N°196 septembre 2016

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

LES «HOW-TO» DU SYSADMIN

Utilisez des filtres et l'API REST pour gérer vos dépôts Debian p.38

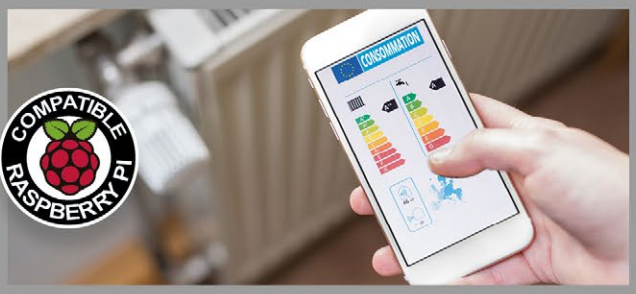
PYTHON / MINMAX

CRÉEZ VOTRE PREMIÈRE INTELLIGENCE ARTIFICIELLE !

p.18

DOMOTIQUE / CONSO ÉLECTRIQUE

Modélisez un système de télé-information p.64



SAUVEGARDE / FICHIERS

Découvrez un projet de sauvegarde en continu p.44

HOWTO / JAVASCRIPT

Développez un serveur REST avec Node.js et Redis

p.50

SÉCURITÉ / JAVA

Discussion autour du Threat Modelling

p.58

UNHOSTED / REMOTESTORAGE

Mettez en place un serveur de stockage distant p.79

ET AUSSI : Réplication et sécurité de PostgreSQL – Création de fichiers MIDI en C

L 19275 - 196 - F: 7,90 € - RD

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

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

NOUVEAU ! 1&1 MANAGED CLOUD HOSTING

Le meilleur de deux mondes

Un pack d'hébergement performant associé à des ressources serveur flexibles et modulables à tout moment : le **nouveau Managed Cloud Hosting 1&1 est arrivé !** Idéal pour les projets Web les plus exigeants en termes de disponibilité, de sécurité et de flexibilité.

- ✓ **Ressources dédiées**
- ✓ **+ de 20 combinaisons de stack**
- ✓ **Géré par les experts 1&1**
- ✓ **Flexible & évolutif**
- ✓ **Prêt en moins d'1 minute**

Cloud Vendor Benchmark 2016
Rising Star Germany
experton GROUP an ISG business

intel XEON inside

Trusted Performance.
Intel® Xeon® processors.

1 MOIS GRATUIT*

1 MOIS POUR ESSAYER

1 CLIC POUR CHANGER DE PACK

1 APPEL UN EXPERT VOUS RÉPOND

0970 808 911
(appel non surtaxé)

1&1

1and1.fr

*1&1 Managed Cloud Hosting : 1 mois d'essai gratuit, puis à partir de 9,99 € HT/mois (11,99 € TTC). Pas de durée minimale d'engagement. Pas de frais de mise en service. Conditions détaillées sur 1and1.fr. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

GNU/Linux Magazine France
est édité par Les Éditions Diamond

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

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

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

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

Périodicité : Mensuel
Prix de vente : 7,90 €

SUIVEZ-NOUS SUR :

https://www.facebook.com/editionsdiamond

@gnulinuxmag

LES ABONNEMENTS ET LES ANCIENS NUMÉROS SONT DISPONIBLES !

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

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

ÉDITORIAL

Pendant que nous profitions de vacances bien méritées, l'IEEE (*Institute of Electrical and Electronics Engineers*) publiait son étude annuelle des langages de programmation les plus populaires. Les données présentées sont issues de différentes sources, dont les recherches Google, les projets GitHub (actifs et créés), Stack Overflow (questions posées et nombre de vues), etc. auxquelles sont attribuées plus ou moins d'importance. Les résultats sont affichés sous la forme d'un tableau interactif où il est possible de filtrer les données par type d'utilisation (développement web, mobile, en entreprise ou embarqué) et par type de classement, chaque classement ayant sa propre configuration en terme de poids appliqué aux différentes sources (classement IEEE, langages ayant la progression la plus rapide, langages recherchés par les employeurs aux États-Unis, langages populaires dans le monde open source et enfin un type personnalisé en cliquant sur **Custom** puis **Edit ranking**) [1]. Dans ces conditions, il est bien sûr très facile de modifier complètement le classement en jouant sur deux ou trois indicateurs. Toujours est-il que ce classement nous donne les grandes tendances concernant les langages qui sont les plus employés et ceux qui le sont de moins en moins. Que peut-on donc retenir du classement de l'IEEE ?

- Les quatre langages les plus employés restent toujours les mêmes : C, Java, Python et C++.
- Le cinquième langage le plus employé est le... R ? Devant C#, PHP et JavaScript ? On constate ici un effet pernicieux de la mesure : plus un langage sera complexe et/ou mal documenté, plus le nombre de recherches sera conséquent... et entraînera son apparition en tête de classement.
- En développement mobile, Python disparaît du classement (manque de confiance des développeurs envers le *framework* Kivy ?) et étrangement Objective C n'apparaît qu'à la huitième place, après Scala. Les deux premières places sont occupées par C et Java (Android).
- En développement web on retrouve Java, Python, C# et PHP et on découvre que l'IEEE considère HTML comme un langage...
- Le langage Go est plus employé qu'en 2015, au contraire de l'usage de Perl qui continue de chuter.
- Enfin, sans surprise, Actionscript et OCaml font partie des langages les moins employés.

Encore une fois, ces classements n'indiquent en aucune façon qu'un langage est « bon » ou « mauvais » ou même « meilleur » qu'un autre, mais ils permettent de suivre la tendance des logiciels produits et donc de savoir quels langages seront le plus à même d'être demandés sur le marché du travail...

Pour finir, sans aucun rapport avec ce qui précède, devant la recrudescence d'auteurs se sentant obligés d'appliquer le « Guide pratique pour une communication publique sans stéréotype de sexe » du Haut Conseil à l'Égalité entre les femmes et les hommes [2], il me paraît important d'effectuer une précision sur les textes que vous pouvez lire dans nos magazines. Sans aucunement souhaiter renier notre lectorat féminin, nous maintiendrons dans nos textes un masculin se voulant généraliste et permettant de garantir la lisibilité d'articles qui peuvent être complexes. En effet, appliquer la marque du féminin et du singulier en les séparant par des points comme préconisé en page 15 dudit guide conduit à des étrangetés du type « les utilisateur.rice.s » qui répétées au fil d'un texte en ralentissent nécessairement la lecture. L'Histoire de l'informatique a été écrite par des hommes, mais aussi des femmes telles que Ada Lovelace et Grace Hopper et le fait de ne pas employer une écriture pointilliste ne nous le fait pas oublier.

Aussi, n'hésitez pas que vous soyez lecteurs ou lectrices à nous proposer des articles sur les technologies que vous employez... qu'elles soient en lien ou non avec le top des langages de l'IEEE :-)

Je vous souhaite à toutes et à tous (et non à tou.te.s) une bonne lecture !

Tristan Colombo

[1] *Interactive: The Top Programming Languages 2016* : http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016

[2] Haut Conseil à l'Égalité entre les femmes et les hommes, *« Guide pratique pour une communication publique sans stéréotype de sexe »* : http://www.haut-conseil-egalite.gouv.fr/IMG/pdf/hcefh__guide_pratique_com_sans_stereo-_vf-_2015_11_05.pdf

1ER ÉVÉNEMENT EUROPÉEN
LIBRE & OPEN SOURCE

La French Tech — Paris Region

EMPOWERING
OPEN INNOVATION
opensourcesummit.paris
#OSSPARIS16

PARIS OPEN SOURCE SUMMIT

16 & 17 NOVEMBRE 2016

DOCK PULLMAN
Plaine Saint-Denis

SPONSOR DIAMOND

Inria — INVENTEURS DU MONDE NUMÉRIQUE

SPONSORS PLATINUM

alter way

Smile — OPEN SOURCE SOLUTIONS

SPONSORS GOLD

BlueMind — Messagerie & espaces collaboratifs

henix

SPONSORS SILVER

axelor — ACCELERATEUR DE PERFORMANCE

XiVO — open-minded telecom systems

YaZiba.net — Powered by zimbra

PARTENAIRES INSTITUTIONNELS

îledeFrance

MAIRIE DE PARIS

POUR TOUTE INFORMATION COMPLÉMENTAIRE :
Email : contact@opensourcesummit.paris – Tel : 01 41 18 60 52

un événement

Systematic — Paris Region Digital Ecosystem

Tarsus — FRANCE

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°196

POSTGRESQL 9.5 : SÉCURITÉ ET RÉPLICATION

Guillaume LELARGE [Contributeur majeur de PostgreSQL,
Consultant Dalibo, auteur du livre « PostgreSQL – Architecture et notions avancées »]

Les développeurs de PostgreSQL travaillent depuis quelque temps sur deux améliorations très attendues : la sécurité au niveau ligne et la récupération rapide d'un ancien maître. Nous allons focaliser cet article sur ces deux améliorations.

Mots-clés : PostgreSQL, Nouveautés, Sécurité, Réplication

Résumé

PostgreSQL est un moteur de bases de données évoluant constamment. Avec une nouvelle version majeure chaque année, il est souvent difficile de suivre les améliorations apportées. Cet article a pour but de détailler quelques fonctionnalités importantes proposées par la version 9.5 pour les DBA.

Un gros manque de PostgreSQL au niveau de la réplication est la possibilité de remettre rapidement un ancien maître dans le cluster de réplication en tant qu'esclave. Le nombre d'étapes pour se faire était bien trop important pour que l'opération soit simple et rapide. Heureusement, un développeur a préparé un outil pour gérer ce problème de façon plus efficace et la version 9.5 l'intègre. De la même façon, un autre manque de PostgreSQL, dans le contexte de la sécurité cette fois, était de pouvoir donner en quelque sorte des droits pour la lecture ou l'écriture de certaines lignes dans une table. Là aussi, un travail d'assez longue haleine a eu lieu et il s'est terminé avec la version 9.5. Dans cet article, nous allons détailler ces deux nouvelles fonctionnalités particulièrement excitantes.

1 Sécurité au niveau lignes

La version 9.5 a apporté une grosse nouveauté au niveau de la sécurité : il est maintenant possible de restreindre les lignes lues par certains utilisateurs suivant une expression donnée. Pour cela, une politique de sécurité est créée indiquant l'expression à respecter sur une table particulière. Chaque requête exécutée par ces utilisateurs se verra ajouter une condition supplémentaire contenant l'expression.

Un exemple complet permet de mieux comprendre l'utilisation de cette nouvelle fonctionnalité.

Commençons par ajouter deux tables :

```
postgres=# CREATE TABLE bloggers(id serial PRIMARY KEY, blogger text);
CREATE TABLE
postgres=# CREATE TABLE posts(id serial PRIMARY KEY, blogger_id integer NOT NULL, title
text, contents text);
CREATE TABLE
```

Notre but est de faire en sorte qu'un blogueur ne puisse lire que les billets liés à son blog. Nous allons maintenant ajouter les blogueurs : tout d'abord en tant qu'utilisateur au niveau PostgreSQL et ensuite dans la table des blogueurs, ainsi que des billets pour les deux blogueurs :

```
postgres=# CREATE ROLE guillaume LOGIN;
CREATE ROLE
postgres=# INSERT INTO bloggers (blogger) VALUES ('Guillaume') RETURNING id;
```

```
 id
----
  1
(1 row)

INSERT 0 1
postgres=# INSERT INTO posts (blogger_id, title) VALUES (1,
'titre 1'), (1, 'titre 2');
INSERT 0 2
postgres=# CREATE ROLE markus LOGIN;
CREATE ROLE
postgres=# INSERT INTO bloggers (blogger) VALUES ('Markus')
RETURNING id;
 id
----
  2
(1 row)

INSERT 0 1
postgres=# INSERT INTO posts (blogger_id, title) VALUES (2,
'titre 1'), (2, 'titre 2'), (2, 'titre 3');
INSERT 0 3
```

Nous donnons les droits de lecture sur les deux tables aux deux nouveaux utilisateurs :

```
postgres=# GRANT SELECT ON bloggers,posts TO guillaume,Markus;
GRANT
```

De ce fait, chaque utilisateur peut lire le contenu complet de ces deux tables :

```
postgres=# \c - guillaume
You are now connected to database "postgres" as user "guillaume".
postgres=> SELECT * FROM posts;
 id | blogger_id |  title  | contents
----+------------+---------+----------
  1 |          1 | titre 1 |
  2 |          1 | titre 2 |
  3 |          2 | titre 1 |
  4 |          2 | titre 2 |
  5 |          2 | titre 3 |
(5 rows)
```

Maintenant, notre but est de faire en sorte que le blogueur **guillaume** ne puisse lire que les billets qu'il a écrits pour son blog. Pour cela, il est nécessaire d'activer la sécurité au niveau ligne sur la table **posts** :

```
postgres=> \c - postgres
You are now connected to database "postgres" as user "postgres".
postgres=# ALTER TABLE posts ENABLE ROW LEVEL SECURITY;
ALTER TABLE
postgres=# \c - guillaume
You are now connected to database "postgres" as user "guillaume".
postgres=> SELECT * FROM posts;
 id | blogger_id | title | contents
----+------------+-------+----------
(0 rows)
```

On s'aperçoit que, dès cette sécurité activée sur une table, les utilisateurs ne peuvent lire aucune ligne de la table. Attention, il n'y a pas d'erreur du style « pas de droit sur la table », mais tout simplement, aucune ligne n'est renvoyée, comme si la table était vide. Nous allons donc ajouter une politique de sécurité pour permettre aux utilisateurs de lire seulement leurs billets :

```
postgres=> \c - postgres
You are now connected to database "postgres" as user "postgres".
postgres=# CREATE POLICY only_yours ON posts FOR ALL TO
guillaume,markus USING (EXISTS(SELECT 1 FROM bloggers WHERE
id=blogger_id AND lower(blogger)=lower(current_user)));
CREATE POLICY
postgres=# \c - guillaume
You are now connected to database "postgres" as user "guillaume".
postgres=> SELECT * FROM posts;
 id | blogger_id |  title  | contents
----+------------+---------+----------
  1 |          1 | titre 1 |
  2 |          1 | titre 2 |
(2 rows)
postgres=> \c - Markus
You are now connected to database "postgres" as user "markus".
postgres=> SELECT * FROM posts;
 id | blogger_id |  title  | contents
----+------------+---------+----------
  3 |          2 | titre 1 |
  4 |          2 | titre 2 |
  5 |          2 | titre 3 |
(3 rows)
```

On voit bien ici que **Guillaume** et **Markus** ne voient pas les mêmes lignes dans la table **posts**. Ils ne voient que les lignes correspondant aux billets de leur blog. Plus exactement, ils ne voient que les lignes de la table respectant l'expression donnée au niveau de la politique de sécurité de cette table. Le plan d'exécution de la requête le montre parfaitement :

```
postgres=> EXPLAIN SELECT * FROM posts;
                                QUERY PLAN
-------------------------------------------------------------------------
 Hash Join  (cost=38.65..59.83 rows=405 width=72)
   Hash Cond: (posts.blogger_id = bloggers.id)
   ->  Seq Scan on posts  (cost=0.00..18.10 rows=810 width=72)
   ->  Hash  (cost=38.58..38.58 rows=6 width=4)
         ->  Seq Scan on bloggers  (cost=0.00..38.58 rows=6 width=4)
               Filter: (lower(blogger) = lower(("current_
user"())::text))
(6 rows)
```

Malgré un **SELECT** simple sur une table, on se retrouve avec une jointure sur la table **bloggers** pour récupérer l'identifiant de l'utilisateur connecté.

Une table peut être associée à plusieurs politiques de sécurité. Elles sont assemblées avec un OU logique. Les

politiques de sécurité sont utilisées pour les lectures et/ou les écritures, suivant la définition des politiques.

Le problème principal de ce système de sécurité est de ne pas se rendre compte de son activation sur certaines tables. Pour prendre un exemple simple, si nous utilisons l'utilisateur **Markus** pour sauvegarder la base avec **pg_dump**, la sauvegarde ne comprendra que les lignes visibles par cet utilisateur et nous n'aurons aucun avertissement que la sauvegarde a été partielle au niveau des données. C'est un risque important que certains utilisateurs se trompent sur leur sauvegarde. Un paramètre, nommé **row_security**, a été ajouté pour permettre à **pg_dump** d'avoir un message d'erreur si la lecture d'une table occulte des lignes, plutôt que de passer sous silence cette occultation.

```
postgres=> SET row_security TO off;
SET
postgres=> SELECT * FROM posts;
ERROR:  query would be affected by row-level security policy for
table "posts"
```

Il faut bien comprendre que ce paramètre ne permet pas de contourner la sécurité en mode ligne. Elle permet de récupérer une erreur plutôt que de passer sous silence une lecture partielle de la table.

Certains utilisateurs ne sont pas concernés par les politiques de sécurité : les super utilisateurs à l'évidence, mais aussi tous les utilisateurs ayant l'attribut **bypassrls** :

```
postgres=> \c - postgres
You are now connected to database "postgres" as user "postgres".
postgres=# ALTER ROLE markus BYPASSRLS;
ALTER ROLE
postgres=# \c - Markus
You are now connected to database "postgres" as user "markus".
postgres=> SELECT * FROM posts;
 id | blogger_id |  title  | contents
----+------------+---------+----------
  1 |          1 | titre 1 |
  2 |          1 | titre 2 |
  3 |          2 | titre 1 |
  4 |          2 | titre 2 |
  5 |          2 | titre 3 |
(5 rows)
```

En terme de performance, cette sécurité se paie au niveau de la planification (lors de la vérification des politiques de sécurité) et à l'exécution (lors de l'application des politiques de sécurité). La vérification n'est faite que pour les tables où la sécurité au niveau ligne est activée, donc elle n'a pas d'incidence sur les tables où cela n'est pas activé. Quant à l'exécution, si PostgreSQL ne le proposait pas, l'application devrait elle-même envoyer la clause **WHERE** et le coût serait donc bien toujours présent. En conclusion, les applications ne l'utilisant pas ne devraient pas payer le coût de cette fonctionnalité, et celles qui l'utilisent subiront évidemment les ralentissements, ces derniers dépendant fortement des expressions contenues au niveau des politiques de sécurité. Il est donc conseillé d'être prudent sur les expressions utilisées.

2 Récupérer rapidement un ancien maître

L'intégration de la réplication a apporté de nombreux utilisateurs à PostgreSQL. Elle a beaucoup évolué depuis la version 9.0 mais il restait encore un cas assez délicat : celui du *switchover* non maîtrisé. L'esclave est basculé maître, mais sans arrêter l'ancien maître qui continue sa vie de son côté. Puis, on souhaite récupérer l'ancien maître comme nouvel esclave. Ce cas-là n'avait comme solution que de renvoyer l'entièreté des données du nouveau maître vers le nouvel esclave. Dans le cas de grosses volumétries, ce n'était pas concevable. La seule solution possible était d'utiliser **rsync** pour diminuer la quantité de données envoyée d'un serveur à l'autre, mais cela nécessitait malgré tout de vérifier l'état de chaque fichier pour vérifier s'il avait été modifié depuis la bascule de l'esclave en maître. Et il ne fallait surtout pas se tromper dans les options fournies à **rsync**, sinon le résultat n'était clairement pas à la hauteur des espérances.

Arrive donc en 9.5 l'outil **pg_rewind**. Cet outil se base sur les journaux de transactions pour connaître la liste des fichiers modifiés, ce qui ne permet de renvoyer que ces fichiers. Voici un exemple montrant son utilisation.

Commençons par créer un répertoire de données, **s1**, qui sera le maître dans un premier temps :

```
$ initdb -D s1
The files belonging to this database system will be owned by user
"guillaume".
This user must also own the server process.
[...]
Success. You can now start the database server using:

    pg_ctl -D s1 -l logfile start
```

Créons le répertoire d'archivage et configurons le serveur **s1** en maître :

```
$ mkdir archives_s1
$ echo "wal_level = hot_standby;
archive_mode = on;
```

```
archive_command = 'cp %p /home/guillaume/article/tests/archives_
s1/%f';
max_wal_senders = 5;
min_wal_size = '32MB';
max_wal_size = '32MB';
hot_standby = on;
wal_log_hints = on;
logging_collector = on;" >> s1/postgresql.conf
$ echo "local replication replication trust" >> s1/pg_hba.conf
```

Enfin, démarrons le serveur **s1** :

```
$ pg_ctl -D s1 start
server starting
LOG:  redirecting log output to logging collector process
HINT:  Future log output will appear in directory "pg_log".
```

Ajoutons l'utilisateur pour la réplication :

```
$ createuser --replication replication
```

Ajoutons une table que nous allons peupler :

```
$ psql postgres
psql (9.5.2)
Type "help" for help.

postgres=# CREATE TABLE t1 (c1 integer, c2 timestamp);
CREATE TABLE
postgres=# INSERT INTO t1 SELECT i, clock_timestamp() FROM
generate_series(1, 1000000) i;
SELECT * FROM t1 ORDER BY c1 DESC LIMIT 1;INSERT 0 1000000
postgres=# SELECT * FROM t1 ORDER BY c1 DESC LIMIT 1;
   c1    |            c2
---------+----------------------------
 1000000 | 2016-04-01 13:17:40.606063
(1 row)

postgres=# \q
```

Créons maintenant l'esclave avec l'outil **pg_basebackup** :

```
$ pg_basebackup -D s2 -R -c fast -U replication
NOTICE:  pg_stop_backup complete, all required WAL segments have
been archived
```

Nous devons corriger la configuration du numéro de port :

```
$ echo "port = 5433" >> s2/postgresql.conf
```

Et ajouter la commande de restauration dans le fichier **recovery.conf** :

```
$ echo "restore_command = 'cp /home/guillaume/article/tests/
archives_s1/%f %p'" >> s2/recovery.conf
```

Nous pouvons enfin démarrer le serveur secondaire :

```
$ pg_ctl -D s2 start
server starting
LOG:  redirecting log output to logging collector process
HINT:  Future log output will appear in directory "pg_log".

$ ps -ef | grep postgres
guillau+  5855     1  0 13:16 pts/0    00:00:00 /opt/postgresql/95/bin/
postgres -D s1
guillau+  5871  5855  0 13:16 ?        00:00:00 postgres: logger process
guillau+  5873  5855  0 13:16 ?        00:00:00 postgres: checkpointer process
guillau+  5874  5855  0 13:16 ?        00:00:00 postgres: writer process
guillau+  5875  5855  0 13:16 ?        00:00:00 postgres: wal writer process
guillau+  5876  5855  0 13:16 ?        00:00:00 postgres: autovacuum launcher
process
guillau+  5877  5855  0 13:16 ?        00:00:00 postgres: archiver process
last was 000000010000000000000009.000000D0.backup
guillau+  5878  5855  0 13:16 ?        00:00:00 postgres: stats collector
process
guillau+  6011     1  0 13:18 pts/0    00:00:00 /opt/postgresql/95/bin/
postgres -D s2
guillau+  6027  6011  0 13:18 ?        00:00:00 postgres: logger process
guillau+  6028  6011  1 13:18 ?        00:00:00 postgres: startup process
recovering 000000010000000000000A
guillau+  6045  6011  0 13:18 ?        00:00:00 postgres: checkpointer process
guillau+  6046  6011  0 13:18 ?        00:00:00 postgres: writer process
guillau+  6047  6011  0 13:18 ?        00:00:00 postgres: stats collector
process
guillau+  6049  6011  0 13:18 ?        00:00:00 postgres: wal receiver process
streaming 0/A000108
guillau+  6050  5855  0 13:18 ?        00:00:00 postgres: wal sender process
replication [local] streaming 0/A000108
guillau+  6052  3147  0 13:18 pts/0    00:00:00 grep --color=auto postgres
```

Parfait. Nous avons **s1** en primaire et **s2** en secondaire, connecté à **s1**.

Insérons de nouvelles données pour nous assurer qu'elles parviennent bien à **s2** :

```
$ psql -p 5432 postgres
psql (9.5.2)
Type "help" for help.

postgres=# INSERT INTO t1 SELECT i, clock_timestamp() FROM
generate_series(1000001, 2000000) i;
INSERT 0 1000000
postgres=# \q
$ psql -p 5433 postgres
psql (9.5.2)
Type "help" for help.

postgres=# SELECT * FROM t1 ORDER BY c1 DESC LIMIT 1;
   c1    |            c2
---------+----------------------------
 2000000 | 2016-04-01 13:19:00.620041
(1 row)

postgres=# \q
```

Parfait également. Maintenant, nous allons faire un *failover* : **s2** devient serveur primaire, mais **s1** ne change pas de rôle.

```
$ pg_ctl -D s2 promote
server promoting
```

Nous avons donc maintenant deux serveurs complètement autonomes :

```
$ psql -p 5432 postgres
psql (9.5.2)
Type "help" for help.

postgres=# INSERT INTO t1 SELECT i, clock_timestamp() FROM
generate_series(2000001, 3000000) i;
INSERT 0 1000000
postgres=# SELECT * FROM t1 ORDER BY c1 DESC LIMIT 1;
   c1    |            c2
---------+----------------------------
 3000000 | 2016-04-01 13:19:44.810024
(1 row)

postgres=# \q
$ psql -p 5433 postgres
psql (9.5.2)
Type "help" for help.

postgres=# SELECT * FROM t1 ORDER BY c1 DESC LIMIT 1;
   c1    |            c2
---------+----------------------------
 2000000 | 2016-04-01 13:19:00.620041
(1 row)

postgres=# INSERT INTO t1 SELECT i, clock_timestamp() FROM
generate_series(2000001, 3000000) i;
INSERT 0 1000000
postgres=# SELECT * FROM t1 ORDER BY c1 DESC LIMIT 1;
   c1    |            c2
---------+----------------------------
 3000000 | 2016-04-01 13:20:17.503301
(1 row)

postgres=# \q
```

Maintenant, nous voulons configurer **s1** en tant qu'esclave de **s2**. Avant la 9.5, il fallait utiliser **pg_basebackup** ou **rsync** pour que les deux répertoires de données soient complètement identiques. Avec la 9.5, nous pouvons passer par l'outil **pg_rewind** pour cela. Commençons par arrêter le serveur **s1** :

```
$ pg_ctl -D s1 stop
waiting for server to shut down..... done
server stopped
```

Et lançons **pg_rewind** en lui indiquant le répertoire à mettre à jour et le nouveau serveur primaire :

```
$ pg_rewind -D s1 --source-server="port=5433 user=guillaume
dbname=postgres"
servers diverged at WAL position 0/E4E23D0 on timeline 1
could not open file "s1/pg_xlog/00000001000000000000000E": No
such file or directory

could not find previous WAL record at 0/E4E23D0
Failure, exiting
```

Cela n'a pas fonctionné car il nous manque des journaux de transactions déjà recyclés. Qu'à cela ne tienne, nous allons les copier des archives :

```
$ cp -i archives_s1/000000010000000000000000[E-F] archives_
s1/000000010000000000000001[0-6] s1/pg_xlog
```

Relançons **pg_rewind** :

```
$ pg_rewind -D s1 --source-server="port=5433 user=guillaume
dbname=postgres"
servers diverged at WAL position 0/E4E23D0 on timeline 1
could not open file "s1/pg_xlog/00000001000000000000000D": No
such file or directory

could not find previous WAL record at 0/DFFFFD8
Failure, exiting
```

Il nous en manque encore un. Copions-le et relançons **pg_rewind** :

```
$ cp -i archives_s1/000000010000000000000000D s1/pg_xlog
```

Il est à noter que j'utilise toujours l'option **-i** pour m'assurer que je n'écrase pas un fichier déjà présent. Son utilité reste à prouver mais ça me rassure.

```
$ pg_rewind -D s1 --source-server="port=5433 user=guillaume
dbname=postgres"
servers diverged at WAL position 0/E4E23D0 on timeline 1
rewinding from last common checkpoint at 0/D402020 on timeline 1
Done!
```

C'est fait en quelques secondes. Il ne reste plus qu'à configurer **s1** en tant qu'esclave de **s2** et à le lancer :

```
$ mv s1/recovery.done s1/recovery.conf
```

Les fichiers **postgresql.conf** et **recovery.conf** doivent être modifiés pour indiquer les bons numéros de port.

```
$ pg_ctl -D s1 start
server starting
LOG:  redirecting log output to logging collector process
HINT:  Future log output will appear in directory "pg_log".
$ psql -p 5432 postgres
Expanded display is used automatically.
psql (9.5.2)
Type "help" for help.
```

```
postgres=# SELECT * FROM t1 ORDER BY c1 DESC LIMIT 1;
   c1    |            c2
---------+---------------------------
 3000000 | 2016-04-01 13:20:17.503301
(1 row)

postgres=# \q
```

s1 a bien récupéré la dernière valeur de la table **t1** telle qu'elle est sur le serveur **s2**. Si on ajoute des données sur **s2**, on devrait les voir arriver sur **s1** :

```
$ psql -p 5433 postgres
psql (9.5.2)
Type "help" for help.

postgres=# INSERT INTO t1 SELECT i, clock_timestamp() FROM
generate_series(3000001, 4000000) i;
INSERT 0 1000000
postgres=# \q
$ psql -p 5432 postgres
psql (9.5.2)
Type "help" for help.
```

```
postgres=# SELECT * FROM t1 ORDER BY c1 DESC LIMIT 1;
   c1    |            c2
---------+---------------------------
 4000000 | 2016-04-01 13:24:38.830395
(1 row)
```

pg_rewind a bien fait son travail. Gageons qu'il sera fortement utilisé par les administrateurs pour simuler des *switchovers* plus ou moins bien contrôlés.

Conclusion

Ce troisième article sur la version 9.5 montre à quel point les développeurs ont pu réussir à intégrer des fonctionnalités attendues depuis longtemps par les utilisateurs. Le prochain article continuera à montrer de nouvelles fonctionnalités de PostgreSQL mais cette fois sur la version 9.6 dont la sortie est prévue ce mois-ci, en septembre. ■

J'EN AI MA CLAQUE, J'ABANDONNE TOUT
ET JE PARS ÉLEVER DES CHÈVRES DANS LE LARZAC

The Cheshire CAT [Everyone here is mad]

On fait un beau métier tout de même non ? Bien loin des contraintes des boulots classiques, on n'a pas d'horaires fixes, pas de contraintes, pas de *dress* code obligatoire. On est les geeks, ceux qui bossent debout sur des bureaux étranges et marchent en chaussettes dans les open space. Mais alors si on est si chanceux, pourquoi tellement d'entre nous sont malheureux ?

Mots-clés : Humeur, Développeur, Équipe, Burnout, Santé, LifeStyle

Résumé

On est censé être des privilégiés. D'ailleurs, soyons honnêtes deux secondes, nous sommes réellement des privilégiés dans le monde du travail. Mais alors pourquoi tant d'entre nous décrochent, changent de vie ou ne semblent jamais heureux mais enfermés dans une cage de tristesse et de ressentiment ? Et si on essayait de comprendre pour éviter de grossir les statistiques de personnes tombant dans le *burnout* ou la tristesse au travail.

Cela m'a frappé il y a quelques semaines. Je connaissais un nombre important de dev ayant tout plaqué pour changer de vie, de pays ou même de métier. Et quand j'ai commencé à faire une liste précise, je me suis rendu compte que c'était même pire que ce que je ne le pensais. Que c'était quasiment une hécatombe. Pour tout vous écrire, la phrase qui sert de titre à mon article, je l'ai moi-même prononcée à de nombreuses reprises au cours des dernières années. Et d'une manière tout à fait sincère, je me demande aujourd'hui si d'ici dix ans, je serai encore un développeur. Autant dire que je suis bien loin de mes jeunes années alors que fringant matou, je commençais à peine à user mes coussinets sur les claviers de la vie active et que je ne voyais alors pas de plus belle avenir que celui de continuer à coder, envers et contre tout. Et au moment de trouver un sujet pour cet article de rentrée, je me suis dit qu'il n'y avait pas de meilleur sujet pour bien démarrer une nouvelle année qu'un papier sur le sujet. Surtout qu'en y regardant de plus près, j'ai l'impression de pouvoir distinguer une caractéristique commune à tous celles et ceux qui ont lâché le métier. Dans tous les cas, c'était des passionnés. Et comme vous lisez en ce moment ces lignes, j'en déduis que vous avez acheté ce magazine et qu'il y a donc de fortes chances que vous soyez vous aussi un ou une passionnée. Et si jamais vous avez récupéré ce magazine dans la salle d'attente de votre docteur ou de votre dentiste, ne le reposez pas tout de suite. Après tout, si vous l'avez ouvert au début, c'est que vous devez être un minimum informaticien, alors on ne sait jamais, peut-être que par hasard, vous allez finir par devenir passionnés, dans ce cas-là, la lecture de cet article pourra vous être utile.

Une dernière remarque avant de me lancer dans le vif du sujet. Vous aurez parfois, peut-être même souvent l'impression de ne lire que des choses que vous connaissez déjà. Ce sera peut-être même vraiment le cas. Mais avant de pester et

de sauter au prochain article, faites une minute d'introspection. Si ce que vous venez de lire n'éveille aucun écho, très bien, l'article suivant vous tend les bras. Dans le cas contraire, restez encore un peu plus longtemps avec mes mots.

1 De l'art de tirer sur la corde, jusqu'à ce qu'elle cède

Ici, je pense que je pourrais faire mien le proverbe parlant des cordonniers et de leurs chaussures. J'ai en effet plus l'habitude de brûler les moustaches par les deux bouts plutôt que de me préserver les coussinets.

1.1 De l'importance d'un rythme soutenable

Pour être passé par un certain nombre d'équipes de développement, je suis bien placé pour savoir que quand le projet est en retard, que les choses commencent à s'échauffer, la variable d'ajustement c'est bien souvent, et bien malheureusement, le temps de travail. Et que passé un certain temps, cette espèce d'état d'urgence de l'équipe de développement, qui devrait, comme tout état d'urgence, être une mesure exceptionnelle, s'installe. On se retrouve alors dans une fuite en avant : bossant d'abord les soirées, puis les week-ends, puis les nuits. C'est ainsi que le fait ne pas avoir d'horaire, de ne pas pointer, se retourne contre nous. Parce que du coup, il n'y pas de trace. On peut bosser quatorze heures, ce n'est pas « grave ». Après tout, on le sait tous, les séries américaines nous l'ont appris, les informaticiens travaillent la nuit, toute la nuit et mangent des pizzas. Sauf qu'en vrai, on le sait tous, dormir n'est pas vraiment un luxe mais bien une nécessité. Le manque de sommeil, même si on peut le camoufler avec caféine et boisson énergétique, vous n'êtes pas sans le savoir, finit toujours par vous rattraper. Et puis si le manque de sommeil ne générait qu'un problème de productivité, cela ne serait « pas si grave ». Mais si vous avez essayé de travailler ainsi en réduisant au maximum votre temps de sommeil, vous avez dû constater qu'en plus d'être moins productif, vous devenez aussi irritable, colérique, sujet à des sautes d'humeur. Ce qui a pour conséquence de ralentir encore plus votre productivité mais aussi, et c'est encore plus impactant, celle de toute l'équipe. Ainsi on tombe dans un cercle infernal de baisse de la productivité qui conduit à une détérioration des conditions de travail qui accentue cette baisse de la productivité et cela jusqu'à ce que l'équipe explose. L'importance d'un rythme de travail soutenable, qui ne sera pas le même pour chaque individu mais qui doit être l'étalon maximal de votre effort est donc primordial.

1.2 De l'assèchement de la créativité

J'ai toujours considéré que notre métier était un métier de création, qu'écrire du code, ce n'est pas simplement taper sur son clavier une suite de mots qui au final finissent par faire un programme. En cela, j'ai toujours été en désaccord avec ceux qui peuvent penser qu'en fait le travail d'un développeur c'est juste de pisser du code et qu'une fois que les spécifications détaillées ont été rédigées, il n'y a plus rien à faire à part lancer les machines outils humaines que l'on nomme développeur. Donc pour pouvoir faire notre travail, il faut à mon sens être créatif. Et donc être capable de faire appel à sa muse créative quand il le faut. Je ne vais pas ici vous conseiller de ne pas trop faire appel à votre créativité parce que je pense que l'on n'assèche pas sa créativité en l'utilisant. C'est même le contraire, je pense que la créativité est comme un muscle. Plus vous l'utilisez, plus elle devient forte. Par contre, je suis persuadé que la monotonie et les automatismes assèchent la créativité. Faire les choses à la manière d'un robot, parce qu'on a toujours fait ainsi, c'est cela qui peut réduire votre faculté à créer et donc à être un bon développeur. C'est la même chose en ce qui concerne l'absence d'activité extérieure au boulot. C'est pour cela que les hobbies sont tellement importants.

Parce que grâce à eux, on peut penser à autre chose, laisser son esprit parcourir d'autres façons de penser et au final être plus créatif, plus alerte lorsqu'il sera temps de se remettre au boulot. Personnellement, j'ai un faible pour l'écriture, les jeux vidéos et la couture. Mais au final ce qui compte, c'est d'arriver à pouvoir faire autre chose, même si c'est disputer un match de curling, pour permettre à votre esprit de se ressourcer.

2 De l'importance du code et de l'équipe

Avoir un rythme de vie au travail correct est donc la première des choses pour ne pas avoir envie de changer de métier au bout de quelques années. Mais ce n'est que la première étape. Les interactions avec l'équipe ainsi que la qualité du code lui-même sont également des facteurs très importants qui permettent, ou pas, de garantir que l'envie de coder restera là.

2.1 Hygiène du code

J'ai remarqué que dans un certain nombre de projets d'envergure, ceux qui feront que vous allez rester plusieurs années dans la même équipe, dans la même entreprise, un même phénomène tend à apparaître. Au début, la base de code est petite, bien maîtrisée, on sait ce que l'on fait et on sait ce que le code fait. Et puis avec l'augmentation du nombre de lignes de code, petit à petit, on finit par ne plus avoir confiance, par avoir peur de sa propre création. Et alors qu'au début, chaque mise en production partielle était une réussite, une occasion de se congratuler, par la suite les mises en production se transforment en monstre dont toute l'équipe a peur. Parce qu'on doute de son code, parce que mettre en production, tout de même, c'est dangereux. Parce qu'on ne sait pas trop les impacts que cela va avoir. Et puis parce qu'il y a « toujours des bugs » et donc quelque chose va finir par casser. Et que cela pourrait être grave. Même dans les équipes qui pratiquent du déploiement continu, j'ai trouvé ce phénomène. Cette sensation de danger, pour le coup permanent, qui finit par complètement paralyser les équipes. Cette sensation de responsabilité qu'implique le commit que l'on va faire et qui finit par devenir tellement écrasante qu'on n'ose plus rien toucher à part ajouter des tests et faire des refactoring minimalistes. Et le plus étrange, bien souvent, c'est que cet ajout de test, cette hypertrophie de la couverture de tests ne sert à rien. Parce que l'équipe a perdu confiance dans le code écrit, voire en elle-même et sa capacité de création. Et c'est à ce moment-là que commencent à naître les idées de tout réécrire de zéro, de tout refaire pour racheter la dette technique, pour changer de techno, parce que celle qu'on utilise n'est plus celle qu'il faut utiliser. Et là encore, on rentre dans un cercle vicieux qui peut bloquer complètement le développement d'un projet, voire avoir pour conséquence de saborder un projet. Mais comment faire en sorte que cela n'arrive pas, si de toute façon la couverture de test ne suffit pas à rassurer l'équipe ? J'ai souvent constaté que dans les projets où il y avait une vraie documentation à la fois technique mais aussi fonctionnelle et surtout à jour, ce phénomène de paralysie de l'équipe avait moins tendance à se produire. Une métrique importante à prendre en compte est également le nombre de lignes de code à connaître pour comprendre un bout du projet. Si pour ajouter une ligne de code dans votre projet actuel vous devez avoir en tête plusieurs milliers de lignes, avec leurs ramifications, leurs impacts, les conséquences de leur activation, bien entendu que vous allez être tétanisé à l'idée de modifier quelque chose. Pour éviter cela, deux bonnes pratiques peuvent, à mon sens, être mises en place. Tout d'abord essayer le plus possible de mettre en place une architecture de type microservice en rendant autonome au maximum chacun de vos petits services. Ensuite, en injectant le minimum d'intelligence métier dans les morceaux de code utilitaires que vous faites pour que l'utilisation de ce qui devrait être des bibliothèques facilitatrices ne finissent pas par devenir plus consommateur de temps que de ne pas les utiliser.

2.2 A plusieurs on va plus loin

Le proverbe africain complet est « Tout seul, on va plus vite. Ensemble, on va plus loin. » Et je trouve que cela représente totalement le développement informatique. Mais pour que cela soit vrai, il faut que l'équipe fonctionne bien, que les gens apprécient le fait de travailler ensemble et fonctionne vraiment en collectif. J'en ai déjà parlé plusieurs fois dans ces pages, mais les interactions entre les différents membres d'une équipe sont pour moi primordiales. Même si j'écris cet article en pleine période d'Euro 2016, je vais vous éviter les métaphores footballistiques. Mais il n'en est pas moins vrai que le développement en équipe s'apparente pour moi à un sport collectif. Les comportements individualistes ne sont pas la meilleure des choses pour le bon fonctionnement de l'organisme « équipe de développement ». Les jugements entre membre d'une même équipe ou l'intransigeance ne sont pas forcément les meilleures choses pour travailler dans un environnement plaisant. J'ai connu des projets où la moindre erreur était impossible, où les reviews de code n'étaient pas un instant de partage sur la façon de coder et d'amélioration collective mais une chasse à la faute, une excuse pour pouvoir se juger, définir une hiérarchie et construire ainsi des structures de pouvoir sclérosantes. Autant vous dire qu'ensuite, l'ambiance n'était pas au beau fixe. Et le pire, c'est qu'au final, l'ambiance globale

était totalement dégradée. Les personnes ayant à subir cette ambiance finissent par se démotiver et soit partir soit ne plus en avoir rien à faire de rien et celles qui se sont auto assignées au rôle de gardien du temple s'enfoncent peu à peu dans une mauvaise humeur et un pessimisme à toute épreuve. Et on se retrouve avec, d'un coté, des aigris râlant sur tout et tout le monde et finissant par refuser de travailler avec tous ceux qui ne valident pas leurs critères d'excellence (autant dire que ce petit groupe d'élus se réduit très rapidement à eux-mêmes) et de l'autre, une équipe démotivée, déresponsabilisée vu que de toute façon, tout ce qu'elle produira ne trouvera jamais grâce aux yeux du « maître des bonnes pratiques ».

2.3 Sans entraînement, pas de salut

Souvent, on utilise le vocabulaire du sport pour imager le développement informatique. Et c'est effectivement une bonne métaphore. Arrive alors un fait que l'on ne peut nier. Les sportifs s'entraînent encore et encore pour délivrer la meilleure des performances possibles. Mais le temps d'entraînement est largement supérieur au temps passer à réaliser une performance réelle. Alors que dans le développement informatique on est toujours en train de délivrer une performance et on ne s'entraîne jamais. Je vous conseille donc vivement à mettre en place vos propres entraînements. Participer à des « coding dojo » est pour moi une excellente façon de s'entraîner. Mettre en place des moments pour simplement apprendre de nouveaux langages et s'entraîner à les pratiquer, même si cela demande un peu plus de motivation et de ténacité, est également, à mon avis, une excellente pratique.

3 Du reste, mais qui n'est pas à négliger

Jusqu'ici, j'avais réussi à regrouper ensemble les causes de mal-être dans une équipe de développement. Dans cette dernière partie, je vais parler du reste, des autres choses qui peuvent aboutir à vous donner envie de changer de boulot. Même si je n'ai pu les regrouper sous un titre commun, ne pensez pas qu'elles sont sans importance, bien au contraire.

3.1 Vaincre le principe de Peter

Ici, on se retrouve sur un mal typiquement français. Je ne sais pas pourquoi, mais dans notre beau pays, si tu es toujours un développeur à trente-cinq ans, tu as raté ta vie. Pour réussir, il faudrait en effet quitter le plus rapidement le poste de dev pour arriver au sacro-saint poste de chef de projet, le seul qui a de la valeur, bien entendu. Sauf que bien entendu, les compétences pour être dev ou chef de projet ne sont pas du tout les mêmes. Et on se retrouve donc en plein dans le principe de Peter. Voir des gens compétents en terme de développement qui évoluent vers leur position d'incompétence en devenant chefs de projet alors qu'ils n'en ont pas forcément l'envie, les compétences ou la formation a toujours été pour moi un crève-cœur. Voir d'autres développeurs être mal considérés parce que justement ils refusent d'être chefs de projet est encore pire. Surtout, lorsqu'au final, ils finissent par signer avec une entreprise étrangère qui a bien compris que l'évolution d'un bon développeur n'est pas chef de projet mais *lead developer*.

3.2 Survivre aux désillusions

Je me souviens encore l'une de mes premières rencontres avec le monde du travail et du développement en entreprise. Plein d'illusions et de rêve, je voulais faire de la création de jeux vidéos. Je mets à dessein le mot création parce que je ne voulais pas être juste un développeur qui pisse du code en fonction du cahier des charges qu'on lui a donné. Je voulais pouvoir participer à la démarche créative des choses, proposer des idées, donner mon avis. Après cinq mois, j'ai bien compris que je pouvais faire une croix sur mes belles illusions, que j'étais juste un codeur et que je n'avais pas droit à la parole. Ce fut la fin de mon expérience dans la création de jeux vidéos. Bien entendu tout le monde ne rêve pas de créer des jeux vidéos. Mais bien souvent les développeurs passionnés se lancent dans le métier avec des rêves, des ambitions de changer le monde. Et puis dix ou quinze ans après, quand on se retourne et que l'on regarde ce que l'on a fait, on se rend compte qu'au final on a simplement vécu une longue suite de métro / boulot / dodo. Et cette prise de conscience n'est pas forcément des plus évidentes à gérer. Pour ma part, pour combattre cela, je contribue à des projets libres. C'est ma façon de faire en sorte, dans une moindre mesure, de « changer le monde » et cela me permet de ne pas passer mon temps libre à mes rêves de grandeur de jeune développeur.

3.3 Gérer sa dette éthique

On entend souvent parler de dette technique et de l'importance de gérer celle-ci, de la limiter, de la racheter le plus vite possible pour continuer à avoir une base de code la plus saine possible. Mais on parle bien moins souvent d'un autre phénomène que j'ai baptisé la dette éthique. Celle qui s'accroît quand vous faites un accro à vos valeurs. Celle qui écorne petit à petit l'idée que vous vous faites de vous-même, lorsque vous acceptez cette mission de prestation alors que vous savez que vous devriez dire non, que vous ne dites oui que parce que « vous n'avez pas le choix », il faut bien facturer. Si vous restez en poste parce que la conjoncture est ce qu'elle est, qu'il faut être content d'avoir un travail et donc il faut subir. Et toute cette dette éthique s'accumule, petit à petit, jusqu'à ce qu'un matin, en vous regardant dans la glace, tout explose...

3.4 Ne pas se scléroser, penser à partager

Le métier de développeur n'est pas forcément de tout repos. Les technologies évoluent, les langages changent, les méthodologies de travail se transforment. Et il faut suivre. Encore et toujours. La tentation est alors forte de refuser le changement et l'effort permanent qu'une remise en question et un apprentissage sans fin demandent. De devenir le vieux grognon (ou la vieille grognonne) qui râle dans son coin, qui fait les choses de la même manière qu'il y a dix ans et qui râlent que c'était mieux avant, quel que soit le sujet dont on parle. Cette lente fossilisation est bien l'une des pires fins possible pour un dev. La meilleure parade pour éviter cela me semble être le partage et la discussion. Aller à des conférences, échanger avec ses pairs et faire preuve d'humilité. Accepter que des petits juniors puissent vous apprendre énormément, autant que vous allez leur apprendre.

Conclusion

Il est bien difficile de garder la flamme des débuts, après dix ou quinze ans de pratique dans la vie active. Et il me semble que dans la pratique de nos métiers, le fait que la passion puisse avoir une part importante dans ce qui nous fait nous lever le matin est un facteur qui peut être aggravant. Et malheureusement bien peu d'entreprises semblent avoir compris qu'avoir des gens heureux permet d'avoir des gens productifs (même s'il semblerait que certains commencent à s'en rendre compte, si j'en crois les grands quatre par trois de publicité pour une assurance ou une mutuelle ou je ne sais quoi parlant d'augmentation de la productivité grâce au bonheur). Si vous voulez continuer à aimer faire votre travail et si vous ne voulez pas tout lâcher et changer de vie sur un coup de sang ou de déprime, il sera de votre responsabilité de faire en sorte d'être heureux dans votre pratique quotidienne du développement informatique. ■

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 TIC TAC TOE UN JEU SIMPLE À DÉVELOPPER ?

Tristan COLOMBO

Il n'y a rien de plus bête que le Tic Tac Toe n'est-ce pas ? D'ailleurs dans les nouveaux programmes d'apprentissage pour le cycle 4 (5ème, 4ème, 3ème), il fait partie des petits jeux que les élèves doivent pouvoir développer. Penchons-nous donc un peu sur ce jeu ô combien facile...

Mots-clés : Tic Tac Toe, Jeu du morpion, Algorithme, Min-max, Enseignement, Intelligence artificielle

Résumé

Le jeu du Tic Tac Toe est un jeu se déroulant sur une grille où deux joueurs posent leurs pions dans le but d'être le premier à en aligner trois. Cet article se propose d'étudier le codage d'un tel jeu depuis la version la plus simple jusqu'à une IA imbattable tout en se questionnant sur la réalisation d'un tel programme par des élèves de 3ème.

Le BO (Bulletin Officiel) nous informe que l'enseignement en informatique qui va être introduit en cycle 4 à la rentrée 2016 *« n'a pas pour objectif de former des élèves experts, mais de leur apporter des clés de décryptage d'un monde numérique en évolution constante »*. À titre d'exemple de projet pouvant être abordé par les élèves se trouve le Tic Tac Toe. Je vous propose donc d'étudier ce jeu et son implémentation de manière à nous faire une petite idée du niveau d' « expertise » nécessaire pour le développer ou pour penser qu'il sera accessible à tous les élèves de cycle 4 (soyons indulgents, considérons qu'il s'agit uniquement de la 3ème).

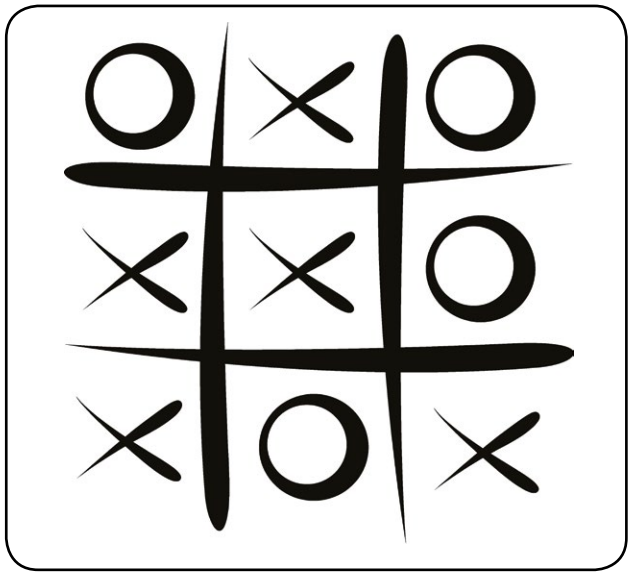


Fig. 1 : Une partie de Tic Tac Toe où les joueurs ont fait match nul. Comme il y a 5 croix et 4 ronds, on en déduit que c'est le joueur possédant les croix qui a commencé la partie.

1 Le Tic Tac Toe ou jeu du Morpion

Le Tic Tac Toe, plus connu chez nous sous le nom de jeu du Morpion, est un jeu qui fait s'opposer deux joueurs sur une grille carrée de trois cases de côté (donc neuf cases en

tout). Chaque joueur dispose d'un ensemble de pions identiques mais différents de ceux de l'adversaire. Généralement il s'agit de ronds et de croix. Le but du jeu est de créer un alignement horizontal, vertical ou en diagonale avec trois de ces pions pour remporter la partie. Comme le nombre de combinaisons est très limité, il n'est pas rare d'assister à des parties nulles où les deux joueurs se sont neutralisés (voir figure 1).

2 Jeu à deux joueurs humains

La version la plus simple du jeu s'adressera à deux joueurs humains et ne fera intervenir aucune intelligence artificielle. D'un point de vue technique, il faudra être capable de stocker un tableau en mémoire et donc de savoir manipuler des variables et pouvoir déterminer quand est-ce qu'un des deux joueurs a gagné.

2.1 Initialisation et affichage de la grille

Pour initialiser et afficher la grille, il faut déjà savoir utiliser des variables, des fonctions, des boucles et des tests conditionnels. Ajoutons à cela un soupçon de listes et nous obtenons le code suivant :

```
01: def jeton(valeur):
02:     if valeur == 0:
03:         return ' '
04:     elif valeur == 1:
05:         return 'X'
06:     else:
07:         return 'O'
08:
09: def affiche(grille):
10:     for ligne in range(3):
11:         for colonne in range(3):
12:             if colonne != 0:
13:                 print('|', end='')
14:             print(jeton(grille[ligne][colonne]), end='')
15:         print()
16:         if ligne != 2:
17:             print('-' * 5)
18:
19: if __name__ == '__main__':
20:     grille = [[0] * 3 for x in range(3)]
21:     affiche(grille)
```

Même si la ligne 20 est plus synthétique telle qu'elle est écrite, pour des élèves de troisième il faudrait préférer l'écriture et la présentation suivante :

```
grille = [[0, 0, 0],
          [0, 0, 0],
          [0, 0, 0]]
```

Note

On pourrait être tenté d'écrire **`[[0] * 3] * 3`** pour l'initialisation de la grille. Cette écriture bien que claire et synthétique ne produit pas le résultat escompté puisque les trois listes **`[0, 0, 0]`** incluses dans la liste principale ne seront pas des copies mais des références : modifier un élément d'une liste entraînera la modification des « autres » puisqu'il ne s'agit en fait que d'une seule et même liste...

De même pour la ligne 17 qu'il faudrait écrire **`print('-----')`** pour une meilleure compréhension.

Nous considérons que la valeur **`0`** indique une case vide, **`1`** la présence d'une croix et **`2`** la présence d'un rond. Il y a ici une notion de codage de l'information et de décodage par la fonction **`jeton()`** des lignes 1 à 7.

La fonction **`affiche()`** utilise deux boucles imbriquées permettant de se déplacer sur les indices de la liste **`grille`**.

On peut améliorer un peu notre code en ajoutant l'affichage de la numérotation des lignes et colonnes :

```
...
09: def affiche(grille):
10:     print('  1 2 3')
11:     for ligne in range(3):
12:         print(str(ligne + 1) + ' ', end='')
...
```

Nous obtiendrons le résultat suivant :

```
$ python3 tictactoe.py
  1 2 3
1  | |
  -----
2  | |
  -----
3  | |
```

Nous venons juste de commencer. À ce stade, on peut raisonnablement penser qu'avec de bonnes explications la majorité des élèves peut encore suivre.

2.2 Questionnement des joueurs

Il faut maintenant créer une boucle dans laquelle nous demanderons aux joueurs dans quelle case ils souhaitent placer leur pion. Cela soulève plusieurs problèmes :

- Par simplification ergonomique, les coordonnées des cases sont exprimées à l'aide d'entiers compris entre **1** et **3** alors que dans la liste les indices sont compris entre **0** et **2**... il faut faire attention de ne pas mélanger les représentations.
- La case est-elle déjà occupée par un pion ? Si oui il faut demander à l'utilisateur de corriger sa saisie.
- Le pion qui va être posé forme-t-il une ligne gagnante ? Si c'est le cas, il faut arrêter le jeu.

Il faut segmenter le problème initial ; ce qui est évident pour nous ne le sera pas forcément pour des élèves de 3ème et il faudra donc les guider dans leur développement.

2.2.1 La case est-elle occupée par un pion ?

Si la valeur d'une case de la grille vaut **0** alors c'est que cette case est vide :

```
...
21: def estLibre(grille, ligne, colonne):
22:     return grille[ligne][colonne] == 0
...
```

Derrière cette syntaxe se cache beaucoup d'informations implicites et il sera sans doute plus simple d'introduire le code suivant :

```
def estLibre(grille, ligne, colonne):
    if grille[ligne][colonne] == 0:
        return True
    else:
        return False
```

2.2.2 Le pion posé forme-t-il une ligne gagnante ?

Il n'y a que huit combinaisons gagnantes qui sont représentées en figure 2. Pour un pion posé en **(ligne, colonne)**, il faut donc vérifier s'il peut former une ligne complète horizontalement, verticalement ou en diagonale. Les cas sont les suivants :

- Pour **ligne** fixée, on parcourt les colonnes ;
- Pour **colonne** fixée, on parcourt les lignes ;
- Si **ligne = colonne** :
  - → si **ligne = 1** (ou **colonne = 1** puisqu'il y a égalité entre les deux), on parcourt les deux diagonales ;
  - → sinon on parcourt la diagonale descendante gauche - droite.
- Si **(ligne, colonne) = (0, 2)** ou **(2, 0)**, on parcourt la diagonale ascendante gauche - droite.

Dans ces parcours de lignes, à partir du moment où on rencontre une case vide ou une case contenant un pion du joueur adverse, on peut arrêter le traitement.

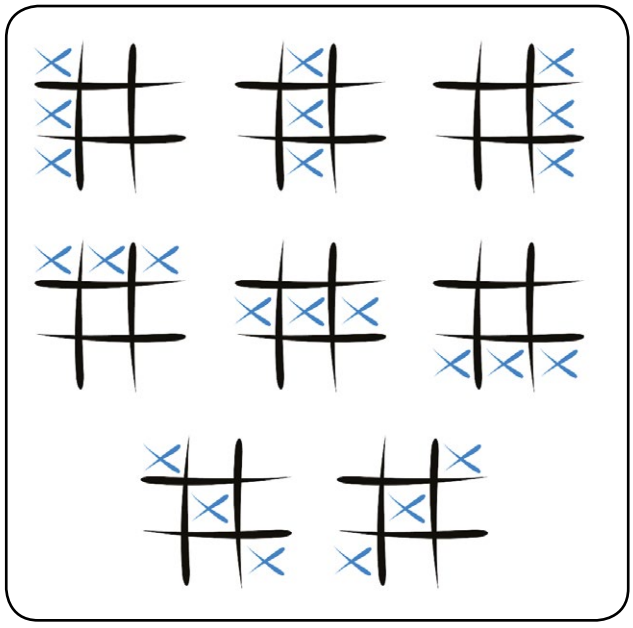


Fig. 2 : Les 8 combinaisons gagnantes du Tic Tac Toe.

```
...
24: def estGagnantLigne(grille, ligne, colonne):
25:     typePion = grille[ligne][colonne]
26:     for col in range(3):
27:         if grille[ligne][col] != typePion:
28:             return False
29:     return True
30:
31: def estGagnantColonne(grille, ligne, colonne):
32:     typePion = grille[ligne][colonne]
33:     for line in range(3):
34:         if grille[line][colonne] != typePion:
35:             return False
36:     return True
37:
38: def estGagnantDiago_1(grille, ligne, colonne):
39:     typePion = grille[ligne][colonne]
40:     for pos in range(3):
41:         if grille[pos][pos] != typePion:
```

```
42:             return False
43:     return True
44:
45: def estGagnantDiago_2(grille, ligne, colonne):
46:     typePion = grille[ligne][colonne]
47:     if grille[1][1] != typePion:
48:         return False
49:     if grille[0][2] != typePion or grille[2][0] != typePion:
50:         return False
51:     return True
52:
53: def estGagnant(grille, ligne, colonne):
54:     if estGagnantLigne(grille, ligne, colonne):
55:         return True
56:
57:     if estGagnantColonne(grille, ligne, colonne):
58:         return True
59:
60:     if ligne == colonne:
61:         if ligne == 1:
62:             return estGagnantDiago_1(grille, ligne, colonne) or \
63:                    estGagnantDiago_2(grille, ligne, colonne)
64:         else:
65:             return estGagnantDiago_1(grille, ligne, colonne)
66:     elif (ligne == 0 and colonne == 2) or (ligne == 2 and colonne
== 0):
67:         return estGagnantDiago_2(grille, ligne, colonne)
68:     else:
69:         return False
...
```

Même si j'ai utilisé une syntaxe plus concise que ce que l'on pourrait proposer à des collégiens, force est de constater que même sans intégration de la moindre parcelle d'intelligence artificielle le problème n'est pas si évident à coder (mais reste réalisable).

2.2.3 La boucle de jeu

Nous allons légèrement modifier le bloc du programme principal de manière à appeler une fonction **demarrerJeu()** qui contiendra l'initialisation de la grille et la boucle de jeu. Une variable **tours** permettra de compter le nombre de tours de jeu et de déterminer si la partie est nulle, la variable **joueur** contiendra le numéro du joueur (**0** ou **1** qu'il faudra donc incrémenter d'une unité à l'affichage et pour placer un pion dans la grille), et une variable booléenne **gagnant** qui indiquera que l'un des deux joueurs a remporté la partie.

```
...
071: def demarrerJeu():
072:     grille = [[0] * 3 for x in range(3)]
073:     tours = 0
074:     gagnant = False
075:     joueur = 0
076:
077:     while not gagnant and tours < 9:
078:         affiche(grille)
079:         valide = False
080:         while not valide:
081:             ligne, colonne = map(int, input('Joueur {}:'.
format(joueur + 1)).split(' '))
082:             valide = estLibre(grille, ligne - 1, colonne - 1)
083:             if not valide:
084:                 print('La case ({}, {}) n\'est pas libre
!'.format(ligne, colonne))
085:         grille[ligne - 1][colonne - 1] = joueur + 1
086:         gagnant = estGagnant(grille, ligne - 1, colonne - 1)
087:         joueur = (joueur + 1) % 2
088:         tours += 1
089:
090:     affiche(grille)
091:
092:     if gagnant:
093:         joueur = (joueur + 1) % 2
094:         print('Le joueur {} a gagné !'.format(joueur + 1))
095:     else:
096:         print('Match nul')
097:
098:
099:
100: if __name__ == '__main__':
101:     demarrerJeu()
```

Je suis tout à fait conscient que la ligne 81 devrait être écrite de manière plus simple dans un souci pédagogique. Ici la fonction **input()** renvoie une chaîne de caractères sur laquelle on applique la méthode **split(' ')** qui nous retourne une liste de chaînes de caractères (découpage suivant le caractère espace), puis nous appliquons la fonction **int()** sur chacun des éléments de la liste via l'appel à **map()**. Voici à quoi correspond cette ligne en version non condensée :

```
saisie = input('Joueur {}:'.format(joueur + 1))
liste_saisie = saisie.split(' ')
ligne = int(liste.saisie[0])
colonne = int(liste.saisie[1])
```

Note

Le fait de convertir le numéro des joueurs entre représentation « réelle » (**1** et **2**) et représentation « numérique » (**0** et **1**) introduit une complexité supplémentaire qui pourrait être évitée en utilisant directement les valeurs **1** et **2** (ce sera le cas un peu plus loin). Toutefois cette écriture permet d'introduire l'opérateur modulo (**%**) et il me semblait intéressant de l'utiliser.

Voici un exemple de partie :

```
$ python3 tictactoe.py
  1 2 3
1  | |
  -----
2  | |
  -----
3  | |
Joueur 1:1 1

  1 2 3
1 X| |
  -----
2  | |
  -----
3  | |
Joueur 2:1 2

...

  1 2 3
1 X|O|
  -----
2 O|X|
  -----
3  | |
Joueur 1:3 3

  1 2 3
1 X|O|
  -----
2 O|X|
  -----
3  | |X
Le joueur 1 a gagné !
```

Je n'ai volontairement introduit aucun test d'erreur de saisie mais il sera forcément important d'aborder le cas (il faut inculquer les bonnes pratiques dès le plus jeune âge) :

```
$ python3 tictactoe.py
  1 2 3
1  | |
  -----
2  | |
  -----
3  | |
Joueur 1:4 4
Traceback (most recent call last):
  File "tictactoe.py", line 101, in <module>
    demarrerJeu()
  File "tictactoe.py", line 82, in demarrerJeu
    valide = estLibre(grille, ligne - 1, colonne - 1)
  File "tictactoe.py", line 22, in estLibre
    return grille[ligne][colonne] == 0
IndexError: list index out of range
```

Il faut donc extraire la partie du code concernant la saisie pour créer une fonction permettant de vérifier que ladite saisie est bien correcte :

```
...
071: def saisie_joueur(joueur):
072:     valide = False
073:     while not valide:
074:         try:
075:             ligne, colonne = map(int, input('Joueur {}:'.
format(joueur + 1)).split(' '))
076:             valide = True
077:         except ValueError:
078:             print('Saisie non valide, veuillez recommencer')
079:     return (ligne, colonne)
...
082: def demarrerJeu():
...
088:     while not gagnant and tours < 9:
089:         affiche(grille)
090:         valide = False
091:         while not valide:
092:             ligne, colonne = saisie_joueur(joueur)
...
```

Il faut également revenir sur la définition de **`estLibre()`** pour ne plus accepter de coordonnées extérieures à la grille :

```
...
021: def estLibre(grille, ligne, colonne):
022:     if ligne < 0 or ligne > 2 or colonne < 0 or colonne > 2:
023:         return False
024:     else:
025:         return grille[ligne][colonne] == 0
...
```

Arriver à amener un élève de troisième jusqu'ici serait déjà un exploit (je ne parle pas d'un passionné mais d'un élève « moyen »). Lorsque l'on parle de développement d'un jeu de Tic Tac Toe, on ne peut pas se satisfaire d'une simple vérification de validité des coups et des conditions de victoire. Dans un contexte où le B.O. nous parle de « notions d'algorithme », nous pensons forcément au développement d'une intelligence artificielle. Sur un tel jeu, ça ne doit pas être bien complexe...

3 IA avec algorithme naïf

La première solution pour permettre à l'ordinateur de jouer est de lui faire poser un pion au hasard sur la grille. Cette solution n'est pas satisfaisante mais permet d'introduire progressivement le jeu de l'ordinateur. Ne prenons pas les élèves pour des idiots, notre algorithme ne sera pas totalement naïf et empêchera la formation d'une ligne dès qu'il détectera que deux pions de l'adversaire sont alignés. Voici les différentes étapes des modifications que nous allons apporter :

- tirage au sort du joueur qui débutera (pour ne pas défavoriser ou favoriser systématiquement l'ordinateur) ;
- alternance de jeu entre humain et ordinateur et jeu de l'ordinateur sans réflexion ;
- jeu de l'ordinateur en empêchant la formation de lignes.

Le découpage permet de tester chaque étape au fur et à mesure et de s'assurer ainsi que l'élève ne va pas tenter de débugger son programme uniquement à la fin (comme certains développeurs adultes peuvent encore le faire).

3.1 Tirage au sort du joueur débutant la partie

Nous considérerons que le joueur humain sera toujours authentifié par le chiffre **0** et donc l'ordinateur par le chiffre **1**.

```
001: import random
...
087: def demarrerJeu():
...
091:     joueur = random.randint(0, 1)
...
```

Il est possible d'ajouter éventuellement une petite phrase pour indiquer qui de l'humain ou de l'ordinateur a gagné le tirage au sort... mais le joueur devrait vite s'en rendre compte.

3.2 Jeu sans réflexion

L'alternance des joueurs humain et ordinateur va se faire dans la boucle de jeu de la fonction **demarrerJeu()** :

```
...
099: def demarrerJeu():
100:     grille = [[0] * 3 for x in range(3)]
101:     tours = 0
102:     gagnant = False
103:     joueur = random.randint(0, 1)
104:
105:     while not gagnant and tours < 9:
106:         if joueur == 0:
107:             affiche(grille)
108:         valide = False
109:         while not valide:
110:             if joueur == 0:
111:                 ligne, colonne = saisie_joueur(joueur)
112:                 ligne = ligne - 1
113:                 colonne = colonne - 1
114:                 valide = estLibre(grille, ligne, colonne )
115:                 if not valide:
116:                     print('La case ({}, {}) n\'est pas libre
!'.format(ligne + 1, colonne + 1))
117:             else:
118:                 ligne, colonne = saisie_ordinateur(grille)
119:                 valide = True
120:         grille[ligne][colonne] = joueur + 1
121:         gagnant = estGagnant(grille, ligne, colonne)
122:         joueur = (joueur + 1) % 2
123:         tours += 1
...
```

Si le joueur est humain (joueur **0**) et qu'il commence à jouer en premier, alors on affiche la grille vide (lignes 106 et 107). De même, pour la saisie du coup, si le joueur est humain nous conservons notre traitement précédent (lignes 110 à 116) en modifiant les valeurs de **ligne** et **colonne** (lignes 112 et 113) de manière à ce qu'elles correspondent aux indices de la liste ce qui rendra le traitement homogène puisque l'ordinateur voit directement les indices de la liste et non une grille indicée de 1 à 3.

Note

Les affectations des lignes 112 et 113 auraient pu être écrites de manières différentes mais moins lisibles. Par exemple :

```
ligne, colonne  = ligne - 1, colonne -1
```

Ou encore :

```
ligne -= 1
colonne -= 1
```

Le jeu de l'ordinateur est déterminé par la fonction **saisie_ordinateur()** de la ligne 118. Cette fonction va créer une liste des cases libres (en utilisant une fonction **casesLibres()**) puis tirer aléatoirement une des cases de cette liste :

```
...
086: def casesLibres(grille):
087:     liste = []
088:     for ligne in range(3):
089:         for colonne in range(3):
090:             if grille[ligne][colonne] == 0:
091:                 liste.append((ligne, colonne))
092:     return liste
093:
094: def saisie_ordinateur(grille):
095:     cases_possibles = casesLibres(grille)
096:     ligne, colonne = cases_possibles[random.randint(0,
len(cases_possibles) - 1)]
097:     return (ligne, colonne)
098:
...
```

Pour déterminer quelles sont les cases libres (fonction **casesLibres()** des lignes 86 à 92), il faut créer une liste vide (ligne 87) qui sera complétée à l'aide de la méthode **append()** par les cases libres (valeur **0**) en parcourant la liste **grille**. Dans la fonction **saisie_ordinateur()** il n'y a plus qu'à « piocher » une valeur au hasard dans la liste résultante.

Note

La ligne 96 devrait sans doute être écrite de manière plus explicite :

```
nb_cases = len(cases_possibles)
choix = random.randint(0, nb_cases - 1)
case_choisie = cases_possibles[choix]
ligne = case_choisie[0]
colonne = case_choisie[1]
```

Lors de l'appel de **randint()**, il faut penser que **nb_cases** est la taille de la liste et qu'une liste de taille **n** a ses éléments indicés de **0** à **n - 1**.

À ce stade il est possible de jouer contre l'ordinateur :

```
$ python3 tictactoe.py
  1 2 3
1 O| |
  -----
2  | |
  -----
3  | |
Joueur 1:2 2
  1 2 3
1 O| |
  -----
2  |X|
  -----
3  | |O
Joueur 1:2 3
  1 2 3
1 O| |
  -----
2  |X|X
  -----
3 O| |O
```

Le dernier coup montre bien l'absence totale de réflexion dans le choix du coup. Je ne pense pas qu'un élève doué d'un minimum de sens critique puisse se satisfaire de cette solution qui présente certainement des difficultés mais n'est pas du tout satisfaisante.

3.3 Jeu en empêchant la formation de lignes

Pour que l'ordinateur empêche le joueur de constituer trop facilement des lignes, il faut modifier la fonction **saisie_ordinateur()** de manière à vérifier qu'aucune ligne n'est en cours de formation (une case libre et deux cases occupées par l'adversaire sur une ligne, une colonne ou une diagonale) :

```
...
094: def saisie_ordinateur(grille):
095:     ligne, colonne = danger(grille)
096:     if ligne is None:
097:         cases_possibles = casesLibres(grille)
098:         ligne, colonne = cases_possibles[random.randint(0,
len(cases_possibles) - 1)]
099:     return (ligne, colonne)
...
```

La fonction **danger()** va renvoyer les coordonnées d'une case où il faut jouer pour éviter la formation d'une ligne. Si aucune case n'est « dangereuse », alors la fonction renverra le tuple **(None, None)**. L'idée est de détecter l'ensemble des cases vides et pour chacune d'elle de vérifier si l'adversaire peut gagner en positionnant un pion sur cette case. Pour cela il va falloir légèrement modifier les fonctions de test **estGagnantLigne()**, **estGagnantColonne()**, etc. de manière à ce qu'elles acceptent comme paramètre le type de pion du joueur que l'on souhaite tester. Cela va entraîner également quelques modifications dans les fonctions qui appelaient ces fonctions de test :

```
...
029: def estGagnantLigne(grille, ligne, colonne, typePion):
030:     for col in set(range(3)) - {colonne}:
031:         if grille[ligne][col] != typePion:
032:             return False
033:     return True
034:
035: def estGagnantColonne(grille, ligne, colonne, typePion):
036:     for line in set(range(3)) - {ligne}:
037:         if grille[line][colonne] != typePion:
038:             return False
039:     return True
040:
041: def estGagnantDiago_1(grille, ligne, colonne, typePion):
042:     for pos in set(range(3)) - {colonne}:
043:         if grille[pos][pos] != typePion:
044:             return False
045:     return True
046:
047: def estGagnantDiago_2(grille, ligne, colonne, typePion):
048:     cases = {(0, 2), (1, 1), (2, 0)} - {(ligne, colonne)}
```

```
049:     for ligne, colonne in cases:
050:         if grille[ligne][colonne] != typePion:
051:             return False
052:     return True
053:
054: def estGagnant(grille, ligne, colonne):
055:     if estGagnantLigne(grille, ligne, colonne, grille[ligne]
[colonne]):
056:         return True
057:
058:     if estGagnantColonne(grille, ligne, colonne,
grille[ligne][colonne]):
059:         return True
060:
061:     if ligne == colonne:
062:         if ligne == 1:
063:             return estGagnantDiago_1(grille, ligne, colonne,
grille[ligne][colonne]) or \
064:                    estGagnantDiago_2(grille, ligne, colonne,
grille[ligne][colonne])
065:         else:
066:             return estGagnantDiago_1(grille, ligne, colonne,
grille[ligne][colonne])
067:     elif (ligne == 0 and colonne == 2) or (ligne == 2 and
colonne == 0):
068:         return estGagnantDiago_2(grille, ligne, colonne,
grille[ligne][colonne])
069:     else:
070:         return False
...
090: def danger(grille):
091:     cases_libres = casesLibres(grille)
092:     for ligne, colonne in cases_libres:
093:         if estGagnantLigne(grille, ligne, colonne, 1):
094:             return (ligne, colonne)
095:         if estGagnantColonne(grille, ligne, colonne, 1):
096:             return (ligne, colonne)
097:         if (ligne, colonne) in ((0, 0), (1, 1), (2, 2)):
098:             if estGagnantDiago_1(grille, ligne, colonne, 1):
099:                 return (ligne, colonne)
100:         if (ligne, colonne) in ((0, 2), (1, 1), (2, 0)):
101:             if estGagnantDiago_2(grille, ligne, colonne, 1):
102:                 return (ligne, colonne)
103:
104:     return (None, None)
...
```

Les lignes 29 à 70 présentent les modifications apportées sur les fonctions de test. Puisqu'il fallait les modifier, j'en ai profité pour améliorer un peu leur comportement en ne testant que les cases autres que la case (ligne, colonne) passée en paramètre. Pour cela, il faut introduire les opérations sur les ensembles. D'un point de vue technique, **`set()`** permet de créer un ensemble à partir des éléments qui lui sont fournis en paramètre sous la forme d'un tuple. Ainsi, **`set(range(3))`** correspond à l'ensemble **`{1, 2, 3}`**.

Notez au passage que l'usage des accolades pour créer des ensembles est autorisé sauf pour l'ensemble vide (**`{}`** est un dictionnaire vide). En utilisant des opérations ensemblistes, on peut donc facilement limiter l'espace de recherche aux cases inconnues (lignes 30, 36, 42 et 48).

La fonction **`danger()`** se contente de récupérer la liste des cases vides en ligne 91 puis de tester chacune de ces cases pour savoir si le joueur adverse peut l'emporter en y plaçant un pion.

Note

Cet algorithme est un petit peu plus intelligent que l'algorithme consistant à déterminer aléatoirement où poser un pion, mais il peut être amélioré : l'objectif de l'ordinateur est d'empêcher son adverse de faire une ligne… y compris s'il a lui-même la possibilité de faire une ligne et de gagner la partie ! La figure 3 illustre un cas possible où l'ordinateur se focalisera sur le fait de bloquer son adversaire plutôt que de faire une ligne.

Il peut être proposé en exercice d'améliorer encore l'algorithme pour que l'ordinateur :

1. cherche à faire une ligne ;
2. bloque son adversaire.

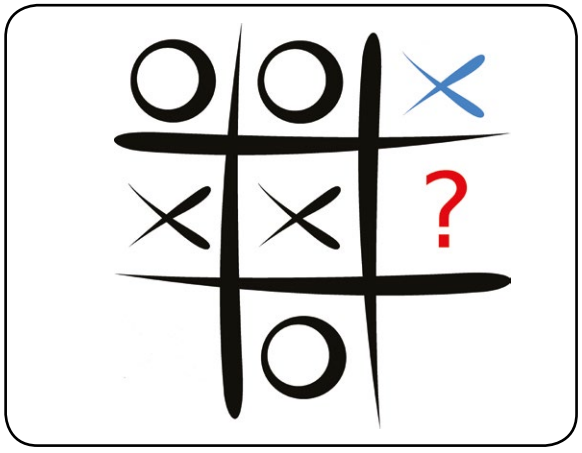


Fig. 3 : Exemple de partie où l'ordinateur bloque son adversaire (pion bleu) plutôt que de gagner la partie.

Voilà à mon avis jusqu'où il faut au minimum aller si l'on veut traiter le jeu de morpion avec des collégiens. Ça ne sera pas évident pour tous les élèves mais cela représente un juste milieu de traitement du problème. La difficulté sera variable en fonction des centres d'intérêts des élèves et de

leur niveau de compréhension des mathématiques. Aller au bout du problème, c'est-à-dire employer le min-max, est complètement utopique !

4 IA avec algorithme min-max

Le principe du min-max est de calculer l'arbre de toutes les combinaisons possibles à partir d'une situation de jeu donnée et de sélectionner le « meilleur » coup calculé à l'aide d'une fonction de score (d'où la notion relative de « meilleur » coup qui peut varier suivant la fonction de score choisie : un mauvais choix et l'algorithme ne sont plus efficaces). Dans le cas du morpion cette fonction (encore appelée fonction d'évaluation) n'est pas très compliquée à construire car il n'y a pas beaucoup de possibilités : si l'ordinateur gagne, on lui attribue un score positif (par exemple **+1000**), s'il perd le score sera négatif (**-1000**) et en cas de match nul il n'y aura pas de point (**0**).

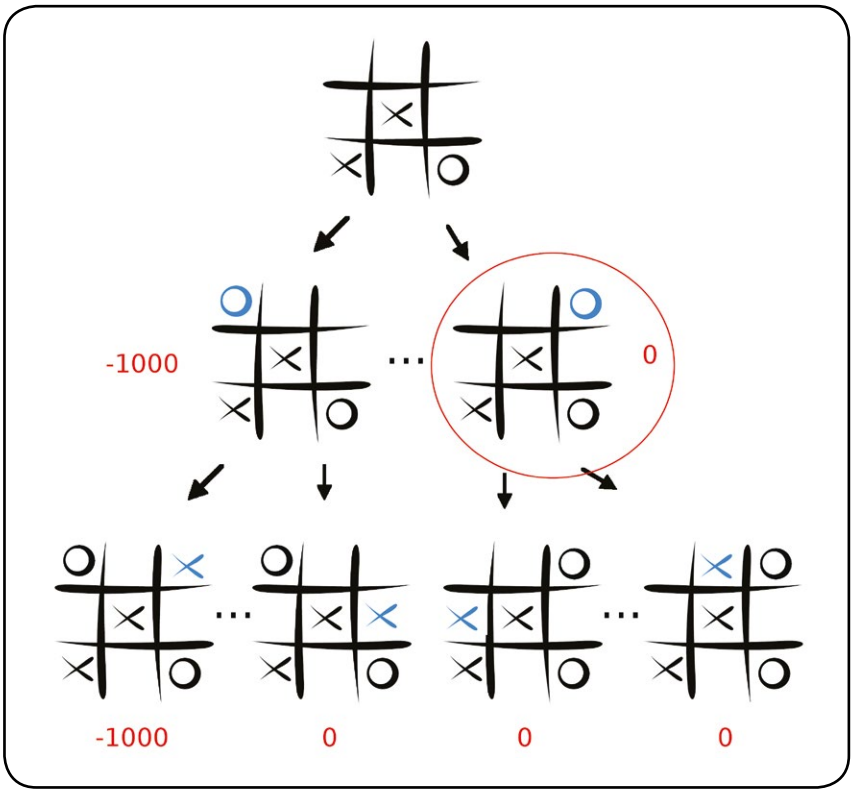


Fig. 4 : Quelques configurations de jeux possibles et scores associés dans un arbre de profondeur 2. L'ordinateur joue avec les ronds et le joueur humain avec les croix. À la profondeur 1 deux configurations sont représentées sur les six possibles et à la profondeur 2 quatre configurations sont représentées sur les trente possibles. Les scores sont indiqués en rouge (les scores à la profondeur 1 sont « remontés » depuis la profondeur 2 en minimisant le score puisque c'est au joueur humain de jouer).

Comme le min-max calcule toutes les combinaisons possibles, il est possible de limiter la profondeur de l'arbre de recherche (pour les jeux complexes tels que les échecs car pour le jeu de morpion on peut raisonnablement calculer toutes les combinaisons). La figure 4 montre un exemple d'arbre à partir d'une situation de jeu donnée. On suppose que l'adversaire jouera toujours le meilleur coup possible pour lui (minimisant la fonction de score) donc les scores obtenus sur les feuilles de l'arbre (configuration de jeu finale pour la profondeur fixée) seront associés aux nœuds parents et l'ordinateur choisira le score qui lui est le plus favorable.

On peut affiner la fonction de score en favorisant les lignes, colonnes et diagonales où se trouvent deux pions d'un même joueur (et une case libre) ou un pion d'un joueur (et deux cases libres). Ainsi, pour notre fonction de score nous pourrions avoir :

```
Si la partie est terminée Alors
    Si égalité Alors retourner 0
    Si l'ordinateur gagne Alors retourner +1000
    Si l'humain gagne Alors retourner -1000
Sinon
    score <- 0
    Pour chaque ligne, colonne et diagonale Faire
        S'il y a deux pions de l'ordinateur et une case vide Alors
            score <- score + 50
        S'il y a deux pions du joueur humain et une case vide Alors
            score <- score - 50
        S'il y a un pion de l'ordinateur et deux cases vides Alors
            score <- score + 20
        S'il y a un pion du joueur humain et deux cases vides Alors
            score <- score - 20
FinSi
Retourner score
```

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

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

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

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

Note

On peut encore améliorer la fonction de score en privilégiant une victoire rapide et donc en pondérant la valeur en cas de victoire (ou de défaite) par le nombre de pions. On peut ainsi favoriser une victoire de l'ordinateur avec un minimum de coups possibles.

Comment allons-nous donc déterminer sur quelle case l'ordinateur va poser son pion ? Pour ne pas casser la structure de notre programme de départ, la fonction **saisie_ordinateur()** va appeler une fonction **minmax()** qui prendra en paramètres la grille de jeu et le joueur devant jouer (c'est une fonction récursive et il faudra donc l'appeler pour l'ordinateur et pour le joueur humain). Elle renverra les coordonnées de la case sur laquelle l'ordinateur doit jouer. Avant cela il faudra bien sûr être capable de déterminer un score :

```
001: import random
002: import copy
...
091: def calculeBonus(humain, ordinateur, vide):
092:     score = 0
093:     if ordinateur == 2 and vide == 1:
094:         score += 50
095:     if humain == 2 and vide == 1:
096:         score -= 50
097:     if ordinateur == 1 and vide == 2:
098:         score += 20
099:     if humain == 1 and vide == 2:
100:         score -= 20
101:     return score
102:
103: def decompte(liste):
104:     humain = 0
105:     ordinateur = 0
106:     vide = 0
107:     for elt in liste:
108:         if elt == 1:
109:             humain += 1
110:         if elt == 2:
111:             ordinateur += 1
112:         if elt ==0:
113:             vide += 1
114:     return (humain, ordinateur, vide)
...
```

La fonction **calculeBonus()** prend en paramètres le nombre de pions du joueur humain, de l'ordinateur et le nombre de cases vides sur une ligne, colonne ou diagonale. Elle renvoie un score associé à la possibilité de compléter la ligne comme expliqué précédemment.

La fonction **decompte()** prend en paramètre une liste représentant une ligne de la grille et compte le nombre de pions de chaque joueur ainsi que les cases vides. Elle retourne un tuple d'entiers.

```
...
115:
116: def score(grille, ligne, colonne):
117:     if estGagnant(grille, ligne, colonne):
118:         if grille[ligne][colonne] == 1:
119:             return -1000
120:         else:
121:             return 1000
122:
123:     for ligne in range(3):
124:         if 0 in grille[ligne]:
125:             break
126:     else:
127:         return 0
128:
129:     score = 0
130:     for ligne in range(3):
131:         humain, ordinateur, vide = decompte(grille[ligne])
132:         score += calculeBonus(humain, ordinateur, vide)
133:
134:     for colonne in range(3):
135:         humain, ordinateur, vide = decompte([grille[0]
[colonne], grille[1][colonne], grille[2][colonne]])
136:         score += calculeBonus(humain, ordinateur, vide)
137:
138:     humain, ordinateur, vide = decompte([grille[0][0],
grille[1][1], grille[2][2]])
139:     score += calculeBonus(humain, ordinateur, vide)
140:
141:     humain, ordinateur, vide = decompte([grille[2][0],
grille[1][1], grille[0][2]])
142:     score += calculeBonus(humain, ordinateur, vide)
143:
144:     return score
...
```

La fonction de score utilise les deux fonctions précédentes pour implémenter l'algorithme de calcul du score présenté précédemment.

```
...
145:
146: def adversaire(joueur):
147:     if joueur == 1:
148:         return 2
149:     else:
150:         return 1
...
```

Pour obtenir rapidement l'identifiant d'un adversaire, j'ai ajouté la fonction **adversaire()**.

À l'aide de toutes ces fonctions nous pouvons maintenant aborder le min-max :

```
...
151:
152: def calculeMinMax(grille, ligne, colonne, profondeur,
joueur, minmax):
153:     if profondeur == 0 or estGagnant(grille, ligne,
colonne):
154:         return score(grille, ligne, colonne)
155:     else:
156:         cases_libres = casesLibres(grille)
157:         if cases_libres == []:
158:             return score(grille, ligne, colonne)
159:         if minmax == 'max':
160:             limite = -32767
161:         else:
162:             limite = 32767
163:         for ligne, colonne in cases_libres:
164:             grille_copie = copy.deepcopy(grille)
165:             grille_copie[ligne][colonne] = joueur
166:             if minmax == 'max':
167:                 valeur_fils = calculeMinMax(grille_copie,
ligne, colonne, profondeur - 1, adversaire(joueur), 'min')
168:                 limite = max(limite, valeur_fils)
169:             else:
170:                 valeur_fils = calculeMinMax(grille_copie,
ligne, colonne, profondeur - 1, adversaire(joueur), 'max')
171:                 limite = min(limite, valeur_fils)
172:
173:         return limite
174:
175: def minmax(grille, profondeur, joueur=2):
176:     cases_libres = casesLibres(grille)
177:     maximum = -32767
178:     for ligne, colonne in cases_libres:
179:         grille_copie = copy.deepcopy(grille)
180:         grille_copie[ligne][colonne] = joueur
181:         valeur_max = calculeMinMax(grille_copie, ligne,
colonne, profondeur - 1, adversaire(joueur), 'min')
182:         if valeur_max > maximum:
183:             maximum = valeur_max
184:             case = (ligne, colonne)
185:
186:     return case
187:
188: def saisie_ordinateur(grille):
189:     return minmax(grille, 6)
...
```

L'appel au **minmax()** en ligne 189 se fait avec une profondeur de **6** (la profondeur maximale pour le jeu de morpion). Il est possible d'introduire ici un réglage du niveau de difficulté de l'ordinateur en diminuant la profondeur.

Les fonctions **minmax()** et **calculeMinMax()** respectent ensuite l'algorithme décrit précédemment : l'ordinateur teste chaque case libre en positionnant un pion (lignes 178 à 180) puis il essaye de minimiser le score du coup suivant de son adversaire (ligne 181). L'alternance de coups ordinateur/adversaire qui se traduit par une maximisation/minimisation du score est réalisée par les appels récursifs à **calculeMinMax()** à qui on indique en dernier paramètre si l'on souhaite maximiser ou minimiser le résultat (chaîne de caractères **'max'** ou **'min'**). Les conditions d'arrêt de la fonction (pour retourner le score) sont l'atteinte de la profondeur maximale demandée, la victoire de l'un ou l'autre joueur (lignes 153 et 154) ou un match nul (lignes 157 et 158).

Pour déterminer la valeur minimale ou maximale à renvoyer, j'initialise la variable **limite** à **32767** ou **-32767** (lignes 160 et 162). Ces valeurs sont des « vestiges » des valeurs limites du **short int** du C car en Python3 tous les entiers sont des entiers longs. Il faut simplement initialiser **limite** avec des valeurs suffisamment grandes et petites pour qu'elles soient remplacées lors des tests avec **min()** ou **max()** (lignes 168 et 171).

Attention !

La copie de la liste est effectuée grâce à la fonction **deepcopy()** du module **copy**. En effet, la copie d'une liste **l** par **new = list(l)** ou **new = l[:]** est plus rapide mais ne copie que les éléments de premier niveau (fonctionne donc pour une liste du type **l = [1, 2, 3]**). En présence de sous-listes, ce sont des références qui sont faites vers ces listes et donc modifier une liste modifie l'autre...

```
> l = [[1, 2, 3], [4, 5, 6]]
> l2 = l[:]
> l2[0][0] = 12
> l2
[[12, 2, 3], [4, 5, 6]]
> l
[[12, 2, 3], [4, 5, 6]]
> import copy
> l3 = copy.deepcopy(l2)
> l3[0][0] = 'l3'
> l3
[['l3', 2, 3], [4, 5, 6]]
> l2
[[12, 2, 3], [4, 5, 6]]
```

J'ai profité de l'écriture de la fonction **adversaire()** pour modifier la fonction **demarrerJeu()** et supprimer la présence du modulo pour passer d'un joueur à l'autre :

```
...
191: def demarrerJeu():
192:     grille = [[0] * 3 for x in range(3)]
193:     tours = 0
194:     gagnant = False
195:     joueur = random.randint(1, 2)
196:
197:     while not gagnant and tours < 9:
198:         if joueur == 1:
```

```
199:             affiche(grille)
200:         valide = False
201:         while not valide:
202:             if joueur == 1:
203:                 ligne, colonne = saisie_joueur(joueur)
204:                 ligne = ligne - 1
205:                 colonne = colonne - 1
206:                 valide = estLibre(grille, ligne, colonne )
207:                 if not valide:
208:                     print('La case ({}, {}) n\'est pas libre
!'.format(ligne + 1, colonne + 1))
209:             else:
210:                 ligne, colonne = saisie_ordinateur(grille)
211:                 valide = True
212:         grille[ligne][colonne] = joueur
213:         gagnant = estGagnant(grille, ligne, colonne)
214:         joueur = adversaire(joueur)
215:         tours += 1
216:
217:     affiche(grille)
218:
219:     if gagnant:
220:         joueur = adversaire(joueur)
221:         print('Le joueur {} a gagné !'.format(joueur))
222:     else:
223:         print('Match nul')
...
```

Note

Le fait de ne pas minimiser le nombre de coups dans notre fonction de score conduit l'ordinateur dans certaines situations à « narguer » son adversaire en ne l'achevant pas tout de suite comme le montre la figure 5. Cela ne modifie pas la validité de l'algorithme puisque l'ordinateur se trouvera en situation de gagner quoiqu'il arrive.

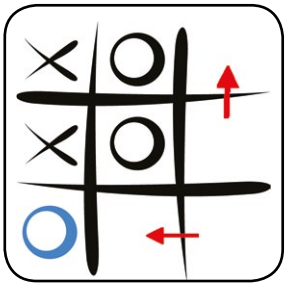


Fig. 5 : Configuration dans laquelle l'ordinateur (qui joue avec les ronds) ne complète pas immédiatement sa colonne. Les flèches rouges montrent les cases qui assureront la victoire de l'ordinateur lorsque ce sera de nouveau à lui de jouer.

Conclusion

Le problème du Tic Tac Toe est un problème intéressant. Le restreindre à l'utilisation d'un algorithme naïf jouant de manière complètement aléatoire n'est pas satisfaisant mais aller jusqu'au minmax peut se montrer problématique pour de nombreux collégiens. Il faut donc trouver un juste milieu permettant de répondre à plusieurs contraintes : l'intérêt de l'élève, l'intérêt algorithmique, le degré de difficulté et... le temps mis à disposition pour l'enseignement de « l'algorithmique et de la programmation », dilué dans les cours de mathématiques.

Cette modification du programme du cycle 4 nous aura permis de nous rafraîchir la mémoire avec un algorithme classique d'intelligence artificielle même si pour les enseignants et les élèves cela ressemble plus à un effet d'annonce... ■

Note

Si vous avez un enfant au collège, ne vous attendez pas à ce qu'il fasse du Python ! Il a été décidé en haut lieu que ce serait du **Scratch**, ce « langage » purement graphique, basé sur Flash (et oui, ça existe encore!), dont vous pouvez voir un exemple en figure 6.

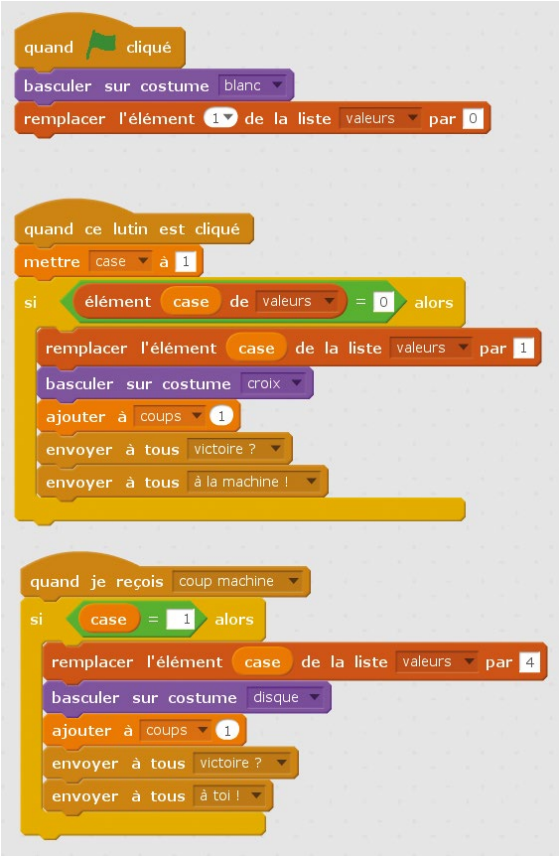


Fig. 6 : Morceau de code Scratch d'un programme de jeu de Morpion (d'après le site Eduscol http://eduscol.education.fr/cid99696/ressources-maths-cycle.html).

FABRIQUER UN CORPS FINI

Nicolas PATROIS [Enseignant dans le secondaire]

Pour corriger un code QR avec erreur ou rature, nous avons besoin de construire le corps fini $\mathbb{F}_{256}$ et son anneau de polynômes.

Mots-clés : Python, Programmation orientée objet, Corps finis, Polynôme, Algorithme

Résumé

La correction d'erreur de Reed-Solomon travaille dans l'anneau des polynômes à coefficients dans $\mathbb{F}_{256}$. Cet article présente une construction de ces deux structures emboîtées en Python orienté objet.

Dans l'article précédent **[1]**, nous avons présenté le minimum mathématique pour une bonne compréhension de ce qui suit.

1 Construction du corps $\mathbb{F}_{256}$

Le corps fini utilisé n'est pas un corps du type $\mathbb{Z}/p\mathbb{Z}$ avec `p` premier, il s'agit, d'un point de vue ensembliste, de $(\mathbb{Z}/2\mathbb{Z})^8$ qui a 256 éléments. Son corps premier est bien $\mathbb{Z}/2\mathbb{Z}=\{0,1\}$, on travaille avec un ordinateur après tout et ses éléments sont modélisables comme des octets. Attention cependant, les opérations effectuées ne sont PAS celles de $\mathbb{Z}/256\mathbb{Z}$ qui n'est PAS un corps, on ne peut donc pas utiliser sans précaution les nombres entiers de `0` à `255`, il faut malgré tout mettre un peu les mains dans le cambouis.

Mathématiquement, on a besoin de quatre structures emboîtées :

- Le corps premier $\mathbb{Z}/2\mathbb{Z}=\mathbb{F}_2$ dont on a vu une construction possible **[1]** (mais `[False,True]` peut très bien convenir) ;
- L'anneau $\mathbb{F}_2$`[X]` des polynômes à coefficients dans $\mathbb{F}_2$;
- Le corps $\mathbb{F}_{256}$ qui est mathématiquement le quotient de $\mathbb{F}_2$`[X]` par un polynôme irréductible dans le corps $\mathbb{F}_2$ (par l'idéal engendré par ce polynôme si on veut être précis), notre polynôme annulateur est $X^8+X^4+X^3+X^2+1$ (si on l'évalue en `X=2`, on obtient `285`, voir plus loin) ;
- Les polynômes à coefficients dans $\mathbb{F}_{256}$ utilisés dans la correction d'erreurs notées $\mathbb{F}_{256}$`[X]`.

Nous avons donc techniquement besoin de polynômes de polynômes. Vous suivez ? Pas trop, alors on va simplifier la construction et ne pas emboîter trop de structures les unes dans les autres, les deux dernières suffiront.

J'ai donc codé dans `qrcorps.py` (disponible dans le GitHub du magazine) un moyen terme entre la structure mathématique abstraite et générale (plus propre mathématiquement mais pénible à utiliser) et celle de **Wikiversity [2]** (qualités et défauts inverses) : je code directement le corps $\mathbb{F}_{256}$ sans passer par $\mathbb{F}_2$ (comme Wikiversity) tout en conservant la création de classes d'éléments du corps et de polynômes (comme Kun **[3]**). Cela me facilitera la correction des éventuelles erreurs tout en conservant la souplesse de la surcharge des opérations. J'espère que ce ne sera ni moche ni pénible.

```
01: #!/usr/bin/python3
02:
03: from qrcodeoutils import *
```

Le code a besoin de quelques fonctions purement mathématiques comme les algorithmes d'Euclide.

2 La classe F256

2.1 Préliminaires

La classe est mémorisée (voir section précédente), on commence par le constructeur :

```
05: annulateur=285
06:
07: @memorise
08: class F256():
09:   def __init__(self,s):
10:     if isinstance(s,str) or hasattr(s,'__iter__') or hasattr(s,'iter'):
11:       self.val=bin2dec(s)
12:     elif type(s) is F256:
13:       self.val=s.val
14:     else:
15:       self.val=s
16:     while True:
17:       ch=bin(self.val)[2:]
18:       if len(ch)<=8:
19:         break
20:       self.val^=annulateur*2**(9-len(ch))
21:     self.corps=F256
```

Une instance de la classe $\mathbb{F}_{256}$ contient deux attributs :

- **val** est un entier entre **0** et **255** par commodité mais on utilise en fait sa décomposition en base **2**, autrement dit sa représentation binaire ;
- **corps** est le nom de notre corps, ce qui permet de ne pas ajouter deux éléments de corps différents (on n'est pas concerné ici mais cela peut empêcher une mauvaise surprise, voir **[1]** pour l'explication de **@memorise**).

Selon le type de l'élément qu'on veut convertir en élément de $\mathbb{F}_{256}$, si c'est :

- une chaîne de caractères ou un tuple qui ne contient que des **0** et des **1**, on convertit l'écriture binaire en un entier ;
- un membre de $\mathbb{F}_{256}$, on recopie la valeur dans l'attribut **val**,
- un entier, de même.

Ensuite, si **val** est supérieur ou égal à **256** on effectue sa division par **285** (c'est-à-dire 100011101_2 ou $11d_{16}$) tant qu'elle est trop grande. Par division, j'entends soustraction ou addition bit à bit, c'est-à-dire un XOR (le shérif de l'espace). Par exemple, **bin(28)** retourne **"0b11100"** voilà pourquoi on ne conserve pas les deux premiers caractères. De même, **hex(28)** retourne **"0x1c"**.

```
22:   def __str__(self):
23:     nb=hex(self.val)[2:]
24:     return nb
25:   def __repr__(self):
26:     return str(self)
```

Quand on veut utiliser **print** ou **str**, on utilise la méthode **__str__**. Quand on affiche une liste qui contient des instances de **F256**, on utilise la méthode **__repr__**. On retourne sa valeur hexadécimale.

2.2. Les opérations

```
27:   def __add__(self,n):
28:     return F256(self.val^n.val)
29:   def __sub__(self,n):
30:     return self+n
31:   def __neg__(self):
32:     return self
```

L'addition est en fait un XOR car le corps $\mathbb{F}_{256}$ est un espace vectoriel de dimension **8** sur le corps $\mathbb{F}_2$, où l'addition a cette table :

+	0	1
0	0	1
1	1	0

En effet, **1+1=2=0** modulo **2**. Comme on ajoute des vecteurs à **8** coordonnées à valeurs dans $\mathbb{F}_2$, on utilise un simple XOR entre les deux valeurs.

Par exemple, si on veut ajouter les vecteurs de coordonnées respectives **(1,0,1,1,0,1,0,1)** et **(0,1,1,1,1,0,1,1)** autrement dit **181** et **123** en représentation décimale, on ajoute coordonnée à coordonnée, ce qui donne **(1,1,0,0,1,1,1,0)** soit **206**. On a déjà utilisé cette technique quand on a démasqué.

Pour la même raison (la caractéristique de $\mathbb{F}_{256}$ vaut **2**), $\forall a,b \in \mathbb{F}_{256}$, **a−b=a+b** et **−a=a**.

```
33:   def __mul__(self,n):
34:     if self.val==0 or n.val==0:
35:       return F256(0)
36:     return F256(F256.exp((self.log()+n.log())))
```

La multiplication utilise une table pré-calculée de puissances d'un élément primitif de $\mathbb{F}_{256}$. Un tel élément existe et engendre complètement $\mathbb{F}_{256}$ à partir de ses puissances (à part bien sûr **0**). On utilise la propriété bien connue d'additivité des exposants : $\forall a \in \mathbb{F}^*_{256}$, $\forall n,m \in \mathbb{Z}$, $a^{m+n}=a^m \times a^n$. Ici, **a** est notre élément primitif et on additionne les deux exposants des puissances de **a** égales à nos deux éléments. Les exposants sont déterminés à partir d'un logarithme discret, pré-calculé un peu plus bas.

Bien entendu, si un des multiplicandes est nul, le produit est nul. Je n'ai pas codé les opérations externes (avec un entier, par exemple **3a=a+a+a**) parce que je n'en ai pas besoin.

```
38:   def __truediv__(self,n):
39:     if n.val==0:
40:       raise ZeroDivisionError
41:     if self.val==0:
42:       return F256(0)
43:     return F256(F256.exp((self.log()-n.log())))
```

La division utilise la même propriété : $\forall a \in \mathbb{F}^*_{256}$, $\forall n,m \in \mathbb{Z}$, $a^{m-n}=\frac{a^m}{a^n}$. On teste d'abord si le diviseur est nul (division par zéro), si le dividende est nul (quotient nul), sinon on utilise la propriété.

```
44:   def __pow__(self,e):
45:     if self.val==0:
46:       if e<0:
47:         raise ZeroDivisionError
48:       elif e==0:
49:         return F256(1)
50:       else:
51:         return F256(0)
52:     return F256(F256.exp((e*self.log())))
```

Pour calculer une puissance, on teste d'abord si le nombre est nul. Dans ce cas, on vérifie si l'exposant est négatif, nul ou positif. Sinon, on utilise la propriété suivante : $\forall a \in \mathbb{F}_{256}$, $\forall n,m \in \mathbb{Z}$, $a^{nm}=(a^n)^m$.

```
53:   def log(self):
54:     if self.val==0:
55:       raise ZeroDivisionError
56:     return F256.log[self.val]
```

Le logarithme discret est pré-calculé, cela évite de le recalculer à chaque fois. En effet, sa détermination par un algorithme meilleur que la recherche exhaustive n'est pas résolue pour le moment. La méthode est en fait un attribut de classe, lequel est un dictionnaire.

```
58: F256.log=dict()
59: exp=[]
60: nb=1
61: for i in range(255):
62:   F256.log[nb]=i
63:   exp.append(nb)
64:   nb*=2
65:   if nb>=256:
66:     nb^=annulateur
67: F256.exp=lambda i:exp[i%255]
68: F256.__name__="F_2^8"
```

L'élément primitif générateur de notre corps fini est **2** (c'est-à-dire le polynôme **X** puisque $2_{10}=10_2$), on calcule ses puissances successives. On stocke l'exposant dans **F256.log** et la puissance dans **F256.exp**. On aurait pu écrire :

```
67: def F256.exp(i):return exp[i%255]
```

Et on définit le nom de la classe.

```
213: if __name__=="__main__":
214:   a=F256(140)
215:   print(a)
216:   b=F256("101")
217:   print(b)
218:   print(a.log())
219:   print(b+a)
220:   print(a*a*a*a*a*a)
221:   print(a**6)
222:   print(F256.log[128])
```

Exécutons ce code :

```
8c
5
49
```

Donc, selon notre écriture, $2^{49}=140$ (soit **10001100** en binaire) car X^{49} modulo $X^8+X^4+X^3+X^2+1$ vaut $X^7+X^3+X^2$ (ces polynômes sont à coefficients dans $\mathbb{F}_2$).

```
89
35
35
7
```

Et on vérifie que l'addition et la puissance sont correctement codées.

3 La classe Polynome

Pour corriger les éventuelles erreurs d'un code QR, on a besoin de polynômes à coefficients dans $\mathbb{F}_{256}$, or les polynômes forment un anneau : les règles de calcul sont celles d'un corps à ceci près que les éléments non nuls ne sont pas tenus d'être tous inversibles pour la multiplication. Par quel polynôme multiplier **X** pour obtenir **1** ?

3.1 Construction et affichage

```
68: @memorise
69: class Polynome(ElementAnneau):
```

La classe **Polynome** est une sous-classe de la classe **ElementAnneau**, présente dans **qrcodeoutils.py** :

```
class ElementAnneau(object):
  def __radd__(self,autre):
    return self+autre
  def __rsub__(self,autre):
    return -self+autre
  def __rmul__(self,autre):
    return self*autre
```

Ici, on s'assure de la commutativité de l'addition et de la multiplication, le code est dans **qrcodeoutils.py**.

Le constructeur :

```
71:   def __init__(self,c):
72:     if type(c) is Polynome:
73:       self.coefficients=tuple(c.coefficients)
74:     elif type(c) is Polynome.corps:
75:       self.coefficients=(c,)
76:     elif not hasattr(c,'__iter__') and not hasattr(c,'iter'):
77:       self.coefficients=(Polynome.corps(c),)
78:     else:
79:       self.coefficients=tuple(c)
80:     try:
81:       while self.coefficients[0]==Polynome.corps(0):
82:         self.coefficients=self.coefficients[1:]
83:     except IndexError:
84:       pass
```

Si on envoie au constructeur :

- un polynôme, il retourne le même polynôme ;
- un élément du corps fini, il retourne un polynôme de degré **0** dont le coefficient de degré **0** vaut cet élément ou le polynôme nul si l'élément est le **0** de $\mathbb{F}_{256}$;
- un tuple, il retourne un polynôme dont les coefficients sont les éléments du tuple.

Les coefficients sont rangés dans l'ordre d'écriture usuel, à savoir que les coefficients du polynôme **$3X^3+4X+1$** sont dans le tuple **(3,0,4,1)**, c'est un choix, Kun **[3]** utilise l'autre convention où **$3X^3+4X+1$** est représenté par **(1,4,0,3)**. Pourquoi un tuple ? Pour pouvoir être mémorisé dans le cache du décorateur.

Si les premiers coefficients sont nuls, on les supprime. Le polynôme nul a une liste de coefficients vide ; **(0,3,0,4,1)** et **(3,0,4,1)** donnent le même polynôme.

```
86:   def estzero(self):
87:     return self.coefficients==tuple()
```

Cette méthode retourne **True** si le polynôme est nul. Si, si.

```
89:   def __str__(self):
90:     if self.estzero():
91:       return "0"
92:     expo={"0":"⁰","1":"¹","2":"²","3":"³","4":"⁴","5":"⁵","6":"⁶","7":"⁷","8":"⁸","9":"⁹"}
93:     return "+".join([str(self.coefficients[i])+"X"+\
          "".join(expo[j] for j in str(self.degre()-i)) for i in range(len(self)) \
          if self.coefficients[i]!=Polynome.corps(0)])

95:   def __repr__(self):
96:     return str(self)
```

On affiche les coefficients non nuls, par exemple si les coefficients sont **(f1,0,4,1)**, on affiche **$f1X^3+4X^1+1X^0$** et les degrés supérieurs ou égaux à **10** sont affichés correctement, comme **$5X^{31}$**.

3.2 Propriétés des polynômes

```
098:  def __call__(self,val):
099:    res=Polynome(tuple())
100:    if type(val) is type(self):
101:      va=val
102:    else:
103:      va=Polynome(tuple([val]))
104:    for c in self:
105:      res*=va
106:      res+=Polynome(tuple([c]))
107:    if type(val) is type(self):
108:      return res
109:    if res.coefficients:
110:      return res[0]
111:    return Polynome.corps(0)
```

On peut évaluer un polynôme en un élément du corps sous-jacent comme en un autre polynôme, ce qui veut dire qu'on peut considérer un polynôme comme une fonction (même si mathématiquement, un polynôme n'est PAS une fonction). Ainsi si **P** est un polynôme, **P(4)** retourne un nombre alors que **P(X+1)** retourne un autre polynôme. On teste donc le type de l'entrée, qu'on transforme en un polynôme si c'est un scalaire (ligne 103). On utilise l'algorithme de Horner (voir **[4]**) des lignes 104 à 106 puis on retourne soit un polynôme soit un scalaire soit zéro si les coefficients sont vides.

```
112:  def __abs__(self):
113:    return len(self.coefficients)
114:  def __len__(self):
115:    return len(self.coefficients)
```

La valeur absolue est utilisée dans la division euclidienne, on l'appelle aussi *stathme euclidien* en mathématiques des anneaux euclidiens (munis d'une division euclidienne), ce que sont les anneaux de polynômes à coefficients dans un corps. La longueur d'un polynôme est la longueur de la liste de ses coefficients.

```
116:  def __iter__(self):
117:    return iter(self.coefficients)
118:  def iter(self):
119:    return self.__iter__()
120:  def __getitem__(self,i):
```

```
121:     return self.coefficients[self.degre()-i]
122:   def __setitem__(self,i,x):
123:     coef=list(self.coefficients)
124:     coef[self.degre()-i]=x
125:     self.coefficients=tuple(coef)
```

On permet d'itérer les coefficients du polynôme, d'y accéder et de les modifier directement sans passer par le constructeur.

Ainsi, on peut utiliser des codes comme **`for c in P: P[-1]=P[0]+c`** où **`P[0]`** désigne le coefficient dominant et **`P[-1]`** le coefficient de plus bas degré.

Attention !

Ne pas confondre **`P[0]`** et **`P(0)`**.

```
126:   def coefficientdominant(self):
127:     return self.coefficients[0]
128:   def degre(self):
129:     return abs(self)-1
```

Le coefficient dominant de $3X^2+4X+5$ est **3**, son degré est **2**.

```
130:   def __eq__(self,autre):
131:     return self.coefficients==autre.coefficients
```

L'égalité entre polynômes est correcte puisqu'on supprime les coefficients nuls s'ils sont au début de la liste des coefficients.

3.3 Les opérations

Je n'ai pas codé les opérations entre les éléments du corps de base et les polynômes, n'en ayant pas besoin.

```
132:   def __add__(self,autre):
133:     somme=[Polynome.corps(0)]*(max(len(self),len(autre))-
len(self))+list(self.coefficients)
134:     for i in range(len(autre)):
135:       somme[-i-1]+=autre[i]
136:     return Polynome(tuple(somme))
137:   def __xor__(self,autre):
138:     return self+autre
139:   def __neg__(self):
140:     return Polynome(tuple((-a for a in self)))
141:   def __sub__(self,autre):
142:     return self+(-autre)
```

L'addition, la soustraction, et le XOR (non utilisé ici) sont trois mêmes opérations : ajouter deux à deux les coefficients de même degré ; l'opposé ne fait rien parce qu'on est en caractéristique 2.

```
143:   def __mul__(self,autre):
144:     if self.estzero() or autre.estzero():
145:       return Polynome(tuple())
146:     prod=[Polynome.corps(0) for _ in range(len(self)+len(autre)-1)]
147:     for i in range(len(self)):
148:       for j in range(len(autre)):
149:         prod[-i-j-1]+=autre[j]*self[i]
150:     return Polynome(tuple(prod))
```

On multiplie deux polynômes en utilisant la formule classique de développement :

$$\left(\sum_{i=0}^{p} a_i X^i\right) \times \left(\sum_{j=0}^{q} b_j X^j\right) = \sum_{k=0}^{p+q} \left(\sum_{m=0}^{k} a_m b_{m-k}\right) X^k$$

Ce n'est pas la plus rapide, de loin, mais elle nous suffira.

```
151:   def __pow__(self,exp):
152:     if exp>0:
153:       prod=Polynome(tuple([Polynome.corps(1)]))
154:       x=Polynome(self.coefficients)
155:       while exp>1:
156:         if exp%2:
157:           prod*=x
158:         x*=x
159:         exp//=2
160:       return prod*x
161:     elif exp==0:
162:       return Polynome(tuple([Polynome.corps(1)]))
```

On calcule la puissance non strictement négative d'un polynôme par l'exponentiation rapide (voir **[1]**). Bien entendu, on ne sait pas calculer les puissances négatives d'un polynôme.

```
164:   def __divmod__(self,diviseur):
165:     quotient,reste=Polynome(tuple()),self
166:     degdiviseur=diviseur.degre()
167:     coefdom=diviseur.coefficientdominant()
168:     while reste.degre()>=degdiviseur:
169:       monomediviseur=Polynome(tuple([reste.coefficientdominant()/
coefdom]+[F256(0)]*(reste.degre()-degdiviseur)))
170:       quotient+=monomediviseur
171:       reste-=monomediviseur*diviseur
172:     return quotient,reste
```

Il s'agit de la division euclidienne de polynômes qui retourne le quotient et le reste de degré inférieur au diviseur, l'algorithme est le même que celui de la division à potence apprise à l'école primaire. Par exemple si on veut diviser dans **`R[X]`** X^5+3X^3+4X-1 par $X2-X+2$, le degré du quotient sera **5−2=3** et celui du reste au plus **2−1=1** :

```
 X^5+0X^4+3X^3+0X^2+4X-1 | X^2-X+2
-X^5+1X^4-2X^3           |-----------
     X^4+1X^3+0X^2+4X-1  | 1X^3+1X^2+2X
    -X^4+1X^3-2X^2       |
        2X^3-2X^2+4X-1   |
       -2X^3+2X^2-4X     |
                     -1  |
```

Donc $X^5+3X^3+4X-1=(X^2-X+2)\times(X^3+X^2+2X)-1$, le reste vaut **-1**, de degré **0≤1**. Remarquez que si le reste est nul, le dividende est factorisable par le diviseur. Le principe est le même dans $\mathbb{F}_{256}[X]$.

```
174:   def __floordiv__(self,div):
175:     q,_=divmod(self,div)
176:     return q
177:   def __mod__(self,div):
178:     _,r=divmod(self,div)
179:     return r
```

Le quotient euclidien est donné par l'opération **//**, l'opération **/** n'existe pas ici puisque notre anneau de polynômes n'est pas un corps, la deuxième opération est le modulo, donné par l'opération **%**. Dans le cas précédent $(X^5+3X^3+4X-1)//(X^2-X+2)$ retourne $1X^3+1X^2+2X$ et $(X^5+3X^3+4X-1)\%(X^2-X+2)$ retourne **-1**.

```
181:   def bezoutpoly(self,p2):
182:     r,u,v=p2,Polynome.construction([1]),Polynome.construction([0])
183:     rr,uu,vv=self,Polynome.construction([0]),Polynome.construction([1])
184:     while not rr.estzero():
185:       q=r//rr
186:       r,u,v,rr,uu,vv=rr,uu,vv,r-q*rr,u-q*uu,v-q*vv
187:     return v,u,r
```

Il s'agit de l'algorithme d'Euclide étendu qui donne les coefficients de Bezout **u** et **v** à l'équation **au+bv=d** où **d** est un PGCD de **a** et de **b**. En effet, le PGCD est défini à un inversible près et ici, les inversibles sont les polynômes de degré **0**, ceux qui n'ont qu'un coefficient non nul et qu'on assimile à $\mathbb{F}^*_{256}$ (mathématiquement, les polynômes de degré inférieur ou égal à **0** sont isomorphes $\mathbb{F}_{256}$). C'est le même algorithme que pour la recherche de l'inverse dans $\mathbb{F}_p$ (voir **[1]**) mais adapté.

```
189:   def der(self):
190:     derive=[Polynome.corps(0) for _ in self]
191:     for i in range(len(self)):
192:       if i%2==1:
193:         derive[-i-1]=self[i]
194:     return Polynome(tuple(derive[:-1]))
```

C'est la dérivation d'un polynôme à coefficients dans un corps de caractéristique **2**. En effet, si $P(X)=\sum_{i=0}^{p} a_i X^i$, $P'(X)=\sum_{i=1}^{p} a_i X^{i-1}$. Or si **i** est pair, **i** vaut **0** modulo **2** d'où le test ligne 192 et on décale bien entendu les puissances de **X**.

```
196: Polynome.corps=F256
197: Polynome.__name__="(%s)[x]"%F256.__name__
198: Polynome.construction=lambda L: Polynome(tuple(F256(x) for x in L))
```

On précise le corps utilisé, le nom de la classe et comment construire rapidement un polynôme à partir de la liste de ses coefficients.

3.4 On joue un peu

Et maintenant, vérifions que tout fonctionne bien.

```
231:   print(Polynome.__name__)
232:   poly=Polynome.construction
233:   p=poly((15,1,12))
234:   q=poly(("11","10110","0"))
235:   print(p,q)
236:   print(p+q)
237:   print(p%q)
238:   print(divmod(p,q))
239:   print(p//q*q+p%q)
240:   print(p^q)
241:   print(p*p)
242:   print(p**2)
243:   print(p(F256(1)))
244:   print(p(q))
245:   print(p(q).der())
```

La ligne 239 vérifie que la division euclidienne se passe bien, c'est-à-dire que le résultat est bien **p**, les lignes 241 et 242 que la puissance **2** est correcte, les lignes 236 et 240 que les deux additions sont identiques et les deux dernières lignes que la dérivation est correcte.

```
(0_20)[x]
fX²+1X¹+cX0 3X²+16X¹+0X0
cX²+17X¹+cX0
4fX¹+cX0
(5X0, 4fX¹+cX0)
fX²+1X¹+cX0
cX²+17X¹+cX0
55X0+1X²+50X0
55X0+1X²+50X0
2
33X0+74X²+16X¹+cX0
16X0
```

Conclusion

Nous sommes prêts à corriger un code QR pas trop abîmé !

Ouverture

On peut rendre cette bibliothèque plus propre, notamment en codant les opérations externes et en gérant mieux les transtypages... ■

Références

[1] PATROIS N., « *Expliquer un corps fini* », GNU/Linux Magazine n°195, juillet/août 2016, p. 16 à 19.

[2] Reed–Solomon codes for coders https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders.

[3] http://jeremykun.com/2014/03/13/programming-with-finite-fields/.

[4] PATROIS N., « *Décoder un code QR* », *GNU/Linux Magazine* n°194, juin 2016, p. 32 à 41.

FONCTIONNALITÉS AVANCÉES D'APTLY

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

Aptly est un gestionnaire de dépôts Debian puissant par le nombre de fonctionnalités offertes. Nous nous intéresserons aujourd'hui à ses fonctions avancées, à savoir les filtres, la fusion d'instantanés ainsi que l'API REST offerte afin de manipuler Aptly via de simples requêtes HTTP.

aptly gpg repository apt-get dépôt Debian

L'OBJECTIF

Dans un précédent article, nous avions présenté les fonctionnalités générales d'**Aptly** (http://www.aptly.info/). Nous aborderons aujourd'hui les fonctionnalités avancées que ce gestionnaire de dépôts Debian offre aux utilisateurs souhaitant rationaliser ou étendre l'utilisation qu'ils font de leurs dépôts pour leur infrastructure.

LES OUTILS

- Un système **GNU/Linux Debian** en version Jessie ;
- gestionnaire de paquets Debian **apt-get**.

PHASE 1

Utilisation des filtres

L'installation et les premiers pas dans Aptly ont été longuement décrits dans mon premier article au sujet de Aptly paru dans GLMF 186, nous continuerons donc l'article en présupposant que la commande **aptly** est installée et disponible sur le système.

Aptly offre la possibilité d'utiliser des filtres dans de nombreuses commandes de manipulation de paquets. Il peut en effet être pénible de créer le miroir d'un dépôt Debian distant, d'autant plus s'il est d'une taille importante. Gâchis de bande passante, d'espace disque et lenteur de la synchronisation avec la source sont à la clé de ce type de dépôt.

De même, si vous constituez un dépôt local, si vous souhaitez importer seulement un ensemble de paquets depuis votre source, copier ou effacer de façon sélective des ensembles de paquets, l'utilisation des filtres vous soulagera d'un grand poids.

Nous allons créer un miroir local du dépôt officiel de Debian Sid (version non-stable de Debian), mais ne contenant que quelques paquets Debian particuliers, à savoir **backupchecker**, **db2twitter** et **retweet**. Nous saisissons la ligne suivante :

```
$ aptly mirror create -architectures="amd64" -filter-with-
deps="true" -filter="backupchecker|db2twitter|retweet" sid-debian-
main http://ftp.fr.debian.org/debian sid main
Downloading http://ftp.fr.debian.org/debian/dists/sid/InRelease...
gpgv: Signature made Sun 22 Nov 2015 21:58:12 CET using RSA key ID
46925553
gpgv: Good signature from "Debian Archive Automatic Signing Key
(7.0/wheezy) <ftpmaster@debian.org>"
gpgv: Signature made Sun 22 Nov 2015 21:58:12 CET using RSA key ID
2B90D010
gpgv: Good signature from "Debian Archive Automatic Signing Key
(8/jessie) <ftpmaster@debian.org>"
```

```
$ aptly mirror create -architectures="amd64" -filter-with-
deps="true" -filter="backupchecker|db2twitter|retweet" sid-debian-
main http://ftp.fr.debian.org/debian sid main
Downloading http://ftp.fr.debian.org/debian/dists/sid/InRelease...
gpgv: Signature made Sun 22 Nov 2015 21:58:12 CET using RSA key ID
46925553
gpgv: Good signature from "Debian Archive Automatic Signing Key
(7.0/wheezy) <ftpmaster@debian.org>"
gpgv: Signature made Sun 22 Nov 2015 21:58:12 CET using RSA key ID
2B90D010
gpgv: Good signature from "Debian Archive Automatic Signing Key
(8/jessie) <ftpmaster@debian.org>"

Mirror [sid-debian-main]: http://ftp.fr.debian.org/debian/ sid
successfully added.
You can run 'aptly mirror update sid-debian-main' to download
repository contents.
```

Notre nouveau miroir local est maintenant créé. L'option la plus importante est ici **-filter="backupchecker|db2twitter|retweet"** qui indique de prendre en compte le paquet backupchecker ou db2twitter ou retweet pour chaque paquet étudié de la source distante.

Nous allons maintenant synchroniser notre miroir local avec la source distante à l'aide de la commande suivante :

```
$ aptly mirror update sid-debian-main
Downloading http://ftp.fr.debian.org/debian/dists/sid/InRelease...
gpgv: Signature made Sun 22 Nov 2015 21:58:12 CET using RSA key ID
46925553
gpgv: Good signature from "Debian Archive Automatic Signing Key
(7.0/wheezy) <ftpmaster@debian.org>"
,,,
Downloading http://ftp.fr.debian.org/debian/pool/main/m/mysql-
connector-python/python3-mysql.connector_2.0.4-1_all.deb...
Downloading http://ftp.fr.debian.org/debian/pool/main/b/
backupchecker/backupchecker_1.7-1_all.deb...

Mirror `sid-debian-main` has been successfully updated.
```

Nous vérifions maintenant le contenu de notre miroir avec la commande suivante pour vérifier que les paquets demandés ainsi que leurs dépendances ont bien été téléchargés depuis notre source distante :

```
$ aptly mirror show -with-packages sid-debian-main
Name: sid-debian-main
Archive Root URL: http://ftp.fr.debian.org/debian/
...
Packages:
  backupchecker_1.7-1_all
  ca-certificates_20150426_all
  db2twitter_0.2-1_all
  debconf_1.5.58_all
  dh-python_2.20151103_all
  mime-support_3.59_all
  ...
  tar_1.28-2.1_amd64
  zlib1g_1:1.2.8.dfsg-2+b1_amd64
```

L'examen du contenu de notre miroir montre bien que nous avons téléchargé les différents paquets et leurs dépendances, jusqu'à l'interpréteur Python3 nécessaire pour faire tourner ces différents programmes.

Bon, avec trois paquets, on ne va pas très loin. Il est possible de spécifier des expressions englobant beaucoup plus de paquets, comme la suivante qui considère tous les paquets commençant par la chaîne **python3** :

```
$ aptly mirror create -architectures="amd64" -filter-with-
deps="true" -filter="(Name (% python3-*)" sid-debian-main http://
ftp.fr.debian.org/debian sid main
```

Puis, comme dans l'exemple précédent, nous lançons la synchronisation avec la source distante à l'aide de la commande suivante :

```
$ aptly mirror update sid-debian-main
Downloading http://ftp.fr.debian.org/debian/dists/sid/InRelease...
gpgv: Signature faite le lun. 23 nov. 2015 09:59:31 CET avec la
clef RSA d'identifiant 46925553
gpgv: Bonne signature de " Debian Archive Automatic Signing Key
(7.0/wheezy) <ftpmaster@debian.org> "
gpgv: Signature faite le lun. 23 nov. 2015 09:59:31 CET avec la
clef RSA d'identifiant 2B90D010
gpgv: Bonne signature de " Debian Archive Automatic Signing Key
(8/jessie) <ftpmaster@debian.org> "
Downloading & parsing package files...
Downloading http://ftp.fr.debian.org/debian/dists/sid/main/binary-
amd64/Packages.bz2...
Downloading http://ftp.fr.debian.org/debian/dists/sid/main/binary-
amd64/Packages.gz...
Applying filter...
Packages filtered: 48416 -> 2669.
Building download queue...
Download queue: 2669 items (1.74 GiB)
Downloading http://ftp.fr.debian.org/debian/pool/main/e/
elasticsearch-curator/python3-elasticsearch-curator_3.4.0-1_all.
deb...
...
Downloading http://ftp.fr.debian.org/debian/pool/main/w/wheel/
python3-wheel_0.26.0-1_all.deb...
Downloading http://ftp.fr.debian.org/debian/pool/main/p/python-
qwt/python3-qwt_0.5.1-1_all.deb...
```

Nous constatons que seuls les paquets dont le nom commence par **python3-** et leurs dépendances sont téléchargés.

Il est également possible d'utiliser les filtres lors d'autres opérations, par exemple pour rechercher dans un dépôt un paquet particulier:

```
$ aptly mirror search sid-debian-main 'retweet'
retweet_0.6-1_all
```

Mais également des expressions plus complexes avec tous les paquets qui commencent par **python3-** dans notre dépôt :

```
$ aptly mirror search sid-debian-main 'Name (% python3-*)'
python3-requests-oauthlib_0.4.0-1_all
python3-chardet_2.3.0-1_all
...
python3-six_1.10.0-1_all
python3-requests_2.8.1-1_all
python3-cryptography_1.1-1+b1_amd64
```

Il est aussi possible d'utiliser des critères de recherche comme **Version** pour le numéro de version, **Architecture** pour le type d'architecture, ou encore **Source** pour le nom du paquet source. Ici un exemple où nous souhaitons récupérer tous les paquets ayant un numéro de version supérieur à **0.3** :

```
$ aptly mirror search sid-debian-main 'Version (>= 0.3)'
python3-requests-oauthlib_0.4.0-1_all
python3-chardet_2.3.0-1_all
,,,
python3-oauthlib_1.0.3-1_all
python3-jwt_1.3.0-1_all
python3-urllib3_1.12-1_all
```

Enfin, astuce utile pour éliminer du contenu indésirable d'un dépôt (pas un miroir), les filtres peuvent être appliqués à la commande **aptly repo remove** de la façon suivante pour, par exemple, ne conserver que le paquet **fpart** dans votre dépôt dont nous affichons le contenu de la façon suivante :

```
$ aptly repo show -with-packages amd64-sid-main
Name: amd64-sid-main
Comment:
Default Distribution:
Default Component: main
Number of packages: 4
Packages:
  backupchecker_1.7-1_all
  db2twitter_0.2-1_all
  retweet_0.6-1_all
  fpart_0.9.2-1_amd64
```

Nous ne souhaitons donc que conserver le paquet fpart, ce qui va nous amener à supprimer tous les paquets qui ne portent pas le nom **fpart** :

```
$ aptly repo remove amd64-sid-main '(!Name (% fpart))'
Loading packages...
[-] db2twitter_0.2-1_all removed
[-] retweet_0.6-1_all removed
[-] backupchecker_1.7-1_all removed
```

À l'utilisation, les filtres apparaissent rapidement indispensables. Un grand nombre de commandes les supporte, il vaut donc mieux investir un peu de temps dans leur compréhension plutôt qu'imaginer des boucles shell autour des commandes de base d'Aptly qui ne prendront par exemple pas en compte les dépendances et seront beaucoup plus lourdes à manipuler que le bon filtre correspondant.

PHASE 2
Fusion d'instantanés

Tout d'abord, un rappel rapide : pour Aptly, un instantané est l'état figé d'un dépôt ou d'un miroir Debian au moment où l'instantané est créé. Les instantanés vont donc incarner différents états d'un dépôt ou d'un miroir dans le temps, par exemple avant et après un changement majeur dans le contenu de votre dépôt Debian local. La fusion d'instantanés quant à elle permet de faire apparaître comme un dépôt unique le contenu de plusieurs miroirs ou dépôts. Cela représente un fort intérêt pour faire apparaître comme un unique dépôt sur votre infrastructure le contenu d'un miroir distant mélangé au contenu de votre propre dépôt privé.

L'exemple suivant propose de fusionner le contenu de deux dépôts contenant chacun un unique paquet (pour simplifier l'exemple). Voici le détail du contenu des deux dépôts en question que nous affichons ici :

```
$ aptly repo show -with-packages backupchecker-debian
Name: backupchecker-debian
Comment:
Default Distribution:
Default Component: main
Number of packages: 1
Packages:
  backupchecker_1.7-1_all
```

Ce premier dépôt contient un unique paquet Debian, celui du logiciel backupchecker.

```
$ aptly repo show -with-packages db2twitter-debian
Name: db2twitter-debian
Comment:
Default Distribution:
Default Component: main
Number of packages: 1
Packages:
  db2twitter_0.2-1_all
```

Ce second dépôt contient également un unique paquet Debian, celui du logiciel db2twitter.

Nous générons maintenant des instantanés pour chacun de ces dépôts. Rappelons que l'instantané d'un dépôt ou d'un miroir correspond à l'état de ce dépôt ou de ce miroir

à un instant **T**. Pour créer les instantanés, nous passons les commandes suivantes :

```
$ aptly snapshot create backupchecker-debian-24-11-2015-1 from
repo backupchecker-debian

Snapshot backupchecker-debian-24-11-2015-1 successfully created.
You can run 'aptly publish snapshot backupchecker-
debian-24-11-2015-1' to publish snapshot as Debian repository.
```

Le premier instantané du dépôt **backupchecker-debian** est ainsi créé. Nous avons indiqué dans son nom la date du jour ainsi qu'un numéro.

```
aptly@portable:~$ aptly snapshot create db2twitter-
debian-24-11-2015-1 from repo db2twitter-debian

Snapshot db2twitter-debian-24-11-2015-1 successfully created.
You can run 'aptly publish snapshot db2twitter-
debian-24-11-2015-1' to publish snapshot as Debian repository.
```

Après avoir créé le second instantané, nous listons maintenant les instantanés disponibles.

```
$ aptly snapshot list
List of snapshots:
 * [backupchecker-debian-24-11-2015-1]: Snapshot from local repo
[backupchecker-debian]
 * [db2twitter-debian-24-11-2015-1]: Snapshot from local repo
[db2twitter-debian]
To get more information about snapshot, run `aptly snapshot show
<name>`.
```

Comme l'indique notre dernière commande, nous avons bien deux instantanés différents pointant chacun sur leur dépôt respectif.

Nous abordons maintenant la partie la plus intéressante, la fusion des instantanés qui va nous permettre de faire apparaître un troisième instantané fusionnant le contenu des deux instantanés sources. Pour cela nous utilisons la commande suivante :

```
$ aptly snapshot merge custom-packages backupchecker-
debian-24-11-2015-1 db2twitter-debian-24-11-2015-1

Snapshot custom-packages successfully created.
You can run 'aptly publish snapshot custom-packages' to publish
snapshot as Debian repository.
```

Notre nouvel instantané a bien été créé. Listons maintenant le contenu offert par cet instantané à l'aide de la commande suivante :

```
$ aptly snapshot show -with-packages custom-packages
Name: custom-packages
Created At: 2015-11-24 17:27:36 CET
Description: Merged from sources: 'backupchecker-
debian-24-11-2015-1', 'db2twitter-debian-24-11-2015-1'
Number of packages: 2
Packages:
  backupchecker_1.7-1_all
  db2twitter_0.2-1_all
```

Nous voyons bien dans la description qu'il s'agit d'une fusion d'instantanés, mais aussi dans la liste des paquets, tout à la fin de la sortie de la commande, les paquets contenus dans les dépôts pointés par les instantanés.

La fusion d'instantanés apporte de très grandes possibilités. Nous pouvons associer l'instantané d'un miroir distant avec celui d'un dépôt local, afin de faire apparaître un unique dépôt local contenant tous les paquets par exemple d'un dépôt officiel mais qui contiendrait aussi nos paquets « maison ». Nous pouvons aussi faire apparaître un ensemble de dépôts locaux chacun dédié à une application bien précise comme un seul et unique dépôt, par exemple auprès des clients APT de l'infrastructure. La grande flexibilité des instantanés rend trivial ce type de manipulations et permet de s'affranchir de la lourdeur de se cantonner à la création d'un dépôt unique qui bien souvent complique considérablement la gestion dudit dépôt à long terme.

PHASE 3

Utilisation de l'API Aptly

Depuis la version 0.9.6, Aptly offre la possibilité d'utiliser une API REST. Cela permet à des programmes externes de donner des ordres à Aptly, par exemple pour mettre à jour un miroir, créer un dépôt ou changer l'instantané couramment employé.

Nous commençons par initialiser l'utilisation de l'API REST avec Aptly :

```
$ curl http://localhost:8080/api/version
{"Version":"0.9.6"}
```

Commençons par lister les dépôts disponibles :

```
$ curl --silent http://localhost:8080/api/repos | python -m json.
tool
[
    {
        "Comment": "",
        "DefaultComponent": "main",
        "DefaultDistribution": "",
        "Name": "backupchecker-debian"
    },
```

```
    {
        "Comment": "",
        "DefaultComponent": "main",
        "DefaultDistribution": "",
        "Name": "amd64-sid-main"
    },
    {
        "Comment": "",
        "DefaultComponent": "main",
        "DefaultDistribution": "",
        "Name": "db2twitter-debian"
    }
]
```

Nous utilisons l'option **--silent** de **curl** afin de ne pas afficher la barre de progression de cet outil pour une meilleure lisibilité. Puis nous envoyons le résultat de cette commande pour traitement au module **json** de la commande Python qui nous fournit un affichage agréable des données retournées par la commande curl. Nous constatons ici que trois dépôts sont disponibles.

Nous procédons de la même façon pour obtenir les détails concernant un paquet Debian, par exemple le paquet backupchecker présent dans le dépôt **backupchecker-debian** :

```
$ curl --silent http://localhost:8080/api/repos/backupchecker-
debian/packages?format=details | python -m json.tool
[
    {
        "Architecture": "all",
        "Depends": "python3, python3:any (>= 3.4~)",
        "Description": " fully automated backup checker\n",
        "Description-Md5": "d5e4001532894d854c13e83fbebdaa1a",
        "Filename": "backupchecker_1.7-1_all.deb",
        ,,,
        "Version": "1.7-1"
    }
]
```

Nous obtenons le contenu détaillé du dépôt nommé **backupchecker-debian**, qui ne contient ici qu'un paquet. Il est également possible d'effectuer des recherches dans la liste des paquets d'un dépôt.

Mais l'API d'Aptly est loin de ne proposer que des commandes consultatives. Voyons maintenant une commande permettant d'envoyer un nouveau paquet vers Aptly et de l'ajouter automatiquement à un dépôt existant :

```
$ curl -X POST -F file=@aptly_0.9.6_amd64.deb http://
localhost:8080/api/files/amd64-sid-main
["amd64-sid-main/aptly_0.9.6_amd64.deb"]
$ curl -X POST http://localhost:8080/api/repos/amd64-sid-main/
file/amd64-sid-main
{"FailedFiles":[],"Report":{"Warnings":[],"Added":["aptly_0.9.6_
amd64 added"],"Removed":[]}}
```

Nous venons ici de rajouter le fichier **aptly_0.9.6_amd64.deb** présent dans notre répertoire de travail vers un répertoire temporaire nommé **amd64-sid-main** sur le serveur d'Aptly à travers une requête JSON de type POST. La seconde requête indique à Aptly d'ajouter tous les paquets Debian présents dans ce répertoire dans le dépôt local **amd64-sid-main**. La première requête retourne une chaîne nous indiquant que le fichier a bien été ajouté au répertoire **amd64-sid-main** alors que la seconde requête nous indique que l'ajout d'un paquet Debian au dépôt a bien été effectué.

C'est une fonctionnalité très intéressante de l'API d'Aptly. Le fait de pouvoir permettre à des applications extérieures de rajouter des paquets dans un dépôt va vous permettre de construire par exemple simplement une solution d'intégration continue poussant automatiquement les nouveaux paquets Debian vers un dépôt Debian central, paquets pouvant être déployés automatiquement sur, par exemple, des environnements de tests et d'intégration.

Il ne suffit pas qu'un dépôt Debian géré par Aptly contienne des paquets Debian pour profiter de toute la puissance et la flexibilité d'Aptly. Pour profiter au mieux des avantages d'Aptly, nous devons utiliser les instantanés. La tâche suivante que nous demanderons à l'API d'Aptly est donc de publier un instantané correspondant à l'état actuel de notre dépôt.

Pour cela, nous passons également par l'API REST d'Aptly :

```
$ curl --silent -X POST -H 'Content-Type: application/json' --data
'{"Name":"amd64-sid-main-snapshot-1"}' http://localhost:8080/api/
repos/amd64-sid-main/snapshots | python -m json.tool
{
    "CreatedAt": "2016-03-07T23:36:43.14721612+01:00",
    "Description": "Snapshot from local repo [amd64-sid-main]",
    "Name": "amd64-sid-main-snapshot-1"
}
```

Nous avons ici créé un instantané nommé **amd64-sid-main-snapshot-1** qui représente l'état actuel de notre dépôt **amd64-sid-main**.

```
$ curl --silent -X POST -H 'Content-Type: application/json' --data
'{"SourceKind": "snapshot", "Sources": [{"Name": "amd64-sid-main-
snapshot-1"}], "Architectures": ["amd64"], "Distribution": "sid"}'
http://localhost:8080/api/publish/amd64-sid-main-snap1 | python -m
json.tool
{
    "Architectures": [
        "amd64"
    ],
    "Distribution": "sid",
    "Label": "",
    "Origin": "",
    "Prefix": "amd64-sid-main-snap1",
    "SourceKind": "snapshot",
    "Sources": [
        {
```

```
            "Component": "main",
            "Name": "amd64-sid-main-snapshot-1"
        }
    ],
    "Storage": ""
}
```

La commande nous retourne le détail de l'instantané qui vient d'être publié, détail qui correspond à ce qui nous est retourné lorsqu'on interroge la liste des instantanés à l'aide de la commande suivante :

```
$ curl --silent http://localhost:8080/api/publish | python
-m json.tool
[
    {
        "Architectures": [
            "amd64"
        ],
        "Distribution": "sid",
        "Label": "",
        "Origin": "",
        "Prefix": "amd64-sid-main-snap1",
        "SourceKind": "snapshot",
        "Sources": [
            {
                "Component": "main",
                "Name": "amd64-sid-main-snapshot-1"
            }
        ],
        "Storage": ""
    }
]
```

Quoiqu'encore incomplète, l'API d'Aptly est déjà largement utilisable pour les besoins courants. Nous pouvons déjà nous en servir pour pousser depuis un serveur de construction de paquets Debian des nouveaux paquets Debian de votre application « maison », paquets qui seront donc immédiatement accessibles à votre chaîne d'outils d'intégration continue pour subir une série de tests fonctionnels par exemple.

LE RÉSULTAT

Aptly s'avère extrêmement puissant grâce à ses fonctionnalités avancées. Vous pouvez filtrer efficacement la longue liste des paquets Debian disponibles dans les dépôts officiels pour n'en extraire qu'un sous-ensemble plus simple à stocker ou à manipuler. Faire passer plusieurs miroirs distants pour un unique dépôt Debian local est maintenant un jeu d'enfant avec les fusions d'instantanés, permettant d'alléger considérablement votre stockage interne mais aussi de rendre cohérent un ensemble de dépôts et de miroirs en ne présentant qu'un unique dépôt aux différents systèmes utilisant vos dépôts Debian. ■

CONTINUOUS DATA PROTECTION FOR GNU/LINUX

Olivier DELHOMME [Développeur de logiciels libres pour mon loisir]

Continuous Data Protection For GNU/Linux (CDPFGL) est un ensemble de logiciels libres qui permettent de sauvegarder vos fichiers. La principale caractéristique différenciatrice des autres (nombreux) logiciels de sauvegarde existants est que cet ensemble permet la sauvegarde des fichiers en continu pendant qu'ils sont créés et modifiés. Cet article vous présentera les principes et la mise en œuvre des logiciels de ce projet (qui est le mien).

Mots-clés : Protection continue des données, Sauvegarde, Serveur sans état, Déduplication à la source, Empreinte mémoire faible

Résumé

Après avoir exposé les idées et les principes qui régissent le projet, j'aborderai rapidement le principe de la déduplication et expliquerai sa mise en œuvre pour le stockage des données du côté serveur. Nous verrons comment télécharger, compiler et installer les dépendances du projet ainsi que le projet lui-même. Enfin bien que d'après moi le projet ne soit pas mûr pour être mis en production nous verrons comment mettre en œuvre les trois programmes qui le composent.

L'idée de départ provient d'une discussion ayant eu lieu dans une liste d'administrateurs système et réseau. La problématique était le temps nécessaire à une commande `rsync` pour terminer la liste des fichiers à sauvegarder avant de lancer effectivement cette sauvegarde. Le nombre de fichiers étant de plusieurs dizaines de millions, le temps en question se comptait en heures. S'est posée pour moi la question de savoir si on ne pouvait pas constituer la liste au fil de l'eau et quitte à constituer cette liste autant faire le travail au fur et à mesure en transmettant les fichiers en question ! Ainsi est né le projet de sauvegarde en continu.

Après avoir testé **ZMQ** et décidé que je n'avais pas besoin d'une telle complexité (et complétude) j'ai choisi HTTP comme méthode de communication entre les clients et le serveur. J'ai décidé d'imposer quelques principes pour borner le fonctionnement du serveur. J'ai choisi d'écrire une interface REST sans état (les grands disent « *stateless* ») achevant la déduplication en ligne, à la source, tout en essayant de préserver la bande passante au maximum. Initialement j'avais dans l'idée d'associer à ce serveur un stockage basé sur **Ceph**. Finalement, le premier stockage codé est un stockage de fichiers à plat dans une arborescence à plusieurs niveaux. Toutefois le code a été prévu pour qu'il soit possible d'utiliser simplement d'autres systèmes de stockage pour le serveur. De son côté, le client doit être résilient aux connexions / déconnexions et continuer dans la mesure du possible son travail même lorsqu'il ne parvient plus à joindre le serveur.

Il découle de tout cela que le serveur a une empreinte mémoire minimale (le plus possible). En tout état de cause cette empreinte mémoire ne dépend en aucune manière de

la volumétrie ou du nombre de blocs « dédupliqués » transmis au serveur (et pas non plus du temps qui passe – du moins j'essaie d'y faire attention).

Le projet est constitué d'un serveur (**cdpfglserver**) et deux programmes clients, un pour réaliser la sauvegarde des fichiers vers le serveur (**cdpfglclient**) et l'autre pour faire la restauration des fichiers depuis ce serveur (**cdpfglrestore**).

À la date de rédaction de l'article, les fonctionnalités de base sont implémentées : sauvegarde en continu (pourvu que l'interface **FANOTIFY** soit compilée dans le noyau) et restauration d'un fichier avec une sélection possible sur le nom et la date.

1 Comment savoir si l'interface FANOTIFY est disponible sur votre machine ?

Il s'agit de trouver les options qui ont été utilisées pour compiler le noyau Linux de votre distribution. Pour ce faire nous devons partir à la recherche d'un fichier texte contenant ces options. Il y a plusieurs façons de procéder car toutes les distributions n'utilisent pas nécessairement les mêmes mécanismes.

Pour trouver la version de votre noyau, utilisez la commande **uname -r**. Elle vous donnera un résultat de la sorte : **3.16.0-4-amd64**. Regardez alors si vous trouvez un fichier nommé **config-3.16.0-4-amd64** dans le dossier **/boot**.

Si vous ne possédez pas ce fichier, il est possible que votre noyau ait été compilé de sorte que le module **configs** se charge de garder les options utilisées. Vous pouvez alors exécuter la commande **modprobe configs** qui chargera le module et donnera accès au fichier **/proc/config.gz** qui contient, dans un format compressé, les informations de configuration du noyau.

Une fois que vous avez trouvé votre fichier, recherchez-y les informations relatives à FANOTIFY comme suit (remplacez **grep** par **zgrep** pour le fichier compressé) :

```
# grep FANOTIFY /boot/config-3.16.0-4-amd64
CONFIG_FANOTIFY=y
# CONFIG_FANOTIFY_ACCESS_PERMISSIONS is not set
```

Dans ce cas l'option est à **y**, c'est-à-dire que le noyau possède l'interface FANOTIFY.

```
# grep FANOTIFY /boot/config-3.16.0-4-amd64
# CONFIG_FANOTIFY is not set
```

Dans ce cas l'option n'est pas renseignée et le noyau ne possède pas l'interface FANOTIFY. Il faudra alors compiler un nouveau noyau (non abordé dans cet article) en spécifiant bien **y** lors du choix de l'activation de cette interface.

2 Le stockage des données par le serveur

2.1 La déduplication

La déduplication est une méthode qui permet de ne garder que les parties non dupliquées d'un fichier, d'un ensemble de fichiers ou d'un ensemble de blocs. Sur un unique fichier, il s'agit de ne garder que les parties « uniques » ou « originales » de ce fichier. Pour ce faire le fichier est généralement découpé en tronçons de taille fixe (des blocs) pour lesquels sont calculés des *hashs*. À deux *hashs* identiques correspondent deux blocs très probablement identiques (la probabilité d'identité est si proche de **1** que l'on commet souvent l'abus de langage de dire qu'ils sont identiques). Un fichier peut être représenté par la liste des *hashs* correspondants aux blocs qui le composent. En ne stockant les blocs aux *hashs* identiques qu'une seule fois, nous diminuons ainsi la place occupée sur le disque.

Note

Un *hash* d'un bloc peut être vu comme un résumé de ce bloc. Il est calculé par un algorithme de hachage à partir des données du bloc. Les algorithmes de hachage les plus connus et qu'il ne faut plus utiliser sont MD4, MD5 et SHA1. Aujourd'hui on utilisera plutôt SHA2 **[0]** et SHA3 **[1]**.

Cette technique est particulièrement intéressante pour la sauvegarde. En effet la déduplication permet d'éviter la copie des blocs qui n'ont pas changé entre deux sauvegardes ou que l'on connaît déjà par ailleurs (sauvegarde de multiples systèmes identiques, fichiers en double, etc.). Dans ce projet elle est réalisée à la source par une communication entre le client et le serveur : le client envoie au serveur la liste des *hashs* (l'algorithme utilisé est SHA256) des blocs d'un fichier et le serveur lui répond la liste des *hashs* des blocs dont il

a besoin (il s'agit des *hashs* qu'il ne connaît pas, c'est à dire des blocs inconnus donc originaux). Le client envoie alors les blocs réclamés correspondants.

Pour le moment, il n'y a aucun mécanisme qui permette de détecter un *hash* identique pour deux blocs aux contenus différents. Il serait intéressant d'avoir un tel mécanisme de détection. En effet la sécurité de SHA256 est réputée être de 2^{128} ainsi avec $2^{128}+1$ blocs nous serions certains d'avoir au moins une collision. C'est-à-dire qu'en terme de stockage, même avec les plus petits blocs de 512 octets, cette certitude représenterait tout de même 144×10^{15} yotta octets ! Toutefois il n'est pas impossible que cette sécurité soit moindre ou que le hasard nous fasse tomber sur une collision. Un tel mécanisme pourrait garantir que l'on sauvegarde bien le bon bloc. Il reste à le concevoir et à l'écrire !

Dans ce projet, cette déduplication réalisée par blocs existe en deux modes : soit la taille des blocs est identique quel que soit le fichier traité (16384 octets par défaut mais vous pouvez choisir votre propre valeur) soit la taille des blocs varie en fonction de la taille du fichier traité (de 512 octets pour les très petits fichiers à 256 Kio pour les fichiers de plus de 130 Mio). En effet, lors de mes essais, il est apparu que le taux d'intra-déduplication (entre les fichiers d'un même système) est plus important lorsque la taille des blocs est petite. Avec cette taille variable, on cherche à augmenter ce taux sans pour autant trop faire exploser le nombre de blocs sauvegardés côté serveur. En effet un fichier de 1 Gio génère 4096 blocs de 256 Kio ou 65536 blocs de 16384 octets selon le mode choisi.

2.2 Mise en cache des modifications

Lorsque le serveur n'est plus joignable, le client enregistre dans sa base de données interne (**sqlite**) l'ensemble des données et métadonnées des fichiers créés et modifiés. Dès que la connexion est à nouveau disponible, la base de données est vidée petit à petit en parallèle des autres actions.

2.3 Stockage des blocs à plat par le serveur

Le système de stockage des blocs reçus par le serveur est effectué dans des fichiers plats où chaque bloc est inscrit dans un fichier. Pour éviter d'avoir une quantité phénoménale de fichiers dans un même dossier, le serveur initialise une arborescence de dossiers (par défaut sur deux niveaux). Pour chaque dossier de chaque niveau, on crée **256** dossiers de **00** à **FF**. Par défaut il y a **65536** dossiers au total de **0000** à **FFFF**. Sur un système de fichiers **ext4** rien que cette structure à deux niveaux représente 256 Mio. Par la suite on range les blocs dans le dossier dont l'arborescence correspond aux premiers octets du *hash* : ainsi le bloc dont le *hash* est **beef3db9718bb771f293a9337181f212ebba690999309fce700e6d0072396324** sera stocké dans le dossier **be/ef/** avec une structure à deux niveaux et **be/ef/3d/b9** avec 4 niveaux. Le fichier ainsi créé a pour nom son *hash* au complet.

Un inconvénient d'une telle structure est qu'avec un niveau « élevé » de 4, le temps de construction de la structure est très loin d'être négligeable : un test de création du niveau 3 (64 Gio pour les seuls dossiers vides) sur un SSD s'est déroulé en une heure ! Le logiciel ne permet pas pour le moment d'utiliser plus de 4 niveaux (leur création n'a jamais été testée – avis aux amateurs (prévoyez de la place disque !)).

L'avantage de cette façon de faire c'est que l'on retrouve facilement les blocs, qu'ils sont bien répartis sur l'ensemble de l'arborescence et qu'elle est commune à l'ensemble des clients. Ainsi lorsque des clients ont les mêmes fichiers, leurs blocs ne se trouvent stockés qu'une seule fois sur le serveur.

Les métadonnées des fichiers sont enregistrées dans des fichiers textes à plat. Un fichier est créé par nom de machine. L'avantage de cette méthode est qu'il est facile de vérifier quels sont les fichiers qui ont été sauvegardés sur le serveur. L'inconvénient c'est que pour s'assurer de la cohérence de ce fichier le serveur utilise un *thread* unique pour écrire dedans. Même si la taille des métadonnées est relativement faible ce *thread* unique est un goulot d'étranglement.

Le principal inconvénient de cette méthode de stockage est qu'elle ne passe sans doute pas à l'échelle sans avoir recours à des « bidouilles » sur le système de fichiers comme pour la partition des dossiers sur plusieurs stockages physiques séparés. De plus, ce goulot d'étranglement sur le serveur limite sans aucun doute le nombre maximal de clients possible. Cette limite n'a pas été testée et notamment on ne sait pas à partir de combien de clients les performances sont impactées.

3 Installation du projet

Si vous êtes sous Debian vous trouverez des paquets binaires tout prêts mais je vous les déconseille pour le moment (à moins que vous ne soyez un développeur Debian et ayez envie de prendre le projet sous votre aile afin d'aider l'auteur dans sa démarche de création de paquets). Nous allons installer les programmes du projet directement à partir des

sources. Quelques dépendances peuvent être installées par le système de paquets de votre distribution car les versions minimales nécessaires ne sont pas récentes. C'est le cas de **glib**, **libcurl**, sqlite ou les outils de compilation comme les **autotools**, **gcc**, et **make**. En revanche pour les autres il convient de prendre une version récente soit parce qu'une fonctionnalité le nécessite soit parce qu'un bug a été corrigé et que cela améliore le programme. Voyons comment installer ces dépendances pour Debian 8 et Centos 7. Commençons par les dépendances pour lesquelles la distribution fournit des versions assez récentes :

- Sous Debian vous pouvez taper les commandes suivantes :

```
$ sudo apt-get install git autoconf automake libtool make
libsqlite3-dev libglib2.0-dev libcurl4-gnutls-dev intltool
```

- Sous Centos la commande est très similaire :

```
$ sudo yum install git autoconf automake libtool make sqlite-devel
glib2-devel libcurl-devel intltool
```

Il s'agit maintenant d'installer une version récente de deux librairies. Cette partie est normalement identique sous Debian et sous Centos. Installons **jansson** une librairie C pour JSON. Nous la récupérons avec **git** puis la compilons de manière classique en prenant soin de générer le script **configure** :

```
$ git clone git://github.com/akheron/jansson.git
$ (cd jansson && autoreconf -f -i && ./configure && make && sudo
make install)
```

Nous procédons de même avec la librairie **libmicrohttpd** qui permet d'embarquer un serveur HTTP dans un code C :

```
$ wget --quiet -c http://ftp.gnu.org/gnu/libmicrohttpd/
libmicrohttpd-0.9.46.tar.gz
$ (tar zxf libmicrohttpd-0.9.46.tar.gz && cd libmicrohttpd-0.9.46
&& ./configure && make && sudo make install)
```

Il reste maintenant à compiler et installer les programmes **cdpfglclient**, **cdpfglserver** et **cdpfglrestore**. La procédure est très classique et parfaitement similaire aux deux précédentes :

```
$ git clone https://github.com/dupgit/sauvegarde.git
$ (cd sauvegarde && ./autogen.sh && ./configure && make && sudo
make install)
```

Les programmes se sont installés dans **/usr/local/bin** et les fichiers de configuration se trouvent dans **/usr/local/etc/cdpfgl**.

Recommencez cette installation sur toutes les machines sur lesquelles vous désirez installer le client. Dans la suite on admet que le serveur est installé sur une machine nommée **save** et que les clients sont installés sur d'autres machines de ce réseau (qui ne filtre pas les requêtes HTTP).

4 Mise en œuvre du projet

4.1 Le serveur

Sur la machine qui fera fonctionner le serveur, dans le dossier de configuration **/usr/local/etc/cdpfgl** copiez le fichier **server.conf** en **server.prod** et éditez ce dernier (en cas de réinstallation vous ne perdrez pas votre configuration). Il convient d'indiquer où le serveur devra enregistrer les blocs et les métadonnées (sans indication, le chemin par défaut est **/var/tmp/cdpfgl**). Pour cela éditez la valeur de **file-directory** dans la section **[File_Backend]** et indiquez le chemin du dossier qui contiendra les données (par exemple **/home/dup/cdpfgl**). Prenez garde à ce que l'utilisateur sous lequel vous ferez fonctionner le serveur puisse écrire à cet endroit (le serveur n'a pas besoin des droits root). Si le dossier n'existe pas, il sera créé ainsi que la sous-arborescence pour stocker les blocs. En fonction de votre système de stockage, cela peut prendre plus ou moins de temps. Soyez patient, par exemple un niveau 2 est créé en un peu moins de 2 minutes sur un **Bananapi M1**. Les autres valeurs par défaut sont correctes pour la réalisation de tests. En production on veillera sans doute à désactiver le mode debug : **debug-mode=false**.

Le serveur se lance avec la commande **cdpfglserver -c /usr/local/etc/cdpfgl/server.prod**. Dans le cas où l'option **debug-mode** est toujours positionnée sur **true**, le serveur vous indique ce qu'il fait à tout moment. Au démarrage il devrait vous indiquer qu'il créé l'arborescence toutefois s'il n'indique rien c'est que cette arborescence a déjà été créée.

4.2 Le client

Maintenant que le serveur est installé et fonctionnel, intéressons-nous aux clients. Vous devez compiler le projet sur chacune des machines clientes. Dans le dossier de configuration **/usr/local/etc/cdpfgl** copiez les fichiers **client.conf** et **restore.conf** respectivement en **client.prod** et **restore.prod** et éditez-les.

Dans la section **[Server]** des deux fichiers indiquez l'adresse IP de la machine sur laquelle vous avez installé

le serveur (à ce stade vous en avez fini avec la configuration du fichier **restore.prod**).

Dans la section **[Client]** du fichier **client.prod**, avec la clef **directory-list** indiquez les dossiers que vous souhaitez sauvegarder (séparés par des points-virgules). Pour le moment la liste doit être explicite. Les expressions rationnelles ne sont pas encore autorisées dans cette liste. La recherche s'effectuant par préfixe, tout sous-dossier d'un dossier indiqué dans cette liste sera automatiquement sauvegardé. Indiquez dans la clef **exclude-list** les dossiers ou fichiers que vous souhaitez exclure de la sauvegarde (là encore séparés par des points-virgules). Les expressions rationnelles simples fonctionnent sur cette liste. Par exemple, pour exclure tous les fichiers dont l'extension est **mp3** on pourra écrire **.mp3$**.

La clef **cache-directory** permet d'indiquer un dossier où seront stockées les métadonnées et les données en cache (en cas de perte de connexion au serveur). Indiquez ici un dossier dans un système de fichier où vous avez de la place et où vous ne risquez pas de bloquer le système. Pour quelques menus tests la valeur par défaut peut convenir mais pour des tests plus poussés ou de la production il faudra préférer une partition non système où il y a suffisamment de place libre.

Par défaut le mode de calcul des blocs des fichiers est adaptatif (**adaptive=true**). Pour pouvoir utiliser une taille fixe, passez cette clef à **false** et indiquez une taille (en octets) dans la clef **blocksize**. Vous n'avez probablement pas besoin de changer les autres valeurs par défaut. Sachez toutefois qu'en mode adaptatif la taille du *buffer* de communication qui correspond à la clef **buffersize** est également adaptative en partie et fixée à 2 Mio pour les fichiers de moins de 128 Mio et de 4 Mio au-delà.

Pour pouvoir utiliser l'interface FANOTIFY le client à besoin d'être lancé sous l'utilisateur « root ». De toute manière, si l'on souhaite sauvegarder un système complet il faut bien pouvoir traverser tous les fichiers et tous les dossiers de chacun des utilisateurs, qu'ils soient utilisateurs normaux ou système. Lancez le client en utilisant la commande **cdpfglclient -c /usr/local/etc/cdpfgl/client.prod**. Le client scanne immédiatement les dossiers et sous-dossiers que vous lui avez demandé de sauvegarder. Lorsqu'ils n'ont pas déjà été sauvegardés, il initie un dialogue avec le serveur pour réaliser cette sauvegarde. Il est aussi prêt à sauvegarder tout fichier qui viendrait à être créé ou modifié dans les dossiers indiqués. Par défaut, en mode debug, il indique tout ce qu'il fait.

4.3 La restauration d'un fichier

Le programme **cdpfglrestore** qui a en charge la restauration souffre de limitations qui devraient disparaître dans les prochaines versions. En effet, la *roadmap* prévoit de nouvelles fonctionnalités pour ce programme dans les versions 0.0.9 et 0.0.10 (à la date où ces lignes sont écrites la version 0.0.8 est pratiquement dans les cartons). Malgré ses limitations, le programme permet la restauration d'un fichier. Utilisez l'option **--help** qui vous donnera l'ensemble des options utilisables par le programme. L'invocation du programme nécessite un serveur en fonctionnement. L'option **-l truc** permet de lister les fichiers ou dossiers dont le nom contient **truc**. Vous pouvez limiter la liste ainsi obtenue en indiquant l'option **-a ladate** pour obtenir uniquement les fichiers ayant une date postérieure à **ladate** et **-b uneautredate** pour obtenir uniquement ceux dont la date est antérieure à **uneautredate**. Si vous remplacez l'option **-l truc** par **-r truc** vous restaurerez le dernier fichier de la liste précédemment obtenue. Si vous souhaitez restaurer ce fichier à un endroit précis indiquez le dossier avec l'option **-w /nom/du/dossier**. Une session de restauration pourrait ressembler à :

```
$ cdpfglrestore -c /usr/local/etc/cdpfgl/restore.prod -l options.c
...
[FILE] 2014-07-16 15:43:38 +0200 /home/dup/Dossiers_Perso/projets/projets_externes/fio/goptions.c
[FILE] 2015-04-06 21:31:34 +0200 /home/dup/Dossiers_Perso/projets/projets_externes/fio/options.c
...
[FILE] 2015-11-07 22:11:13 +0100 /home/dup/Dossiers_Perso/projets/sauvegarde/monitor/options.c
[FILE] 2015-11-21 21:09:04 +0100 /home/dup/Dossiers_Perso/projets/sauvegarde/monitor/options.c
[FILE] 2015-10-25 22:35:12 +0100 /home/dup/Dossiers_Perso/projets/sauvegarde/restaure/options.c
[FILE] 2015-10-25 22:35:12 +0100 /home/dup/Dossiers_Perso/projets/sauvegarde/serveur/options.c
[FILE] 2015-11-07 22:11:13 +0100 /home/dup/tmp/sauvegarde-0.0.6/monitor/options.c
[FILE] 2015-10-25 22:35:12 +0100 /home/dup/tmp/sauvegarde-0.0.6/restaure/options.c
[FILE] 2015-10-25 22:35:12 +0100 /home/dup/tmp/sauvegarde-0.0.6/serveur/options.c

$ cdpfglrestore -c /usr/local/etc/cdpfgl/restore.prod -l options.c -a 2015-10-26 -b 2015-11-20
...
[FILE] 2015-11-07 22:11:13 +0100 /home/dup/Dossiers_Perso/projets/sauvegarde/monitor/options.c
[FILE] 2015-11-07 22:11:13 +0100 /home/dup/tmp/sauvegarde-0.0.6/monitor/options.c

$ cdpfglrestore -c /usr/local/etc/cdpfgl/restore.prod -r /home/dup/tmp/sauvegarde-0.0.6/monitor/
options.c -d 0 -a 2015-10-26 -b 2015-11-20 -w /tmp
...
[files.c, 505] Erreur ou avertissement pour le fichier (/home/dup/tmp/sauvegarde-0.0.6/monitor/
options.c) : La définition de l'attribut standard::type n'est pas prise en charge
```

```
$ ls -ls /tmp/options*
20 -rw-r--r-- 1 dup dup 16926 nov.   7 22:11 /tmp/options.c

$ stat /tmp/options.c
  Fichier : " /tmp/options.c "
   Taille : 16926       Blocs : 40         Blocs d'E/S : 4096   fichier
Périphérique : fe00h/65024d     Inœud : 135505      Liens : 1
 Accès : (0644/-rw-r--r--)  UID : ( 1000/     dup)   GID : ( 1000/    dup)
 Accès : 2015-11-18 20:48:54.563022000 +0100
Modif. : 2015-11-07 22:11:13.207030000 +0100
Changt : 2016-04-26 23:10:29.207030693 +0200
  Créé : -
```

L'avertissement que l'on note lors de la restauration est « normal » : il s'agit d'un avertissement de la librairie glib qui renseigne sur l'impossibilité d'indiquer la valeur de l'attribut **standard::type** car le système de fichiers ne le prend pas en compte. Pas d'inquiétude, le fichier est néanmoins bien restauré avec les bons attributs comme les commandes **ls** et **stat** l'indiquent.

Conclusion

Ce projet est une formidable opportunité pour apprendre et tenter de mettre en fonctionnement des tas de « nouveautés » pour moi. Vous l'aurez compris sans peine à la lecture de l'article, il reste encore beaucoup à faire avant d'obtenir un ensemble de logiciels satisfaisant qui puisse sauvegarder en continu puis restaurer vos données rapidement et en toute confiance.

Bien entendu, s'agissant de logiciels libres, si le projet vous intéresse, vous êtes particulièrement encouragés à participer de quelque manière que ce soit – en codant dans le noyau Linux pour ajouter les événements manquants à l'interface FANOTIFY, en ajoutant au client l'interface de notification de votre système préféré, en ajoutant une traduction ou en chiffrant les communications entre les clients et le serveur (qui a dit HTTPS ?), ...

N'ayant rien d'autre à offrir que la liberté de mon projet, je prends garde à ce que les contributions soient rendues à leurs auteurs respectifs. À la racine du projet, vous trouverez un fichier **AUTHORS** à jour contenant l'ensemble des auteurs ayant déjà contribué. Par ailleurs, j'indique également les noms des contributeurs à une version donnée dans le fichier **NEWS** (également à la racine) qui me sert à écrire ce que je souhaite raconter sur les différentes versions lors de leur sortie. ■

Références

[0] SHA2 : https://en.wikipedia.org/wiki/SHA-2

[1] SHA3 : https://en.wikipedia.org/wiki/SHA-3

Remerciements

Je tiens à remercier toutes les personnes qui œuvrent pour le logiciel libre, toutes celles qui contribuent directement ou indirectement à un logiciel libre et l'ensemble des logiciels libres que j'utilise pour mon projet (sqlite, glib, jansson, **curl**, **libmicrohttpd**, autotools, **pkg-config**, ...). Parmi ceux-là, je tiens à remercier plus particulièrement Christian Grothoff du projet libmicrohttpd pour sa réactivité à corriger un bug sur lequel j'étais tombé par mon utilisation intensive et répétitive de requêtes POST. Si tous les mainteneurs de logiciels libres pouvaient prendre exemple il y aurait certainement plus de contributeurs. Encore merci !

DÉVELOPPEMENT RAPIDE D'UN MICRO GESTIONNAIRE DE TICKETS AVEC NODE.JS

Tristan COLOMBO

Parfois certains développements ne nécessitent pas de se compliquer inutilement la tâche. C'est ce que nous allons voir avec la création d'un système de gestion de tickets basé sur un serveur REST.

REST Nodes.js Redis yaml JavaScript

L'OBJECTIF

Développer un gestionnaire de tickets dont le format sera défini à l'aide d'un fichier de configuration. Le gestionnaire sera un serveur fonctionnant sur le modèle REST et un client pourra obtenir le format des tickets, la liste des tickets et pourra bien sûr ajouter un nouveau ticket.

LES OUTILS

- **Node.js** ;
- **npm**.

PHASE 1

Lecture du fichier de configuration

Le fichier de configuration utilisera le format yaml de manière à pouvoir être modifié simplement. Celui-ci contiendra obligatoirement une entrée **host** indiquant à quelle adresse et quel port le serveur devra être lancé. Les autres entrées permettront de définir le type de donnée qui sera demandée à l'utilisateur et auront toutes le même format et contiendront obligatoirement les champs suivants :

- **type** : indique le type de la donnée (choix dans une liste, texte, image, etc.) ;
- **title** : nom du champ à afficher.

En fonction du type, d'autres champs pourront apparaître comme **items** dans le cas d'une liste pour décrire les entrées possibles.

Voici un exemple de fichier de configuration du serveur, **config.yml** :

```
01: host: localhost:8080
02:
03: categorie:
04:     type: list
05:     title: Catégorie
06:     items:
07:         - espaces verts
08:         - voirie
09:         - bâtiments
10:     mails:
11:         - mail@adresse.com
12:         - mail@adresse.com, mail_2@autre_adresse.net
13:         - mail_2@autre_adresse.net
14:
15: status:
16:     type: list
```

```
17:     title: Priorité
18:     items:
19:         - critique
20:         - urgente
21:         - faible
22:         - moyenne
23:         - résolu
24:
25: description:
26:     type: text
27:     title: Description
28:
29: image:
30:     type: image
31:     title: Photo(s)
32:     optionnal: true
33:     multiple: true
```

Pour lire un fichier yaml à l'aide de Node.JS nous allons utiliser le module **js-yaml** :

```
$ npm init # Pour créer le fichier package.json
$ npm install --save js-yaml
```

La lecture d'un fichier se fait alors de manière très simple en utilisant la fonction **`safeLoad()`**. Voici un exemple de lecture de notre fichier de configuration dans **`server.js`** :

```
01: yaml = require('js-yaml');
02: fs   = require('fs');
03:
04: try {
05:   var config = yaml.safeLoad(fs.readFileSync('config.yml',
'utf8'));
06:   console.log(config);
07: } catch (e) {
08:   console.log(e);
09: }
```

Au lancement du programme nous obtenons les données dans la variable **`config`** au format json :

```
$ node server.js
{ host: 'localhost:8080',
  categorie:
   { type: 'list',
     title: 'Catégorie',
     items: [ 'espaces verts', 'voirie', 'bâtiments' ],
     mails:
      [ 'mail@adresse.com',
        'mail@adresse.com, mail_2@adresse.com',
        'mail_2@adresse.com' ] },
...
  image:
   { type: 'image',
     title: 'Photo(s)',
     optionnal: true,
     multiple: true } }
```

À ce stade nous n'avons traité aucune erreur. Nous allons donc nous assurer que les champs utilisés dans le fichier de configuration sont corrects et que les champs obligatoires sont bien présents.

PHASE 2

Validation du fichier de configuration

Nous pouvons maintenant structurer notre code de manière à vérifier la validité des données :

- la fonction **`readConfiguration()`** sera chargée de lire le fichier yaml de configuration (adaptation du code précédent) ;
- **`validateConfiguration()`** vérifiera la validité des données du fichier.

```
01: var yaml = require('js-yaml');
02: var fs   = require('fs');
03:
04: function syntaxError(msg) {
05:   console.error('Syntax error in configuration file : ' +
msg);
06:   process.exit(2);
07: }
08:
09: function readConfiguration() {
10:   try {
11:     return yaml.safeLoad(fs.readFileSync('config.yml',
'utf8'));
12:   } catch (e) {
13:     if (e.errno == -2) {
14:       console.error('Error : Can\'t open file ' + e.path);
15:       process.exit(1);
16:     }
17:     else if (e.name == 'YAMLException') {
18:       syntaxError(e.reason);
19:     }
20:     else {
21:       console.log(e);
22:       process.exit(3);
23:     }
24:   }
25: };
26:
27: function validateConfiguration(config) {
28:   if (!('host' in config)) {
29:     syntaxError('host key MUST be present');
30:   }
31:
32:   host = config['host'].split(':');
33:   if (host.length != 2) {
34:     syntaxError('host key format is url:port');
35:   }
36:   if (isNaN(parseInt(host[1], 10))) {
37:     syntaxError('in host key the port MUST be an integer');
```

```
38:   }
39:
40:   for (key in config) {
41:     if (key == 'host') {
42:       continue;
43:     }
44:     if (!('type' in config[key])) {
45:       syntaxError(key + ' key has no \'type\' entry');
46:     } else {
47:       if (config[key]['type'] == 'list') {
48:         if (!('items' in config[key])) {
49:           syntaxError(' no \'items\' entry in list ' + key);
50:         }
51:       }
52:     }
53:     if (!('title' in config[key])) {
54:       syntaxError(key + ' key has no \'title\' entry');
55:     }
56:
57:   }
58: };
59:
60: config = readConfiguration();
61: validateConfiguration(config);
62: console.log(config)
```

Au niveau de la validation (lignes 27 à 58), nous testons la présence de l'entrée **host** (lignes 28 à 30) puis nous vérifions que l'adresse a bien le format **url:port** où **port** est un entier (lignes 32 à 38). Enfin, la boucle des lignes 40 à 57 parcourt les clés du tableau **config** : s'il s'agit de l'entrée **host** on ne traite pas le cas (lignes 41 à 43), et on vérifie la présence des entrées **type** (lignes 44 à 46) et **title** (lignes 53 à 55). Si le type de l'entrée est **list**, alors nous vérifions la présence de l'entrée **items** (lignes 47 à 51) qui définie les entrées de ladite liste.

La place étant limitée, nous n'avons testé ici que les éléments principaux du fichier mais il pourrait être intéressant de tester toutes les entrées pour s'assurer qu'il n'y ait pas d'éléments incompatibles ou non traités.

PHASE 3

Journey pour un routeur json rapide à développer

Journey est un module permettant de développer rapidement un routeur json. Par « routeur JSON » on entend un programme capable d'analyser une url et d'y répondre par des informations en json. Comme tout serveur REST notre routeur comprendra les requêtes get, post, delete et put.

L'installation est classique :

```
$ npm install --save journey
```

Voici un petit exemple d'utilisation :

```
01: var PORT = 8080;
02:
03: var journey = require('journey');
04: var http = require('http');
05:
06: var router = new(journey.Router);
07:
08: function sendJSON(res, json, code) {
09:   if (typeof code === 'undefined') {
10:     code = 200;
11:   }
12:   res.send(
13:     code,
14:     {'Content-Type': 'application/json'},
15:     json
16:   );
17: }
18:
19: router.get('/hello').bind(function (req, res, params) {
20:   if (params.name === undefined) {
21:     sendJSON(res, {}, 400);
22:   } else {
23:     sendJSON(res, {msg: 'Hello, ' + params.name + '!'});
24:   }
25: });
26:
27: router.post('/hello').bind(function (req, res, params) {
28:   if (params.name === undefined) {
29:     sendJSON(res, {}, 400);
30:   } else {
31:     sendJSON(res, {msg: 'Hello (POST), ' + params.name +
'!'});
32:   }
33: });
34:
35: server = http.createServer(function (req, res) {
36:   var body = '';
37:
38:   req.addListener('data', function (chunk) {
39:     body += chunk;
40:   });
41:   req.addListener('end', function () {
42:     router.handle(req, body, function (result) {
43:       res.writeHead(result.status, result.headers);
44:       res.end(result.body);
45:     });
46:   });
47: });
48: server.listen(PORT);
49:
50: console.log('Server running on ' + PORT);
```

Le module journey est chargé en ligne 3 et le routeur est créé en ligne 6. Nous définissons une fonction **sendJSON()** dans les lignes 8 à 17. Cette fonction est chargée de renvoyer le code http, l'en-tête et les données json. Par défaut le code http de retour sera **200** (OK - succès).

Il faut ensuite définir les urls qui pourront être servies. Dans cet exemple **/hello** pourra être servi en get (lignes 19

à 25) ou en post (lignes 27 à 33) et attendra un paramètre **name**. Si ce paramètre est absent nous renverrons un code http **400** (*Bad Request*).

Le serveur est créé et lancé dans les lignes 35 à 48 avec mise en place du dispatcheur dans les lignes 41 à 46.

Pour tester notre programme, après avoir lancé le serveur par **node server.js**, nous utiliserons **curl** permettant de lancer des requêtes en get ou en post :

```
$ curl localhost:8080/hello?name=Tristan
{"msg":"Hello, Tristan!"}
$ curl localhost:8080/hello?toto=10
{}
$ curl --data 'name=Tristan' localhost:8080/hello
{"msg":"Hello (POST), Tristan!"}
```

PHASE 4

Incorporation de Journey à notre projet

Pour notre projet, un client doit pouvoir accéder aux services suivants :

- **/ticketFormat** (get) : renvoi le format du ticket (pour l'instant nous nous contenterons des données lues dans le fichier de configuration) ;
- **/tickets** (get) : renvoi la liste des tickets actifs ;
- **/newTicket** (put) : ajoute un nouveau ticket.

En dehors du premier service, nous aurons besoin d'une base de données et nous laisserons donc temporairement les méthodes incomplètes :

```
001: var yaml = require('js-yaml');
002: var fs   = require('fs');
003: var http = require('http');
004: var journey = require('journey');
005:
006:
007: function sendJSON(res, json, code) {
...
016: }
017:
018: function syntaxError(msg) {
...
021: }
022:
023: function readConfiguration() {
...
039: };
040:
041: function validateConfiguration(config) {
...
072: };
073:
074: function createRouter(config) {
075:   var router = new(journey.Router);
076:
077:   router.get('/ticketFormat').bind(function (req, res,
params) {
078:     if (Object.keys(params).length !== 0) {
079:       sendJSON(res, {}, 400);
080:     } else {
081:       sendJSON(res, config);
082:     }
083:   });
084:
085:   router.get('/tickets').bind(function (req, res, params) {
086:     if (Object.keys(params).length !== 0) {
087:       sendJSON(res, {}, 400);
088:     } else {
089:       sendJSON(res, {msg: 'WORK IN PROGRESS'});
090:     }
091:   });
092:
093:   router.put('/newTicket').bind(function (req, res, params) {
094:     if (Object.keys(params).length === 0) {
095:       sendJSON(res, {}, 400);
096:     } else {
097:       sendJSON(res, {msg: 'WORK IN PROGRESS'});
098:     }
099:   });
100:
101:   return router
102: }
103:
104: function startServer() {
105:   var config = readConfiguration();
106:   validateConfiguration(config);
107:
108:   var host = config['host'].split(':');
109:
110:   var router = createRouter(config)
111:
112:   var server = http.createServer(function (req, res) {
113:     var body = '';
114:
115:     req.addListener('data', function (chunk) {
116:       body += chunk;
117:     });
118:     req.addListener('end', function () {
119:       router.handle(req, body, function (result) {
120:         res.writeHead(result.status, result.headers);
121:         res.end(result.body);
122:       });
123:     });
124:   });
125:   server.listen(host[1]);
126:
127:   console.log('Server running on ' + host[1]);
128: }
129:
130:
131: startServer();
```

Il n'y a rien de nouveau dans ce code, seulement une réorganisation des codes précédents. On peut juste noter l'utilisation du test sur **`Object.keys(params).length`** dans les lignes 78, 86 et 94. Il s'agit d'une façon de tester si le tableau associatif **`params`** est vide sans utiliser de *framework*. Avec Node.js en version 6, EcmaScript 2015 autorise la déclaration d'un paramètre par défaut dans l'en-tête des fonctions, ce qui simplifie nettement l'écriture.

Pour tester le serveur, il faut là encore le lancer puis utiliser curl :

```
$ curl localhost:8080/ticketFormat
{"host":"localhost:8080","categorie":{"type":"list","title":"Catég
orie","items":["espaces verts","voirie","bâtiments"],"mails":[...]
},"status":{"type":"list","title":"Priorité","items":[...]},"descr
iption":{"type":"text","title":"Description"},"image":{"type":"ima
ge","title":"Photo(s)","optionnal":true,"multiple":true}}
$ curl localhost:8080/ticketFormat?name=toto
{}
$ curl localhost:8080/tickets
{"msg":"WORK IN PROGRESS"}
$ curl localhost:8080/tickets?param=1
{}
$ curl -X PUT -d '{param: 1}' localhost:8080/newTicket
{"msg":"WORK IN PROGRESS"}
$ curl -X PUT localhost:8080/newTicket
{}
```

PHASE 5

Et la base de données ?

L'utilisation de Node.js a été dictée par notre volonté de développer un serveur « léger ». Pour la base de données, nous allons rester dans la même optique en optant pour une base NoSQL **Redis** (base de type clé/valeur). Comme toujours avec Node.js, il existe un module :

```
$ npm install --save redis
```

Attention !

Le module se nomme **node-redis** mais il est référencé en tant que **redis** dans npm. Les informations que vous pourriez trouver sur un module node-redis parlent donc bien du même module.

Pour comprendre comment utiliser Redis avec Node.js nous allons y stocker manuellement des données correspondant au serveur et au format d'un ticket. Notre fichier s'appellera **`redis_test.js`** :

```
01: var redis = require('redis')
02:
03: var client = redis.createClient();
04:
05: client.on('connect', function () {
06:   console.log('Connecté à la base Redis');
07: });
08: client.on('error', function (err) {
09:   console.log('Erreur : ' + err);
10: });
11:
12: client.set('host', 'localhost:8080', function (err, reply) {
13:   if (err !== null) {
14:     console.warn('Erreur : ' + err);
15:   }
16: });
17:
18: client.hmset('ticketFormat', {
19:   'categorie': {
20:     'type': 'list',
21:     'title': 'Catégorie',
22:     'items': [
23:       'espaces verts',
24:       'voirie',
25:       'bâtiments'
26:     ],
27:     // ...
28:   },
29:   // ...
30:   'description': {
31:     'type': 'text',
32:     'title': 'Description'
33:   },
34:   // ...
35: }, function (err, reply) {
36:   if (err !== null) {
37:     console.warn('Erreur : ' + err);
38:   }
39: });
40:
41: client.exists('ticketFormat', function (err, reply) {
42:   if (reply === 1) {
43:     console.log('Clé ticketFormat trouvée');
44:   } else {
45:     console.log('La clé ticketFormat n\'existe pas');
46:   }
47: });
48:
49: client.get('host', function (err, reply) {
50:   if (err === null) {
51:     console.log('Valeur de host : ' + reply);
52:   }
53: });
54:
55: client.hgetall('ticketFormat', function (err, reply) {
56:   if (err === null) {
57:     console.log('Valeur de ticketFormat : ');
58:     console.log(reply);
59:   }
60: });
61:
62: client.del('host', function(err, reply) {
63:   if (err === null) {
64:     console.log('La clé host a été supprimée');
65:   }
66: });
```

En ligne 1 nous commençons par charger le module **`redis`** puis en ligne 3 nous créons l'objet de connexion (par défaut l'adresse employée sera 127.0.0.1:6379, pour une

autre adresse du serveur redis vous pourrez utiliser **redis.createClient(port, host)**). Dans les lignes 5 à 10 nous testons l'état de la connexion de manière à afficher un message en cas de problème. L'insertion d'un couple clé/valeur « simple » se fait à l'aide de la méthode **set()** comme vous pouvez le voir dans les lignes 12 à 16. Le dernier paramètre des méthodes d'accès à Redis est une fonction anonyme permettant de traiter les erreurs (**err** différent de **null**) et les données renvoyées (**reply**). Pour l'insertion d'un objet nous utilisons la méthode **hmset()** (lignes 18 à 39). Toutes les méthodes d'accès suivant le même schéma il y a peu d'explications à donner : **exists()** permet de tester l'existence d'une clé (lignes 41 à 47), **get()** permet d'accéder à la valeur stockée pour une clé (lignes 49 à 53), **hgetall()** a la même fonction mais pour une valeur stockée sous forme d'objet (lignes 55 à 60) et **del()** permet de supprimer une entrée (lignes 62 à 66).

Au lancement le résultat ne sera pas celui attendu...

```
$ node redis_test.js
Erreur : Error: Redis connection to 127.0.0.1:6379 failed -
connect ECONNREFUSED 127.0.0.1:6379
Erreur : Error: Redis connection to 127.0.0.1:6379 failed -
connect ECONNREFUSED 127.0.0.1:6379
Erreur : Error: Redis connection to 127.0.0.1:6379 failed -
connect ECONNREFUSED 127.0.0.1:6379
```

En y réfléchissant bien nous essayons de nous connecter au serveur Redis... mais l'avons-nous installé et lancé ? Bien sûr que non ! Si vous disposez d'une distribution basée sur Debian, il vous faudra exécuter :

```
$ sudo apt-get install redis-server
$ sudo service redis-server start
```

Nous pouvons maintenant relancer notre programme :

```
$ node redis_test.js
Connecté à la base Redis
node_redis: Deprecated: The HMSET command contains a argument of
type Object.
This is converted to "[object Object]" by using .toString() now
and will return an error from v.3.0 on.
Please handle this in your code to make sure everything works as
you intended it to.
node_redis: Deprecated: The HMSET command contains a argument of
type Object.
This is converted to "[object Object]" by using .toString() now
and will return an error from v.3.0 on.
Please handle this in your code to make sure everything works as
you intended it to.
Clé ticketFormat trouvée
Valeur de host : localhost:8080
Valeur de ticketFormat :
{ categorie: '[object Object]', description: '[object Object]' }
La clé host a été supprimée
```

Vous pouvez constaté que l'usage que nous avons fait de **hmset()** est considéré comme déprécié. Cela est dû au fait que Redis ne supporte pas les objets imbriqués : toutes les valeurs de propriété de l'objet seront converties sous forme de chaînes avant d'être stockées. On peut avoir l'impression que cela ne change rien... erreur ! La clé **categorie** fait référence à **'[object Object]'** c'est-à-dire à une chaîne de caractères et en aucun cas un objet !!! **hmset()** et **hgetall()** sont donc à réserver pour les objets « simples » ne contenant aucun niveau d'imbrication. Dans les autres cas, nous utiliserons **JSON.stringify()** pour sérialiser notre objet et **JSON.parse()** pour le désérialiser. Voici les modifications à apporter au code :

```
...
18: client.set('formatTicket', JSON.stringify({
19:   'categorie': {
20:     'type': 'list',
...
30:   'description': {
31:     'type': 'text',
32:     'title': 'Description'
33:   },
34:   // ...
35: }), function (err, reply) {
36:   if (err !== null) {
37:     console.warn('Erreur : ' + err);
38:   }
39: });
...
55: client.get('formatTicket', function (err, reply) {
56:   if (err === null) {
57:     console.log('Valeur de formatTicket : ');
58:     console.log(JSON.parse(reply));
59:   }
60: });
...
```

PHASE 6

Intégration de Redis à notre projet

Après avoir compris le fonctionnement de Redis, il n'y a plus qu'à l'intégrer à notre projet tout en gardant à l'esprit que la logique d'une base NoSQL est bien différente d'une base relationnelle : nous ne gérerons que des couples de clés/valeurs (il faudra notamment être capable d'identifier les tickets par la suite).

```
001: var yaml = require('js-yaml');
002: var fs   = require('fs');
003: var http = require('http');
004: var journey = require('journey');
```

```
005: var redis = require('redis');
006: var colors = require('colors');
007:
008: colors.setTheme({
009:   redis_ok: 'green',
010:   redis_warn: 'yellow',
011:   redis_debug : 'blue',
012:   redis_error: ['red', 'bold']
013: });
...
083: function createRouter(config, clientRedis) {
084:   var router = new(journey.Router);
085:
086:   router.get('/ticketFormat').bind(function (req, res, params) {
087:     if (Object.keys(params).length !== 0) {
088:       sendJSON(res, {}, 400);
089:     } else {
090:       clientRedis.get('ticketFormat', function (err, reply) {
091:         if (err === null) {
092:           console.log(('[REDIS] Clé ticketFormat trouvée!').
redis_ok);
093:           console.log(reply.redis_debug);
094:           sendJSON(res, reply);
095:         } else {
096:           console.warn(('[REDIS] La clé ticketFormat n\'existe
pas').redis_warn);
097:         }
098:       });
099:     }
100:   });
...
121: function connectRedis() {
122:   var client = redis.createClient();
123:
124:   client.on('connect', function () {
125:     console.log('[REDIS] Connecté à la base Redis'.redis_ok);
126:   });
127:   client.on('error', function (err) {
128:     console.log(('[REDIS] Erreur : ' + err).redis_error);
129:   });
130:
131:   return client;
132: }
133:
134: function addRedis(client, key, object) {
135:   client.set(key, JSON.stringify(object), function (err, reply) {
136:     if (err !== null) {
137:       console.warn(('[REDIS] Erreur : ' + err).redis_error);
138:     } else {
139:       console.log('[REDIS] Objet ajouté à la base'.redis_ok);
140:       console.log(JSON.stringify(object).redis_debug);
141:     }
142:   });
143: }
144:
145: /*function getRedis(client, key) {
146:   client.get(key, function (err, reply) {
147:     if (err === null) {
148:       console.log(('[REDIS] Clé ' + key + ' trouvée!').redis_ok);
149:       console.log(reply.redis_debug);
150:       return reply;
151:     } else {
152:       console.warn(('[REDIS] La clé ' + key + ' n\'existe pas').
redis_warn);
153:     }
154:   });
155: }*/
156:
157: function startServer() {
158:   var config = readConfiguration();
159:   validateConfiguration(config);
160:
161:   var host = config['host'].split(':');
162:
163:   delete config['host'];
164:
165:   var client = connectRedis();
166:
167:   client.exists('ticketFormat', function (err, reply) {
168:     if (reply === 1) {
169:       console.log('[REDIS] Format des tickets déjà présent dans
la base'.redis_warn);
170:     } else {
171:       addRedis(client, 'ticketFormat', config);
172:       console.log('[REDIS] Format des tickets ajouté à la base'.
redis_ok);
173:     }
174:   });
...
```

En ligne 5 il faut bien entendu charger le module redis. Les lignes 6 à 13 ont été ajoutées pour gérer la couleur en mode console grâce au module **colors** :

```
$ npm install --save colors
```

Pour accéder à la base, j'ai défini trois fonctions :

- **`connectRedis()`** dans les lignes 121 à 132 : il s'agit de la connexion à la base telle que définie précédemment ;
- **`addRedis()`** (lignes 134 à 143) : pour ajouter un objet à la base en le sérialisant ;
- et **`getRedis()`** dans les lignes 145 à 155 pour lire un objet dans la base... Cette fonction est commentée et n'est pas utilisée dans **`createRouter()`** (voir lignes 90 à 94) à cause d'un problème de traitement asynchrone. J'ai préféré laisser visible la première version de ma fonction, celle que l'on aura tendance à écrire de prime abord avant de constater que ça ne marche pas... La solution consiste à utiliser une fonction de rappel pour s'assurer de la fin du traitement. La fonction **`getRedis()`** devient alors :

```
145: function getRedis(client, key, callback) {
146:   client.get(key, function (err, reply) {
147:     if (err === null) {
148:       console.log(('[REDIS] Clé ' + key + ' trouvée!').
redis_ok);
149:       console.log(reply.redis_debug);
150:       callback(reply);
151:     } else {
152:       console.warn(('[REDIS] La clé ' + key + ' n\'existe
pas').redis_warn);
153:     }
154:   });
155: }
```

Et l'appel de **getRedis()** dans **createRouter()** est :

```
083: function createRouter(config, clientRedis) {
084:   var router = new(journey.Router);
085:
086:   router.get('/ticketFormat').bind(function (req, res,
params) {
087:     if (Object.keys(params).length !== 0) {
088:       sendJSON(res, {}, 400);
089:     } else {
090:       getRedis(clientRedis, 'ticketFormat', function
(reply) {
091:         console.log(reply.redis_debug);
092:         sendJSON(res, reply);
093:       });
094:     }
095:   });
...
```

Pour revenir aux explications du code du serveur, en ligne 163 on supprime la clé **host** de **config**, en ligne 165 on se connecte à la base à l'aide de **connectRedis()** et dans les lignes 167 à 174 si l'entrée **ticketFormat** n'existe pas on la crée et on y stocke le contenu de **config**.

LE RÉSULTAT

Nous avons écrit et testé la structure d'un gestionnaire de tickets configurable par fichier yaml. Pour que celui-ci soit pleinement fonctionnel il faudrait encore compléter les commandes REST **tickets** et **newTicket** (ce qui ne présente plus de difficulté) et lui adjoindre un client autre que **curl**...

Pour une mise en production de l'outil, il faudrait s'assurer que le serveur est sécurisé. Pour cela je vous renvoie au hors-série 85 de GNU/Linux Magazine avec les articles de Sébastien Gioria et Nicolas Gadachoit sur le sujet... ■

CHEZ LES BARBUS – JAVA & SÉCURITÉ

Romain PELISSE [Sustain Developer @Red Hat] & François Le Droff [Achitecte @Adobe]

« Chez les Barbus - Java & Sécurité » c'est le titre de la conférence donnée par François Le Droff et Romain Pelisse à l'occasion de Devoxx France 2015. Cet article propose d'en reprendre le contenu de manière plus didactique et adaptée à ce nouveau support, sous la forme d'un dialogue.

Mots-clés : Java, Sécurité, DevOps, HTTPS/SSL, OAuth, JHipster

Résumé

À l'aide d'un cas pratique, cet article décrit, sous la forme d'un dialogue entre deux experts, les deux auteurs, l'ensemble des problématiques liées à la mise en place de la sécurité autour d'une application « Web » - aussi simple et peu critique soit-elle.

François : Je suis en train de monter une sympathique petite application en interne, mais je m'inquiète un peu de l'aspect sécurité, je ne me sens pas armé pour un audit ! On en est encore au stade du prototype, mon architecture n'est pas sèche et mes développements sont encore en cours... Je n'ai même pas encore ***staffé*** toute l'équipe... Bref, comment m'y prendre ?

Romain : Les audits *adhoc*, qui arrivent à la fin des développements, quand justement les personnes clés du projet sont parties, ça ne marche pas. La sécurité se fait en équipe, dans la durée, tout au long du projet. Cependant, il est vrai qu'il est difficile de trouver un « bon point de départ ».

1 Threat Modelling

François : Comme « point de départ » je te propose une méthode que je trouve très intéressante. Elle est poussée par **Microsoft** et **OWASP**. Elle s'appelle le *« Threat Modeling »*. Elle donne une bonne vision d'ensemble; elle permet d'une part d'identifier les menaces, et d'autre part, de définir la priorité des points à adresser à l'aide de deux acronymes: **STRIDE** et **DREAD**.

Romain : Commençons alors par STRIDE, qu'est-ce que ça veut dire ?

François : L'acronyme STRIDE reflète une catégorisation des attaques possibles selon les types d'exploitation qui leur sont associés (**S** pour *Spoofing*, **T** pour *Tampering*, **R** pour *Repudiation*, **I** pour *Information disclosure*, **D** pour *Denial of service*, **E** pour *Elevation of privilege*). À chaque catégorie correspond des mesures à mettre en place.

Romain : Restons didactiques et brefs, sinon, on va y passer tout l'article ! Prenons donc juste un exemple concret: que signifie le **R** de *Repudiation* ?

François : La menace, le risque de *Repudiation* sont ceux de voir ses utilisateurs contester les transactions faites avec leurs identifiants et depuis leur machine. Une des solutions pour réduire cette menace est de mettre en place un *« audit trail »* sur chacun de tes tiers applicatifs.

Romain : En effet, si on doit imputer à quelqu'un une transaction malhonnête, c'est bien de pouvoir le prouver sans l'ombre d'un doute. Passons au second acronyme: DREAD.

François : Avec l'acronyme DREAD, le *Threat Modeling* propose un système de classification pour comparer, qualifier et définir l'importance des risques associés à chaque menace. C'est une mesure du risque de sécurité. Elle se calcule sur la base de la moyenne des valeurs assignées à chacune des catégories suivantes: *Damage potential, Reproductibility, Exploitability, Affected users, Discoverability.*

Romain : Le résultat final: un schéma d'architecture orientée sécurité offrant une vue d'ensemble des menaces pesant sur ton système.

François : C'est aussi un bon outil d'aide à la décision... malheureusement souvent ignoré par les développeurs.

Romain : Triste réalité, car cette méthode semble être une arme parfaite pour prendre conscience des menaces qui pèsent sur un système. Et ceci, dans un langage et un format qu'un *manager* (même récalcitrant) pourra comprendre et réutiliser pour convaincre sa propre hiérarchie.

François : Et comme par un effet miroir, ce même inventaire peut permettre au *manager* de vérifier que ses développeurs n'ont pas (par naïveté, par manque de connaissance ou simplement par manque de temps) négligé l'aspect sécurité.

Romain : Vaste sujet que le *Threat modeling*. Maintenant que nous avons une méthodologie pour avoir un bon point de départ, regardons donc ta petite application.

1.1 Notre cas d'étude

François : C'est une application Web très simple. Elle n'est pas conçue pour spéculer sur les marchés financiers ni pour aider au marketing ou toute autre surveillance généralisée. Elle n'est donc pas très *big data*, tu n'y trouveras ni *hadoop*, ni *data lake*, ni *machine learning*.

Romain : Ce n'est pas très bon pour ton aura, rassure-moi, tu utilises au moins **Docker**, non ?

François : Non. C'est juste une bonne vieille application de gestion, une plomberie et une API http faite en **Java** avec une interface en **html**. Et pour gagner du temps, je l'ai générée avec **jHipster [1]**.

Romain : C'est quoi jHipster ? Un énième *framework* MVC ? Un truc qui **SpringBoot** pour lancer un container **Scala** écrit en **JRuby** qui exécute des greffons **Groovy** ?

François : Presque! C'est un générateur d'applications Web basé sur **Yeoman**; l'application générée s'appuie sur une architecture assez simple et au goût du jour: Java avec SpringBoot pour la partie serveur, et **css**, html, **JavaScript** et **AngularJS** pour la partie cliente.

Romain : Très bien, mais du point de vue sécurité, tout ceci ne semble pas apporter grand-chose...

François : Si, sous le capot tu trouves **Spring Security [2]**, et ce *framework* Java offre une bonne base de travail pour non seulement gérer l'authentification, les autorisations et les rôles, mais aussi pour contrer des attaques assez « classiques » dans le monde Web.

Romain : Oui, comme les attaques de type *man in the middle*.

François : Oui, en mettant en place **HSTS [3]**, qui indique au client que seul le protocole https sera utilisé.

Romain : Oui, mais encore...

François : Tu y trouveras également des stratégies pour éviter le *click-jacking* et les attaques XSS et les attaques de type *Cross site scripting forgery*, pour lesquelles il fournit une solution à base de *CSRF Token* **[4]**.

Romain : Je vois : Spring Security te fournit donc plein d'acronymes en anglais pour te la péter... Et sinon niveau audit ?

François : Il fournit aussi une bonne trame pour mettre en place l'audit et les logs relatifs à la sécurité.

Romain : OK, tu me l'as vendu !

2 Firewall

François : Mais tout ça, au fond, n'est peut-être pas si utile: nos serveurs sont à l'abri, dans un VLAN cloisonné en intranet, derrière une belle ribambelle de *firewalls*.

Romain : Et alors ? Quel rapport ? Un firewall n'a jamais servi à sécuriser une infrastructure, ça se saurait... Un *firewall*, ça sert juste à faire de la qualité réseau ; en gros, à épargner le CPU du système: à ignorer chaque trame réseau correspondant à un port « non utilisé ».

François : Les *firewalls* réduiront donc le trafic « inutile » et donc potentiellement « nuisible », non ?

Romain : Oui, sauf que généralement quand tu as un service qui tourne sur un port, c'est que tu en as besoin, donc du coup, le port est ouvert. Se contenter de démarrer un *firewall* pour bloquer les ports inutilisés ne suffit pas !

François : Qu'est-ce qu'on fait des ports ouverts alors ?

Romain : On en vérifie l'usage ! Si tu as ouvert un port pour envoyer des mails, tu ne laisseras passer que du SMTP. Mais attention, une politique agressive de fermeture systématique des ports ne donne qu'une illusion de sécurité...

François : ... et casse les pieds à tout le monde dans la société...

Romain : On est bien d'accord. La bonne solution est d'implémenter un filtrage applicatif et vérifier ainsi que le contenu de l'échange correspond au protocole utilisé. Certains *firewalls* savent le faire, mais ce filtrage sera souvent mis en place à coup de *reverse proxy*.

François : Force est de constater que les équipes de développement ont rarement la main sur la configuration du *firewall*, souvent sous la responsabilité exclusive des équipes d'infrastructures. De plus, ces équipes dédiées aux configurations de *firewalls* n'auront jamais une connaissance suffisante de la solution pour mettre en place son filtrage applicatif de manière efficace et pérenne. Je serai donc ravi de mettre en place un *reverse proxy*. Par où je commence ?

3 Reverse Proxy

Romain : Une solution simple est de mettre un serveur Web en frontal et de s'en servir comme d'un *reverse proxy* (RP). Pour faire simple, tu diriges tout le trafic à travers ton serveur « Web » frontal: ce serveur frontal vérifiera que les requêtes sont conformes aux attentes de l'application, avant de les faire suivre au serveur d'application.

François : Tu pourras ainsi te servir de ton *reverse proxy* pour mettre en place la validation de requêtes http, pour ajouter et valider des *tokens CSRF*, ou encore pour implémenter du *throttling* ou du *rate limiting*... Mais ce *reverse proxy*, c'est comme tout le reste, on peut tout autant me le « hacker », non ?

Romain : Oui, mais plus difficilement. Dans les faits, la plupart des « hacks » se basent sur une manipulation de la réaction du serveur distant. On abuse ce dernier, en lui envoyant des requêtes qu'il accepte, mais qui l'amènent à avoir un comportement invalide. Or le *reverse proxy* n'interagit pas directement avec ces requêtes. Il se contente d'en analyser le contenu et de, le cas échéant, ne pas les transmettre au serveur cible. Il reste possible d'abuser son mécanisme d'analyse, mais c'est une fenêtre de tir beaucoup moins grande qu'avant. En outre, comme il n'y a pas d'interaction directe entre le *reverse proxy* et l'attaquant, il n'est pas aisé pour l'attaquant de réaliser que ses requêtes malicieuses ne seront jamais transmises et interprétées par le serveur cible.

François : Pour résumer Romain, on peut dire qu'une des clés d'une bonne sécurité applicative sur une application « Web », c'est la mise en place d'un filtrage applicatif adapté. Si on se contente de simples *firewalls*, c'est un peu comme disposer d'une frontière sans douanier.

4 Données et chiffrement

Romain : On filtre, on nettoie mais on veut aussi recevoir et envoyer des données sur ce fameux réseau, non ?

François : Bien sûr et je ne veux surtout pas que ces données soient compromises. Mon application fournit un beau *mashup* de données, c'est un simple *« employee productivity tool »*, mais, mine de rien, voilà les informations qui y transitent :

- l'adresse, le numéro de téléphone des employés – soit du **PII** (*Personally Identifiable Information*), et ceci toute société est tenue de le garder confidentiel ;
- des infos sur les employés, comme leurs salaires – et ça, c'est aussi confidentiel, mais aussi super sensible ;
- et enfin des informations internes sur des marchés en cours de négociation – c'est non seulement des informations « internes » mais en plus d'accès restreint !

Romain : Ce qui veut dire que tu ne peux rien faire circuler en clair. Mais ce n'est pas surprenant, c'est le lot commun de la plupart des applications d'entreprise.

François : Oui, il va falloir chiffrer de bout en bout: chiffrer les flux de données qui passent dans les câbles, mais aussi les données au repos, sur les disques.

Romain : Comment as-tu mis ça en place ?

5 Chiffrer le front

François : Alors, commençons par le front : on prévoit évidemment de tout servir en https.

Romain : SSL c'est *secure*, mais faire ça correctement c'est tout un art.

François : Oui, garantir une implémentation robuste du protocole https demande une surveillance et une veille techno constante. Même les géants du Web doivent gérer des situations de crise : faire face à des choix d'algorithmes de chiffrement vieillissants et donc vulnérables, faire face à des vols de certificats, ou encore à la propagation de vrais-faux certificats : certificats créés par des attaquants, associés à leurs domaines ou sous domaines et qui sont « trustés » par la plupart des navigateurs.

6 Chiffrer le back

Romain : Admettons que l'échange entre le serveur et ses clients soit correctement chiffré, il faut maintenant s'assurer que la donnée reste confidentielle, et donc chiffrée quand elle circule (entre ton serveur d'application et ta base de données) et quand elle est au repos, *persistée* sur un disque.

François : C'est fait, mais ce fut laborieux. Pour la base de données, nous avions choisi **MongoDB**; et à l'époque, le support SSL n'était pas fourni dans les distributions MongoDB ; il a donc fallu reconstruire MongoDB depuis les sources et *patcher* l'authentification par certificat (imposé par le transport SSL dans MongoDB) dans le *stack* « Spring-data ».

Romain : Bien, mais quid du chiffrage au repos ?

François : Là, j'hésite encore. On pourrait chiffrer le volume, mais je crains un impact sur les performances et une complexité dans l'administration de la base de données.

Romain : Reste l'option du chiffrage applicatif. C'est l'application, elle-même, qui va chiffrer ses données. Elle sera seule à pouvoir les déchiffrer.

François : Oui, du coup l'administration de MongoDB redevient plus facile, mais ça implique une charge de développement et une gestion d'un secret supplémentaire. De plus, si tu utilises un bon chiffrage (donc un chiffrage asymétrique) tu ne pourras faire ni recherche ni indexation sur les champs chiffrés.

Romain : Il n'y a pas de solution miracle. Comme toujours, c'est une question de compromis. Maintenant, l'avantage aussi du chiffrage applicatif, c'est aussi d'avoir une meilleure granularité. Si l'application est compromise, seules ses données pourront être exploitables par l'attaquant. Les données des autres applications resteront illisibles (du moins quelque temps).

7 Authentification

Romain : Je constate que jHipster propose par défaut sa propre base de données d'utilisateur. C'est pratique pour développer un prototype, mais alors par contre, c'est inenvisageable pour la production.

François : Oui, c'est une véritable plaie, non seulement pour la sécurité mais également pour l'expérience utilisateur. En tant qu'utilisateur, il est assez insupportable de devoir créer un énième nom d'utilisateur et mot de passe aux contraintes absurdes.

Romain : Surtout que la plupart des entreprises ont des solutions de gestions d'identité en place.

François : Oui, c'est notre but : notre serveur d'application se contentera de n'être qu'un fournisseur de service, l'authentification sera déléguée à un fournisseur d'identité mutualisé (s'appuyant sur le standard **SAML**) et l'autorisation à un serveur implémentant le standard **oAuth2**.

8 SAML

Romain : Voilà encore une pluie d'acronymes. Commençons par SAML - qu'est-ce que c'est exactement ?

François : C'est un standard, un protocole de sécurité, il permet de mettre en place l'authentification unique (en anglais *Single Sign-On* ou **SSO**) sur le Web.

Romain : Et si je comprends bien, une fois identifié sur un site, tu peux te connecter à d'autres sites, avec le même navigateur, sans avoir à entrer à nouveau tes identifiants. C'est effectivement bien mieux en termes d'expérience utilisateur. Est-ce compliqué à mettre en place ?

François : Pas tant que ça, pour démarrer je te conseille le site *« SSO Circle »* **[5]**, il t'aidera à tester ta solution.

Romain : Comment ça marche ? Qu'est-ce que je dois coder ? Est-ce qu'implémenter SAML est à la portée de n'importe quel développeur Java ?

François : Oui, et comme souvent, la seule complexité réside dans la configuration. Tu dois récupérer les métadonnées du fournisseur d'identité (en anglais *Identity Provider* ou **IdP**) et les transmettre au moteur de ton client SAML. Ces métadonnées contiennent certificats et URLs associés à la redirection en vue d'une authentification ou tout au moins d'une vérification d'identité.

Romain : Et de ton côté, j'imagine aussi que ton appli fournit quelques métadonnées supplémentaires à ton IdP, telles que des certificats aussi et d'autres URLs à associer à la redirection en vue d'une identification (*login*) et d'une déconnexion (*logout*).

François : Exactement, tu as tout compris. Regardons l'enchaînement des échanges sur ce diagramme ci-dessous.

Romain : Donc si je me connecte à une page, à une ressource protégée, sans avoir été préalablement identifié, le fournisseur d'identité va retourner un *« Not authorized »* et me refuser l'accès à la ressource.

François : Oui, et de là tu vas devoir prouver ton identité auprès de ce fournisseur. Une fois ceci effectué, ce dernier retourne vers mon application, et si tu es probablement identifié, mon application te donnera accès à la ressource.

Romain : Enfin, j'imagine que l'authentification n'est pas le seul facteur, je n'ai peut-être pas le droit d'accès à cette ressource en particulier.

François : Oui, bien sûr, lors du retour du fournisseur d'identité, une enveloppe SAML est transmise, son contenu te permettra de déduire le rôle que tu associeras à l'utilisateur, et donc de lui donner ou non l'accès à un service ou à une ressource.

Romain : Mais, alors dis-moi, ton super outil jHipster, il supporte le SAML ?

François : Non, mais il vient avec Spring Security qui lui le supporte, quelques lignes de code suffisent.

Romain : Cette authentification unique doit être l'objet de notre attention, elle se doit d'être robuste et fiable.

François : Oui, et c'est pour ça qu'on a aussi implémenté, dans notre serveur d'identité, une authentification à deux étapes **[6]**.

Romain : Excellente idée car c'est vraiment devenu essentiel de nos jours... Mais pour aborder ce sujet, il faudrait un autre article ! ■

Références

[1] Jhipster : https://jhipster.github.io/

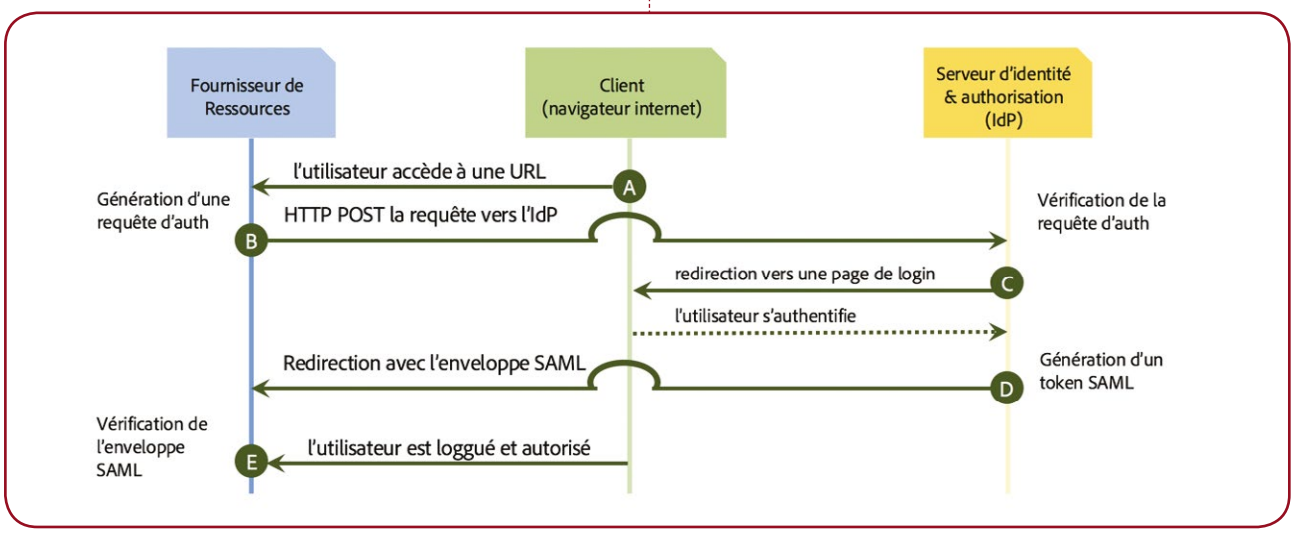
[2] Spring Security : http://projects.spring.io/spring-security/

[3] HTTP Strict Transport Security (HSTS) : https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security

[4] Cross-site request forgery (CSRF) : https://en.wikipedia.org/wiki/Cross-site_request_forgery/

[5] Le site SSO Circle : http://www.ssocircle.com

[6] Authentification à deux étapes : https://en.wikipedia.org/wiki/Two-factor_authentication



DOUANES & DROITS INDIRECTS

INFORMATICIENS

REJOIGNEZ LA DOUANE !

Mettez vos talents au service de la protection du territoire.

RÉSEAU INTERNATIONAL DE LA DOUANE

PROTÉGER LES CITOYENS ET L'ENVIRONNEMENT

VALIDATION

RECHERCHE

PARTICIPER AU FINANCEMENT DES SERVICES PUBLICS

ACCOMPAGNER LES ACTEURS DU COMMERCE MONDIAL

Contactez Infos Douane Service

0811 20 44 44 Service 0,06 € / min + prix appel

#LaDouaneRecrute

douane.gouv.fr / recrutement

Sur iOS et Android : douanefrance.mobi

@douane_france

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DES FINANCES ET DES COMPTES PUBLICS

MODÉLISATION D'UN SYSTÈME DE TÉLÉINFORMATION EDF

Michel RAMBOUILLET [Professeur honoraire de Génie Électrique option Informatique et Télématique]

Le sujet a certes déjà été traité à plusieurs reprises dans la littérature informatique ; nous envisagerons ici de replacer ce projet dans un cadre domotique plus général d'informatique répartie et d'insister sur la conception et la réalisation logicielle, notamment à l'aide de diagrammes. En particulier la téléinformation peut permettre, outre la surveillance de la consommation électrique, de piloter par exemple le système de chauffage (pompe à chaleur / chaudière fuel) en fonction des différentes périodes tarifaires EDF.

Mots-clés : UML, RaspBerry Pi, C++, Python, Web, Service

Résumé

Après avoir présenté le contexte de l'application et défini un cahier des charges, nous ferons une rapide analyse des cas d'utilisation et nous choisirons une architecture matérielle et logicielle. La partie matérielle concerne l'interface avec le bus de téléinformation EDF et l'intégration dans le réseau local ; y seront détaillés la réalisation de l'interface au moyen d'une carte de prototypage Adafruit et la configuration adéquate de l'ordinateur monocarte Raspberry Pi choisi.

La partie logicielle s'articule autour du service de gestion de la téléinformation et d'un serveur HTTP. Le service de téléinformation sera réalisé en C++ et nous aborderons la conception des classes, la compilation (fichier `Makefile`) et le démarrage du service sous systemd. Pour le serveur nous avons retenu Python et la bibliothèque wsgiref.

Tout au long de cette étude, nous utiliserons une notation inspirée d'UML (*Unified Modeling Language*) ; on pourra consulter l'ouvrage [1] comme guide pour la modélisation UML. Nous ne prétendons cependant pas respecter scrupuleusement cette notation mais simplement montrer comment des schémas compréhensibles par tous les lecteurs peuvent permettre de documenter le travail de conception en illustrant l'architecture d'une application.

Le cahier des charges de notre application domotique s'appuie sur deux constatations :

- Les solutions commerciales actuelles sont fermées et incompatibles entre elles. On trouve facilement des émetteurs/récepteurs radio pour la télécommande sans fil de lampes ou volets roulants ; mais les centrales de commande ou de surveillance utilisent des protocoles propriétaires et restent chères, fermées à toute adaptation par l'utilisateur (les fabricants de matériel de régulation pour piscine n'échappent pas à cette règle). Ceci conduit les utilisateurs à disposer de télécommandes spécifiques et de systèmes incapables de communiquer entre eux.
- Tous les foyers sont maintenant équipés d'un réseau local même si celui-ci est souvent réduit à un routeur ADSL. En fait, qu'il soit câblé, Wi-fi ou utilisant

le réseau EDF via des boîtiers CPL, le réseau local de l'habitation permet de mettre en œuvre aisément une solution domotique reposant sur une architecture réseau distribuée. On trouve pour quelques dizaines d'euros des nano-ordinateurs sous Linux ou systèmes divers (**Raspberry-Pi**, **Arduino**, etc.) ou pour quelques euros des microcontrôleurs, des modules Bluetooth ou Wi-fi permettant de réaliser toute application domotique.

1 Contexte et cahier des charges

Notre système de téléinformation s'intègre à l'architecture réseau de l'habitation comme indiqué sur la figure 1. On exploite ainsi le réseau local câblé et/ou Wi-fi, le réseau électrique via des boîtiers CPL ainsi qu'une box Internet pour fournir un accès à certains équipements hors du domicile.

Outre les éventuelles télécommandes d'origine des équipements, le système permet de contrôler ceux-ci et de visualiser les informations qu'ils fournissent avec des moyens informatiques standards (ordinateur personnel, tablette ou smartphone) via un navigateur internet par exemple ou des applications spécifiquement développées.

Le système de téléinformation doit fournir les fonctionnalités suivantes :

- **Signalisation de la période tarifaire en cours :** l'abonnement EDF est de type **EJP** et comporte donc deux périodes tarifaires (heures normales et heures de pointe mobile). Les changements de périodes tarifaires seront diffusés sur le réseau local (datagrammes) afin de permettre le délestage de certains appareils ; il seront aussi visualisés par des leds à proximité du serveur (jaune : préavis EJP, rouge : pointe mobile). Une led verte sur le serveur permettra de vérifier que l'on se trouve en période normale. Il est facile d'adapter le système à d'autres types d'abonnement.
- **Enregistrement de l'historique de la consommation :** l'évolution de la consommation électrique sera enregistrée avec indication de la période tarifaire. Cet historique permettra d'analyser les variations de consommation selon les périodes et par exemple de valider le compromis Pompe à Chaleur / Chaudière Fuel en fonction de leur coût respectif.

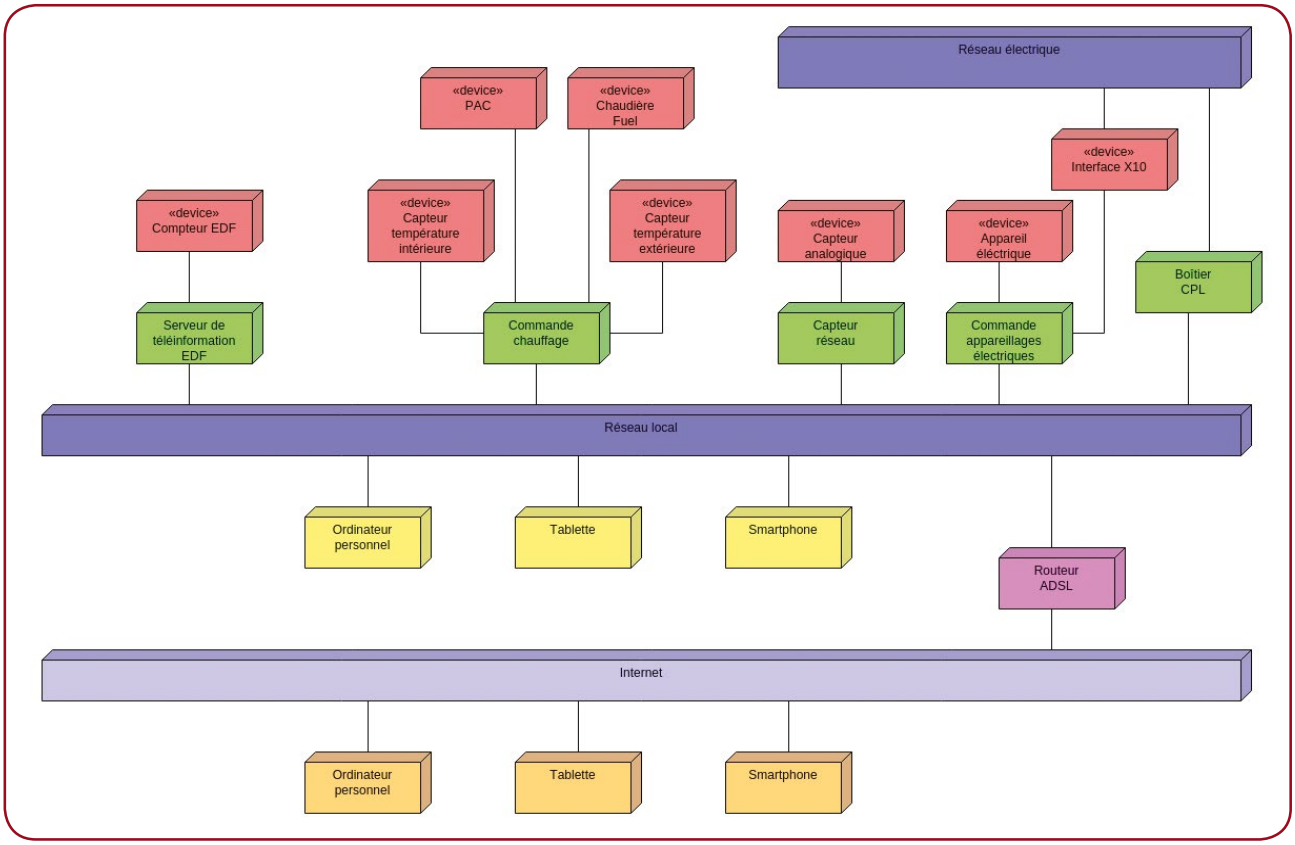


Fig. 1 : Architecture du système domotique

Fig 2 : L'installation en service

- **Serveur HTTP :** afin d'exploiter les données de téléinformation (consommation instantanée, historique) un serveur HTTP fournira des pages HTML consultables à partir d'un navigateur quelconque sur un ordinateur connecté au réseau local, voire de n'importe où si l'on a pris soin de configurer convenablement le routeur ADSL (pour plus de détails sur la manière d'accéder à un réseau local situé derrière un routeur ADSL et principalement une Livebox « Orange », voir la référence **[2]** en fin d'article). Ce choix évite de développer des applications spécifiques pour smartphones par exemple.

La figure 2 montre l'installation en service. On y voit le système à base de Raspberry-Pi fixé sous le tableau électrique. Une prise 220V permet d'alimenter le montage et de fournir l'accès au réseau local via un boîtier CPL. Le fil sortant sous le tableau fournit le signal de téléinformation EDF (il est relié aux bornes I1-I2 du compteur). Le fil qui se trouve à droite du montage permet d'allumer deux leds fixées sur la porte du logement prévu pour le tableau électrique (voir incrustation).

Les figures 3 et 4 montrent respectivement la consultation de la consommation électrique et de l'historique via un navigateur internet.

Figure 3 : La page « consommation » depuis un smartphone

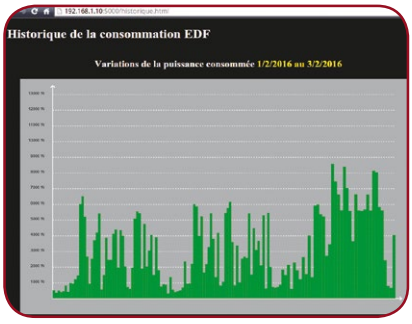


Figure 4 : Une vue de la page « historique » depuis le réseau local

2 Description des cas d'utilisation

Les **acteurs** du système sont les suivants (voir figure 5) :

- **Utilisateur** : la personne qui désire consulter les données (instantané ou historique) de téléinformation ;
- **Bus de téléinformation EDF** : le système qui fournit les données de téléinformation. On trouvera sous la référence **[3]** la documentation utile ;

- **Système automatisé** : les systèmes qui utiliseront les données de téléinformation (essentiellement la période tarifaire en cours) ; il s'agit du système de téléinformation lui-même (visualisation par leds de la période tarifaire) ou d'un système de contrôle du chauffage par exemple ;

On identifie les cas d'utilisation suivants :

- Cas « **Gérer téléinformation** » : il s'agit du cas d'utilisation principal où le système analyse les données fournies par le bus de téléinformation, les enregistre et signale les événements tarifaires ;
- Cas « **Consulter téléinformation** » : l'utilisateur consulte les données de téléinformation, consommation ou historique, via un navigateur Web.

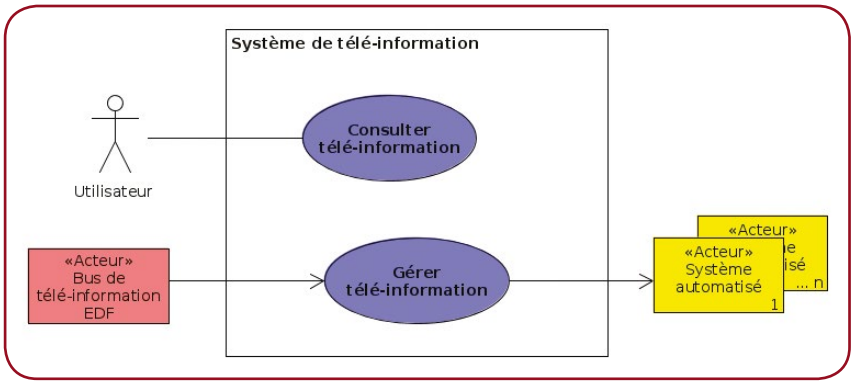


Figure 5 : Diagramme des cas d'utilisation

2.1 Cas « Gérer téléinformation »

Ce cas est déclenché par l'acteur « Bus de téléinformation EDF »

Scénario :

1. Le bus de téléinformation émet une trame de données (voir la description d'une trame dans **[3]**).
2. Le système analyse les données de la trame.
3. S'il s'agit d'une nouvelle trame (des données ont changé de valeur par rapport à la trame précédente), le système enregistre cette trame qui devient la trame courante et met à jour l'historique.
4. Si le système détecte un événement tarifaire (changement de période), il signale cet événement aux différents systèmes automatisés du réseau local ou directement connectés au système lui-même.
5. Le cas reprend à l'étape 1.

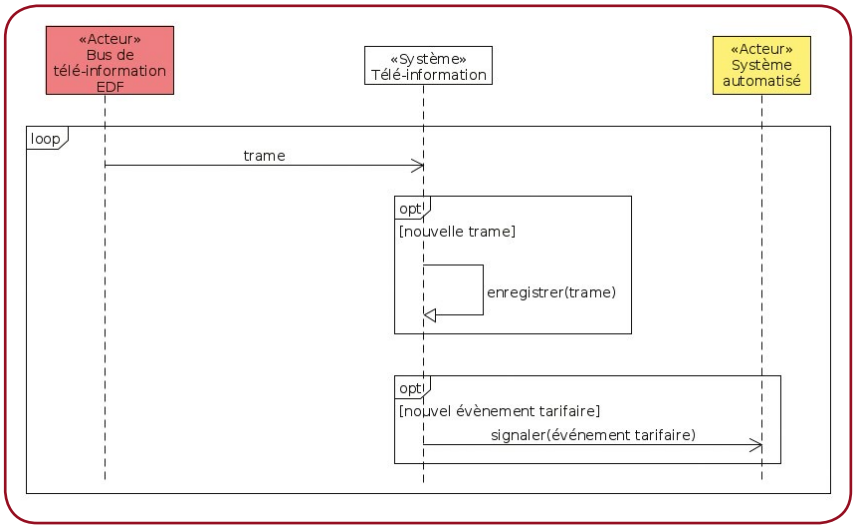


Figure 6 : Diagramme de séquence système du cas « Gérer téléinformation »

La figure 6 présente les opérations système associées au cas « Gérer téléinformation » et leur enchaînement dans un diagramme de séquence système.

2.2 Cas « Consulter téléinformation »

Ce cas est déclenché par l'utilisateur (internaute) qui désire connaître les données enregistrées ; il comporte deux variantes.

Scénario 1 :

1. L' utilisateur effectue une requête de consommation.
2. Le système affiche une page fournissant les données de consommation instantanée (données du contrat EDF, intensité instantanée et maximale, période tarifaire, quantités consommées par période).

Scénario 2 :

1. L' utilisateur effectue une requête d'historique.
2. Le système affiche une page fournissant les données de l'historique sur deux jours (j-3 à j-1) par tranche de 30 minutes sous la forme d'un graphique (voir figure 4).

La couleur des barres du graphique varie suivant la période tarifaire (rouge : heures de pointe mobile, vert : heure normale).

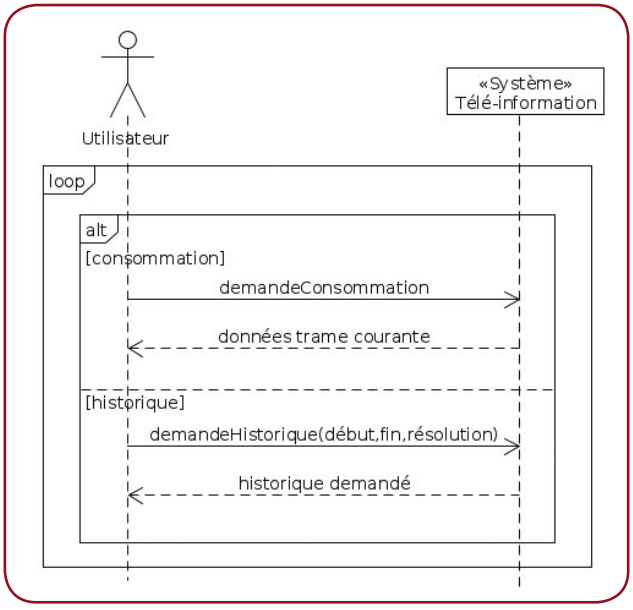


Figure 7 : DSS du cas « Consulter téléinformation »

3. L'utilisateur peut modifier les dates de début et de fin de l'historique ainsi que la résolution souhaitée.
4. Le système affiche une page fournissant les données de l'historique avec les nouveaux paramètres.
5. Le cas reprend à l'étape 3.

Le diagramme de séquence système correspondant est visible en figure 7.

3 Architecture du système de téléinformation

Physiquement, notre système doit satisfaire à plusieurs exigences :

- être capable de se connecter au bus de téléinformation EDF : les signaux émis sont de type série (1200 bauds) avec modulation d'amplitude d'une porteuse à 50 kHz ; une interface de type série (RS232 ou TTL) peut donc convenir après isolation et filtrage des signaux (voir référence **[3]**) ;

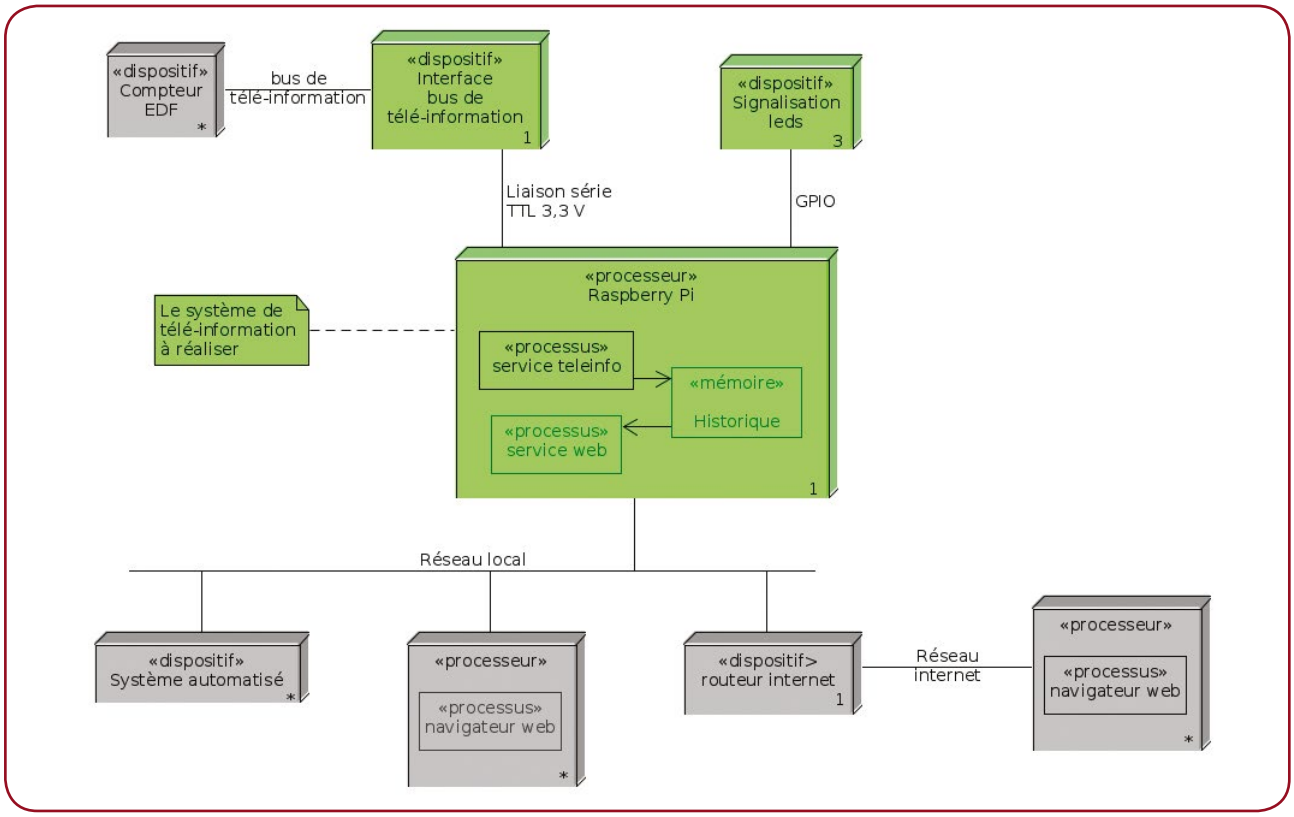


Figure 8 : Architecture de système de téléinformation

- se situer à proximité du compteur EDF en raison du câblage filaire au compteur ;
- être capable de piloter (si possible directement) un système de signalisation à leds ;
- disposer d'une connexion au réseau local.

Notre choix s'est donc porté sur un **système embarqué de type Raspberry Pi** (pour nous un modèle B) sous **Raspbian** équipé d'une **carte de prototypage Adafruit** supportant le câblage de l'interface au bus de téléinformation (via la liaison série TTL 3,3 V) et la commande des leds (via GPIO). Les cas d'utilisation seront implémentés en tant que services Linux, l'historique étant mémorisé sur la carte SD.

La figure 8 montre le diagramme de déploiement de l'application.

4 Conception et réalisation matérielle

4.1 Interface avec le bus de téléinformation

L'interface doit assurer l'isolation galvanique du montage ainsi que l'adaptation des signaux du bus à l'interface série TTL du Raspberry Pi. Une recherche sur internet nous a permis de trouver un schéma à base d'un optocoupleur **6N138** réalisant l'isolation suivie d'un filtrage et d'une mise en forme au moyen d'un trigger de Schmitt et de quelques portes logiques. Après quelques essais, nous avons pu simplifier ce schéma pour obtenir celui de la figure 9, adapté à l'alimentation 3,3 V du Raspberry. Le filtrage de la porteuse est suffisamment efficace pour ne générer que quelques erreurs de réception qui s'avèrent sans importance lors de la réception des trames de téléinformation.

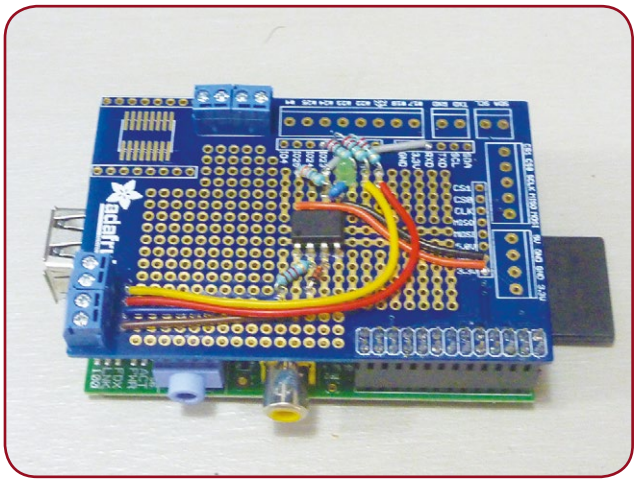


Figure 10 : Le câblage sur la carte de prototypage

Sur la figure 10, on distingue l'arrivée du signal de téléinformation sur le bornier en bas à gauche par les fils marron (I1) et bleu (I2) ainsi que la résistance de 1,2 kΩ et la diode **1N4148** acheminant le signal au 6N138 en éliminant une alternance. La sortie mise en forme est reliée à la broche RX du connecteur d'entrées/sorties du Raspberry (fil gris).

4.2 Interface avec la signalisation à leds

Nous avons choisi de signaler les événements tarifaires de manière visuelle, ne serait-ce que pour penser à ne pas utiliser le lave-vaisselle pendant les périodes de pointe mobile. Ceci est réalisé au moyen de trois leds reliées au connecteur GPIO du Raspberry Pi au moyen d'une résistance de 220 Ω :

- Sur réception d'un préavis EJP, 1 h avant la période de pointe mobile, une led jaune clignote jusqu'au début de cette période.
- Pendant les périodes de pointe mobile, une led rouge clignote alternativement avec la led jaune afin d'attirer l'attention des usagers.

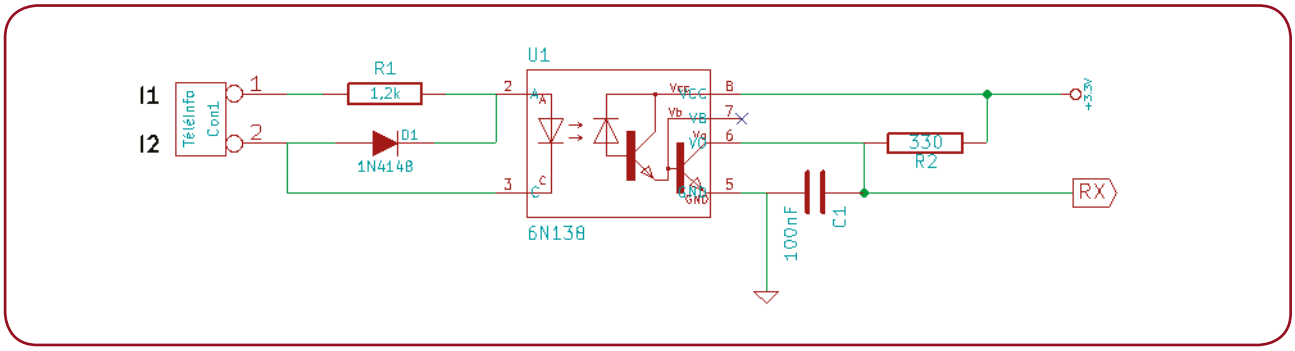


Figure 9 : Schéma de l'interface

- Une led verte clignote lentement lorsque la période tarifaire est normale afin de signaler que le système est en service.

La led verte est câblée directement sur la plaque de prototypage et reliée à GPIO 4 (IO23, broche 16 - voir note).

Les leds jaune et rouge sont déportées via un bornier (fils jaune et rouge) afin d'être visibles aisément des usagers ; elles sont reliées respectivement à GPIO 3 (IO22, broche 15) et GPIO 2 (IO21/27, broche 13).

Note

La référence des broches du connecteur d'entrées/sorties du Raspberry Pi supporte différentes désignations ; le système **IO n** ou **#n**) fait référence au marquage de la carte Adafruit et correspond à la notation **BCM n** du circuit SoC (*System on Chip*) du Raspberry alors que la référence **GPIO n** correspond à certaines bibliothèques disponibles (comme **wiringPi**) pour manipuler les entrées/sorties.

Ainsi la broche 16 du connecteur peut être référencée IO23 ou GPIO 4. On pourra consulter les liens **[4]** et **[5]** pour plus de détails.

4.3 Installation, configuration et tests

Il convient tout d'abord d'installer le système raspbian sur une carte SD si ce n'est déjà fait. On pourra télécharger le système sur le site https://www.raspberrypi.org/downloads/raspbian. Il suffit ensuite de suivre le guide d'installation en ligne. La configuration spécifique à notre application peut alors être réalisée directement sur la carte SD en éditant les fichiers adéquats avant insertion de la carte dans le Raspberry Pi ou après démarrage du Raspberry Pi, à l'aide d'une connexion ssh depuis un ordinateur sous Linux et de l'utilitaire **raspi-config**. On trouvera des compléments d'information sur http://raspbian-france.fr/installer-raspbian-premier-demarrage-configuration.

Par défaut le système raspbian utilise le port série du connecteur (*/dev/ttyAMA0*) pour gérer un terminal. Il faut donc empêcher cela en éditant le fichier **/etc/inittab** pour mettre la dernière ligne en commentaire :

```
#Spawn a getty on Raspberry Pi serial line
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

On supprimera aussi la référence à la console **ttyAMA0** (caractères en gras ci-dessous) dans le fichier **/boot/cmdline.txt** :

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 console=tty1 root=/dev/
mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait
```

Si on veut éviter de brancher un écran et un clavier pour un premier démarrage, il convient en outre de vérifier la configuration réseau ainsi que l'activation du démon ssh..

Les tests peuvent alors commencer en se connectant au système à l'aide de la commande **ssh -l pi <adresse ip>**, le mot de passe par défaut étant **raspberry**.

4.3.1 Test de l'interface avec le bus de téléinformation

Il faut configurer le port série utilisé pour la lecture des données du bus EDF avec les paramètres suivants (voir référence **[3]**) :

1200 bauds, parité paire, 7 bits de données, 1 bit de stop.

Ceci est réalisé au moyen de la commande **stty evenp speed 1200 raw --file=/dev/ttyAMA0**.

Si on a pris soin de relier l'interface aux fils I1,I2 du compteur EDF, la commande **cat /dev/ttyAMA0** permet alors de lire les données émises par le bus de téléinformation et de vérifier immédiatement le fonctionnement de l'interface ; on trouvera ci-dessous le résultat de la commande **hexdump -C /dev/ttyAMA0** qui présente l'avantage de mieux montrer la composition des trames avec les caractères de contrôle :

```
pi@pi-edf ~ $ hexdump -C /dev/ttyAMA0
00000000  38 34 31 38 30 30 37 20  48 0d 0a 50 54 45 43 20  |8418007 H..PTEC |
00000010  48 4e 2e 2e 20 5e 0d 0a  49 49 4e 53 54 20 30 32  |HN.. ^..IINST 02|
00000020  39 20 22 0d 0a 49 4d 41  58 20 30 35 39 20 4d 0d  |9 "..IMAX 059 M.|
00000030  0a 4d 4f 54 44 45 54 41  54 20 30 30 30 30 30 30  |.MOTDETAT 000000|
00000040  20 42 0d 03 02 0a 41 44  43 4f 20 30 33 39 35 30  | B....ADCO 03950|
00000050  31 30 32 33 36 33 31 20  38 0d 0a 4f 50 54 41 52  |1023631 8..OPTAR|
00000060  49 46 20 45 50 2e 20 22  0d 0a 49 53 4f 55 53 43  |IF EP. "..ISOUSC|
00000070  20 36 30 20 3c 0d 0a 45  4a 50 48 4e 20 32 34 33  | 60 <..EJPHN 243|
00000080  37 39 36 33 32 38 20 51  0d 0a 45 4a 50 50 4d 20  |796328 Q..EJPPM |
00000090  30 30 38 34 31 38 30 30  37 20 48 0d 0a 50 54 45  |00841800 7 H..PTE|
000000a0  43 20 48 4e 2e 2e 20 5e  0d 0a 49 49 4e 53 54 20  |C HN.. ^..IINST |
000000b0  30 32 39 20 22 0d 0a 49  4d 41 58 20 30 35 39 20  |029 "..IMAX 059 |
^C
pi@pi-edf ~ $
```

4.3.2 Test de l'interface de signalisation à leds

On suppose maintenant que les leds externes sont convenablement reliées au bornier de la carte de prototypage.

La commande **gpio** du système Raspian permet de gérer les entrées/sorties de Raspberry Pi. On peut commencer par **gpio -v** qui permet de vérifier le modèle du Raspberry Pi, ce qui peut affecter la manière dont on référence les broches du connecteur.

```
pi@pi-edf ~ $ gpio -v
gpio version: 2.20
Copyright (c) 2012-2014 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
  Type: Model B, Revision: 2, Memory: 512MB, Maker: Sony
```

Il convient tout d'abord de configurer les broches GPIO utilisées en sorties (par défaut, toutes les broches sont configurées en entrées) ; on utilisera pour cela la commande **gpio mode 2 output** qui configure GPIO 2 en sortie (la commande **gpio** utilise par défaut les références wiringPi pour désigner les broches du connecteur - voir note). On répétera évidemment cette commande pour GPIO 3 et GPIO 4.

Le résultat peut être visualisé au moyen de la commande **gpio readall** :

```
pi@pi-edf ~ $ gpio readall
 +-----+-----+---------+------+---+-Model B2-+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |  OUT | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |  OUT | 0 | 15 || 16 | 0 | OUT  | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |  28 |  17 | GPIO.17 |   IN | 0 | 51 || 52 | 0 | IN   | GPIO.18 | 18  | 29  |
 |  30 |  19 | GPIO.19 |   IN | 0 | 53 || 54 | 0 | IN   | GPIO.20 | 20  | 31  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+-Model B2-+---+------+---------+-----+-----+
pi@pi-edf ~ $
```

On peut alors allumer/éteindre une des leds au moyen de la commande **gpio write <broche> <valeur>** qui permet de forcer à **<valeur>** (**0** ou **1**) la broche désignée ; ainsi la commande **gpio write 2 1** permet d'allumer la led rouge.

On pourra évidemment obtenir un complément d'information sur la commande **gpio** en consultant le manuel en ligne (**man gpio**) sous réserve que la page du manuel soit installée sur le système.

Conclusion

Après analyse du cahier des charges, nous avons défini les opérations système et choisi une architecture matérielle et logicielle pour notre application. Nous avons montré ensuite comment réaliser et tester la plateforme matérielle de notre système embarqué. La modélisation du système de téléinformation pourra se poursuivre par la définition des classes d'objets permettant la réalisation des opérations système et l'étude de leurs collaborations.

Ceci fera l'objet d'un prochain article, où nous aborderons la conception et la réalisation logicielle du service de téléinformation et du service web. ■

Références

[1] ROQUES P., *« Les cahiers du programmeur UML 2 - 4ème Edition »*, Eyrolles, 2006.

[2] RAMBOUILLET M., *« Concevoir un service DynDNS pour Livebox depuis votre Raspberry Pi »*, Linux Pratique n° 90, juillet/août 2015, p. 26 à 37.

[3] Sorties de téléinformation client des appareils de comptage électroniques utilisés par ERDF : http://www.erdf.fr/sites/default/files/ERDF-NOI-CPT_02E.pdf

[4] Documentation des GPIOs du Raspberry : https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/README.md

[5] Documentation de la librairie GPIO WiringPi : http://wiringpi.com/pins/

FORMAT MIDI : COMPOSEZ EN C !

Vincent MAGNIN [Maître de conférences à Polytech Lille et l'IEMN, administrateur du projet gtk-fortran]

Avec le format MIDI, vous pouvez mêler les plaisirs de la composition musicale et de la programmation. Nous allons nous intéresser à ce format binaire et écrire un court programme en C permettant de l'exploiter.

Mots-clés : MIDI, Format SMF, Musique Assistée par Ordinateur, Langage C

Résumé

Un fichier MIDI est un fichier binaire composé d'événements musicaux codés par des séries d'octets. Nous allons dans ce premier article écrire un petit programme en C pour créer ce type de fichier. Dans un second article, nous utiliserons ce programme pour explorer la musique et ses relations avec les mathématiques ou l'algorithmique.

Nous avions commencé à explorer l'univers MIDI dans le hors-série n°29 « MUSIQUE & SON » de Linux Pratique **[1]**. MIDI *(Musical Instrument Digital Interface),* lancé en 1983, définit un protocole de communication entre instruments de musique où les informations sont transmises à une vitesse de 31250 bauds. En 1989, s'y ajoute le format de fichier SMF *(standard MIDI file)* **[2]** qui permet de stocker des séquences d'instructions MIDI dans des fichiers très compacts ayant pour extension `.mid` ou `.midi`. Il s'agit d'un format binaire dont nous allons explorer les bases en écrivant un programme en C permettant de créer de tels fichiers.

1 Le programme

Vous pouvez récupérer les fichiers `midi.c` et `demo.c` sur le dépôt GitHub du magazine. Le premier fichier contient toutes les procédures et fonctions qui nous seront utiles. Le second fichier est une courte démonstration des possibilités du format MIDI qui contient le programme principal et les procédures créant les pistes. Nous n'allons pas implémenter toutes les fonctionnalités du format MIDI mais celles qui sont essentielles pour créer un fichier fonctionnel. Une fois compris les grands principes, vous n'aurez pas de difficultés à ajouter d'autres instructions MIDI en vous basant sur les pages de la documentation en ligne qui vous seront données en références.

Après compilation avec `gcc` et exécution, la commande `hexdump` permet de visualiser facilement le début du fichier MIDI généré :

```
$ gcc demo.c
$ ./a.out
$ hexdump demo.mid -C | head
00000000  4d 54 68 64 00 00 00 06  00 01 00 02 00 80 4d 54  |MThd..........MT|
00000010  72 6b 00 00 00 0b 00 ff  51 03 07 a1 20 00 ff 2f  |rk......Q... ../|
00000020  00 4d 54 72 6b 00 00 06  6f 00 c0 5a 00 90 3c 40  |.MTrk...o..Z..<@|
00000030  40 80 3c 00 00 90 3d 40  40 80 3d 00 00 90 3e 40  |@.<...=@@.=...>@|
00000040  40 80 3e 00 00 90 3f 40  40 80 3f 00 00 90 40 40  |@.>...?@@.?...@@|
00000050  40 80 40 00 00 90 41 40  40 80 41 00 00 90 42 40  |@.@...A@@.A...B@|
00000060  40 80 42 00 00 90 43 40  40 80 43 00 00 90 44 40  |@.B...C@@.C...D@|
00000070  40 80 44 00 00 90 45 40  40 80 45 00 00 90 46 40  |@.D...E@@.E...F@|
00000080  40 80 46 00 00 90 47 40  40 80 47 00 00 90 48 40  |@.F...G@@.G...H@|
00000090  40 80 48 00 00 c0 00 00  90 45 40 81 00 80 45 00  |@.H......E@...E.|
```

Au fil de cet article, vous pourrez revenir à ce *dump* et y repérer les valeurs de certains octets.

2 Programme principal

Un fichier MIDI est organisé en *chunks* (terme que je traduirai par blocs), chacun commençant par un tag de quatre caractères ASCII, suivi d'un entier sur quatre octets codant la taille en octets des données contenues à sa suite dans ce bloc. Il existe deux types de bloc dans un fichier MIDI : le bloc d'en-tête (tag `MThd`) et les blocs des pistes (tag `MTrk`). Le programme principal, situé dans le fichier `demo.c`, va tout d'abord ouvrir en écriture un fichier binaire `demo.mid`, puis appeler la procédure qui écriera l'en-tête MIDI, et enfin les procédures qui écrieront chaque piste (ici deux), avant de finalement fermer le fichier :

```
27: int main(void) {
28:     FILE *fichier_midi = fopen("demo.mid", "wb") ;
```

```
29:     MIDI_ecrire_en_tete(fichier_midi, 1, 2, noire) ;
30:     ecrire_piste1(fichier_midi) ;
31:     ecrire_piste2(fichier_midi) ;
32:     fclose(fichier_midi) ;
33: }
```

3 En-tête MIDI

Les octets à écrire dans le fichier sont stockés dans un tableau de variables du type **unsigned char**. En effet, d'après la norme du langage C **[3]**, une variable de ce type occupe exactement un *byte*, unité définie comme composée de **CHAR_BIT** bits. La valeur de la constante **CHAR_BIT** vaut bien sûr 8 en général, mais la norme prévoit qu'elle puisse être plus grande. À noter que la norme ne définit pas la taille des différents types d'entiers, mais seulement une taille minimale : un **short** doit occuper au minimum deux octets et un **long** quatre octets. Dans ce programme, par souci de rigueur nous utiliserons donc des variables de type **unsigned char**, **unsigned short** ou **unsigned long** en fonction du nombre d'octets exigés par la norme MIDI. Sur un microprocesseur 32 bits, un **long** occupera quatre octets et huit octets sur un 64 bits, mais cela n'a pas d'importance car nous n'utiliserons à chaque fois que le nombre d'octets de poids faibles nécessaires.

Le bloc d'en-tête commence par quatre octets formant la chaîne **Mthd**, soit **0x4d546864** en hexadécimal (ligne 48) :

```
42: void MIDI_ecrire_en_tete(FILE *fichier, unsigned char SMF, unsigned
short pistes, unsigned short nbdiv) {
43:     if ((SMF == 0) && (pistes > 1)) {
44:         printf("ERREUR : une seule piste possible en SMF 0 ! \n") ;
45:         exit(EXIT_FAILURE) ;
46:     }
47:
48:     unsigned char octets[14] = {0x4d, 0x54, 0x68, 0x64, 0x00, 0x00,
0x00, 0x06} ;
49:     octets[8]  = 0 ;
50:     octets[9]  = SMF ;
51:     octets[10] = pistes >> 8 ;
52:     octets[11] = pistes ;
53:     octets[12] = nbdiv  >> 8 ;
54:     octets[13] = nbdiv ;    // Nombre de divisions de la noire
55:     fwrite(&octets, 14, 1, fichier) ;
56: }
```

Les quatre octets suivant la chaîne **Mthd** codent la longueur en octets de la suite de l'en-tête, valeur qui dans la pratique vaut toujours 6. Les deux octets suivants (8 et 9) codent le type de fichier (paramètre SMF). Il n'existe que trois types de fichiers :

- **SMF=0** : il n'y a qu'une seule piste (plage) dans le fichier. D'où le test en début de procédure qui arrête le programme si vous vous êtes trompés.
- **SMF=1** : le fichier contient plusieurs pistes qui seront jouées simultanément. On s'en servira généralement pour séparer les différentes voies d'une partition musicale. Le format **1** est le plus utilisé et le plus pratique, c'est donc celui que nous utiliserons dans cet article.
- **SMF=2** : le fichier contient plusieurs pistes qui seront jouées séquentiellement. Chaque piste pourra donc correspondre à un morceau de musique différent, comme les plages d'un album. Ce format est très peu utilisé.

Le nombre de pistes est codé sur deux octets et peut donc aller jusque **65535**. La valeur de la variable **pistes** doit être recopiée dans les octets **10** et **11** de l'en-tête. L'octet de poids faible est directement recopié dans la variable **octets[11]**, la variable de type **short** étant implicitement convertie en une variable de type **char**. Pour récupérer l'octet de poids fort de notre valeur, nous décalons de **8** bits vers la droite ses bits, avant conversion implicite en **char**. Pour être plus explicite, on pourrait aussi écrire ces lignes ainsi :

```
octets[10] = (char) (pistes >> 8) & 0xFF ;
octets[11] = (char) pistes & 0xFF ;
```

Les deux derniers octets de l'en-tête MIDI codent le nombre de divisions de la noire. Deux codages existent : si le bit le plus à gauche vaut **1**, on utilise le codage temporel SMPTE **[4]** utilisé dans l'audiovisuel, sinon on définit le nombre de tics correspondant à la noire (soit au maximum **32767** puisqu'il ne reste que **15** bits pour coder la valeur). C'est cette dernière méthode que nous allons utiliser. Attention de ne pas confondre cette grandeur avec le tempo (nombre de noires par minute), qui sera défini plus loin. Il s'agit ici de définir la résolution temporelle dans notre fichier. On choisira de préférence une puissance de deux, par exemple **128**, la croche valant une demi-noire, la double croche un quart, etc.

Ligne 55, l'instruction **fwrite()** écrit dans notre fichier 14 blocs d'un octet (d'où le **1**), stockés en mémoire à partir de l'adresse **&octets** de notre tableau. Vous pouvez vérifier que l'en-tête a bien été correctement écrit en utilisant la commande suivante :

```
$ file demo.mid
demo.mid: Standard MIDI data (format 1) using 2 tracks at 1/128
```

4 Structure d'une piste

Après l'en-tête du fichier MIDI vient l'en-tête de la première piste puis ses données, suivies de l'en-tête de la deuxième piste et ses données, etc. L'en-tête d'une piste commence par quatre octets codant la chaîne **Mthd**, soit

0x4d54726b en hexadécimal. Les quatre octets suivants codent le nombre d'octets composant la suite de la piste. Problème : nous ne connaissons pas forcément ce nombre avant d'avoir écrit les données ! Nous écrivons donc pour l'instant zéro à cet endroit et la fonction **MIDI_ecrire_en_tete_piste()** renvoie grâce à la fonction **ftell()** la position que nous avons atteinte dans le fichier juste après l'écriture de ce zéro :

```
 99: unsigned long MIDI_ecrire_en_tete_piste(FILE *fichier) {
100:     unsigned char octets[8] = {0x4d, 0x54, 0x72, 0x6b, 0x00,
0x00, 0x00, 0x00} ;
101:     fwrite(&octets, 8, 1, fichier) ;
102:     return ftell(fichier) ;
103: }
```

L'en-tête est immédiatement suivi des données. Pour indiquer la fin de la piste, il faudra par convention écrire la valeur **0xFF2F00** grâce à cette procédure :

```
105: void MIDI_fin_de_la_piste(FILE *fichier) {
106:     MIDI_delta_time(fichier, 0) ;
107:     unsigned char octets[3] = {0xFF, 0x2F, 0x00} ;
108:     fwrite(&octets, 3, 1, fichier) ;
109: }
```

Nous pourrons alors revenir dans l'en-tête de la piste afin d'y écrire le nombre d'octets écrits après l'en-tête de la piste grâce à la procédure **ecrire_taille_finale_piste**. Lors de l'écriture d'une piste, nous stockons dans une variable **marque** la valeur renvoyée par la fonction **MIDI_ecrire_en_tete_piste()**. Nous soustrayons donc à la position actuelle renvoyée par **ftell(fichier)** la valeur qui est dans **marque**, puis nous soustrayons **4** pour nous retrouver juste après le tag **Mthd**. Ligne 114, la fonction **fseek()** avec pour dernier paramètre **SEEK_SET** permet de « rembobiner » à l'endroit voulu, où nous écrivons ensuite les quatre octets codant la taille désormais connue (en utilisant des décalages de **24**, **16**, **8** et **0** bits suivant le même principe que précédemment vu). Ligne 120, la fonction **fseek()** avec pour dernier paramètre **SEEK_END** permet de revenir à la fin actuelle du fichier afin de pouvoir y écrire éventuellement d'autres pistes :

```
111: void ecrire_taille_finale_piste(FILE *fichier, unsigned long
marque) {
112:     unsigned char octets[4] ;
113:     unsigned long taille = ftell(fichier) - marque ;
114:     fseek(fichier, marque-4, SEEK_SET) ;    // On rembobine
115:     octets[0] = taille >> 24 ;
116:     octets[1] = taille >> 16 ;
117:     octets[2] = taille >> 8 ;
118:     octets[3] = taille ;
119:     fwrite(&octets, 4, 1, fichier) ;
120:     fseek(fichier, 0, SEEK_END) ;
121: }
```

5 Piste des métadonnées

Si vous utilisez le format SMF 1, la première piste doit être réservée aux méta-données (en SMF 0, elles seront écrites au début de l'unique piste). Dans le cadre de cet article, nous restreindrons ces méta-données au strict nécessaire, c'est-à-dire au tempo qui est codé sous forme du nombre de micro-secondes correspondant à la durée d'une noire. Nous prenons ici une valeur de **500000** micro-secondes (ligne 5 du fichier **demo.c**), donc une demi-seconde, ce qui est la valeur MIDI par défaut et correspond à un tempo de **120** (nombre de noires par minute) :

```
1: #include "midi.c"
2:
3: void ecrire_piste1(FILE *fichier) {
4:     unsigned long marque = MIDI_ecrire_en_tete_piste(fichier) ;
5:     MIDI_tempo(fichier, 500000) ;
6:     MIDI_fin_de_la_piste(fichier) ;
7:     ecrire_taille_finale_piste(fichier, marque) ;
8: }
```

Toutes les métadonnées commencent par un octet **0xFF**. La valeur du tempo est ainsi codée sur trois octets précédés de la valeur **0xFF5103** :

```
58: void MIDI_tempo(FILE *fichier, unsigned long duree) {
59:     MIDI_delta_time(fichier, 0) ;
60:     unsigned char octets[6] = {0xFF, 0x51, 0x03} ;
61:     octets[3] = duree >> 16 ;
62:     octets[4] = duree >> 8 ;
63:     octets[5] = duree ;
64:     fwrite(&octets, 6, 1, fichier) ;
65: }
```

Une noire durera donc entre une microseconde et **256** au cube microsecondes, soit environ **16,7** secondes. Voilà qui est une plage largement suffisante !

D'autres méta-données peuvent être écrites comme la mesure (4/4 par défaut), l'armure de la clé, le nom de la piste, les paroles, etc.

6 Événements MIDI

Les données d'une piste MIDI sont composées d'une suite d'événements MIDI précédé systématiquement d'un délai (éventuellement nul) appelé *delta-time,* durée exprimée en tics devant s'écouler depuis le précédent événement MIDI :

```
38: void MIDI_delta_time(FILE *fichier, unsigned long duree) {
39:     ecrire_variable_length_quantity(fichier, duree) ;
40: }
```

Il convient pour cela de se référer au nombre de tics composant une noire, valeur qui a été définie dans l'en-tête du fichier.

Décimal	Hexadécimal	Binaire	VLQ (binaire)	VLQ (hexadécimal)
0	0x00	00000000	00000000	0x00
127	0x7F	01111111	01111111	0x7F
128	0x80	10000000	10000001 00000000	0x8100
16383	0x3FFF	00111111 11111111	11111111 01111111	0xFF7F
16384	0x4000	01000000 00000000	10000001 10000000 00000000	0x818000
268435455	0x0FFFFFFF	00001111 11111111 11111111 11111111	11111111 11111111 11111111 01111111	0xFFFFFF7F

Afin d'économiser des octets (n'oublions pas que MIDI est une norme des années 80 où les informations circulent entre instruments à environs **31250** bauds), les *delta-times* sont, selon leur valeur, stockés sur un à quatre octets suivant un format nommé *variable length quantity* **[5]**.

6.1 Variable Length Quantity

Dans ce codage des entiers, les chiffres binaires sont groupés par sept, le bit le plus significatif de l'octet valant **1** pour indiquer qu'il y a encore des octets à lire et **0** s'il s'agit du dernier octet. On peut ainsi représenter des entiers arbitrairement grands tout en limitant l'espace mémoire utilisé. Le tableau suivant présente quelques exemples de valeurs codées dans ce format (tableau ci-dessus).

Dans la norme MIDI, on se limite à quatre octets maximum et la valeur la plus grande est **0x0FFFFFFF** puisque le bit le plus significatif de chacun des quatre octets est réservé. Par sécurité, notre procédure vérifie tout d'abord que nous n'avons pas dépassé cette valeur :

```
 16: void ecrire_variable_length_quantity(FILE *fichier, unsigned
long i) {
 17:     bool continuer ;
 18:
 19:     if (i > 0x0FFFFFFF) {
 20:         printf("ERREUR : delai > 0x0FFFFFFF ! \n") ;
 21:         exit(EXIT_FAILURE) ;
 22:     }
 23:
 24:     unsigned long filo = i & 0x7F ;
 25:     i = i >> 7 ;
 26:     while (i != 0) {
 27:         filo = (filo << 8)  + ((i & 0x7F) | 0x80) ;
 28:         i = i >> 7 ;
 29:     }
 30:
 31:     do {
 32:         fwrite(&filo, 1, 1, fichier) ;
 33:         continuer = filo & 0x80 ;
 34:         if (continuer) filo = filo >> 8 ;
 35:     } while (continuer) ;
 36: }
```

Ensuite, nous récupérons les sept bits les moins significatifs de l'entier **i** (ligne 24), que nous plaçons dans une variable nommée **filo** car elle servira de pile de type *First In Last Out*. Ligne suivante, les bits de **i** sont décalés vers la droite de sept positions. La première boucle « Tant Que » va continuer de récupérer des groupes de sept bits et de les empiler dans la variable **filo**, en mettant si nécessaire à **1** le bit le plus significatif de l'octet (opération **| 0x80**).

La deuxième boucle, lignes 31 à 35, va récupérer chaque octet de **filo**, en commençant par le moins significatif, et l'écrire dans le fichier MIDI, jusqu'à qu'ils aient tous été écrits : on continue tant que le bit le plus significatif de l'octet qui vient d'être écrit vaut **1**.

6.2 Choix d'un instrument

La norme MIDI définit seize canaux (le dixième canal est spécial puisque réservé aux percussions). Chaque canal ne peut jouer qu'un instrument à la fois (mais on peut changer d'instrument au cours du temps). En 1991, la norme General MIDI (GM) a normalisé la numérotation et le nom de 128 instruments répartis en 16 familles (cf. tableau suivant) ainsi que de 47 percussions **[6]**.

0	7	Pianos
8	15	Percussions Chromatiques
16	23	Orgues
24	31	Guitares
32	39	Basses
40	47	Cordes
48	55	Ensembles et chœurs
56	63	Cuivres
64	71	Instruments à anche
72	79	Instruments à Vent
80	87	Lead (Synthétiseurs)
88	95	Pad (Synthétiseurs)
96	103	Effets (Synthétiseurs)
104	111	Instruments ethniques
112	119	Percussions
120	127	Effets sonores

La numérotation des canaux et des instruments commence informatiquement à zéro (attention, la liste en référence **[6]** commence à **1**). L'instruction MIDI pour changer l'instrument du canal **n** est **0xCnXX** où **n** est le chiffre hexadécimal correspondant et **XX** le numéro de l'instrument en hexadécimal (par exemple **00** pour le grand piano acoustique). Pour limiter les problèmes, nous utilisons l'opérateur **%** (modulo) afin de rester dans les bornes autorisées même si l'utilisateur se trompe (sinon les octets pourraient être interprétés comme d'autres instructions MIDI) :

```
 67: void MIDI_Program_Change(FILE *fichier, unsigned char canal,
unsigned char instrument) {
 68:     unsigned char octets[2] ;
 69:     MIDI_delta_time(fichier, 0) ;
 70:     octets[0] = 0xC0 + canal % 16 ;
 71:     octets[1] = instrument % 128 ;
 72:     fwrite(&octets, 2, 1, fichier) ;
 73: }
```

6.3 Débuter et relâcher une note

Les événements MIDI consistant à débuter ou à relâcher une note (*Note On* et *Note Off*) commencent respectivement par les octets **0x9n** et **0x8n** où **n** est le numéro du canal en hexadécimal.

Le deuxième octet code la note à jouer : le système MIDI définit 128 notes numérotées de **0** *(do -2)* à **127** *(sol 8),* sur plus de dix octaves. Comme indiqué figure 1, le *la 3* (celui de la troisième octave, à 440 Hz) correspond au numéro **69** (**0x45**), le *la 3* dièse a pour numéro **70**, le *si 3* le numéro **71**, le *do 4* le numéro **71**, etc. Vous trouverez sur Wikipedia une figure vous permettant de trouver le numéro de chaque note **[7]**, mais attention, la numérotation des octaves est différente dans la notation anglo-saxonne : le *la3* est noté C4 (Figure 1).

Le troisième octet correspond à la *vélocité* avec laquelle la touche est enfoncée et correspond concrètement au volume sonore. La valeur minimale est **1** et la valeur maximale est **127** (**0x7F**). La valeur **0** peut être utilisée pour arrêter la note (relâchement de la touche) à la place d'un message *Note Off*.

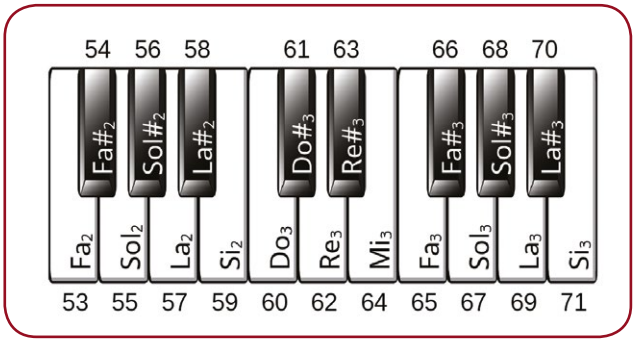


Fig. 1. Numérotation des notes MIDI.

Nous utiliserons la même procédure pour débuter ou relâcher une note :

```
 84: void MIDI_Note(unsigned char etat, FILE *fichier, unsigned
char canal, unsigned char Note_MIDI, unsigned char velocite) {
 85:     unsigned char octets[3] ;
 86:     octets[0] = etat + canal % 16 ;
 87:     octets[1] = Note_MIDI % 128 ;
 88:     octets[2] = velocite % 128 ;
 89:     fwrite(&octets, 3, 1, fichier) ;
 90: }
```

Pour cela, le premier paramètre **etat** devra prendre pour valeur **ON** ou **OFF**, constantes définies au début du code source :

```
  3: #include <stdio.h>
  4: #include <stdlib.h>
  5: #include <stdbool.h>
  6:
  7: #define noire 128
  8: #define ON  0x90
  9: #define OFF 0x80
 10: #define C3  60
 11: #define percu 9
 12: #define reverb 0x5B
 13: #define chorus 0x5D
 14: #define phaser 0x5F
```

Dans une partition, il est courant qu'un même instrument joue plusieurs notes simultanément mais qui ne se terminent pas forcément en même temps et en particulier avant que d'autres notes commencent. Cela complique beaucoup le calcul des *delta-times* entre événements. Dans le cadre de cet article, nous allons simplifier les choses en créant une procédure permettant à la fois de débuter et de relâcher une même note (paramètre **durée**). Aucune autre note ne pourra alors commencer pendant la durée de cette note :

```
 92: void Note_unique_avec_duree(FILE *fichier, unsigned char
canal, unsigned char Note_MIDI, unsigned char velocite, unsigned
long duree) {
 93:     MIDI_delta_time(fichier, 0) ;
 94:     MIDI_Note(ON,  fichier, canal, Note_MIDI, velocite) ;
 95:     MIDI_delta_time(fichier, duree) ;
 96:     MIDI_Note(OFF, fichier, canal, Note_MIDI, 0) ;
 97: }
```

Si vous voulez jouer plusieurs notes simultanément, par exemple pour jouer un accord, vous devrez donc gérer vous-même les états **ON** et **OFF** de chaque note avec la procédure **MIDI_Note()**.

> **Attention !**
>
> MIDI définit un mode *Running status* qui permet de ne pas répéter le premier octet d'une instruction destinée à un canal, par exemple `0x90` quand plusieurs notes se succèdent. Nous ne l'utilisons pas dans cet article afin de ne pas compliquer la programmation inutilement.

6.4 Contrôler les paramètres MIDI

L'instruction MIDI *Control Change,* dont le premier octet est `0xBn`, permet de régler sur le canal `n` un paramètre MIDI (`type`) à une `valeur` donnée (comprise en `0` et `127`) :

```
 75: void MIDI_Control_Change(FILE *fichier, unsigned char canal,
unsigned char type, unsigned char valeur) {
 76:     unsigned char octets[3] ;
 77:     MIDI_delta_time(fichier, 0) ;
 78:     octets[0] = 0xB0 + canal % 16 ;
 79:     octets[1] = type % 128 ;
 80:     octets[2] = valeur % 128 ;
 81:     fwrite(&octets, 3, 1, fichier) ;
 82: }
```

On pourra par exemple agir sur le niveau de réverbération (`0x5B`), de chorus (`0x5D`), de l'effet phaser (`0x5F`), du panoramique (`0x0A`), des paramètres du son *(Attack, Release...),* contrôler le niveau des molettes et pédales que l'on trouve sur tout synthétiseur, etc. On pourra également changer de banque sonore (`0x00`), à condition que la fonte sonore utilisée dispose de plusieurs banques, ce qui permet d'outrepasser la limite de 127 instruments. Vous trouverez la liste des *Control Changes* sur le site officiel **[8]**.

7 Exemple de piste musicale

Dans notre programme d'exemple `demo.c`, nous nous contenterons de créer une seconde piste (la première étant celle des méta-données). Une première boucle joue avec l'instrument `90` (Pad 3 polysynth) une gamme chromatique (3e octave) : *do, do#, ré, ré#, mi, fa, fa#, sol, sol#, la, la#, si, do.* Puis une seconde boucle passe en revue les 128 sons de la norme General MIDI en jouant un *la 3* (neuf notes au-dessus du *do3* dont le numéro est stocké dans la constante `C3`) :

```
10: void ecrire_piste2(FILE *fichier) {
11:     unsigned long marque = MIDI_ecrire_en_tete_piste(fichier) ;
12:
13:     MIDI_Program_Change(fichier, 0, 90) ;
14:     for(int i=C3 ; i<=C3+12 ; i=i+1){
15:         Note_unique_avec_duree(fichier, 0, i, 64, noire/2) ;
16:     }
17:
18:     for(int i=0 ; i<=127 ; i=i+1){
19:         MIDI_Program_Change(fichier, 0, i) ;
20:         Note_unique_avec_duree(fichier, 0, C3 + 9, 64, noire) ;
21:     }
22:
23:     MIDI_fin_de_la_piste(fichier) ;
24:     ecrire_taille_finale_piste(fichier, marque) ;
25: }
```

La première boucle joue des croches (durée valant **noire/2**), la seconde des noires. Dans les deux cas, le volume (vélocité) est fixé à **64**, la moitié de la valeur maximale permise. Le canal MIDI utilisé est le canal **0**.

Conclusion

Pour jouer le fichier **demo.mid**, vous pouvez installer le logiciel **TiMidity++** (paquet **timidity** dans le cas d'une distribution Ubuntu) **[9]**. Tapez ensuite :

```
$ timidity demo.mid
```

Remarquez que le fichier ne fait que 1688 octets ! Le rendu sonore dépend principalement de la qualité de la fonte sonore *(soudfont)* utilisée. TiMidity++ utilise par défaut une fonte nommée **freepats [10],** constituée de sons échantillonnés *(samples)* de divers instruments, mais qui n'est pas complète (le fichier **/etc/timidity/freepats.cfg** contient la liste des instruments présents). Il est préférable d'utiliser une des fontes sonores libres les plus réputées, la *Fluid R3 General MIDI soundfont* disponible dans le paquet **fluid-soundfont-gm** (142 Mio) :

```
$ timidity demo.mid -x "soundfont /usr/share/sounds/sf2/FluidR3_GM.sf2"
```

Si vous avez suffisamment de mémoire vive, vous pouvez également télécharger la fonte *Crisis General Midi 3.01* **[11]** qui fait 1,6 Gio une fois décompactée !

TiMidity++ dispose de nombreuses options. Vous pouvez par exemple exporter votre musique en **.wav** facilement en ajoutant les options **-o demo.wav -Ow**, en **.ogg** avec l'option **-Ov**, ou au format FLAC *(Free Lossless Audio Codec)* avec **-OF**. Vous pouvez même tracer un spectrogramme en temps réel avec l'option **-g .05** (définition temporelle).

Enfin, pour visualiser la partition de votre morceau, vous pouvez utiliser le séquenceur **Rosegarden [12, 13]**. Vous pouvez également récupérez vos pistes dans **LMMS [14]** où vous pourrez les manipuler et les faire jouer par tout un tas de *plugins* synthétiseurs.

Vous disposez désormais des bases minimales nécessaires pour créer vos propres fichiers MIDI, *from scratch*. Dans un second article, nous explorerons plus avant les possibilités offertes par ce programme en mêlant musique, algorithmes et mathématiques. ■

Références

[1] Magnin V., *« Avec MIDI, lancez-vous dans la musique assistée par ordinateur »*, Linux Pratique Hors-Série n°29, p. 36-47.

[2] *Standard MIDI Files :* https://www.midi.org/articles/about-midi-part-4-midi-files

[3] Brouillon de la norme C 2011 (ISO/IEC 9899:201x) : http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1570.pdf

[4] Code temporel SMPTE : https://fr.wikipedia.org/wiki/Timecode_%28temporel%29

[5] Codage *Variable Length Quantity :* https://en.wikipedia.org/wiki/Variable-length_quantity

[6] Liste des instruments General MIDI : https://www.midi.org/specifications/item/gm-level-1-sound-set

[7] Numérotation MIDI des notes : https://upload.wikimedia.org/wikipedia/commons/3/36/NoteNamesFrequenciesAndMidiNumbers_V3.svg

[8] *Control Change Messages :* https://www.midi.org/specifications/item/table-3-control-change-messages-data-bytes-2

[9] Lecteur MIDI Timidity++ : http://timidity.sourceforge.net/

[10] Fonte sonore freepats : http://freepats.zenvoid.org/

[11] Fonte CGM3.01 : http://www.bismutnetwork.com/04CrisisGeneralMidi/Soundfont3.0.php

[12] Séquenceur Rosegarden : http://www.rosegardenmusic.com/

[13] MAGNIN V., *« Composez librement avec le séquenceur Rosegarden »*, Linux Pratique Hors-Série n°29, p. 48-57.

[14] MAGNIN V., *« Lancez-vous dans la "dance music" avec Linux MultiMedia Studio ! »*, Linux Pratique n°82, p. 56-63.

À LA DÉCOUVERTE D'UNE APPLICATION « UNHOSTED » : LITEWRITE ET PHP REMOTESTORAGE

Stéphane MOUREY [Mousse sur le Seeraiwer]

Le concept d'applications « unhosted » a été créé dans l'espoir de rendre aux utilisateurs le contrôle de leurs données. Il s'agit d'applications servies par un serveur Web mais fonctionnant entièrement dans le navigateur et capables de sauvegarder leurs données sur un autre serveur Web. Faisons connaissance avec ce concept inhabituel en découvrant une de ses applications : Litewrite.

Mots-clés : Unhosted, RemoteStorage, Contrôle, Vie privée, Serveur de données

Résumé

Nous découvrirons comment utiliser Litewrite avec un stockage distant, avant d'aborder la mise en œuvre de nos propres instances de Litewrite et RemoteStorage.

Développer des applications Web en ligne qui, de par la façon même dont elles sont conçues, laissent l'utilisateur entièrement maître de ses données est la préoccupation d'un certain nombre de développeurs militants du logiciel libre. Le scandale PRISM et l'exploitation intensive des données par les grands acteurs des réseaux sociaux ont mis en évidence la pertinence de telles solutions. Pour autant, un tel mode de fonctionnement n'est pas rentré dans nos habitudes et reste exotique pour la plupart d'entre nous. Faisons donc connaissance avec une de ces applications, Litewrite, d'abord en tant que simple utilisateur, avant d'étudier la mise en œuvre de notre propre instance.

1 Découverte de l'application

1.1 Présentation de Litewrite

Litewrite est un simple outil de prise de note. Pour l'utiliser, il suffit de se connecter sur https://litewrite.net. Dès la première ouverture de la page, une session de travail vous attend. Il n'existe qu'une seule note pour le moment, qui vous présente le service en français. Lorsque vous aurez ajouté d'autres notes et si vous ne l'avez pas modifiée, celle-ci sera supprimée lors de votre prochaine connexion. Mais vous la retrouverez facilement en accédant au site avec un autre navigateur. Par ailleurs, la seule information vraiment utile qu'elle comporte tient dans les quelques raccourcis clavier à la fin.

Pour créer une nouvelle note, il suffit de cliquer sur le signe « plus » en haut à gauche. Chaque note est listée en dessous de ce signe, identifiée par son titre. Par défaut, elle s'appellera d'abord « Écrire ... ». Le focus est immédiatement placé dans la zone de travail, vous pouvez commencer à taper tout de suite. La première ligne servira de titre. L'outil ne vous propose aucune possibilité de mise en forme : il s'agit de prise de notes, pas de traitement de texte. Pour supprimer une note, il suffit d'effacer tout son contenu. Vous trouverez en bas à droite un lien « Partager ». Si vous cliquez dessus, deux autres liens apparaissent : « ouvrir » et « cacher ».

« cacher » permet simplement de retourner à l'état précédent et « ouvrir » ouvre une nouvelle fenêtre présentant votre note en lecture seule. Vous pourrez alors copier l'adresse de cette page pour la communiquer à vos correspondants.

Vous aurez remarqué que, contrairement à la plupart des applications en ligne, il n'est pas nécessaire de créer un compte et de vous identifier pour pouvoir l'utiliser et même bénéficier de la persistance des données. C'est que celles-ci sont stockées en local dans votre navigateur. Mais alors quel intérêt de faire une application Web s'il faut toujours utiliser le même navigateur ? N'est-on pas lié du coup à un poste de travail ? Le seul bénéfice est-il seulement de se passer d'une installation ?

Heureusement, Litewrite ne se limite pas à cela, mais pour bénéficier de votre session dans vos autres environnements de travail, il faut faire intervenir un deuxième acteur.

1.2 Présentation du RemoteStorage de 5Apps

L'idée est de déléguer le stockage des données en ligne à un autre service en ligne. Là, il vous sera nécessaire de créer un compte, mais au moins n'aurez-vous à le faire qu'une seule fois pour toutes les autres applications compatibles. Nous verrons plus loin comment monter nous-mêmes notre propre stockage, mais pour le moment nous nous contenterons d'utiliser une instance publique en bêta offerte par **5Apps**.

Pour créer votre compte, rendez-vous sur https://5apps.com/users/sign_up?site=storage. Vous pouvez utiliser un identifiant **GitHub** ou **Persona** pour vous identifier, ou encore votre adresse email. Cela fait, identifiez-vous pour accéder à votre page personnelle.

De fait, comme il ne s'agit que d'un espace de stockage, il n'y a que très peu de fonctionnalités associées à cette page. Une jauge en haut à droite vous permet d'évaluer le volume de données que vous utilisez. Vous disposez de 1 Gio, de quoi voir venir dans un premier temps. Bien que cela n'apparaisse pas encore, c'est également ici que vous pourrez revenir sur les autorisations d'utiliser vos données que vous avez accordées aux applications. Chaque application s'y trouve listée avec l'indication du dossier auquel elle a accès et en quel mode (lecture seule ou lecture et écriture). Vos données sont en effet structurées en dossiers, et constituent un véritable système de fichiers **[1]** permettant de gérer les droits des différentes applications – et éventuellement de les faire collaborer, ce que ne permet pas le stockage dans le navigateur seul.

Vous trouverez également sur cette page un élément important : il s'agit de votre adresse pour ce compte. Elle ressemble à un email ayant la forme votrelogin@5apps.com, mais personne ne pourra vous y écrire. Cette adresse est à utiliser dans les applications compatibles.

1.3 Liaison entre Litewrite et RemoteStorage

Retour à Litewrite. Sur la page en haut à gauche vous trouverez un lien ***Connect remote storage***. Lorsque vous cliquez dessus, une boîte de dialogue s'ouvre vous invitant à saisir votre adresse RemoteStorage, donc votrelogin@5apps.com. 5Apps vous demande alors d'autoriser l'application litewrite.net à accéder à votre espace de stockage dans le dossier `/documents`. Cliquez sur ***Allow***. Et le tour est joué.

Vous retournez alors automatiquement sur Litewrite et vos données sont désormais stockées sur 5Apps. Le lien ***Connect remote storage*** est remplacé par sa seule icône, ce qui vous permet de savoir que vous êtes identifié sur votre espace de stockage. Si vous cliquez sur l'icône, une boîte de dialogue vous permet de vérifier l'identité active, de forcer une synchronisation immédiate et enfin de déconnecter votre session Litewrite de votre espace de stockage.

1.4 Première conclusion

Le fait que l'on puisse utiliser Litewrite avant même de se préoccuper du stockage des données permet de ne pas gêner l'utilisateur avec cette question, un peu plus complexe qu'à l'ordinaire et pour beaucoup inhabituelle, avant qu'il en ressente le besoin (pour des raisons d'ubiquité ou de sauvegarde) et donc qu'il en comprenne le sens : il comprendra mieux son action, la réussira plus facilement et s'y pliera plus volontiers que si elle lui était imposée *a priori* avant même de pouvoir expérimenter le produit. Ce n'est malheureusement pas le cas de toutes les applications : **Laverna**, un autre outil de prise de notes, plus élaboré que Litewrite, exige pour commencer une adresse RemoteStorage ou un compte **DropBox**.

Quant à l'ensemble de la démarche proprement dite, les applications qui se connectent à nos comptes sociaux pour des raisons plus ou moins légitimes ou celles que vous installez sur votre téléphone nous ont habitués à répondre à des demandes d'autorisation. Du coup, le pli à prendre est déjà pris pour beaucoup. Le plus difficile sera surtout de comprendre l'intérêt de créer un compte sur un serveur uniquement pour stocker des données utilisées par d'autres services... Bref, notre travail pédagogique consistant à expliquer ce que la liberté signifie en informatique continue.

2 Nos propres instances

2.1 Litewrite

Commençons par Litewrite. Vous devez disposer d'un serveur Web sur lequel déposer votre instance. Il n'y a pas d'autres prérequis. Dans le dossier de votre choix, disons **/var/www/html**, exécutez la commande :

```
# git clone https://github.com/litewrite/
litewrite.git
```

Ouvrez maintenant votre navigateur à l'adresse correspondante, dans notre cas http://localhost/litewrite. Et là, vous devriez accéder à une page identique à celle que vous aviez vu lorsque vous vous étiez connecté à http://litewrite.net. La connexion à votre RemoteStorage ne devrait poser aucun problème.

2.2 Remote Storage

L'installation d'un RemoteStorage est un peu plus compliquée. Rien de surprenant à cela, étant donné qu'il est chargé de stocker nos données, quelques précautions concernant la sécurité sont utiles à prendre. Voyons comment nous devons procéder.

Il existe plusieurs projets de RemoteStorage **[2]** et vous n'aurez qu'à choisir celui qui a votre préférence. Pour ma part j'ai choisi **PHP RemoteStorage [3]**.

Nous allons procéder à une installation dans un environnement classique, à savoir un serveur **Apache** avec PHP activé (**mod_php**). Mais les prérequis ne s'arrêtent pas à cela : notre serveur Web devra supporter les connections en HTTPS et PHP RemoteStorage devra fonctionner sur son propre domaine. En ce qui concerne HTTPS, il est recommandé de vérifier sa configuration avec certains services en ligne **[4]** afin de s'assurer d'un bon fonctionnement avec tous les navigateurs. En ce qui concerne le domaine dédié, il ne s'agit pas d'une obligation, mais cela simplifie considérablement la tâche de configuration. En tout état de cause, si vous voulez dans un premier temps seulement *tester* votre propre instance de PHP RemoteStorage, rien ne s'oppose à ce que vous utilisiez https://localhost pour avoir un aperçu. Pour ma part, lors de mes tests initiaux, afin d'éviter la complexité inhérente à la mise en place d'une telle configuration dans un environnement déjà existant, j'ai procédé à l'intérieur d'un **Docker** en mappant le port **443** de l'hôte sur celui du Docker :

```
# docker run -p 443:443 -i -t debian /bin/bash
```

En premier lieu, assurons-nous que toutes les dépendances requises sont présentes :

```
root@7918c12fad52:/# apt-get update
root@7918c12fad52:/# apt-get install apache2 php5 php5-curl libapache2-mod-xsendfile php5-
sqlite git
```

Téléchargeons PHP RemoteStorage au bon emplacement :

```
root@7918c12fad52:/# cd /var/www
root@7918c12fad52:/var/www# git clone https://github.com/fkooman/php-remote-storage.git
root@7918c12fad52:/var/www/php-remote-storage# cd php-remote-storage
```

Pour procéder à la mise à jour des dépendances PHP du projet, il faut recourir à **Composer**, qui n'est pas installé dans mon Docker. J'y remédie de cette façon :

```
root@7918c12fad52:/var/www/php-remote-storage# php -a
php > $composer = file_get_contents('https://raw.githubusercontent.com/composer/
getcomposer.org/master/web/installer');
php > file_put_contents('composer_install.php',$composer);
php > rm
All settings correct for using Composer
Downloading...

Composer successfully installed to: /var/www/php-remote-storage/src/composer.phar
Use it: php composer.phar
php > exit
root@7918c12fad52:/var/www/php-remote-storage# rm composer_install.php
root@7918c12fad52:...# mv composer.phar /usr/local/bin/composer
root@7918c12fad52:...# composer install
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
- Installing symfony/yaml (v2.8.1)
Downloading: 100%
phpunit/phpunit suggests installing phpunit/php-invoker (~1.1)
Generating autoload files
```

Mettons en place la configuration par défaut, créons le répertoire destiné à recevoir les données (Git ne versionne pas les dossiers vides...) et créons un premier utilisateur (la base de données n'en contient pas par défaut) :

```
root@7918c12fad52:/var/www/php-remote-storage# cp config/server.yaml.example config/
server.yaml
root@7918c12fad52:...# mkdir data
root@7918c12fad52:...# chown www-data.www-data data
root@7918c12fad52:...# bin/add-user utilisateur m@tDe_pa55e
```

Procédons maintenant à la génération du certificat SSL :

```
root@7918c12fad52:/var/www/php-remote-storage# openssl genrsa -out /etc/ssl/private/
storage.local.key 2048
root@7918c12fad52:...# chmod 600 /etc/ssl/private/storage.local.key
root@7918c12fad52:...# openssl req -subj "/CN=storage.local" -sha256 -new -x509 \
-key /etc/ssl/private/storage.local.key \
-out /etc/ssl/certs/storage.local.crt
```

En dernier lieu, paramétrons Apache avec la configuration par défaut fournie par PHP RemoteStorage, et activons les modules nécessaires :

```
root@7918c12fad52:/var/www/php-remote-storage# cp contrib/storage.local.conf.ubuntu /etc/
apache2/sites-available/storage.local.conf
root@7918c12fad52:...# a2enmod rewrite
root@7918c12fad52:...# a2enmod headers
root@7918c12fad52:...# a2enmod ssl
root@7918c12fad52:...# a2ensite default-ssl
```

Si vous ne travaillez pas dans un Docker, il vous suffira de relancer le service Apache :

```
# service apache2 restart
```

Si, comme moi, vous êtes dans une image Docker, Apache ne peut fonctionner en service, il faut le lancer depuis la ligne de commande :

```
root@7918c12fad52:/var/www/php-remote-storage# /usr/sbin/apache2ctl -D FOREGROUND
```

À partir de là, ouvrez votre navigateur à l'url https://localhost. Vous devrez accepter une exception de sécurité étant donné que nous avons mis en place un certificat autosigné. Dès lors, vous devriez découvrir la page par défaut d' Apache. C'est que la configuration que nous avons mise en place vaut pour le nom d'hôte **storage.local**. Or ce nom ne correspond pour le moment à rien. Il faut modifier notre fichier **/etc/hosts** pour qu'il soit reconnu par notre système :

```
# echo 127.0.0.1 storage.local >> /etc/hosts
```

Maintenant, lancez votre navigateur sur https://storage.local. Le résultat sera radicalement différent : une page vous annonce la bienvenue sur RemoteStorage et il vous est possible de vous identifier en cliquant sur la silhouette à droite. Nous avions créé plus tôt un utilisateur, il suffit de reprendre ses paramètres.

2.3 Lier tout cela

Pour terminer notre démonstration, retournons sur notre instance locale de Litewrite http://localhost/litewrite. Et connectons-nous avec notre utilisateur, en utilisant l'adresse utilisateur@storage.local. La procédure se déroule sensiblement de la même façon avec les instances publiques, à ceci près que Litewrite fonctionnant en HTTP, nous avons droit à un petit avertissement de la part de PHP RemoteStorage nous indiquant que le jeton de sécurité utilisé pourrait être compromis. À noter donc pour une mise en production : utiliser une connexion sécurisée.

Bien évidemment pour une mise en production, il est nécessaire d'utiliser un domaine accessible en DNS pour éviter d'avoir à manipuler le fichier **hosts** de toutes les machines à partir desquelles vous voulez accéder à vos données.

Conclusion

La démonstration est faite qu'il est possible de développer des applications fonctionnant dans le navigateur seul offrant en option l'ubiquité des données en préservant leur confidentialité. Ces applications sont très simples à utiliser et à mettre en œuvre, la seule difficulté étant de mettre en place son propre serveur de données pour ceux qui le souhaitent. On regrettera en particulier la nécessité pour PHP RemoteStorage de fonctionner à la racine du domaine, ce qui rend plus complexe sa mise en œuvre dans un environnement existant. Mais cette tâche n'est pas insurmontable, surtout pour un lecteur de GNU/Linux Magazine France ! ■

Références

[1] Une application existe d'ailleurs pour le parcourir, remoteStorage Brower : https://remotestorage-browser.5apps.com

[2] Une liste des solutions existantes : wiki.remotestorage.io/Servers

[3] github.com/fkooman/php-remote-storage

[4] SSL Decoder (ssldecoder.org/) et SSL Labs (www.ssllabs.com/ssltest/) sont recommandés par le site de PHP RemoteStorage.

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