsqu'il suffit de prendre l'écouteur en bout de ligne ? C'est ce que la NSA a fait avec le programme PRISM. Depuis, malgré les objections, d'autres propositions de même nature ont fleuri partout dans le monde.

Dès lors qu'un double des clés de votre coffre circule à votre insu, que pouvez-vous faire ? Changer de coffre ? Si la solution paraît séduisante de prime abord, vous en viendrez tout de même à vous dire que ce qui s'est produit une première fois pourra se reproduire une seconde, quel que soit le paradis numérique de votre choix. Une autre solution consiste à ne plus placer vos données

directement dans le coffre, mais de les placer dans un plus petit coffre intermédiaire qui ira lui-même dans l'ancien coffre. Ainsi, celui qui obtiendra les clés du premier coffre d'une façon ou d'une autre devra toujours percer le second...

L'une de ces solutions est alors de concevoir des applications Web résistantes à la compromission du serveur qui les hébergent, une *host-proof web app*. Cela signifie que toutes les données doivent être chiffrées par le client avant d'être transmises vers le serveur, qui n'y aura jamais accès sous une forme claire. Le rôle du serveur est alors réduit à celui de fournisseur de l'application et hébergeur de données chiffrées.

Voilà ce que vous propose la cryptographie JavaScript : de ne plus envoyer vos données en clair sur le réseau ou même au travers d'un protocole crypté, mais de les chiffrer d'abord en local, sur votre machine, dans votre navigateur, avant de les communiquer au serveur. La première difficulté à laquelle le développeur d'une telle application est confronté est l'identification de l'utilisateur afin de lui communiquer les données qui lui appartiennent. Un protocole a été développé à cette fin, le protocole *SRP* **[1]** (*Secure Remote Password protocol*). Défini par l'Université de Stanford, il est devenu le principal standard pour résoudre ce problème.

Il est à noter que ce n'est pas parce que vous utilisez SRP que vos données vont être miraculeusement sécurisées et que votre application deviendra *host-proof*. Cela ne concerne que votre mot de passe et votre ouverture de session. Vous pouvez très bien utiliser ce protocole pour protéger les mots de passe d'une application classique sans rien changer à son fonctionnement par ailleurs. Ou vous pouvez la réécrire pour utiliser vos mots de passe comme clé de cryptage.

Un dernier avertissement avant de se lancer dans le vif du sujet : le point de vue adopté ici n'est pas celui d'un expert en sécurité, mais d'un développeur Web soucieux de sécuriser ses applications. L'auteur n'estime pas avoir les compétences nécessaires pour faire des recommandations « à l'épreuve des balles ». Les choix proposés ici sont raisonnables et permettent d'envisager l'architecture d'une application Web dans une démarche de sécurisation, mais ils n'y mettront certainement pas un point final. Par ailleurs, il ne s'agit pas de se passer des outils de sécurité existants mais d'en concevoir de nouveaux qui s'ajoutent aux premiers. En particulier, l'utilisation d'un protocole sécurisé, comme https, est impératif pour se protéger d'attaques du type *man-in-the-middle*.

1 Choix d'une bibliothèque

Il existe plusieurs bibliothèques JavaScript permettant de faire de la cryptographie. Malheureusement, rares sont celles qui implémentent le protocole SRP **[2]**. Mon choix s'est d'abord porté sur la **Stanford Javascript Crypto Library [3]** (*SJCL*) étant donné que cette bibliothèque est développée dans les mêmes murs qui ont vu naître SRP. Malheureusement, la documentation est succincte et, sans tutoriel détaillé, je ne suis pas parvenu à réaliser une implémentation utilisable de SRP...

Je me suis rabattu sur *thinbus-php Demo2* **[4]**. Il ne s'agit pas à proprement parler d'une bibliothèque, mais d'une démonstration permettant de tester l'ouverture d'une session SRP, tant en ce qui concerne les aspects clients que serveurs. La partie cliente est écrite en JavaScript comme attendu, et la partie serveur en PHP, langage le plus répandu pour le développement de sites Web côté serveur : l'idéal, en somme, pour le développeur Web que je suis. Il n'y a plus qu'à démonter le code afin de comprendre les différents rouages avant de l'implémenter sur mes propres instances.

2 Installation

L'installation est on ne peut plus simple. Étant donné qu'il s'agit d'une démonstration, les prérequis ont été réduits au minimum : **git**, un serveur Web et PHP. À partir de là, il vous suffit de vous placer avec votre terminal préféré dans un dossier lisible par votre serveur Web et de lancer la commande :

```
$ git clone https://bitbucket.org/taophp/thinbus-php-demo2.git
```

Je n'ai pas utilisé ici le dépôt principal du projet, mais un *fork* que j'ai réalisé pour cet article. En effet, je compte en commenter certains points en utilisant les numéros de ligne : le *fork* était le moyen le plus simple de m'assurer que, même si le développeur les modifiait entre le moment où j'écris ces lignes et celui où vous les liriez, vous pourrez les suivre sans aucun souci.

Les données utilisées par le serveur seront stockées dans une base de données **SQLite** dans `/tmp` par défaut. Il ne faudra donc pas être surpris si, après un redémarrage de votre machine, vos identifiants ne fonctionnent plus.

Voilà, c'est tout, rien de plus à faire, il n'y a plus qu'à passer à la manipulation.

3 Découverte du fonctionnement

3.1 Point de vue de l'utilisateur

Ouvrez votre navigateur sur le dossier où vous avez installé thinbus-php Demo2. L'application est immédiatement fonctionnelle. L'interface est épurée au possible : quelques mots d'explications en anglais pour commencer, un formulaire et quelques mots d'adieu.

L'essentiel se passe naturellement avec le formulaire. Il comprend deux champs et deux boutons. Les champs correspondent à une adresse email utilisée comme identifiant et au mot de passe. Le choix du bouton détermine le fonctionnement du formulaire : avec ***Register***, vous procédez à la création d'un nouveau compte ; avec ***Login***, vous effectuez une tentative d'identification. Vous aurez compris comment utiliser la démo : créer un compte, puis utiliser les mêmes paramètres pour vous loguer. Vous êtes alors redirigé sur une page encore plus succincte si vous avez réussi.

3.2 Point de vue du développeur

3.2.1 Architecture client

Regardons maintenant un peu comment les choses se passent sous le capot. En premier lieu, commençons à examiner le fichier **`index.php`**, point d'entrée de l'application. En réalité, celui-ci ne contient que peu de choses fonctionnelles, principalement le texte affiché sur la page et la définition de l'interface utilisateur. Mais il commence par un appel à un autre fichier qui, lui, va se révéler beaucoup plus instructif : **`demo.inc.php`**. C'est à partir de là que se met en place l'architecture côté client.

Ce fichier débute par quelques lignes qui, si elles ne relèvent pas précisément du point qui nous intéresse dans cet article, se révèlent tout de même intéressantes en ce qui concerne la sécurité d'une application Web. Car à quoi bon implémenter une solution aussi complexe que SRP si on le fait dans un contexte non sécurisé ? Le script envoie donc deux en-têtes http permettant de s'assurer que le contexte est sain :

- à la ligne 4 :

```
header("Content-Security-Policy: default-src 'self'; frame-src
'self'; style-src 'self' 'unsafe-inline';");
```

Cet en-tête, appelé *directive CSP,* interdit, pour les navigateurs qui la supportent, le chargement de contenu depuis des sources extérieures à notre site afin d'éviter des injections de code malicieux depuis une source non maîtrisée, mais également l'exécution de Javascript contenu dans la page proprement dite (événement **`onClick="alert('coucou');"`** par exemple, lien de type **`href="javascript:alert('coucou');"`** ou encore balise **`<script>alert('coucou');</script>`**).

- à la ligne 7 :

```
header("X-Frame-Options: DENY");
```

Cet en-tête interdit de placer la page dans une *frame*, ce qui permet de s'assurer qu'il n'y a pas une tentative pour duper l'utilisateur quant au site sur lequel il se trouve, ce qui pourrait être utilisé dans le cadre d'un scénario pour lui soutirer son mot de passe.

Suit l'inclusion d'une série de fichiers Javascript qui sont le réel cœur de notre application. En commentaire, à la ligne 24, signalons l'inclusion possible d'un fichier optimisé reprenant tout leur contenu à utiliser à leur place en production. Examinons maintenant la liste des fichiers Javascript liés (je n'indique pas les chemins complets de chacun pour plus de clarté) :

- **`biginteger.js`** : la cryptographie fait appel à de grands entiers, que Javascript ne sait pas manipuler de manière native ; ce fichier permet de corriger le problème ;
- **`isaac.js`** : un script générateur de nombres pseudo-aléatoires sécurisés ;
- **`random.js`** : un *wrapper* qui permet d'avoir recours au fichier précédent si la fonction Javascript **`window.crypto.getRandomValues`** n'est pas implémentée par le navigateur ; cette dernière permet effectivement un générateur de nombres pseudo-aléatoire natif présent sur les navigateurs modernes ;
- **`sha256.js`** : fichier extrait de la bibliothèque **CryptoJS [5]** qui permet de calculer la signature SHA256 d'une chaîne ;
- **`thinbus-srp6client.js`** : le client SRP proprement dit qui va s'appuyer sur les outils précédents ;
- **`thinbus-srp-config.js`** : contient la configuration à utiliser par le client ; en commentaire, il est indiqué qu'un grand nombre premier y est défini et qu'il est recommandé de le remplacer par un autre pour plus de sécurité ; **il faudra indiquer le même nombre**

premier dans le fichier PHP équivalent (**/server/lib/thinbus-srp-config.php**) pour que la communication avec le serveur se déroule correctement **[6]** ;

- **thinbus-srp6a-config-sha256.js** : classe permettant de configurer le hachage SHA256 à l'aide de grands entiers ;
- **client.js** : contient des fonctions simplifiant la communication AJAX avec le serveur ;
- **demo.js** : contient d'autres fonctions utilitaires.

L'essentiel du fonctionnement de notre client est défini dans ces fichiers, et il nous faudra nous pencher plus particulièrement sur **thinbus-srp6client.js** qui est le cœur du processus. C'est tout pour le fichier **demo.inc.php**.

Retournons dans le fichier **index.php**. L'interface utilisateur y est d'abord définie à la suite de l'inclusion de **demo.inc.php**. Une fois terminé l'en-tête HTML, on y trouve principalement des explications sur cette démonstration puis sur le fonctionnement de SRP en général et celui de cette implémentation en particulier. Cela se révèle d'une grande aide pour comprendre le déroulement de l'identification.

La définition des éléments fonctionnels de l'interface utilisateur se trouve au milieu de tout cela, prise dans une mise en forme à base de tableaux : aux lignes 43 et 49 se trouvent les champs de saisie ; et à la ligne 55 les deux boutons. Il n'y a pas de balise **<form>** associée à ces éléments de formulaires, ils sont seulement gérés en Javascript.

Le fichier **index.php** ne sert finalement qu'à mettre en place l'architecture du client SRP et à donner à lire de la documentation. Où se cache donc le service avec lequel notre client va dialoguer ?

La réponse nous est révélée dans le fichier **demo.js**. Il contient en effet la définition de l'URL avec laquelle le client va communiquer en AJAX :

```
06: var ajaxurl = window.location.protocol + '//' + window.
location.host + removefilefrompath(window.location.pathname) +
'/server/server.php';
```

Cela semble bien élaboré, mais cela permet simplement de s'abstraire du contexte d'exécution. Le seul présupposé est que le script destiné à traiter les requêtes AJAX se trouve dans un sous-dossier **/server/server.php** par rapport au document courant. En l'état actuel du projet, c'est le cas. Mais si vous décidiez d'en reprendre des éléments dans une nouvelle architecture et que cette contrainte n'était pas préservée, c'est dans **demo.js** que vous devrez effectuer les corrections requises. La suite du fichier est également intéressante. Les lignes 12 à 19 définissent l'injection d'un fichier Javascript à exécuter à la fin du chargement du DOM. Le nom du fichier est calculé à partir du nom de la page courante, changeant simplement l'extension en **.js**. Le principe de cette fonctionnalité peut se révéler pratique pour charger automatiquement un fichier JavaScript particulier pour chaque page qui le nécessite, à condition que leur nombre justifie la mise en place d'un système si complexe – et une requête supplémentaire au serveur pour chaque page. J'avoue avoir été surpris de trouver une solution si élaborée dans le cadre de cette démonstration. J'ai interrogé le développeur à ce sujet : effectivement, le fichier aurait pu être simplement inclus. Mais plusieurs raisons lui ont fait préférer cette technique. La plus importante est qu'il s'agit d'un code particulier à cette page, mais qui doit être externalisé vers un fichier car, du fait de la directive CSP, le Javascript en ligne est proscrit. Ce problème étant général sur un site utilisant cette directive – ce qu'il faut bien recommander – une solution globale était souhaitable. L'implémenter en Javascript permet de mieux séparer Javascript et PHP.

Le reste du fichier **demo.js** ne fait que définir d'autres fonctions utilitaires. C'est donc sur **index.js** qu'il nous faut nous pencher maintenant. Et c'est effectivement dans celui-ci que tout va se lier. Ce fichier comprend quatre fonctions : une anonyme, exécutée à la fin du chargement de la page et qui permet de lier les éléments de l'interface graphique avec les fonctions du code ; **getinput()** qui récupère les éléments saisis par l'utilisateur dans les différents champs texte ; **srpregister()** qui va demander l'enregistrement d'un nouvel utilisateur dans la base de données et **srplogin()** qui va demander d'identifier l'utilisateur.

Il y a peu de choses à dire sur la fonction anonyme : le lien entre les boutons de commande et les fonctions correspondantes est défini de la ligne 15 à la ligne 23, le reste n'étant que de l'embellissement pour la documentation présente sur la page.

De même, il y a lieu d'être bref sur la fonction **getInput()** : celle-ci est utilitaire et récupère le contenu des champs dont les identifiants sont **email** et **pass**, les valide avant de les rassembler au sein d'un objet qu'elle va utiliser comme valeur de retour. Si l'un des champs n'est pas valide, alors il reçoit le focus et la fonction renvoi **false**. Cette fonction est purement utilitaire pour **srpregister()** et **srplogin()**, mais elle est dépendante du contexte de notre démonstration : si vous voulez implémenter SRP sans adopter exactement le même formulaire, vous aurez des modifications à y faire.

3.2.2 Processus d'enregistrement de l'utilisateur

srpregister() et **srplogin()** sont les deux fonctions Javascript qui sont au cœur du processus du côté du client. Pour le coup, examinons le code de chacune d'elles en détail, en commençant par la procédure d'enregistrement de l'utilisateur :

```
83: function srpregister(event)
84: {
85:   var input = getinput(); // Get email and password from
input fields
87:   if(input == false) // test if input is valid
88:     return;
90:   document.body.style.cursor  = 'wait';
92:   var srpclient = new SRP6JavascriptClientSessionSHA256();
93:   var srpsalt = srpclient.generateRandomSalt();
94:   var srpverifier = srpclient.generateVerifier(srpsalt, input.
email, input.pass);
96:   // Send the data to the server with an ajax call
97:   jsonrpc(ajaxurl, 'register', {email: input.email, srpsalt:
srpsalt, srpverifier: srpverifier}, 0, function (data, error, id)
98:   {
99:     document.body.style.cursor  = 'default';
101:    if(error)
102:    {
103:      alert(data);
104:      return;
105:    }
107:    alert('You have with success registered user: ' + input.
email);
108:  });
109: }
```

Les choses sérieuses commencent à la ligne 92 avec **SRP6JavascriptClientSessionSHA256()**. Cette fonction définie dans **thinbus-srp6a-config-sha256.js** est en réalité une classe étendant **SRP6JavascriptClientSession()** définie dans **thinbus-srp6client.js**. **SRP6JavascriptClientSession()** renvoie un objet contenant toutes les méthodes nécessaires pour établir une authentification SRP. Ici, nous avons d'abord recours à la méthode **generateRandomSalt()** qui génère ce que l'on appelle un *grain de sel* qui permet de sécuriser notre travail cryptographique en y ajoutant un élément aléatoire. Puis ligne 94, ce grain de sel avec les données soumises par l'utilisateur est utilisé pour générer un vérificateur, qui a la forme d'une longue chaîne hexadécimale. Toutes ces données sont ensuite encodées en JSON et envoyées au serveur à la ligne 97. La destination est le script **server/server.php**., les données sont envoyées en utilisant la méthode http POST et l'action demandée en paramètre au script est **register**.

Passons maintenant du côté serveur. En examinant le script **server.php**, on s'aperçoit qu'il passe alors la main, ligne 46, à la fonction **srpregister**. Celle-ci est définie à la ligne 79 du même fichier et se contente d'enregistrer les informations fournies par le client, y compris le grain de sel et le vérificateur, en base de données, si l'utilisateur indiqué n'existe pas encore. Il est maintenant possible de tester l'aspect suivant, celui de l'identification.

3.2.3 Processus d'identification de l'utilisateur

Penchons-nous donc maintenant sur la fonction **srplogin()** de notre fichier **index.js** :

```
114: function srplogin(event)
115: {
116:   var input = getinput(); // Get email and password from input
fields
118:   if(input == false) // test if input is valid
119:     return;
121:   document.body.style.cursor  = 'wait';
123:   var srpclient = new SRP6JavascriptClientSessionSHA256();
125:   srpclient.step1(input.email, input.pass);
127:   // Send the data to the server with an ajax call (first step)
128:   jsonrpc(ajaxurl, 'login1', {email: input.email}, 0, function
(data, error, id)
129:   {
130:     if(error)
131:     {
132:       document.body.style.cursor  = 'default';
133:       alert(data);
134:       return;
135:     }
137:     var credentials = srpclient.step2(data.salt, data.b);
139:     jsonrpc(ajaxurl, 'login2', {email: input.email, A:
credentials.A, M1: credentials.M1}, 0, function (data, error, id)
140:     {
141:       document.body.style.cursor  = 'default';
143:       if(error)
144:       {
145:         alert(data);
146:         return;
147:       }
149:       if(!srpclient.step3(data.M2))
150:       {
151:       alert('Wrong password specified');
152:       return;
153:       }
155:       // Store the session in session storage (session is
now the same for both server and client and it has never been
transmitted)
156:       sessionStorage['sessionkey'] = srpclient.
getSessionKey();
158:       // Redirect the browser to the success page (this also
clear all values)
159:       window.location.assign(window.location.protocol
+ '//' + window.location.host + removefilefrompath(window.
location.pathname) + '/success.php');
161:     });
162:   });
163: }
```

Jusqu'à la ligne 123 incluse, l'initialisation est exactement la même que pour `srpregister()`, jusqu'à l'obtention d'un objet client SRP. Puis à la ligne 125 commence la première étape de l'identification, grâce à la méthode `step1()`. Celle-ci est définie dans le fichier `thinbus-srp6client.js`, à la ligne 259. Elle se contente, après les vérifications d'usage, d'associer les valeurs soumises par l'utilisateur à notre nouvel objet.

Par le même mécanisme que nous avons vu auparavant, ces données sont transmises au serveur qui va utiliser la fonction `sprlogin1` définie dans `server.php`. Celle-ci, après avoir vérifié si l'utilisateur existait, crée un objet `ThinbusSrp` avec les paramètres nécessaires. Puis il va appeler la méthode `step1()` (définie à la ligne 239 de `thinbus-srp.php`) de cet objet. Celle-ci renverra un challenge, un défi, créé à partir du grain de sel et du vérificateur stockés en base sous la forme d'un long nombre hexadécimal. Puis la session de l'utilisateur est alimentée avec ces données pour les conserver pendant la suite du processus, avant de les renvoyer au client.

Le client passe alors à l'étape 2 à la ligne 138 de notre fichier `index.js`, grâce à la méthode `step2()` de l'objet `SRP6JavascriptClientSessionSHA256` définie à la ligne 314 de `thinbus-srp6client.js`. Cette méthode tente de résoudre le défi, et le résultat est envoyé au serveur. Cet appel est traité avec la fonction `srplogin2()` (fichier `server.php`, ligne 129) qui vérifie si le client a résolu le défi. Il est important de signaler ici que le serveur a les moyens de vérifier si le client a réussi, mais pas ceux de le réussir à sa place. Si le défi est résolu, la session, déjà ouverte, est associée avec l'utilisateur maintenant identifié et l'information renvoyée au client sous la forme d'un nombre hexadécimal que le client vérifiera à son tour en passant à l'étape 3 à la ligne 150 d'`index.js`. Enfin, si tout le monde est d'accord, le jeu s'arrête presque là.

À la ligne 156, le client stocke l'identifiant de session dans le navigateur, dans une base de données qui n'est conservée que tant que l'utilisateur se sert de son navigateur. Cela fait, ligne 159, une redirection est effectuée vers la page de succès. Cette redirection est importante, car elle permet en même temps de s'assurer que toutes les valeurs contenues sur la page, y compris les saisies de l'utilisateur comme le mot de passe, ne sont plus accessibles par le reste de l'application.

Dernier point, sur lequel je ne m'étendrai pas : dans le fichier `success.js`, injecté lors du chargement de la page de succès, se trouve le nécessaire pour clore la session identifiée. Le pendant du côté du serveur est exécuté par la fonction `srplogout` à la ligne 165 de `server.php`. Là aussi des adaptations seront nécessaires selon votre contexte.

4 Difficulté non levée

Différents problèmes pèsent sur les applications Javascript résistantes à une compromission de serveur, suffisamment importantes pour hypothéquer leur avenir. Sans les passer tous en revue maintenant, il en est un majeur qui se manifeste déjà dans le processus que nous venons de décrire et qui fait que bon nombre de personnes préoccupées de sécurité rejettent Javascript dans le navigateur comme un outil pertinent. Voici le problème : si le code utilisé pour chiffrer les données est fourni par le serveur, qu'est-ce qui empêcherait un serveur compromis de le modifier pour pouvoir accéder aux données ? On s'imagine mal relire tout le code source de l'application à chaque fois qu'on y accède pour vérifier qu'aucune backdoor n'y a été placée depuis sa dernière utilisation...

Des solutions ont été proposées pour tenter d'y remédier, malheureusement aucune n'est pleinement satisfaisante. Par exemple, le développement d'un module navigateur permettant de s'assurer de l'intégrité du code : dans ce cas, puisqu'il est nécessaire de demander l'installation de ce module à l'utilisateur, pourquoi ne pas l'inviter à installer une véritable application ?

L'implémentation du protocole SRP en Javascript présente tout de même un avantage majeur : en effet, seules les données enregistrées après la compromission du serveur peuvent être éventuellement exploitées, et elles ne peuvent pas l'être sans modification vérifiable du code. Ce n'est pas rien.

Conclusion

À l'issue de cet article, vous serez sinon convaincu, du moins sensibilisé à l'intérêt d'utiliser la cryptographie en Javascript dans le navigateur. La mise en œuvre de SRP dans le contexte Web n'est pas si complexe qu'il y paraît au premier abord, dès lors que l'on dispose d'une bonne bibliothèque correctement documentée. Dans la mesure où SRP n'est pas utilisé pour chiffrer autre chose que le mot de passe, il n'est pas contradictoire avec la possibilité de réinitialiser le mot de passe. Il n'y a donc aucun changement visible pour l'utilisateur final. Alors, pourquoi s'en priver ?

Pour aller plus loin

- *thinbus-php Demo2* par Benny Nissen est un *fork* de *thinbus-php* (https://bitbucket.org/simon_massey/thinbus-php/src), lui-même dérivé de *thinbus-srp-js* (https://bitbucket.org/simon_massey/thinbus-srp-js), tous les deux par Simon Massey : il peut être utile de consulter ces deux projets qui contiennent une documentation éclairante sur SRP.
- Très critique, mais toujours d'actualité pour l'essentiel, je vous recommande la lecture de l'article du blog du NCC GROUP intitulé *Javascript Cryptography Considered Harmful* (https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2011/august/javascript-cryptography-considered-harmful/). Cela vous convaincra peut-être de ne pas utiliser JavaScript pour faire de la cryptographie. Mais au moins aurez-vous une connaissance plus approfondie des lourds problèmes que la cryptographie pose à Javascript. ■

Références

[1] Site officiel du protocole SRP : http://srp.stanford.edu

[2] Site officiel de la Stanford Javascript Crypto Library : http://bitwiseshiftleft.github.io/sjcl

[3] Je n'en ai trouvé qu'une autre, la *Javascript Crypto Library* (https://clipperz.is/open_source/javascript_crypto_library/) sous licence BSD, développée pour le service en ligne **Clipperz** (voir l'article *« Clipperz »* dans *Linux Pratique n°93* p. 46) mais qui ne m'a pas parue aussi convaincante.

[4] Sur Bitbucket: https://bitbucket.org/beastybeast/thinbus-php-demo2

[5] Projet de librairie cryptographique en JavaScript, dont le développement n'est plus poursuivi aujourd'hui. Site officiel : https://code.google.com/p/crypto-js/

[6] Le projet initial **Thinbus Javascript Secure Remote Password** (https://bitbucket.org/simon_massey/thinbus-srp-js) a été réalisé en utilisant Java pour la partie serveur. Ce projet inclut un outil permettant de mettre en forme un grand nombre premier produit par OpenSSL de façon à pouvoir le copier/coller dans les fichiers **`thinbus-srp-config.[js|php]`**. Vous pouvez consulter le paragraphe *Creating A Custom Large Safe Prime* (https://bitbucket.org/simon_massey/thinbus-srp-js#markdown-header-creating-a-custom-large-safe-prime) pour connaître la procédure à utiliser. Un service en ligne est également proposé. Il vous permettra de gagner du temps en vous passant de Java et de l'installation des outils, mais pas de OpenSSL.

UN SERVICE WEB EN 10 MINUTES AVEC BOTTLE


Tristan Colombo

Il arrive que l'on ait à développer des « microservices » pour fournir deux ou trois informations en ligne. Dans ces moments-là, autant utiliser un framework hyper léger qui ne nous fera pas perdre une journée en développement !

Mots-clés : Botte, Python, Framework, Web, Service

Résumé

Inutile de sortir un bazooka pour tuer un moustique : pour développer un petit service Web autant utiliser les outils appropriés. Nous allons employer Python et le module Flask en ce sens en finalisant notre développement par une mise en production sur un serveur Apache.

Tout part d'une question toute bête d'un ami qui voulait pouvoir servir des fichiers depuis une url en fournissant une sorte de code en paramètre get. Par exemple, http://www.monsite.com/giveme?file=12234 renverrait le fichier **texte.odt** et http://www.monsite.com/giveme?file=5456 renverrait **audio.ogg**. Ici il est clairement inutile de partir sur une solution « lourde » du type **Django**.

Je vous propose de résoudre ce problème en utilisant le *framework* **Bottle [1]** et d'en profiter pour découvrir son fonctionnement.

1 Une bouteille à la mer

Python dispose nativement d'un serveur http avec **http.server [2]** pour Python 3 et **SimpleHTTPServer [3]** pour Python 2.7. Ces solutions sont très pratiques et simples à mettre en place mais la gestion des *templates* doit se faire à la main et il est donc préférable de les conserver pour des microprojets (même si notre service serait tout à fait réalisable en utilisant cette technique). Il vaut donc mieux s'orienter vers un petit *framework* disposant de quelques outils. Comme mon projet va être une sorte de service hybride incluant l'affichage d'une page html en cas d'erreur, mon choix s'est porté sur Bottle.

L'installation se fait classiquement depuis **pip** :

```
$ sudo pip3 install bottle
```

Comme l'indique la documentation, la création du serveur se fait en quelques lignes :

```
01: from bottle import route, run
02:
03: @route('/hello')
04: def hello():
05:     return "Hello GLMF!"
06:
07: run(host='localhost', port=8080, debug=True)
```

Dans les lignes 3 à 6 nous lions l'url hello à la fonction **hello()** et en ligne 7 nous lançons le serveur en local sur le port **8080** en activant le mode *debug*. Exécutez le code

et rendez-vous sur la page http://localhost:8080/hello pour voir le message s'afficher dans votre navigateur.

```
$ python3 start.py
Bottle v0.12.8 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

127.0.0.1 - - [21/Oct/2015 14:51:04] "GET /hello HTTP/1.1" 200 11
```

Bien sûr ici nous serons loin d'un document html respectant les recommandations du W3C avec notre « Hello GLMF ! ». C'est là que les vues rentrent en scène.

2 Terre en vue

Les vues sont gérées en utilisant des *templates* de code html dans lesquels il est possible d'insérer des variables à l'aide d'identifiants encadrés par **{{** et **}}**. Reprenons l'exemple précédent en créant un fichier **hello.tpl** :

```
01: <!doctype html>
02: <html lang="fr">
03:   <head>
04:     <meta charset="utf-8" />
05:     <title>{{title}}</title>
06:   </head>
07:
08:   <body>
09:     <div>Hello {{name}} !</div>
10:   </body>
11: </html>
```

Il faut alors modifier le fichier **start.py** pour pouvoir utiliser ce fichier de *template* :

```
01: from bottle import route, run, view
02:
03: @route('/hello')
04: @view('hello')
05: def hello():
06:     values = {
07:       'title' : 'Test de Bottle',
08:       'name'  : 'GLMF'
09:     }
10:     return values
11:
12: run(host='localhost', port=8080, debug=True)
```

La valeur des variables utilisées dans le fichier de *template* doit être indiquée dans un dictionnaire. Déconcertant de simplicité non ?

Revenons à notre problème initial où nous souhaitions récupérer un paramètre depuis l'url.

3 Quels sont les paramètres de navigation capitaine ?

L'objet **request.query** va permettre d'utiliser très simplement les paramètres. Modifions le fichier **start.py** de manière à ce que le nom de **Hello {{name}}** soit déterminé depuis une valeur de l'url :

```
01: from bottle import route, run, view, request
02:
03: @route('/hello')
04: @view('hello')
05: def hello():
06:     if request.query.name == '':
07:         request.query.name = 'GLMF'
08:
09:     values = {
10:       'title' : 'Test de Bottle',
11:       'name'  : request.query.name
12:     }
13:     return values
14:
15: run(host='localhost', port=8080, debug=True)
```

Désormais en appelant http://localhost:8080/hello on obtient « Hello GLMF ! » et avec une url du style http://localhost:8080/hello?name=toto on obtient « Hello toto ! ». Là encore rien de bien compliqué...

Si vous souhaitez faire une distinction entre les méthodes get et post vous pourrez spécifier le paramètre **method** dans la route de votre fonction. Par exemple, notre ligne 3 peut devenir :

```
03: @route('/hello', method='get')
```

Ceci était sous-entendu dans le code précédent puisque pour récupérer des paramètres en post il faut utiliser **request.forms**. L'objet **request.params** permet, lui, d'accéder à la fois à **request.query** (méthode get) et à **request.forms**.

4 Hissez la grand-voile !

Imaginons que nous ayons besoin d'un fichier de style, d'images ou d'un fichier Javascript. Il nous faudra donc servir des fichiers statiques et il suffit pour cela de spécifier dans le fichier **start.py** dans quel répertoire ils se trouveront. Nous en profiterons pour déplacer le fichier de *template* **hello.pl** dans un répertoire spécifique :

```
01: from bottle import route, run, view, request, static_file
02:
03:
04: @route('/static/<filename:path>')
05: def server_static(filename):
06:     return static_file(filename, root='./static')
07:
08:
09: @route('/hello')
10: @view('static/template/hello')
11: def hello():
12:     if request.params.name == '':
13:         request.params.name = 'GLMF'
14:
15:     values = {
16:       'title' : 'Test de Bottle',
17:       'name'  : request.params.name
18:     }
19:     return values
20:
21: run(host='localhost', port=8080, debug=True)
```

Ce sont les lignes 4 à 6 qui permettent de déterminer comment seront servis les fichiers statiques. Ils seront cherchés dans le répertoire **static** et correspondront à l'url http://localhost:8080/static (nous aurions pu choisir deux noms différents dans les lignes 4 et 6). En ligne 10 nous avons modifié le chemin du fichier de *template* et il faut donc le déplacer :

```
$ mkdir -p static/template
$ mv hello.tpl static/template
```

Pour nos tests, nous ajouterons un fichier de css dans **static/style/style.css** :

```
01: div {
02:     color : blue;
03: }
```

Enfin, il faut charger cette feuille de style dans le fichier de *template* **hello.tpl** :

```
01: <!doctype html>
02: <html lang="fr">
03:   <head>
...
06:     <link rel="stylesheet" href="static/style/style.css" />
07:   </head>
...
```

5 Pirates à bâbord !

Les données saisies en tant que paramètres get ou post sont protégées et il n'est donc pas possible de réaliser une injection de code. Vous pouvez essayer de vous rendre sur http://localhost:8080/hello?name= <script>alert('oops');</script>, vous ne modifierez pas le comportement de la page et l'injection SQL ne fonctionnera pas non plus.

6 Souquez ferme moussaillons !

Revenons une nouvelle fois à notre problématique de service de fichiers. Nous avons vu comment servir les fichiers statiques habituels que sont les **.css**, **.js**, etc et il n'y aura donc pas de difficulté à servir un autre type de fichier. Si vous le souhaitez vous pourrez en profiter pour personnaliser la page d'erreur 404 grâce au décorateur **@error** :

```
01: from bottle import route, run, view, request, static_file, error
02:
03: @route('/static/<filename:path>')
04: def server_static(filename):
05:     return static_file(filename, root='./static')
06:
07: @route('/giveme', method='get')
08: def sendFile():
09:     return static_file(request.query.key + '.odt', root='./files')
10:
11: @error(404)
12: @view('static/template/404')
13: def error404(error):
14:     return None
15:
16: run(host='localhost', port=8080, debug=True)
```

Dans les lignes 3 à 5 j'ai réactivé le service des pages statiques en vue de pouvoir lire un *template* et une image pour ma page d'erreur personnalisée des lignes 11 à 14 (notez le **return None** de la ligne 14 puisque je n'ai pas paramétré ma page). Ce sont les lignes 7 à 9 qui vont me permettre de servir des fichiers en fonction du code reçu. Ici il s'agit seulement de fichiers **.odt** dont le nom est le même que la clé fournie mais les modifications sont minimes pour obtenir quelque chose de plus généraliste.

7 Et si nous changions de galion ?

La fonction **run()** de Bottle permet de lancer rapidement un serveur de développement sur un seul thread basé sur **wsgiref [4]**. Il est possible d'utiliser des adaptateurs pour des versions *multi-threads* comme **CherryPy** ou **Paste**. Il existe même un adaptateur pour **Gunicorn**. Je vous propose de partir sur une autre implémentation en utilisant un serveur externe qui sera en l'occurrence **Apache** avec le module **wsgi**.

Je suppose que vous avez déjà un serveur Apache installé et configuré. Il faut tout d'abord s'assurer que le module

wsgi soit bien présent et activé. Voici les commandes pour un système basé sur Debian :

```
# aptitude install libapache2-mod-wsgi
# a2enmod wsgi
# service apache2 restart
```

Créez ensuite un répertoire pour héberger votre site/application. En général on place tout dans **/var/www**. Je vais partir du travail que nous avons effectué précédemment et simplement déplacer le répertoire **FileServer** contenant tous les fichiers :

```
# mv FileServer /var/www
# chown -R www-data:www-data /var/www/FileServer
```

Notre fichier **start.py** va changer de structure. Tout d'abord nous allons le renommer en **FilerServer.wsgi** :

```
# cd /var/www/Fileserver
# mv start.py FileServer.wsgi
```

Ensuite il faut changer de répertoire dans le script de manière à pouvoir lire les fichiers externes et il ne faut plus lancer le serveur depuis le script (la variable de retour doit forcément s'appeler **application**) :

```
01: from bottle import route, run, view, request, static_file, error
02: import os
03: import bottle
04:
05: os.chdir(os.path.dirname(__file__))
...
20: application = bottle.default_app()
```

Il reste ensuite à créer le fichier de configuration Apache. Comme ici nous allons fournir un service, je vais considérer que nous sommes sur un sous-domaine et je vais donc travailler avec un fichier **/etc/apache2/sites-available/fileserver.domaine.fr** :

```
01: NameVirtualHost fileserver.domaine.fr:80
02:
03: <VirtualHost fileserver.domaine.fr:80>
04:     ServerAdmin tristan@gnulinuxmag.com
05:     DocumentRoot /var/www/FileServer
06:     ServerName fileserver.domaine.fr
07:     ServerAlias www.fileserver.domaine.fr
08:
09:     WSGIDaemonProcess FileServer user=www-data group=www-data
processes=1 threads=5
10:     WSGIScriptAlias / /var/www/FileServer/FileServer.wsgi
11:
12:     <Directory /var/www/FileServer>
13:         WSGIProcessGroup FileServer
14:         WSGIApplicationGroup %{GLOBAL}
15:         Order Deny,Allow
16:         Allow from all
17:     </Directory>
18:
19:     # Fichiers de log
20:     CustomLog /var/log/domaine/FileServer/access_log common
21:     ErrorLog /var/log/domaine/FileServer/error_log
22:     LogLevel debug
23: </VirtualHost>
```

Activez ensuite la configuration :

```
# a2ensite fileserver.domaine.fr
# service apache2 restart
```

Vous n'avez plus qu'à vous connecter sur l'url configurée pour utiliser votre service !

Attention !

Il est de petites erreurs qui peuvent se révéler très coûteuses en temps... Par exemple, si vous effectuez vos tests sur une machine avant de tout transférer en production, pensez à vérifier que vous avez bien installé Bottle sur le serveur...

Conclusion

Nous avons pu développer et mettre en production très rapidement notre petit service qui aurait pu être un service REST en fonction des besoins. Il est inutile de chercher des solutions trop complexes quand de petits *frameworks* peuvent fournir tout ce dont on a besoin ! ■

Références

[1] Site officiel du *framework* Bottle : http://bottlepy.org/docs/dev/index.html

[2] Documentation de http.server : https://docs.python.org/3/library/http.server.html

[3] Documentation de SimpleHTTPServer : https://docs.python.org/2/library/simplehttpserver.html

[4] Documentation de wsgiref : https://docs.python.org/3.5/library/wsgiref.html

****
******** ********
******** Saurez-vous déchiffrer le message ci-dessus ? ********
******** ********
****

Avec la clé 0x4245424F50 en utilisant la méthode inventée par un certain Blaise (1523)
et en utilisant pour alphabet (dans l'ordre) les 26 lettres Latines majuscules,
le point, l'espace, le symbole "deux-points", et le symbole slash.

Rendez-vous sur le lien caché dans le message pour envoyer votre solution
et son explication.

Parrot®

****
**** Rejoignez les 900 passionné(e)s qui inventent les drones de demain ****

..... parrot.recrute.com .....