NOUVELLE FORMULE | NOUVELLE FORMULE | NOU

programmez.com

[Programmez!]

Le magazine des développeurs

217 AVRIL 2018

Choisir son moteur 3D

Unity, Unreal Engine, CryEngine, Three.JS, Babylon.JS, Ogre, Irrlicht

Yocto

Construire son Linux embarqué

Créer un cluster de Raspberry Pi avec OpenFaaS

Delphi

Le retour en force du Pascal

© DR

Pourquoi les tests sont-ils indispensables ?

M 04319 - 217 - F: 6,50 € - RD

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

Printed in EU - Imprimé en UE - BELGIQUE 7 € - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 € - TOM 1020 XPF - MAROC 55 DH

embarcadero

RX RAD Studio™ 10.2

RAD Studio est la solution la plus rapide pour coder, compiler, packager et déployer des applications natives multiplateformes.

Avec la version test de RAD Studio, vous découvrirez:

Des applications multiplateformes – Pour ne coder qu'une fois et déployer sur des cibles multiples

Les contrôles VCL et composants FireMonkey: Pour concevoir des applications natives attractives sous Windows, MacOS, Android et iOS

Des accès simplifiés aux API, aux capteurs et aux services de l'appareil

La seule solution pour créer des applications natives compilées à partir d'une seule base de code source

Lire l'article complet sur RAD Studio page 48 dans ce magazine

Téléchargez votre manuel "Object Pascal" par Marco Cantu

Le manuel "Object Pascal" gratuit par Marco Cantu est un guide complet de 500 pages couvrant les principales fonctionnalités du langage, la programmation orientée objet avec Object Pascal et les dernières fonctions (génériques, méthodes anonymes, réflexion) des compilateurs Delphi.

Ultilisez ce lien pour télécharger le manuel: http://bit.ly/2FLEQ5J

Commencez à développer des applications de nouvelle génération grâce au manuel "Object Pascal"

Applications bureautiques ou client-serveur, modules pour serveurs Web de masse, middleware, automatisation bureautique, applications pour les derniers téléphones et tablettes, systèmes d'automatisation industrielle, réseaux téléphoniques virtuels par Internet... Object Pascal est aujourd'hui utilisé couramment pour tout ces types de solutions.

L'objectif de cet ouvrage est d'expliquer les concepts clés et de présenter de brèves démonstrations afin que le lecteur puisse les essayer, les expérimenter et les compléter pour parfaitement en maîtriser le fonctionnement.

Ce manuel offre une couverture complète de la dernière version du langage Object Pascal ; il est destiné tant aux nouveaux développeurs qu'à ceux ayant expérimenté des langages similaires.

Le manuel "Object Pascal" utilise une approche logique et progressive couvrant le fonctionnement du langage et ses bonnes pratiques.

Contactez notre revendeur agréé Barnsten pour votre essai gratuit

Web: www.barnsten.com Tel.: 0972192887

EDITO

Si Philippe Kahn voyait ça…

Retour en arrière. Un des outils de développement phare des années 80 était le fameux Turbo Pascal qui a fait la joie et le malheur de tant de développeurs. Philippe Kahn arrive dans l'univers informatique dès la fin des années 1970. Mais il devient un acteur incontournable à partir de 1983 lorsqu'il cofonde Borland.

C'est en 1995 que Delphi apparaît, utilisant le langage Pascal. C'est rapidement un joli succès. Mais cette année est aussi l'année des crises et problèmes stratégiques. Philippe Kahn part définitivement en 1996. Pourtant, Delphi évolue tranquillement année après année malgré tous les aléas : changements de noms, rachats, etc. L'aventure Kylix, une version Linux de l'IDE, se solde par un échec cuisant.

Aujourd'hui Delphi est redevenu un des outils phares de la gamme IDE d'Embarcadero et évolue à son rythme annuel. Delphi dépassé ? Non car il intègre les évolutions et les technologies les plus récentes. Trop centré sur Windows ? Non, Il cible de multiples plateformes. Toujours adepte du Pascal ? Ah là effectivement, si vous n'aimez pas le langage Pascal, on va avoir un problème :-)

Bref, il n'y a pas que Visual Studio, PHP Edit, IdealJ, XCode ou Eclipse dans la vie. Delphi est un IDE solide. Par contre je ne dis pas que tous les vieux IDE sont utilisables. J'ai réinstallé Visual Basic pour DOS 1.0 sur un vieux PC sous MS-DOS et Windows 3.1. Le truc que je n'avais plus vu depuis 1993… Bizarrement, il ne m'a pas manqué…

Autre sujet intemporel pour le développeur : les tests. Non, ne partez en courant. Le test n'est pas le sujet le plus excitant mais difficile de s'en passer. Si vous êtes en développement agile, le test y est incontournable. Enfin, plutôt un domaine précis des tests. Car les tests couvrent une réalité très large. Comme vous le verrez dans notre dossier, chaque type de test(s) se complète. Vous aurez des tests au niveau code pur, d'autres plus orientés utilisateurs et interface. Et chaque test s'attaque à un niveau précis de votre application.

Je vous laisse à votre magazine préféré, j'installe BeOS.

François Tonic
ftonic@programmez.com

SOMMAIRE

Dans le prochain numéro !
Programmez! #218, dès le 27 avril 2018

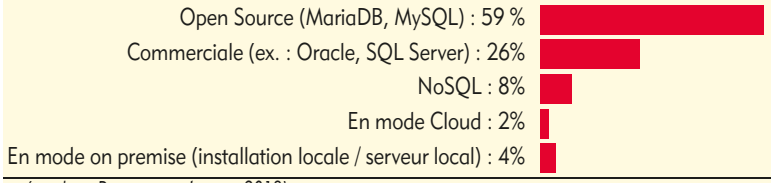
SPÉCIAL 1998-2018

Programmez! a 20 ans.

Parlons technologies, codes et futurs.

Un numéro à ne pas rater !

Vous utilisez un SGBD, quelle est sa principale caractéristique ?



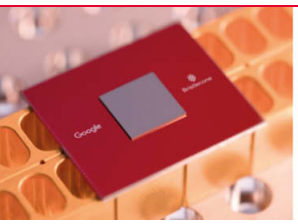
(sondage Programmez!, mars 2018)

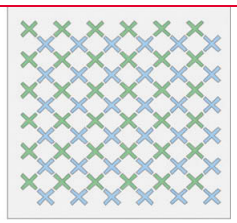
Le **No Code** est-il utopique et irréaliste ? Il semble que oui…

Zend Framework est clairement en perte de vitesse depuis le rachat de Zend par Rogue Wave en 2015. Aujourd'hui deux frameworks PHP sont à surveiller : Laravel et Codeigniter. Sans oublier Symfony.

Depuis presque 2 mois, les failles **Spectre** et **Meltdown** ont disparu des medias ! Eh hop, plus de failles…

Python 2.7 s'arrêtera officiellement le 1er janvier 2020 ! A cette date : plus aucune mise à jour !





Processeur quantique chez Google : vers la suprématie quantique ?

Google a dévoilé début mars un processeur quantique de 72 Qubits : Bristlecone. Cette annonce montre clairement la grande ambition de Google dans ce domaine. Mais nous n'en sommes qu'au début. Une des difficultés pour les processeurs quantiques est de réduire les taux d'erreurs et donc garantir un fonctionnement optimal. Car le nombre de qubits ne fait pas tout…

L'annonce officielle : https://research.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html

Oracle fait le ménage dans certaines piles Java

Oracle a dévoilé une évolution de la roadmap de Java Client. Java Client inclut la partie déploiement et les interfaces (Java UI). Le document de mars indique les principaux points suivants :

- les mises à jour de Java SE 8 sont étendues jusqu'au début 2019 ;
- Oracle continuera à supporter Java SE 8 Web Start même si Oracle incite les développeurs à migrer vers d'autres mécanismes ;
- JavaFX sera sorti de la pile Java avec Java SE 11, la technologie devra trouver des communautés / fondations / autres pour continuer à vivre ;
- Swing et AWT resteront supportés jusqu'en 2026 ;
- l'avenir de Swing, JavaFX et AWT reste à définir au-delà des dates annoncées ;
- les applets seront retirés à partir de Java SE 11. Conséquence logique car les navigateurs ont retiré les plug-in Java.

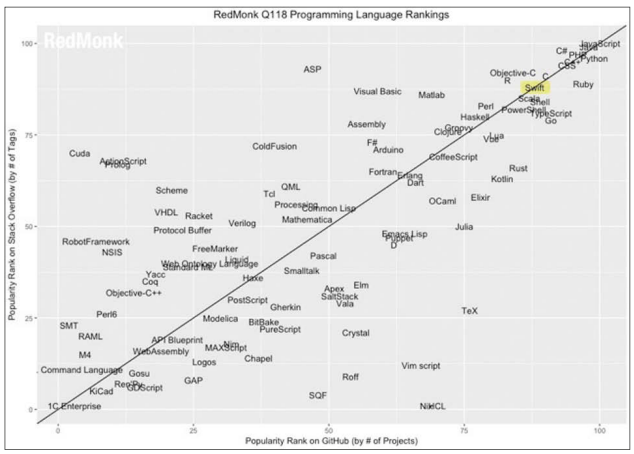
Les langages à surveiller / à apprendre en 2018

Go et éventuellement	Peu de surprises à attendre pour 2018 / valeurs sûres	Les langages en perte de vitesse
DART	C++	Ruby
Rust	Java	ASP.Net
Scala	C#	VB.Net
Langages fonctionnels	PHP	Objective-C
R	Swift	Groovy
Kotlin		PHP

UNE STARTUP PEUT AUSSI FERMER…

Avoir des startups c'est bien mais il ne faut pas oublier qu'au-delà, il faut des marchés, des clients pour pouvoir exister. Comme une entreprise classique, la startup ne doit pas l'oublier. La French Tech s'affiche partout. C'est très bien mais ce n'est pas un gage de réussite. Giroptic a fermé ses portes le 5 mars dernier. L'un des espoirs du fondateur était un rapprochement avec un acteur du smartphone mais rien n'arriva. Et la société avait vendu 20 000 caméras en 2017. Des retards de production avaient pénalisé l'entreprise et la concurrence n'a pas attendu pour s'emparer du marché. Résultat : on ferme.

Le classement des langages selon GitHub



Chaque mois nous vous donnons quelques indices sur la popularité des langages. Nous reprenons un index selon l'usage des langages dans les projets sur GitHub. Le gros peloton de tête reste : JavaScript, Java, Python, PHP, C#, CSS (sic).

Objective-C reste légèrement devant Swift mais la tendance est désormais là : Swift supplante Objective-C. Rien de plus logique : Apple a clairement dit que son langage historique, issu de NextStep, n'était pas l'avenir…

RASPBERRY PI 3 MODEL+ : du mieux et de la frustration

La fondation Raspberry Pi a sorti une évolution de la Pi 3 avec le Model B+. Cette itération améliore un peu les caractéristiques matérielles de la carte :

CPU : Cortex-A53 64-bit 1,4 Ghz
Ethernet : 1 Gb sur USB 2
Wifi : 2,4 & 5 Ghz 802.11 b/g/n/ac et Bluetooth 4.2

Le processeur gagne un léger surcroît de puissance, la partie réseau sans fil fait des progrès. La partie Ethernet était limitée au port 10/100 classique. Le nouveau modèle affiche un port 1 Gb mais sur USB 2. Ce qui peut refroidir notre joie. Car le fait de passer par l'USB 2 bride les débits et la fiche officielle prévient : 300 Mbps maximum mais c'est toujours mieux que la version précédente ! La mémoire vive reste bloquée à 1 Go ce qui retire l'intérêt du processeur 64-bit. Espérons que la prochaine itération proposera du mieux.

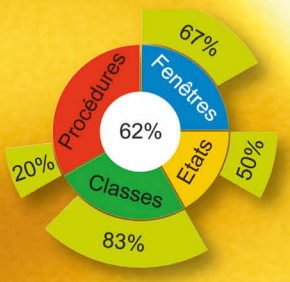
Bonne nouvelle : le prix ne change pas.

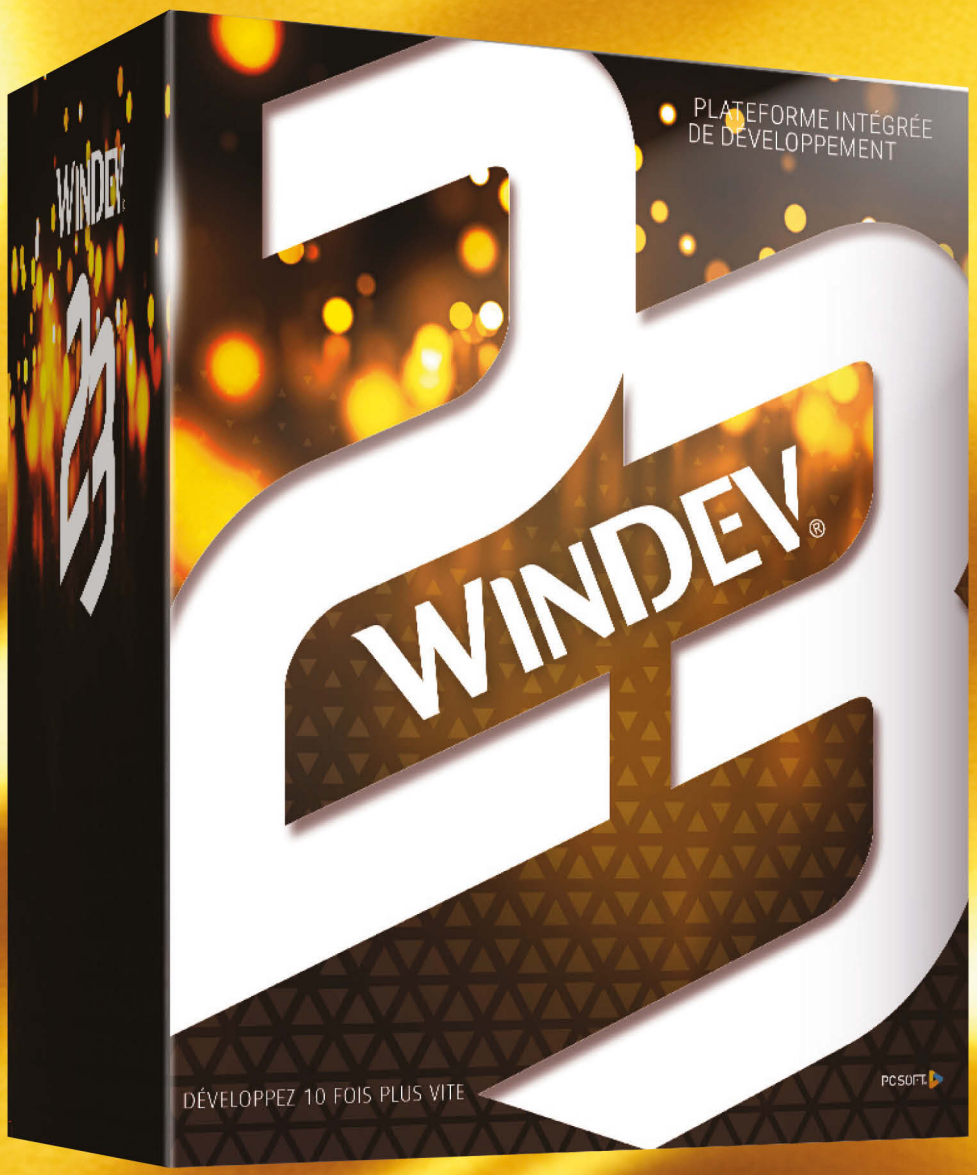
TESTS AUTOMATISES

La phase de **tests automatisés** est une partie importante du cheminement qui amène la qualité du logiciel.

WINDEV 23 propose les tests de **non-régression**, la réalisation de tests unitaires, de tests d'intégration, de tests de charge, de tests d'IHM, et le Code Coverage.

Par simple enregistrement de vos manipulations, des scénarios de test sont générés et ensuite rejouables à l'envi pour **valider** chaque nouvelle version de vos logiciels.





COMMUNAUTÉS

Annoncez vos meetups, conférences sur Programmez ! :
ftonic@programmez.com

GDG Toulouse
17 mai : WebVR et intégration continue.

ParisJUG
15 mai : spécial 10 ans de ParisJUG !

AVRIL

Add Fab : 11 & 12 avril – Paris
Pour sa deuxième édition, Add Fab rassemblera les 11 et 12 avril 2018, Porte de Versailles, les acteurs les plus importants de l'industrie de l'Impression 3D ou Fabrication Additive. Regroupant les sociétés les plus représentatives et novatrices du secteur, de la Start-up à l'entreprise internationale, afin de présenter une offre exhaustive du secteur : logiciels, imprimantes, prototypage, matériaux, équipements, outillage et formation. En parallèle se déroulera un cycle de conférences, d'ateliers et de formations en direct.

MakemeFest Angers revient en avril !
Le nouveau salon maker reviendra en avril à Angers. L'événement 2017 avait réuni plus de 70 startups et makers et plus de 3000 visiteurs. Pourquoi pas vous ?
Site officiel : http://makeme.fr

MAI

PHP Tour 2018
PHP Tour 2018 s'arrêtera à Montpellier les 17 et 18 mai. Le programme s'annonce chargé, comme toujours : moteur PHP 7, RGPD, tests, Rex, Synfony, le rôle de la documentation, l'asynchronisme, OpenAPI, GraphQL, etc. Programme complet : https://event.afup.org/phptourmontpellier2018/programme/

JUIN

EclipseCon 2018
Les 13 et 14 juin, la grande conférence EclipseCon se tiendra une nouvelle fois à Toulouse. L'appel à conférence se terminera le 19 mars. N'hésitez pas à proposer des sessions : https://www.eclipsecon.org/france2018/

DevFest Lille
Le Devfest Lille 2018 se passera le 21 juin 2018 sur une seule journée pour 400 personnes prévues. La conférence prendra place dans les locaux de l'IMT Lille-Douai (20 Rue Guglielmo Marconi, 59650 Villeneuve-d'Ascq). Le site est accessible via https://devfest.gdglille.org/
Le CFP (http://devfestlille.cfp.io/) est en cours jusqu'au 1 avril 2018 (pour un programme disponible début mai).

Hack in Paris
Du 25 au 29 juin, Hack in Paris sera la grande conférence sur la sécurité et le hacking en France. Un événement à ne pas râter !

À VENIR

Oracle Cloud 2018
Oracle aime les développeurs et veut le prouver avec cette journée 100 % code, 100 % développeur. L'événement se déroulera dans plusieurs villes dans le monde dont Paris. Nous y trouvons des sessions, des ateliers. La date française n'est pas encore connue.

Conférences techniques

Xebia organise 4 conférences techniques :

- DataXDay, conférence autour du futur de la Data Science, des prochaines innovations en matière de temps réel et des incontournables en architecture Big Data. 17 mai 2018 - dataxday.fr
- Paris Container Day, conférence pionnière en France dédiée à l'écosystème des conteneurs. Cette année, le thème sera : "Vivre avec l'Orchestration". 26 juin 2018 - paris-container-day.fr/
- FrenchKit, la première conférence française dédiée aux développeurs iOS et macOS. 20 et 21 septembre 2018 - frenchkit.fr/
- XebiCon, la conférence qui vous donnera les clés pour tirer le meilleur des dernières technologies : Data, Architecture, mobilité, DevOps, etc. 20 novembre 2018 - xebicon.fr

Donnez le meilleur à vos développeurs et à vos équipes !

Abonnez-les à [Programmez!] Le magazine des développeurs

© bowie15

Depuis 20 ans, le magazine Programmez! est un véritable guide mensuel de veille, de décryptage, de tendances, d'articles et de dossiers qui va à l'essentiel de l'actualité dans toutes les sphères de la programmation.

Nous proposons aux entreprises et aux écoles des tarifs spéciaux pour les équipes à partir de 5 personnes.

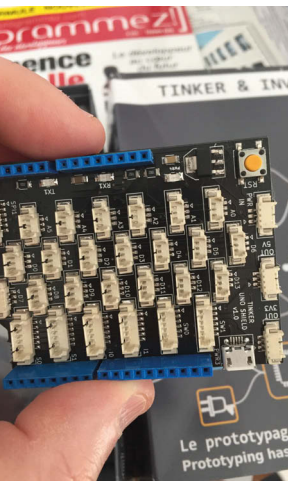
Contactez-nous dès aujourd'hui pour un devis personnalisé.

Contact : Olivier Pavie
06 12 36 29 36
opavie@programmez.com

Disponible en version papier et en version PDF. France et étranger.

François Tonic

Tinker : une alternative aux modules Grove !



Le shield

Hack&Invent est une jeune société française que nous avions découverte à la dernière Maker Faire Paris avec une solution ambitieuse et excitante : une alternative aux composants Grove de Seeed Studio que nous utilisons régulièrement dans nos montages. Prototypage rapide, mais parfois un peu cher ! Nous étions impatients de tester le kit de démarrage. C'est rapide et de qualité !

L'idée est simple : proposer des capteurs et composants que l'on intègre facilement à son prototype et sans soudure. Le système Grove doit son succès à sa connectique rapide et à un développement simplifié : une librairie où on code les interactions / comportements. Pour réaliser le montage, on utilise une shield d'extension que l'on connecte typiquement à une Arduino Uno.

Le shield de connexion

Tinker reprend ce concept Plug&Play. Le shield Tinker propose bien plus d'extension que Grove (version 1 du shield) :

- 6 connecteurs analogiques : A0 à A5 ;
- 14 connecteurs digitaux : D0 à D13 ;
- 2 connecteurs I2C ;
- des connecteurs SPI, série, 3,3V et 5V en sortie.

On dispose de deux types de connecteurs, plus ou moins larges. Les ports Ax et Dx exigent moins de connexions, 3 fils. Les ports I2C, SPI, série, UART exigent jusqu'à 6 fils. Par exemple, en SPI, il faut connecter MOSI, MISO, SCK, CS1 & 2, GND. En Grove, nous avons un unique connecteur de 4 broches. Les ports série sont utiles pour des modules nécessitant un port série. Tous les ports Dx ne sont pas alimentés.

Ce qui est intéressant ici est la possibilité de chaîner les modules, ce que Grove ne permet pas, sauf sur quelques composants précis. Le constructeur distingue plusieurs catégories de modules :

- affichage ;
- actionneurs ;
- capteurs ;
- logiques.

Premiers pas

Pour commencer, il nous faut une Arduino UNO et le shield Tinker. On enfonce le shield sur la UNO. Il vous faudra télécharger la librairie TinkerLib qui permet de piloter et de manipuler les capteurs depuis le shield.

Attention : le zip disponible depuis le Github officiel comprend la librairie et tous les exemples. Depuis l'IDE Arduino, chargez uniquement le dossier lib quand vous allez rajouter la TinkerLib depuis les menus. C'est tout.

On connecte la UNO. On vérifie que la carte UNO est bien déclarée et que le port COM est le bon. La programmation diffère fondamentalement assez peu. Il faudra juste apprendre les réflexes Tinker : déclarer la librairie et sur les déclarations des modules… C'est une question de quelques heures…

Vous voulez utiliser un composant standard du marché par exemple en écran OLED 0,96'' via un port I2C ou SPI ? Pas de souci, on dénude un fil Tinker, on repère les bonnes broches, un peu de soudure et c'est bon. Dans notre cas, nous avons un OLED générique I2C. Tout d'abord, on vérifie que le module I2C est bien reconnu (mieux vaut le vérifier). Pour ce faire, on utilise un simple I2C Scanner. Quand tout est validé, on va tester l'OLED avec un simple code d'exemple venant avec la librairie Adafruit_SSD1306. On pensera à installer la librairie complémentaire : Adafruit GFX.

Dans le code Arduino, on ne change rien. On rajoute deux lignes au début :

```
#define TINKER_UNO_SHIELD
#include "tinkerlib.h"
```

On compile, on téléverse. Et notre OLED s'anime !

Un autre attrait de Tinker est la possibilité d'utiliser des capteurs indépendamment du Shield. On peut construire des chaînages simples. Par exemple, on alimente le montage avec module Jack pour brancher une pile 9V. Ce module permet de connecter jusqu'à 4 montages (4 ports). Idéal pour débuter en douceur.

Notre avis

Nous étions déjà enthousiastes à la Maker Faire Paris 2017. Et cette bonne impression s'est confirmée. La fabrication est de qualité, il existe de nombreux modules, l'UNO Shield offre beaucoup de connecteurs. La programmation n'est pas très compliquée même s'il faudra apprendre de nouvelles fonctions. Le chaînage des éléments ou encore la possibilité de créer des montages indépendants de la carte de prototypage, deux très bonnes idées ! Le petit livret inclus dans les kits permet un démarrage rapide.

À noter que Tinker peut aussi se programmer avec Ardublocky et mBlock, environnements de programmation pour les enfants.

La documentation disponible est encore limitée pour pouvoir explorer l'univers Tinker. À l'usage, la taille des connecteurs utilisée n'est pas trop pratique, surtout quand on a de gros doigts. Nous ne pouvons qu'encourager Hack&Invent et espérer que la communauté va rapidement croître ! •

LES KITS DISPONIBLES

Hack&Invent propose 3 kits :

- Box starter : inclut les modules les plus courants (bouton, LEDs, UNO Shield, etc.). Quelques fils supplémentaires seraient un plus. 77 euros
- Box Pro : un peu plus fourni. 99 euros
- Box Advanced : le plus complet. 120 euros

Aucun kit ne propose tous les modules Tinker et les kits n'incluent qu'un unique UNO Shield. Dommage. Par contre, le prix des modules à l'unité est très compétitif ce qui n'est pas toujours le cas des modules Grove.

Les +	Les –
Simplicité des montages	**Documentation incomplète**
Qualité de fabrication	**Pas de LCD ou d'OLED officiel**
Tinker Shield	**Communauté faible**
Les nombreux modules	**Pas de coffret incluant tous les modules**
Compatibilité Arduino	

Questions – réponses avec Badr (Hack&Invent)

Comment est né Tinker ?

Au départ nous réalisions des drones sur mesure et différentes cartes électroniques (stations inertielles, capteurs embarqués...), pour le compte de la société DRONETAIR SAS et pour différents clients. Avant de produire les cartes finales (complexes et coûteuses), nous réalisons des sous-modules pour les différents capteurs à intégrer. De cette manière on s'assure du bon fonctionnement des sous-modules avant d'intégrer la carte finale. Nous étions donc amenés à développer plusieurs modules de prototypage avant de produire la carte finale. D'où l'idée de mettre en place des cartes TINKER pour prototyper.

Au départ nous avions uniquement les headers à souder dans les cartes de prototypage, nous les utilisions avec des fils Dupont. Mais rapidement nous nous sommes rendu compte que des faux contacts se produisaient. Nous étions donc amenés à utiliser des connecteurs pour fiabiliser le raccordement. Le grand avantage de ces modules de prototypage est de faire abstraction de la complexité éventuelle du circuit électronique pour se concentrer sur le projet final. L'utilisation des fils Dupont a des limites pendant le prototypage.

Puisque les modules étaient utiles pour nous, nous voulions commencer à les commercialiser.

Quels publics / utilisateurs ciblez-vous ?

Les modules TINKER sont destinés à tout public programmant sur Arduino par exemple. Nous avons eu des sessions de présentations avec des profs de technologie de collèges. TINKER peut être utilisé par un public assez jeune. Avec l'abstraction des modules, on se concentre sur le code et le projet.

Qu'est-ce qui a été le plus difficile dans ce projet ? Concevoir les modules, la partie purement électronique, le shield ou la librairie de développement ?

Il y eut plusieurs difficultés, mais deux principales :

- nous sommes passés par plusieurs itérations afin de trouver les bons connecteurs et la bonne combinaison de modules afin de mettre en place un écosystème de modules assez complet. Aujourd'hui nous avons environ 100 modules, dont une trentaine disponibles. Nous mettrons en vente les modules au à fur et à mesure.
- vu la concurrence des fournisseurs chinois, nous voulions garder des prix compétitifs. Il était difficile de trouver des fournisseurs européens de cartes PCB acceptant de réaliser des cartes à un prix compétitif. Finalement nous avons trouvé un fabricant qui soutient notre projet à condition d'avoir un volume mensuel relativement important. Ce qui nous a permis de lancer le projet en phase semi-industrielle.

Vous proposez des modules Tinker logic, quelles sont leurs utilités dans un montage et comment sont-ils utilisés dans le code ?

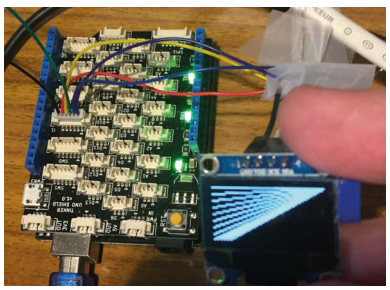
L'avantage des modules Logic est de permettre l'apprentissage de la logique combinatoire en manipulant des boutons et des Leds par exemple. Ils permettent aussi de réaliser des montages simples interactifs sans avoir besoin d'utiliser un microcontrôleur et à le programmer.

Il est aussi possible de les associer à un microcontrôleur pour faire des logiques simples locales sans avoir à tirer un fil complémentaire vers le microcontrôleur. Les modules Logic sont une de nos originalités

Vous allez sortir un module OLED, pourquoi aucun afficheur LCD ou OLED n'était disponible ?

Les afficheurs ne sont pas encore sortis, car nous étions dans une réflexion structurante. Nous comptons abandonner les connecteurs 6p et n'utiliser que les connecteurs 3p. De cette manière un I2C aura deux connecteurs 3p, un pour la SDA et le second pour la SLC. C'est ce qui a retardé sa sortie, ainsi que plusieurs modules qui utilisent du I2C/SPI/UART.

Nous sommes en ce moment en phase de lancement élargi. Jusqu'à aujourd'hui nous étions en phase bêta. Nous avons récupéré des commentaires du corps enseignant, des fablabs ou makerspaces. Nous avons pris en compte leurs commentaires pour les modules en cours de production. La phase industrielle commence ce mois-ci.



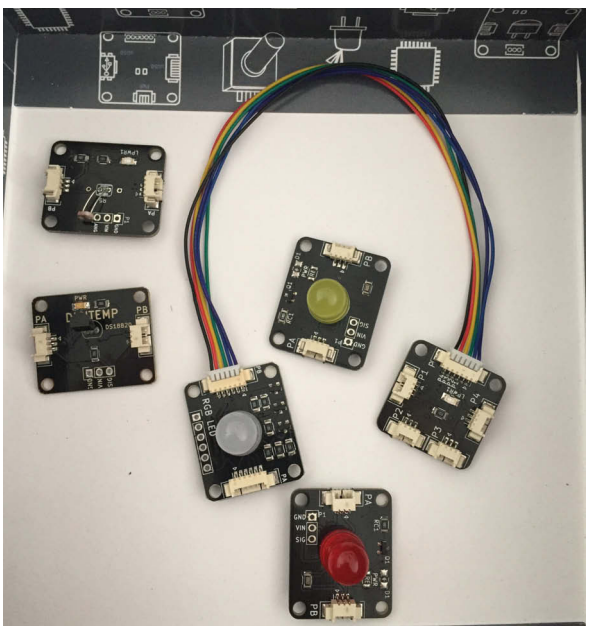
EN VRAC

Si je veux utiliser un capteur du marché : problème ou pas ?

Il est possible de raccorder des capteurs du marché aux modules TINKER. Nous avons prévu des emplacements pour headers à souder dans chaque module. L'utilisateur peut raccorder de cette manière les modules TINKER avec des fils Dupont.

Comment utiliser le connecteur SPI ?

Aujourd'hui nous n'avons pas de modules SPI. Jusqu'à maintenant, nous l'utilisons pour un module d'inertie pour nos drones. Il s'agit d'une station inertielle redondée que nous avons développée pour un client.



Vive les tests

François Tonic

Le terme test, ou plus précisement test logiciel, recouvre une multitude de réalités. Et il est vrai que ce n'est pas toujours simple de s'y retrouver surtout quand on développe des apps pour son plaisir ou que l'on est freelance, voire dans une petite entreprise. Les tests ne sont pas toujours une priorité ou éventuellement une certaine catégorie de tests. Un des réflexes du développeur est d'utiliser la vérification à la volée du code dans son IDE. C'est vrai que c'est pratique. Cela élimine une partie des erreurs les plus "classiques". On peut remercier l'autocompletion et l'intellisense. Mais ce n'est que le premier niveau. Basiquement, on retiend généralement 4 grands types :

- tests manuels ;
- tests d'interface ;
- tests d'intégration ;
- tests unitaires.

Les tests unitaires sont les plus communs car très proches du code, de la classe, du module à tester. Ainsi chaque classe, chaque module aura son test unitaire (d'où son nom). Généralement, on écrit les tests unitaires en même temps que le code, voire avant celui-ci. Il va servir à valider son développement très rapidement : en entrée, en sortie, son comportement. Le test unitaire est à la base de toute bonne pratique de développement et encore plus dans les méthodes agiles. Donc, quels sont les tests possibles et imaginables ? En voici une petite liste, non exhaustive :

- tests de recettes ;
- tests système ;
- tests d'intégration ;
- tests de non-régression ;
- tests de montée en charge et de performances ;
- tests de stress ;
- tests de robustesse ;
- tests d'interface et d'ergonomie ;
- tests manuels ;
- smoke tests ;
- tests fonctionnels.

Bref, quelle que soit la problématique que vous avez, il y a forcément un test :-)

Un test ne s'improvise… enfin pas trop

Théoriquement, les phases de tests ne s'improvisent pas. Il faut les effectuer avec méthodes et avec les bons outils. Si vous effectuez des tests sans avoir une démarche globale, vous allez avoir une couverture partielle du code et de l'application. Par exemple, si vous faites uniquement des tests unitaires, votre code sera effectivement plus ou moins validé mais impossible de savoir si tous les bouts de codes s'intégreront ensuite. Idem pour l'interface si elle est correcte selon le cahier des charges. Impossible non plus de savoir si les performances seront bonnes, etc. Pareillement, si vous vous limitez aux tests fonctionnels, vous n'aurez qu'une vision restreinte de votre projet…

J'ai déjà entendu des étudiants dire : les tests d'intégrations dans une approche d'intégration continue suffisent. Eh bien non, là encore, le test d'intégration ne vous fournit qu'une vue limitée de votre code même si en effet, vous détecterez un certain nombre d'anomalies.

Bref, pour être véritablement efficace, vos tests doivent couvrir plusieurs surfaces car ils sont complémentaires entre eux. N'oubliez jamais que chaque type de tests couvre un domaine précis. Avoir une stratégie de tests est donc indispensable surtout quand vous travaillez en équipe et dans les projets d'importance.

Un plan de tests, des jeux de tests servent réellement à quelque chose (si, si). Les tests de non-régression sont là pour éviter "je ne comprends pas la fonction était ok hier et je n'ai pourtant pas changé grand chose dans le code". Et les tests d'interfaces / d'ergonomie peuvent aussi aider à ne pas entendre "notre app web s'affiche pas mal sur Safari, à moitié sur Edge, mais très bien sur Chrome, par contre, en version mobile, c'est une catastrophe".

Les tests ne sont pas gratuits !

Même si vous utilisez des outils open source, le test coûte cher en ressources humaines et en temps. Il ne faut surtout pas sous-estimer le temps nécessaire pour les réaliser. Traditionnellement, on estime que l'effort de testing représente de 20 à 30 % du projet global ou de son temps de développement. Dans les projets, encore trop souvent, les délais sont trop courts ou alors les équipes ont dépassé le planning initial. On veut mettre en production le plus vite possible, quitte à proposer une app ou un site web instable et bogué. La tentation est grande couper les tests et de raccourcir les heures allouées, au risque de bâcler les tests et de ne pas pouvoir tous les faire. La panoplie des environnements de tests est tellement vaste que l'on trouve facilement des outils adaptés en open source et dans les offres commerciales. Et les plateformes cloud permettent maintenant de créer des labs de tests complets, des bancs de tests que l'on provisionne uniquement en cas de besoin. Cela permet d'éviter de payer des licences perpétuelles ou encore de déployer sur ses serveurs. Les tests unitaires sont généralement les moins coûteux car ils sont automatisés et parfaitement outillés. Ce sont des tests de bases et connus. Ils s'exécutent naturellement dans le processus de développement. Par contre, les tests manuels, d'interface, de montée de charge, de stress, exigent plus de temps à mettre en place et à exécuter et parfois, la logistique en termes d'outils et de jeux de tests est bien plus importante.

Utiliser un environnement de tests ne s'improvise pas. Le réflexe RTFM(1) existe toujours. •

QUELQUES CONSEILS NON EXHAUSTIFS :

1. le test est un job, une compétence ;
2. le test est un réel effort à réaliser. Ne pas le sous-estimer ;
3. pour faciliter les tests et les rendre crédibles, il est indispensable d'avoir des jeux de tests dès le début du développement ;
4. le test s'intègre dans le projet dès le début. N'attendez pas la fin du projet pour commencer à tester. Cela ne servira plus à rien !
5. plus un bug est découvert tôt moins il coûte cher à corriger ! Merci les tests !
6. bien définir les systèmes cibles d'exécution de son app pour pouvoir recréer les mêmes conditions pour ces tests ;
7. les tests vivent et évoluent avec votre projet. Des tests non adaptés seront obsolètes, voire, vous induiront en erreur ;
8. un bon codeur est un codeur avec les outils ! En test c'est pareil ;
9. si des morceaux de codes s'avèrent difficiles à tester, à valider, une phase de refactoring sera à envisager ;
10. 100 % du code est couvert. Bravo ! Mais c'est insuffisant, et surtout, votre couverture de tests sera partielle ;
11. plus les tests sont clairs et simples mieux c'est !
12. automatiser les tests ;
13. ne vous dites pas : "un autre de l'équipe testera !"
14. tenez à jour vos protocoles de tests, votre boîte à outils. Vérifier qu'ils sont parfaitement compatibles avec les IDE, frameworks et langages utilisés ;
15. un outil de tests peut coûter très cher : pour optimiser votre budget, regardez les alternatives open source quand elles existent (attention au périmètre de l'outil) ou opter pour un service à la demande (cloud).

A vous de jouer !

(1) Pour rester poli : lisez le gentil manuel !

NE RATEZ AUCUN NUMÉRO

Abonnez-vous ! [Programmez!] Le magazine des développeurs

Nos classiques

1 an ..... 49€*
11 numéros

2 ans ..... 79€*
22 numéros

Etudiant ..... 39€*
1 an - 11 numéros

** Tarifs France métropolitaine*

Abonnement numérique

PDF ..... 35€
1 an - 11 numéros

Souscription uniquement sur
www.programmez.com

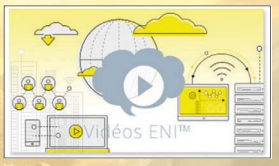
Option : accès aux archives 10€

Nos offres d'abonnements 2018

1 an ..... 59€
11 numéros
+ 1 vidéo ENI au choix :

- Arduino*
  Apprenez à programmer votre microcontrôleur
- jQuery*
  Maîtrisez les concepts de base

eni Editions



2 ans ..... 89€
22 numéros
+ 1 vidéo ENI au choix :

- Arduino*
  Apprenez à programmer votre microcontrôleur
- jQuery*
  Maîtrisez les concepts de base

Offre limitée à la France métropolitaine
** Valeur de la vidéo : 34,99 €*

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT à retourner avec votre règlement à :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex.

☐ **Abonnement 1 an :** 49 €
☐ **Abonnement 2 ans :** 79 €
☐ **Abonnement 1 an Etudiant :** 39 €
Photocopie de la carte d'étudiant à joindre

☐ **Abonnement 1 an :** 59 €
11 numéros + 1 vidéo ENI au choix :
☐ **Abonnement 2 ans :** 89 €
22 numéros + 1 vidéo ENI au choix :

☐ Vidéo : Arduino
☐ Vidéo : jQuery

PROG 217
Offre limitée, valable jusqu'au 27 avril 2018

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

email indispensable pour l'envoi d'informations relatives à votre abonnement

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !
☐ Je souhaite régler à réception de facture

*** Tarifs France métropolitaine**

© AvesJun

N'oubliez pas les tests

A l'occasion de la Journée Française des Tests Logiciels du 10 avril, Programmez! se plonge dans les tests. Quelles sont les nouvelles tendances du testing ? Comment le développeur doit-il utiliser le test ? Dans ce dossier, nous aborderons différents outils et méthodes, particulièrement en DevOps. La rédaction.

Par Olivier Denoo
Président du Comité Français des Tests Logiciels

« Je veux de la qualité, *et plus vite que ça !* »

CFTL
Comité Français des Tests Logiciels

« Que de temps perdu à gagner du temps ! », disait déjà Paul Morand, il y a presque cinquante ans. Rien n'est sans doute plus vrai aujourd'hui encore, dans cette société moderne qui, soumise aux diktats d'une dopamine facile, voue un véritable culte à la performance et à l'immédiateté.

Homo Modernicus modifie son environnement

L'Homo Modernicus s'est affranchi du temps : il mange vite (fast food, micro-ondes), n'attend pas son taxi (Uber), reçoit ses colis même avant de les avoir commandés (Amazon, e-commerce), s'engage vite (job dating, Tinder), communique vite (Twitter, (fake ?) news) et, bien sûr, change d'avis tout le temps(1).

Cette course éperdue contre la montre(2), rendue possible par l'informatisation extensive et systématique de notre environnement, repose sur deux failles majeures dans le chef des Informaticiens(3) :

- Notre incapacité à dire non aux demandes des utilisateurs, même lorsqu'elles nous semblent inacceptables (parce que d'autres diront oui, quoi qu'il en soit) ;
- Notre côté tantôt narcissique, tantôt masochiste qui nous pousse à les encourager, à leur faire croire que tout est possible, de plus en plus vite…et de mieux en mieux.

Pas le temps pour les bouchons, sur nos autoroutes de l'information ! Et puis qu'importe le voyage quand seule la destination compte. Alors nous composons avec les contraintes immuables et les dures contingences de la réalité :

- La version actuelle ne donne pas pleine satisfaction ? Il manque des fonctionnalités essentielles ? Pas grave ! La version suivante corrigera tout cela, elle sort demain. Et si ce n'est pas suffisant la suivante est prévue six heures après.
- Vous préfériez la version d'avant ? Mais mon bon monsieur il faut vivre avec son temps ! Et non vous n'avez pas le choix, c'est du SaaS…
- Documenter ? pour qui ? pour quoi ? De toute façon tout change si rapidement et en mode itératif, rien ne dure vraiment…
- Tester ? Oui, mais juste pour nous assurer que la promesse est à peu près tenue. C'est la loi de la démo permanente, du concept perpétuel. Mieux vaut donc tester moins mais être capable de relivrer plus et plus vite, en « récupérant » nos erreurs, la plupart ne s'en rendront même pas compte…

D'aucuns diront que je force le trait, que je verse dans la caricature facile.

Certes, mais que dire alors de cette obsolescence (programmée ou non) toujours plus rapide ; de ces millions de smartphones passant de « must have » à « has been » en moins d'un an ; des cycles de vie de nos produits et services, toujours plus courts, toujours plus comprimés ; de cette incessante course à la nouveauté qui, en abolissant l'attente, réduit d'autant le désir et nous pousse à toujours surconsommer plus. L'argument philosophique ne trouverait sans doute pas sa place dans ces pages, s'il n'était pas étayé par des choix technologiques aux conséquences potentiellement dramatiques.

Ne pas perdre de vue les réalisations techniques et technologiques

Alors que notre techno-sphère s'engage à la fois vers le plus et le plus vite, elle tend à sous-estimer la complexité croissante à laquelle elle doit faire face :

- En nous adressant à un public d'utilisateurs toujours plus large – et non plus à des informaticiens – nous multiplions les problématiques liées à la compréhension, à l'interprétation ou à l'utilisabilité (en contexte ou non). Les conséquences directes en sont, paradoxalement, un accroissement de la fracture numérique ainsi que des problématiques, parfois dramatiques, liées aux différences de perception ou d'interprétation(4) des capacités réelles du produit.
- En offrant des produits et services de plus en plus intégrés, de plus en plus sophistiqués, nous induisons, consciemment ou non, une complexité d'autant plus difficile à gérer qu'elle est à la fois en partie exogène (intégration de plus en plus généralisée de composants tiers ou externalisés) et qu'elle tend à dépasser les limites de la perception humaine (à la manière de l'arbre qui cache la forêt)(5). L'effet est bien évidemment multiplié si les délais de production se trouvent dramatiquement raccourcis au nom de la sacro-sainte performance opérationnelle.
- En décorrélant la réalité informatique de la réalité physique, nous finissons par perdre nos repères ; au risque de ne plus pouvoir mesurer les conséquences de nos actes. Une même touche de clavier, un même bouton poussoir peut, en fonction du contexte, allumer une simple ampoule ou fermer les vannes alimentant un circuit de refroidissement d'une centrale nucléaire… avec des conséquences bien différentes toutefois.

En matière de programmation informatique, le passage progressif de la ligne de commande aux solutions de métalangages de plus en plus modulaires, visuels ou orientés-utilisateur, permet à la fois une vulgarisation de l'usage mais aussi un risque accru de biais divers et variés, dont la déprofessionnalisation des fonctions dans les TIC.

- En substituant un développement itératif en forme d'essais-erreurs à une vision à plus long terme, certains en arrivent à générer – bien plus vite – beaucoup plus de coûts et de déchets que les approches dites « traditionnelles » dont elles veulent se démarquer.
  Je parle ici de ces approches chaotiques qui se réclament de l'agilité sans en adopter les codes, qui adoptent dogmatiquement les rituels comme on entre en religion, qui se concentrent sur la méthode plutôt que sur la valeur et qui délèguent à d'autres les choix qui leur appartiennent.
  Peut être celles-là se verront-elles mises au ban d'un futur mouvement techno-écologique émergent ; un peu comme les tendances « slow food », vegan et autres labels bio, en vogue aujourd'hui, grignotent peu à peu des parts de marché aux géants du secteur agro-alimentaire(6).
- En appliquant de manière systématique et récursive les méthodes et composants « qui marchent », au prétexte qu'ils sont moins chers et plus performants, nous induisons un risque démultiplié de catastrophe généralisée (sorte d'industrialisation du « single point of failure »)(7). Sans nier leur apport évident à la société actuelle, les oracles « mainstream » n'ont pas non plus réponse à tout. Google, par exemple, ignore tout du « Dark Web » et Siri ou Facebook me brossent un peu trop dans le sens du poil (et pas à gratter en l'occurrence) pour rester crédibles à mes yeux(8).
- En favorisant des approches reposant uniquement sur la livraison progressive de solutions de plus en plus abouties, nous nous reposons uniquement sur la confiance des utilisateurs. Soit parce qu'ils nous signent d'avance un chèque en blanc et financent à long terme un produit qu'ils espèrent un jour pleinement répondre à leurs attentes (en se contentant d'approximations plus ou moins fonctionnelles entretemps) ; soit parce qu'ils passent continuellement la main au portefeuille pour acquérir toujours plus de fonctionnalités ou de services…au risque de se lasser ou de se laisser tenter par les sirènes de la concurrence.
  Si un tel mode de fonctionnement convient sans doute aux « start-ups », il me semble beaucoup plus risqué pour des produits et services déjà en place. Il ne suffit pas de changer le mode de gestion des projets et d'emballer des « scrum masters » sous des avalanches de « post-its » pour devenir (ou re-devenir ?) une « start-up ». Ce qui n'est pas un but en soi d'ailleurs !
- En reportant les problématiques de qualité et en les sacrifiant à la montre, nous confondons tests et Assurance Qualité. Ironiquement, comme certaines approches qui galvaudent l'agilité le préconisent sans pour autant lui définir des règles ou un contexte, le test devient l'affaire de tous – y compris et surtout de ceux qui y sont le moins préparés. En conséquence, le test devient donc l'affaire de personne (car tester serait subitement devenu « has been », chronophage et sans intérêt). Il en va de même de l'analyse-métier, sacrifiée sur l'autel d'un improbable Product Owner qui n'avait pourtant rien demandé, ou d'un management intermédiaire qui ne sait plus trop quel indicateur suivre. Toute ressemblance…

Loin d'être un adversaire engoncé dans quelque « c'était bien mieux avant » aux relents réactionnaires(9), je suis convaincu que les approches agiles, intégrées ou itératives sont des vecteurs positifs du progrès pour autant qu'elles s'accompagnent d'une nécessaire vision (au-delà d'une simple itération, d'une approche réductrice ou ego centrée), d'une assurance qualité intrinsèque à la méthode et surtout qu'elles se donnent le temps nécessaire à la réflexion au lieu de foncer tête baissée sur le premier demi-concept venu.

La qualité est bien évidemment l'affaire de tous, mais elle s'apprend, elle se cultive et se prépare. Elle se nourrit de réflexions, elle s'interroge et se projette. Elle se mesure à l'aune du progrès et des valeurs sociétales au lieu de se limiter aux simples tests. Elle couvre toute la chaîne - le cycle de vie – et se conjugue comme un art(10) ou un métier à part entière.

« *Le temps c'est de l'argent* », disait Benjamin Franklin, certes, mais si vous n'avez pas le temps de bien faire, où trouverez-vous le temps de le refaire ?

Et puis, comme disait le regretté Raymond Devos, « A force de tuer le temps, on ne sait plus quoi faire de ses temps morts ». •

---

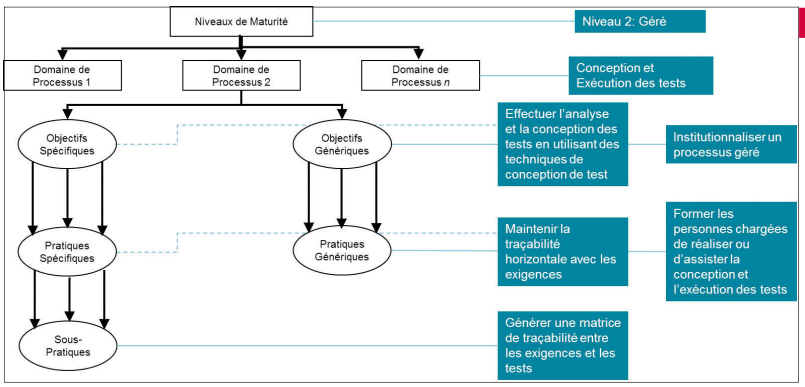
(1) Pour ne pas paraître imbécile peut être, car comme le dit le dicton…
(2) Le temps gagne toujours : hora fugit…disaient les romains
(3) Toutes disciplines confondues
(4) J'en veux pour exemple les récents accidents liés à l'utilisation de voitures autonomes aux États Unis. Dans les deux cas, la compréhension ou l'usage de la notion d'autonomie avaient été mal comprise par les conducteurs qui pensaient que leur véhicule était pleinement autonome, ce qui a été apprécié différemment par la justice et les compagnies d'assurances impliquées en aval. Sans parler de cet autre « conducteur » au taux d'alcoolémie élevé qui comptait sur son véhicule autonome pour le ramener à bon port.
(5) Ainsi les logiciels embarqués dans nos véhicules haut de gamme comportent plusieurs millions de ligne de code, pour ne citer qu'un exemple parmi tant d'autres.
(6) Même les chaînes de hamburger offrent des salades et des burgers « veggie » au menu, c'est dire !
(7) J'en veux pour preuve, parmi tant d'autres, les récents déboires liés à la sécurité du protocole Bluetooth et autres puces Intel (Meltdown, Spectre)
(8) Tout flatteur…, disait La Fontaine
(9) Pierre Dac disait à ce sujet que parfois les réactionnaires peuvent être plus rapides, les avions à réaction allant plus vite que les autres…
(10) En référence à l'ouvrage de Gleford Myers « The Art of Software Testing » 1979

Eric RIOU du COSQUER,
Certifié ISTQB niveau avancé complet
Lead Assessor TMMi et responsable du Comité TMMi pour la France

« TMMi : un dispositif pour évaluer l'efficacité d'une équipe de test, la faire progresser et la certifier ! »

Une certification pour un Centre de Test

A la différence du schéma de certification ISTQB qui concerne les compétences des personnes, le modèle TMMi (Test Maturity Model integration) s'applique à une équipe ou un centre de test, c'est-à-dire à un groupe de personnes qui effectue un ensemble d'activités de test pour différents projets informatiques de la même société ou d'une société cliente. TMMi est avant tout un recueil de bonnes pratiques spécifiques aux activités de test logiciel. Ces bonnes pratiques sont réparties dans différents processus, par exemple la Gestion des Environnements de Test ou encore la Maîtrise de la Stratégie de Test, et assorties d'exemples concrets pour leur mise en œuvre. Les processus sont répartis sur 5 niveaux et il est possible d'évaluer, pour une équipe ou un centre de test, si les pratiques recommandées pour un certain niveau sont en place. 1



1 *Structure du modèle en niveaux, objectifs et pratiques*

Un dispositif rapide et simple à utiliser

Contrairement au CMMI (Capability Maturity Model Integration) qui traite un nombre important de processus, par exemple sur le développement avec une couverture de la Gestion de Projet, de la Gestion de Configuration, du Développement des Exigences, ou encore de l'Intégration de Produits, TMMi est concentré sur les activités de test, ce qui permet d'avancer rapidement, en quelques mois sur ce sujet. Le modèle TMMi identifie précisément les profils à rencontrer, le type d'information à rechercher et permet de bâtir un plan d'amélioration concret avec des objectifs mesurables grâce à un système d'évaluation précis. La Certification TMMi pourra être attribuée pour les niveaux 2, 3, 4 ou 5, à une organisation qui aura été évaluée avec succès par un organisme habilité ayant fait intervenir un Lead Assessor TMMi. La Certification est un excellent moyen de valoriser les compétences d'une équipe de test et aussi de mettre en évidence son efficacité afin d'inciter d'autres entités ou clients à la solliciter. 2

Une dimension internationale

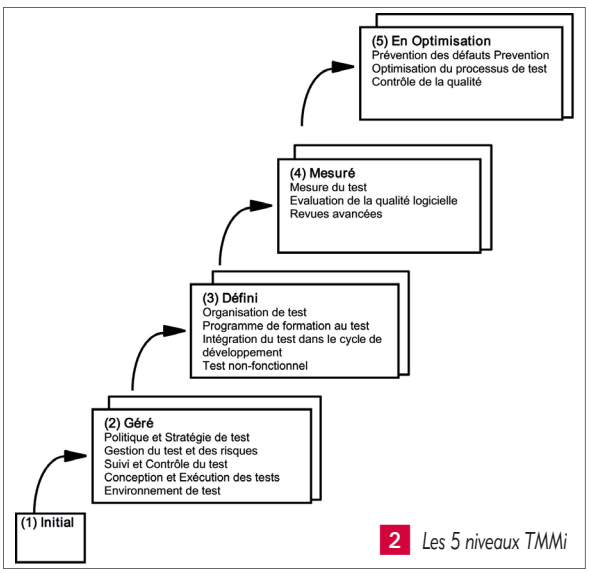
TMMi connaît un réel engouement dans le monde entier y compris en France depuis quelques années avec par exemple la certification au niveau 3 du Centre de Test de BNP Paribas Cardif ou encore du Centre de Qualification Fonctionnelle de Crédit Agricole Payment Services qui font désormais partie de la liste des organisations certifiées au niveau 3.

Une démarche en trois phases

Après un travail interne au centre de test, l'approche se déroule généralement en 3 phases. Le responsable du Centre de Test sollicite un organisme certifcateur accrédité par la fondation TMMi pour une évaluation jusqu'au niveau souhaité. Cette première phase donne lieu à la production d'une notation par rapport au niveau visé et d'un plan d'amélioration pour atteindre le niveau visé. Ensuite, les membres du Centre de Test mettent en œuvre le plan d'actions, avec le soutien ponctuel de l'organisme certificateur. Enfin, lorsque le plan d'amélioration a été mis en œuvre, l'organisme certificateur intervient à nouveau pour réévaluer le centre de test et, si la mise en œuvre des pratiques attendues est confirmée, délivrer le certificat de certification au Centre de Test.

Une formation pour apprendre à utiliser l'outil TMMi

Il existe aussi depuis peu, pour les personnes, une certification TMMi dont l'objectif est de valider la capacité d'une personne à comprendre le modèle et à l'utiliser de façon autonome dans le cadre de la mise en place ou de l'accompagnement d'une équipe de test. Cette formation certifiante garantie la capacité d'une personne à bien comprendre à utiliser le modèle TMMi. A ce jour, plus de 50 personnes en France sont des « Professionnels Certifiés TMMi » Vous trouverez sur www.tmmi.org toutes les informations utiles, comme le modèle TMMi, la liste des sociétés certifiées, la liste des Lead Assessor et la liste des sociétés habilitées à réaliser des audits dans les différents pays du monde. Par ailleurs, le Comité Français des Tests Logiciels, qui a pris la responsabilité de « Comité TMMi pour la France » se fera un plaisir de répondre à toutes vos questions sur le sujet TMMi, à info@cftl.fr •



2 *Les 5 niveaux TMMi*

Bruno Legeard
Expert Testing – CFTL

ATDD / BDD : le test devient une documentation vivante des projets agiles

L'expression du besoin en agile, centrée sur les User Stories et leurs critères d'acceptation, permet de définir ce qui doit être implémenté et testé dans l'itération (ou la suivante). Mais les user stories ne permettent pas de connaître le comportement du produit, car, une fois implémentées, elles sont oubliées et peuvent être invalidées par de nouvelles user stories.

La mise en pratique croissante de l'ATDD – Acceptance Test Driven Development – et du BDD – Behavior Driven Development, de mieux en mieux supportée par les outils de test, offrent une double réponse :

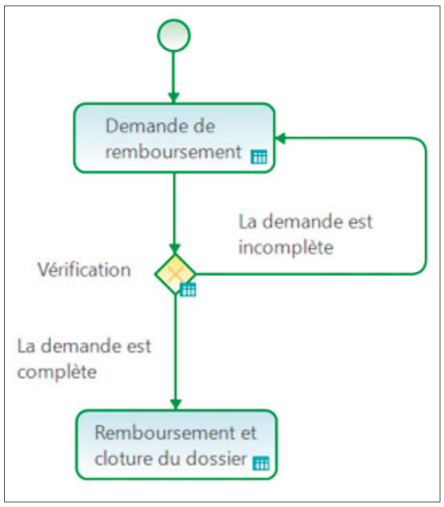
- Les scénarios de tests d'acceptation sont exprimés de façon intelligible, tant pour l'équipe agile que pour les experts Métier, et ils participent de la clarification du besoin ;
- Ces scénarios ATDD et BDD sont automatisés et maintenus comme tests de non-régression, constituant un matériel à jour sur ce que fait le produit.

L'idée est bien de faire d'une « *pierre deux coups* », en réalisant deux choses à la fois avec une seule action : écrire les tests d'acceptation le plus tôt possible et constituer une documentation vivante du produit.

Ces pratiques sont de mieux en mieux prises en charge par les outils de tests, tant au sein des plateformes de test en Agile, qu'au niveau des outils de conception et d'automatisation des tests. Deux approches principales sont proposées : utiliser une description textuelle ou une description visuelle des scénarios. Prenons l'exemple d'un système de gestion du remboursement des frais de déplacement et la User Story suivante : « *En tant que membre du département RH, je veux pouvoir vérifier la note de frais d'un employé et lui indiquer si besoin les éléments manquants ou erronés, afin de m'assurer que la demande est valide et qu'elle pourra être remboursée par le comptable* ». Voyons comment les scénarios de tests d'acceptation ATDD /BDD peuvent être définis de façon textuelle, ou de façon visuelle pour cette User Story.

Scénarios sous une forme textuelle – Le langage « Étant donné – Quand – Alors »

Cette représentation des scénarios, typique du BDD, est aussi connue sous le nom de langage de Gherkin. Typiquement, pour notre User Story, nous pourrions avoir un scénario dans ce langage : Étant donnée une demande de remboursement réalisée par un employé. Quand il manque un reçu de paiement pour une dépense. Alors la demande n'est pas validée par la RH et le message indiquant de mettre le dossier à jour est envoyé à l'employé.



Ce scénario est exécutable : l'implémentation du texte en mots d'actions d'automatisation (en keywords) permet d'obtenir un code de test exécutable pour l'intégration continue. Cette approche est outillée par de nombreux frameworks d'automatisation BDD tels que Cucumber ou SpecFlow, ou dans les plateformes de test en agile telles que Hiptest ou QASymphony, pour n'en citer que quelques-uns.

Scénarios sous une forme visuelle – Le pouvoir de l'image

La description visuelle du parcours applicatif au sein duquel la User Story est implémentée permet de produire les deux scénarios d'acceptation souhaités comme montré ci-dessous : en 1re colonne le parcours applicatif, et ensuite les deux scénarios produits comme tests d'acceptation. Voir l'image ci-dessus.

De façon similaire au textuel, les scénarios issus du visuel sont automatisés par une approche par mots d'action (Keywords) : « demande de remboursement », « vérification » et « remboursement et clôture du dossier » sont des keywords implémentés pour produire un code de tests automatisés. Le visuel permet aussi de produire la documentation de chaque test, par exemple pour alimenter le gestionnaire de tests pour une exécution manuelle, et pour relier ce test à la User Story en JIRA (ou dans tout autre gestionnaire de Backlog). Cette approche visuelle est prise en charge par les outils de conception de tests en agile tels que ARD de CA et Yest de Smartesting (le visuel précédent est réalisé avec Yest).

Quels avantages réciproques du visuel et du textuel ? Sur cette question, il y a d'abord une question de goût : préférez-vous un graphique ou du texte pour communiquer avec les utilisateurs Métier et maintenir une documentation vivante du produit ?

Une autre différence est dans le périmètre adressé : le parcours applicatif visuel décrit un workflow Métier lisible et donne une perspective couvrant généralement plusieurs User Story. C'est un atout important pour la documentation vivante, car cela permet une visualisation globale et synthétique des comportements applicatifs. Un scénario textuel « Etant donné – Quand – Alors » vise à formaliser un scénario d'acceptation d'une User Story unique : la lecture globale est plus difficile, car il faut passer en revue tous les scénarios concernés.

Pour finir, il reste la question : quelle différence entre ATDD et BDD ?

La compréhension la plus commune du BDD aujourd'hui est l'écriture des scénarios de test automatisés en Gherkin.

ATDD est une expression plus générale, recouvrant l'écriture des scénarios de test au moment de la définition des User Story et des critères d'acceptation associés, et l'utilisation de ces scénarios comme documentation vivante du produit. La scénarisation visuelle réalisée par le testeur agile au moment du raffinement du besoin est une approche typiquement ATDD permettant l'expression de workflows Métier (documentation vivante du produit) exécutables sous la forme de tests d'acceptation et de non-régression.

Ces approches sont au cœur de la transformation actuelle des pratiques du test, comme le montre la Journée Française des Tests Logiciels — JFTL 2018, avec plusieurs présentations et retours d'expérience sur ces techniques. •

Glenn Everitt
Technical Product Manager
chez **Compuware**

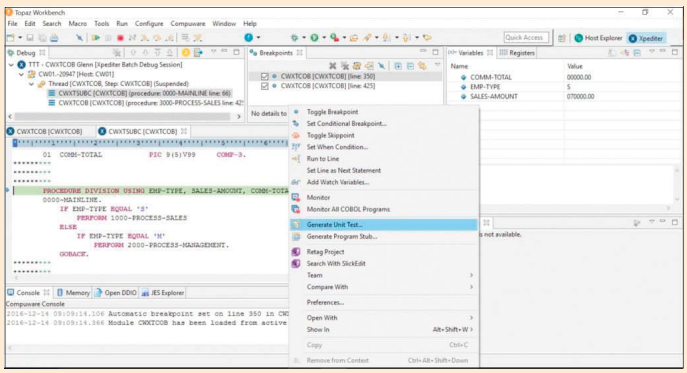
Tests unitaires automatisés : une nécessité pour les applications mainframe ?

Lorsqu'on pense aux systèmes de codage Mainframe, on les assimile souvent à de vieux codes obsolètes auxquels personne ne comprend rien à part le collègue sénior qui va bientôt partir à la retraite. Aujourd'hui, nombreux sont ceux qui qualifient le code COBOL d'obsolète. Pourtant, les faits sont là : que l'on parle de Java, de Ruby ou de n'importe quel autre code, tous sont obsolètes à partir du moment où aucun test n'est prévu pour vérifier qu'ils fonctionnent correctement.

Il est vrai toutefois que les démarches de test ne sont pas toujours faciles à mener sur les logiciels Mainframe. En effet, ces derniers proposent uniquement des « *feuilles de route* » pour tester manuellement les programmes de manière unitaire. Or ce process est particulièrement compliqué et chronophage en raison du nombre de programmes mainframe existants et la fréquence à laquelle ils doivent évoluer pour des raisons économiques ou de conformité. C'est pourquoi de nombreux codes sur mainframe ne sont pas testés aussi bien et aussi souvent qu'ils le devraient. Or des erreurs de programmation conséquentes augmentent les risques internes et externes d'une entreprise. Un bug dans un programme mainframe peut entraîner de gigantesques erreurs de calcul. Une mauvaise évaluation des risques peut pousser à prendre de mauvaises décisions et vous amener jusque devant les Commissaires aux Comptes.

Tests unitaires automatisés : une chance pour les programmes mainframe

La base de tout process de vérification est le test unitaire qui permet aux développeurs de détecter et réparer les bugs mineurs d'une application avant de passer aux tests d'intégration à plus grande échelle. Toutefois, nous avons souligné combien mettre en place des tests unitaires à la main était chronophage et fastidieux. C'est pourquoi les tests unitaires automatisés constituent une solution optimale car ils permettent d'exercer une validation fréquente et régulière, tout en faisant gagner un temps précieux aux développeurs pour un travail tout aussi efficace.



Aujourd'hui quand la direction générale demande au service IT de procéder à des changements au sein d'un programme, les développeurs se font des cheveux blancs à l'idée que les changements qu'ils devront opérer au sein du programme mainframe n'expose ce dernier à toute une série de bugs qui les forcera à rester au bureau tout le week-end. Les tests unitaires automatisés permettront donc aux développeurs de gagner un temps considérable tout en étant sûr que le code continuera à fonctionner correctement malgré les mises à jour. Enfin, la mise en place des tests unitaires automatisés permet d'identifier les bugs plus rapidement et plus efficacement. Avec des tests manuels, le bug peut n'être identifié que plusieurs mois plus tard alors que le développeur peut ne pas se rappeler du code écrit ou peut avoir quitté la société. Les tests unitaires automatisés permettent de résoudre rapidement ces problèmes grâce à un process de construction continue. En effet, les développeurs peuvent rapidement chercher la cause des bugs et rapidement en déterminer l'origine si le test unitaire automatisé échoue presque instantanément suite à l'introduction du bug.

Si les dysfonctionnements sont rapidement identifiés avant que les développeurs n'oublient les modifications effectuées dans le code, il sera plus facile pour ces derniers de corriger le code mis à jour.

Optimisation des tests unitaires automatisés

Utiliser le moins de données possible

En premier lieu, il est important que le test ne soit mené qu'avec le strict minimum de données pour les raisons suivantes :

- **Gagner du temps.** Quand un test unitaire utilise plus de données que nécessaire et qu'il échoue, le développeur devra, pour identifier et résoudre le problème, passer au crible plus de code et plus de données, ce qui nécessitera plus de temps et plus d'énergie. D'autre part, l'usage massif de données risque d'amener les développeurs à utiliser les mêmes lors de tests différents. Or, utiliser les mêmes données ne permettra pas d'identifier de nouveaux problèmes. De plus, en cas de dysfonctionnement d'un programme, plus on aura utilisé de données pour le faire, plus le diagnostic sera difficile à faire et plus l'exécution sera longue. Et le tout sans aucune valeur ajoutée.
- **Limiter les tests du code par le contrôle unitaire** : un test unitaire qui consomme trop de données exécutera plus de code de programmation pour traiter ces dernières. Par conséquent limiter la donnée utilisée dans un seul test peut également limiter le code exercé.

Créer des ensembles de données séparés

Pour mener différents contrôles unitaires, il est plus efficace de séparer la donnée en plusieurs ensembles. Cela permet en effet aux développeurs de se concentrer sur une unité de code à la fois. Un excès de donnée ne fera qu'accroître le temps nécessaire au diagnostic d'un éventuel échec d'un test unitaire. Cette pratique est d'autant plus nécessaire lorsqu'il faut traiter des données ayant subi une modification. Ainsi, même si un programme fonctionnait correctement lorsqu'il a été déployé, les données de saisies initialement utilisées pour le créer peuvent avoir évolué sans que le développeur n'ait pu l'anticiper, ce qui peut engendrer des bugs et des dysfonctionnements. C'est pourquoi les process de validation doivent évoluer conjointement avec les données. Par conséquent, les nouveaux codes doivent également être conçus pour anticiper les changements possibles. Cela signifie que des tests supplémentaires devront être menés pour gérer ces nouvelles données saisies. Cela sera plus facile à faire si les données ont été séparées lors des tests unitaires.

Cela devrait devenir une pratique incontournable, de comprendre les règles opérationnelles conjointement à la donnée, et de s'assurer que les tests unitaires valident bien le code qui est à la base de chaque règle opérationnelle. Bien souvent, les règles opérationnelles ont été elles-mêmes mal conçues et ne peuvent pas prendre en compte d'éventuels changements au sein de la donnée.

Conclusion

Il est temps pour les équipes mainframe d'être plus souples et de produire des rendus de meilleure qualité pour faire face aux nouvelles demandes digitales. Les tests automatisés sont la pierre angulaire essentielle pour accélérer le développement des applications et conserver la qualité du code. En un mot, pour être plus agiles, les développeurs doivent automatiser la création des tests unitaires et leur exécution, tout en concentrant leurs tests sur des sections spécifiques du code et en utilisant le moins de données possible. •

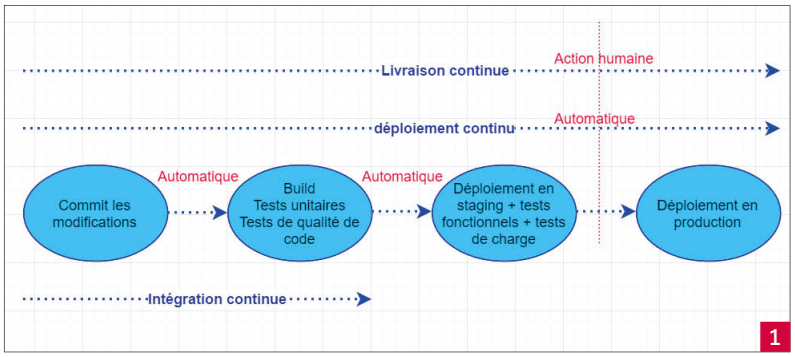
Paquet Judicaël
Coach Agile & Devops, Judicaël accompagne les entreprises en France et en Suisse comme Renault Digital ou la RATP dans leur transformation avec Myagile Partner qu'il a créé en janvier 2017. http://blog.myagilepartner.fr

MYAGILE PARTNER

Agilité & DevOps : les tests, de l'utile au risque

Alors que l'excellence technique avait été oubliée durant une grosse dizaine d'année, le coût de cette errance s'est vite fait ressentir ; nos structures prennent conscience de l'importance d'instaurer à nouveau différents types de tests. Il est nécessaire d'assurer une qualité minimale du code, de limiter au maximum les régressions et de limiter profondément les futures maintenances de Legacy.

Dans l'univers des méthodes agiles, il est fortement conseillé de mettre des tests en place afin de garantir une excellence fonctionnelle ainsi qu'une excellence technique. Si un produit doit être le reflet du besoin des utilisateurs clés, leur garantir de la non régression et de bonnes performances contribue fortement à leur satisfaction.



Automatisation des mises en production dans le monde Devops

Qu'est-ce qu'une méthode agile ?

Les méthodes agiles sont des méthodes de gestion de projet (on dit même « gestion de produit » pour accentuer l'aspect user-centric) doivent respecter 4 valeurs fondamentales ainsi que 12 principes sous-jacents à ces valeurs. Ces principes sont décrits au sein du manifeste agile qui a été établi par 17 spécialistes de la gestion de projet en 2001.

Voici les 4 valeurs fondamentales :

- Les individus et leurs interactions plus que les processus et les outils ;
- Des logiciels opérationnels plus qu'une documentation exhaustive ;
- La collaboration avec les clients plus que la négociation contractuelle ;
- L'adaptation au changement plus que le suivi d'un plan.

Rappel du manifeste Agile : nous reconnaissons la valeur des seconds éléments, mais privilégions les premiers.

Si ces 4 valeurs restent superficielles sur l'aspect « excellence technique », on retrouve l'un des 12 principes présentés par ce manifeste Agile qui est « *une attention continue à l'excellence technique et à une bonne conception renforce l'Agilité* » sous-jacent des 4 valeurs citées précédemment.

Il est intéressant de voir ces valeurs et l'un de ces principes car d'autres mouvements sont nés à la suite dans le monde du développement qui vont considérablement renforcer l'aspect « excellence technique » des produits.

Que dit le Scrum méthode agile ultra-populaire ?

Le Scrum est la méthode agile la plus populaire au sein des DSI aujourd'hui. Elle offre un cadre léger très apprécié par ceux qui veulent se lancer dans les méthodes agiles.

Cependant le Scrum est une méthode agile qui ne précise rien sur le « comment » atteindre l'excellence technique ; ce sont des pratiques venant de l'Extreme Programming qui reviennent en force depuis quelques années grâce à l'émergence de deux mouvements agiles récents : le devops et le software craftsmanship.

Qu'est-ce que le devops ?

Comme on l'avait vu dans mon précédent article au sein de Programmez, le Devops est un mouvement né en 2009 qui privilégie la mise en place d'un alignement de l'ensemble de la DSI autour d'objectifs communs ; le terme devops est la concaténation de dev pour développeur et ops pour opérationnels. Il s'agit donc des ingénieurs responsables des infrastructures.

Si le mouvement devops est un mélange de 70% de culture et 30% de technique, on retrouvera dans cette partie technique une couche d'intégration continue et une couche de livraison continue (voire de déploiement continu). La couche d'intégration continue proposera l'automatisation d'un grand nombre de types de tests différents. 1

Avoir une équipe enfermée dans une pièce totalement isolée des équipes de développeurs pour mettre en place des solutions d'intégration continue ou de livraison continue ne correspond pas à ce concept Devops. C'est pourtant cette façon de faire que nous voyons de plus en plus aujourd'hui au sein des entreprises. L'ingénieur Devops, quand il y a besoin d'en mettre, devra être régulièrement au sein des équipes pour qu'elles apprennent à évoluer au coeur de ces nouvelles technologies pas toujours simples à maîtriser.

Afin de garantir la qualité technique des produits, le Devops est souvent associé aux outils tels que Docker pour la containérisation des plateformes, kubernetes ou autres orchestrateurs de conteneurs ou des plate-

formes micro-services pour séparer totalement les différents services le tout hébergé en cloud.

Qu'est-ce que le software crafsmanship ?

Le craftsmanship est une approche agile très axée sur la qualité technique des réalisations. La démarche fait suite à une longue période où la baisse des coûts et l'accélération des sorties (Time-To-Market) se faisaient au détriment de la qualité.

Ce mouvement est donc né pour remettre la qualité technique au centre des préoccupations des développeurs, et même au delà d'ailleurs, en faisant référence à la qualité des produits artisanaux.

Si un livre "software craftsmanship" est publié en 2001 pour différencier un développeur enfermé dans un monde industriel et un développeur agile, c'est en 2008 qu'on considère que le mouvement se lance vraiment avec la proposition d'une cinquième valeur au manifeste agile par Uncle Bob : "l'artisanat plus que l'exécution".

Maintenant ce mouvement se fait de plus en plus connaître et des coachs craftsman apparaissent dans les grands groupes pour accompagner les équipes techniques à revenir vers des produits bien conçus.

Le manifeste du software craftsmanship

Voici les 4 valeurs qui ont été écrites par les craftsmen se voulant de ce mouvement agile :

- Pas seulement des logiciels **opérationnels**, mais aussi des logiciels bien **conçus**
- Pas seulement l'adaptation aux **changements**, mais aussi l'ajout constant de **valeur**
- Pas seulement les individus et leurs **interactions**, mais aussi une **communauté** professionnelle
- Pas seulement la **collaboration** avec les clients, mais aussi des **partenariats** productifs

« *C'est-à-dire qu'en recherchant les éléments de gauche, nous avons trouvé que les éléments de droite sont indispensables*. »

On peut indirectement revoir dans ce mouvement, un retour de l'Extreme Programming qui se voulait mettre en avant l'excellence technique.

Types de tests différents

Il existe différents types de tests dans le monde du développement qui auront pour but de couvrir ensemble un maximum de choses :

- tests unitaires (voire TDD)
- tests fonctionnels (BDD)
- tests end-to-end
- tests de performance
- tests de charges
- tests de qualimétrie.

Tous ces tests vont permettre de couvrir différents périmètres et tous les mettre à un coût initial non négligeable. Nous allons d'ailleurs profiter de cet article pour voir les coûts, voire les risques, de mettre chacun de ces tests.

Il est vrai que cela fait bien de dire que l'équipe a mis en place l'ensemble de ces tests. Cependant, à force d'accompagner des équipes, je vois bien que cela est rarement le cas - voire impossible - pour certaines équipes où la maturité de l'excellence technique (voire l'envie) empêche totalement ces équipes de couvrir l'ensemble des périmètres.

La TDD (tests unitaires)

Pour faire simple pour les personnes n'étant pas familiarisées avec ces termes, un test unitaire est un test qui couvre une seule fonction technique afin de s'assurer que les résultats retournés sont cohérents.

La TDD (Test Driver Development) est une technique de développement logiciel qui consiste à écrire les tests unitaires d'une fonction avant d'écrire le contenu de cette fonction. C'est une pratique très connue dans le domaine de l'intégration continue.

Cycle de la TDD

La TDD propose un cycle de travail aux développeurs pour obtenir une qualité optimale de la mise en place des tests unitaires :

- Ecrire le test unitaire
- Lancer celui-ci et vérifier qu'il échoue (classe pas encore codée)
- Ecrire la classe à tester avec le minimum pour faire marcher le test
- Lancer le test et vérifier qu'il fonctionne
- Finir le code complet de la classe
- Vérifier que le test fonctionne toujours (non-régression).

Outils de tests unitaires

Chaque langage possède son outil de tests unitaires facilement exploitable et tous basés sur les mêmes principes : Php Unit (PHP), JUNit (Java), NUnit (.Net), Unit.js (JS), CppUnit (C++).

Ces outils vous permettront de réaliser vos tests unitaires et de les lancer manuellement. Cependant il existe de nombreux outils qui permettent d'automatiser le lancement des tests unitaires comme Team City, Jenkins ou Travis (outil SaaS directement connecté avec Git Hub).

La difficulté de mettre ce type de tests

Si mettre en place ce concept de TDD est vraiment intéressante, il ne faudra pas l'imposer aux développeurs comme le font certains grands groupes. Avec l'expérience, on s'aperçoit que ce changement de comportement dans la façon de coder n'est pas ouvert à toutes les équipes de développement.

Avant même de mettre de la TDD, il est essentiel que les équipes de développement comprennent concrètement ce qu'apporte la TDD avant de les forcer à en faire. Rappelez-leur que la TDD c'est :

- Un code de meilleure qualité
- Un outil pour diminuer considérablement les bugs futurs
- Un gain de temps dans un avenir proche pour faire du projet et non plus du débogage
- Une méthode de développement très aimée sur un CV

La TDD est une pratique qui peut énormément apporter à la qualité, mais l'amener sans accompagnement peut avoir des conséquences sur la productivité des équipes. En effet, quand l'ensemble de l'équipe ne choisit pas de faire de la TDD, on voit rapidement des comportements contre-productifs se mettre en place.

L'agilité rappelle que l'équipe de développement est responsable technique du produit ; elle doit donc décider ensemble de la mise en place ou non de concepts tels que la TDD. Cependant le Scrum Master devra vérifier que cela ne sera pas un frein pour certains membres de l'équipe et si c'est le cas de voir comment faire accompagner ces développeurs afin qu'ils ne perdent pas en productivité.

S'il y a de plus en plus de développeurs habitués à ce type de techniques d'ingénierie logicielle, il reste un grand nombre de développeurs qui n'aiment pas (ou n'arrivent pas) à se plier à ce type de tests. Il faut bien prendre conscience, que cette façon de faire impacte le comportement de développement.

BDD (Behavior Driven Development)

Le BDD (Behavior Driven Development) est une pratique Agile créée par Dan North en 2003 qui a pour but de créer des tests fonctionnels avec un langage naturel compris de tous.

Cette pratique encourage le rapprochement des équipes techniques et des équipes fonctionnelles comme le font les méthodes agiles dans leur ensemble. Regardez l'article BDD/ATTD pour en savoir plus.

Des Product Owners pas toujours préparés

Souvent en Agile, nous demandons aux Product Owners d'écrire eux-mêmes les tests d'acceptances (pour faire du BDD) qui permettent souvent de renforcer la qualité fonctionnelle des livrables.

Bien que j'aie déjà vu que cela se faisait par les développeurs dans certaines entreprises, ça n'a pas beaucoup de sens d'imposer ce travail à ceux-ci. En effet, si cela va probablement améliorer un peu la qualité générale, je ne suis pas convaincu qu'ils écrivent tous ces tests avec beaucoup d'envie (et donc de rigueur).

En tant que coach, je vois de plus en plus de Définition of Ready inclure des tests d'acceptances au sein des user-stories (demandes fonctionnelles) ; les coachs agiles y sont sûrement pour quelque chose car nous essayons d'amener les équipes à chercher constamment à améliorer la qualité des produits livrés.

Cependant cela fonctionne pour des Product Owners (PO) qui ont de l'expérience dans ce poste mais la tâche peut s'avérer plus compliquée pour les Product Owners qui découvrent le métier. Il est indispensable que les PO inexpérimentés soient accompagnés individuellement par un coach si on ne veut pas que l'écriture de ces tests devienne un point de blocage.

D'ailleurs, il est également conseillé que les développeurs connaissent bien également ce type de test pour que les développements soient efficaces. Il est utopique de penser que l'équipe (développeurs et Product Owner) mettra ce type de test rapidement sans la moindre formation.

Je connais très peu d'équipes qui arrivent aujourd'hui à mettre du BDD en place : les équipes non perfectionnistes et curieuses n'auront que très peu de chance d'arriver à mettre cette pratique en place.

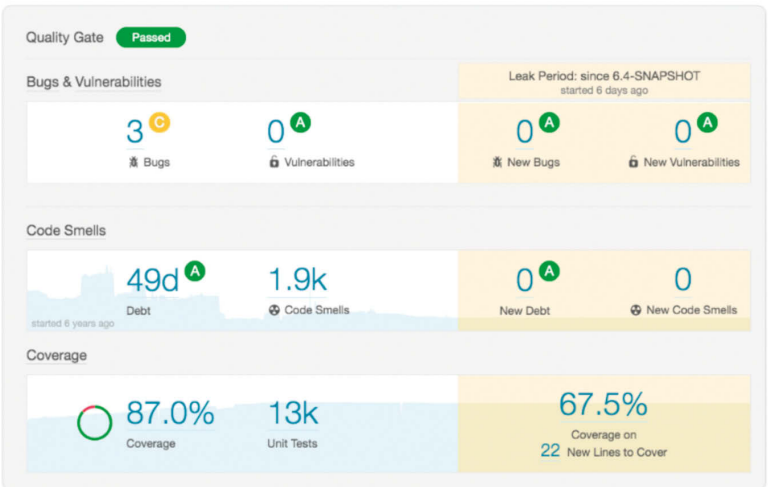
Des outils de qualimétrie de code

Afin de mettre en place une vérification automatique de la qualité du code, il existe aujourd'hui de nombreux outils de qualimétrie utilisés permettant de s'assurer que le code partagé - voire envoyé en production - est d'excellente qualité.

Qualimétrie avec SonarQube

SonarQube est devenu très populaire dans les groupes car il s'intègre dans la couche d'intégration continue pour mesurer régulièrement la qualité du code des équipes. Le but n'étant pas de « fliquer » les équipes mais surtout d'accompagner les équipes à faire du code de plus en plus propre. 2

L'outil parle lui-même « d'inspection continue » pour expliquer le service qu'il offre à ses utilisateurs. Il est d'ailleurs devenu très populaire. La raison en est qu' au-delà du fait de proposer des mesures de grande qualité, il sait aussi se connecter très facilement aux nombreux outils d'intégration et de tests du marché : Bamboo, Travis CI, Jenkins, Team City, Visual Studio, Maven…



SonarQube 2

Comment est reçu SonarQube par les développeurs ?

Les développeurs en général s'habituent rapidement à l'utilisation de ce type d'outil et j'ai rarement vu des équipes revenir en arrière sur cette pratique de mesure de qualité de code. L'installation est relativement simple et l'utilisation n'est pas contraignante.

En revanche, certains experts en développement considèrent que SonarQube n'est pas un outil suffisant pour dire que cela impose une véritable qualité technique au code.

Des tests end-to-end

Afin de vérifier les parties les plus critiques de vos applications, il est possible d'utiliser des tests end-to-end. On fait ces scénarios avec des outils tels que Selenium.

L'outil permet de simuler l'utilisation d'une application par un utilisateur clé en suivant un parcours type. Cela peut-être fort pratique pour vérifier que tout est ok sur l'application. Cependant en cas de changements majeurs sur la partie IHM, il est possible d'être contraint de réécrire l'ensemble des tests sous Selenium.

Je conseille fortement de limiter le nombre de tests end-to-end aux parties critiques de l'application et de ne pas multiplier à grand nombre ces types de tests. Cependant ceux qui en mettent un grand nombre sont plutôt satisfaits des résultats attendus car le moindre bug de comportement est détecté avec ce type d'outil.

Les développeurs ne sont pas impactés par ce type de test souvent écrit par le Product Owner ou un testeur.

Des tests de charges

Si ces tests sont plus souvent mis en place par les ops, ils permettent de tester la capacité des applications à tenir debout selon différentes situations.

Tests de charges classiques

Nous allons tester que l'application répond de correctement en augmentant considérablement un nombre d'utilisateurs simulés.

Tests de stress

Nous allons dans ce type de tests de charge, regarder le comportement de l'application lorsque celle-ci sera confrontée à des charges totalement extrêmes.

Tests de capacités

Nous allons tester spécifiquement chaque test de stress et mesurer le nombre maximum d'utilisateurs simulés simultanés avant de voir le système dérailler totalement.

Tests d'endurance

Nous allons tester le comportement de l'application lors d'une exécution d'un test de charge sur la durée.

S'agit-t-il de tests utiles et coûteux ?

La mise en place de ce type de tests est plutôt utile. Quand ils sont en place et automatisés dans des outils tels que Jenkins, il y a peu de chance d'avoir un besoin de revoir ce type de tests. C'est certes un investissement de base mais qui sera très vite rentabilisé avec l'avancement du projet.

Des tests de performances

Les tests de performances permettent de vérifier que le rendu est suffisant pour l'utilisateur de l'application ; on sait que l'ajout de 100ms sur le chargement des pages a un impact très fort sur le site d'Amazon, donc il est nécessaire de garder de bonnes performances sur les applications surtout pour celles qui reçoivent des utilisateurs finaux.

Rapport qualité/prix ?

La mise en place automatisée de ce type de tests est vraiment facile et ne demandera pas une charge insurmontable. Vous pouvez sans soucis investir sur ce type de tests surtout si vous avez des clients finaux sur vos applications.

Quels sont les risques ?

Si l'agilité insiste sur le fait qu'il faut avoir une « excellence technique » au sein des produits que les équipes réalisent, il est rarement possible de mettre tous les types de tests différents en place en même temps ; cela implique d'avoir des équipes techniques avec une excellente maturité sur le sujet.

L'entreprise ne doit pas imposer ces pratiques à l'ensemble de leurs équipes mais doit laisser chacune d'entre elles décider des pratiques qu'elle mettra en place pour assurer la qualité technique du produit. Si l'entreprise tente vraiment d'imposer une batterie de tests, les risques ne seront pas négligeables.

Faire des tests à tout prix ?

Si l'excellence technique est importante, cela ne doit pas empiéter sur la valeur délivrée pour les clients. Dans les entreprises agiles, le but est de livrer un maximum de valeur au client et les tests ne doivent pas en être un frein. Il faut faire attention au culte du test qui s'installe de plus en plus au sein des entreprises. Si mettre plein de tests peut être vu comme une bonne sécurité pour le produit (non régression, KPI laissant rêveur), les équipes peuvent voir leur productivité baisser.

Les méthodes agiles parlent de la nécessité d'avoir une véritable « excellence technique » sur l'un de ces 12 principes mais la première valeur agile « **Les individus et leurs interactions** plus que les processus et les outils » rappelle bien qu'il est préférable de laisser les équipes choisir les outils qu'elles veulent utiliser et d'éviter de leur imposer ceux-ci.

Agilité à l'échelle

Dans les grands groupes, c'est un sujet délicat car avec une agilité à l'échelle, on a tendance à vouloir imposer les mêmes règles pour l'ensemble des équipes : cela souvent pour avoir des KPI à suivre en pensant à le faire pour imposer une qualité irréprochable au produit.

Mais soyons clairs, des tests mal faits avec du code écrit selon les normes imposées par tel ou tel outil n'en feront pas un code de qualité ; si les développeurs n'ont pas la motivation et la maturité de faire des tests, les résultats seront contre-productifs. Nous aurons un investissement fort dans les tests pour des résultats quasiment nuls. Si nos outils deviennent de plus en plus performants, ils n'ont pas l'intelligence de dire : « ta fonction a un nom bizarre ». Cela serait peut-être vrai dans 20 ans, mais d'ici là, j'aurai écrit un nouvel article adapté sur le sujet.

Mais alors si on laissait tout simplement chaque équipe choisir les outils et les types de tests qu'elles veulent mettre en place ; restons agile, n'imposons pas les processus et les outils.

En imaginant que l'on ait une petite application interne et un site e-commerce ouvert au grand public ; est-ce vraiment utile d'avoir des tests de performances, voire l'ensemble des tests de charges existants, sur la petite application interne ? Notre exigence sur ces points ne serait-elle pas plutôt pour le site e-commerce ?

Si une équipe n'a pas du tout de maturité pour faire du TDD et du BDD, ne risque-t-on pas de frustrer l'équipe et de baisser considérablement sa productivité ? Je peux vous assurer que les résultats d'imposer des tests de type TDD ou BDD à des équipes qui ne veulent pas en faire n'amène pas à un produit de qualité… C'est même le contraire. En effet, décider de mettre tous les tests sur tous les projets n'a pas de sens. Chaque produit a-t-il son besoin d'exigence et chaque équipe sa capacité de faire des tests, ou non ?

Vouloir n'est pas pouvoir

Je rencontre également des équipes qui veulent vraiment faire des batteries de tests et principalement du TDD et BDD. Si les développeurs motivés se mettent facilement aux tests de type TDD, elles ont plus de difficulté avec le BDD car il impose un minimum de maîtrise.

Dans ces cas là, en tant que coach, j'amène le Product Owner à monter en compétence sur l'écriture des tests en Gherkin et l'équipe apprend peu à peu à les intégrer. Lors des premières séries de user-stories, le Product Owner ne saura pas forcément comment écrire son test d'acceptance, mais le coach l'aidera à aller peu à peu vers une maturité certaine.

Si l'équipe complète a vraiment envie de faire ce type de tests, cela viendra très vite ; il faudra juste que le Product Owner rencontre quelques subtilités afin d'apprendre à gérer. Une bonne écriture des user-stories lui permettra également d'avoir un meilleur cadre pour l'écriture de ses tests d'acceptance.

Conclusion

Mettre en place des tests est un excellent moyen d'amener le produit à une excellence technique et de diminuer les régressions. Cependant, n'essayez pas de l'imposer même dans les équipes agiles et laissez les équipes choisir leur stratégie pour aller vers l'excellence technique car, imposés, les tests pourraient devenir contre-productifs. Il faut que les développeurs gardent en tête que la priorité dans le monde de l'agilité est de servir un maximum de valeur au client, et qu'il faut savoir mettre le niveau suffisant d'excellence technique sans pour autant aller vers l'extrême. •

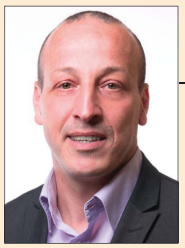
« L'avenir de l'entreprise est entre les mains des développeurs ! »

Lancée par Otto Berkes, Chief Technology Officer de CA Technologies, lors de la conférence CA World en novembre 2017 à Las Vegas, cette petite phrase en dit long sur les transformations constatées pour la fonction de développeur, dans tous les secteurs, au cours des dernières années.

Transformer l'entreprise en usine logicielle moderne

Pour rester compétitives et innover avec agilité, les entreprises sont tenues d'adopter le modèle de la Modern Software Factory (usine logicielle moderne) qui leur permet de passer rapidement du stade de l'idée à sa concrétisation en application. Et cette transformation repose sur les développeurs : alors qu'ils se contentaient d'écrire du code en Cobol au temps des mainframes, ils jouent aujourd'hui un rôle clé dans l'économie des applications, en adoptant, pour certains, la certification au framework SAFe.

« *Notre objectif est donc de donner plus de contrôle et de maîtrise à tous les développeurs, à chaque phase du développement,* précise Mos Amokhtari, Directeur Technique de CA Technologies France. *Et n'allez pas croire que le mainframe n'ait pas lui aussi évolué : la moyenne d'âge de nos développeurs mainframe est de 26 ans et ils travaillent en mode agile sur des technologies comme la blockchain !* »



« Shift left testing » : intégrer la phase test au plus tôt dans le cycle de développement

Une première étape dans ce processus de transformation fut l'avènement des architectures microservices et des containers. Fini les applications monolithiques constituées de plusieurs millions de lignes, place aux modules à taille humaine réutilisables que de petites équipes agiles peuvent faire évoluer de façon incrémentale. Ensuite, le *shift-left* : les développeurs doivent pouvoir tester les performances d'une appli et vérifier la qualité du code beaucoup plus tôt dans le cycle de production, afin de réduire à la fois le *time-to-market* et les coûts. Enfin, la sécurité : une étude récente de CA Veracode montre que 77 % des applications présentent au moins une faille dès la première analyse et 88 % des applis Java contiennent au moins un composant les exposant à des attaques. Il est donc urgent que les développeurs puissent se tourner vers des solutions DevSecOps de premier plan afin d'intégrer la sécurité applicative dès les phases initiales de développement.

« *Un autre point crucial pour les développeurs est de bien définir les exigences d'une appli de bout en bout, car près de 60 % des bugs ou des comportements déviants en production sont liés à une mauvaise compréhension initiale des besoins* », explique Mos Amokhtari. Un outil de collecte des besoins, d'automatisation et de conception des scénarios et plans de test tel que CA Agile Requirements Designer permet aux organisations de développer et de commercialiser plus rapidement des applications de qualité.

Tests, performances et gestion des API

Au niveau des performances, qui sont une exigence phare des développeurs, des solutions comme CA BlazeMeter leur donnent la possibilité de faire des tests de charge, simulant l'activité de milliers d'utilisateurs, très tôt dans le cycle pour vérifier que le code les supporte. Et pour éviter les failles de sécurité applicative, dommageables en termes de pertes de données et au niveau financier (RGPD oblige(1)), les développeurs peuvent s'appuyer sur le DevSecOps pour le scan de leur code applicatif afin de s'assurer que la sécurité est optimale, par exemple en utilisant CA Veracode Greenlight. Autre processus clé de la transformation numérique, la gestion des API. Il faut donner aux développeurs les moyens d'exposer leurs API de façon sécurisée, dans un environnement hautement évolutif qui permet par exemple de faire des retours arrière si nécessaire. L'API est-elle disponible ? Répond-elle dans les temps ? Son contenu est-il conforme à ce qui est exigé ? Les réponses sont fournies par des outils tels que CA APM ou CA Runscope en mode Saas. « *Un exemple typique est une application qui utilise la géolocalisation via Google Maps* », explique Bilel Sfaxi, Principal Pre-sales Consultant chez CA Technologies. « *Pour s'assurer que les API de Google Maps fonctionnent correctement, les développeurs peuvent réaliser très facilement des tests de disponibilité, de temps de réponse et de vérification de contenu.* »

Un écosystème de services et de support

Proposer les bons outils, c'est bien. Mais donner aux développeurs les moyens d'en exploiter pleinement le potentiel est encore mieux. Les entreprises doivent pouvoir se tourner vers des fournisseurs capables de délivrer un riche écosystème de support : consultants experts, e-learning, tutoriels en vidéo, formations… « *Nous organisons de nombreuses réunions avec les communautés de développeurs afin de voir comment nous pouvons faire évoluer nos solutions par rapport à leurs besoins* », ajoute Mos Amokhtari. Et demain ? Pour accompagner au mieux les développeurs, les éditeurs de logiciels vont devoir leur fournir plus de données à valeur ajoutée. « *Nous travaillons sur une plateforme SaaS qui exploite certains métriques collectés par nos outils afin d'optimiser par exemple l'analyse de risque ou la gestion de la capacité,* précise Bilel Sfaxi. *Nous proposons déjà aux développeurs des solutions, des services et du support, il est temps d'y ajouter de l'intelligence.* » •

(1) Le non-respect de la directive européenne RGPD exposera toute entreprise basée dans l'Union Européenne à des sanctions pouvant aller jusqu'à 4% du chiffre d'affaires mondial.

Bernard Homès
Président **TESSCO sas** - *Fondateur et ex-président du Comité Français des Tests Logiciels.* bhomes@tesscogroup.com
Avec plus de 35 ans d'expérience en Tests de logiciels dans des postes clefs à dimension internationale, Bernard exerce en qualité de Consultant Senior au sein d'entreprises renommées dans la banque, l'aéronautique, le spatial et les télécommunications. Il possède de nombreuses certifications (Scrum Master, CMAP, ISTQB Full Advanced, IREB, REQB) et se focalise sur le transfert de connaissances et s'implique au sein d'organisations professionnelles comme l'IEEE Standards Association.

Tests, livraisons continues et versions

Le concept de livraison continue est présent et continuera à se développer. Déclinée dans DEVOPS, la livraison continue répond aux changements de priorité chez les clients et utilisateurs. Il y a cependant des éléments importants à ne pas oublier : quel niveau de confiance peut-on avoir dans les composants déployés ? Le composant est-il bien celui qui a été développé et testé, n'a-t-il pas été modifié – intentionnellement ou non – avant d'être déployé ?

Le concept de blockchain est relativement récent et propose un mécanisme assurant un niveau de confiance sans nécessiter une autorité centrale. L'utilisation de composants logiciels d'origines diverses (p.ex. containers ou open source) implique de s'assurer du niveau de confiance de ceux-ci. Une solution type blockchain (publique ou privée) peut assurer ce niveau de confiance pour les composants constituant le logiciel. Faut-il une confiance du niveau assuré par les « blockchains » ou une simple gestion de configuration suffit-elle ?

Gestion de configuration ?

Nous connaissons l'importance de la gestion de versions (impactée par le niveau de granularité des composants). Toute modification du produit entraîne la création d'une branche à gérer, tester et suivre séparément. L'intégration de cette branche dans le produit, nécessitera de s'assurer l'absence d'effets de bord indésirables. Pour être complet, il faut associer tout ce qui est lié à cette modification. Par exemple, exigences et résultats d'exécution des tests, documentation, paramétrage, interfaces, version d'OS et des outils comme l'IDE et les automates des tests, version des matériels et composants logiciels utilisés, etc. Cette exhaustivité – exigée pour les logiciels à sécurité critique – est souvent ignorée pour des logiciels de gestion. Divers aspects sont à prendre en compte, par exemple :

- Gestion interne ou externe (p.ex. Open Source) des composants ?
- Quelle est la granularité des composants ?
- Criticité et/ou complexité des systèmes ?

Simple pour un logiciel monolithique et des livraisons limitées, la gestion de configuration se complexifie avec l'accroissement du nombre de composants, d'interfaces, d'acteurs et de clients. Des outils existent et mais leur implémentation laisse parfois à désirer.

Les mauvaises nouvelles

Malheureusement nous devons prendre en compte des réalités pas toujours agréables, comme :

1. L'explosion exponentielle du nombre de combinaisons et de versions de composants. Pour limiter cette explosion combinatoire, des solutions de virtualisation peuvent être utilisées. Ces machines virtuelles (VM) gèrent les interfaces avec l'infrastructure physique et encapsulent un OS (souvent une déclinaison de Linux) au sein duquel s'exécute l'application. L'utilisation de VM nécessite des machines plus puissantes car plusieurs OS tournent simultanément. Des solutions plus légères (Containers et Docker) sont proposées partageant un moteur (container engine) qui s'exécute dans l'OS. Ceci simplifie la gestion des environnements (p.ex. Dev/Test/Prod, etc.), mais pas la confiance à accorder aux composants.

2. L'automatisation des tests permet de réduire les durées d'exécution des tests, mais augmente le nombre de composants à gérer. À moins de réduire drastiquement les versions et environnements supportés, une explosion combinatoire aura lieu. Pour limiter celle-ci, il faudra limiter les branches (et versions) supportées, voire augmenter les délais entre les livraisons.

L'automatisation des tests peut être nécessaire pour les clients : ils doivent identifier les impacts des modifications subies par leurs logiciels. La gestion de configuration sera associée à des campagnes de tests automatisés dans des environnements – virtuels ou physiques – séparés. L'augmentation du nombre de composants et de versions livrées augmentera exponentiellement la charge de travail.

3. L'obsolescence des composants et le besoin de maintenance des tests impliquent de les gérer en configuration. Certaines fonctionnalités peuvent devenir redondantes, être décommissionnées, remplacées ou modifiées. La gestion des versions devra donc prendre en compte, outre les évolutions fonctionnelles des applications, la maintenance des scripts de tests automatisés et de leurs environnements d'exécution (p.ex. compatibilité de Sélénium et des versions 55 et suivantes de Firefox).

Bonnes nouvelles

Heureusement des solutions existent et se mettent en place :

1. Les environnements de développement intégrés (p.ex. IDE Eclipse ou Visual Studio, …) regroupent les éléments constitutifs d'un produit logiciel (exigences, code source, tests, etc.). Ces outils assurent gestion de configuration, traçabilité et suivi des versions des composants ;

2. Les outils de gestion de configuration (p.ex. GIT) utilisent les fonctions de hashage pour identifier de façon unique les composants et groupes de composants. Cela permet d'obtenir un niveau de fiabilité accru dans les composants ;

3. Les tests automatisés s'intègrent avec les IDE, pour de la vérification des exigences (p.ex. : TDD ou ATDD), le test fonctionnel (Sélénium, UFT, etc.) ou les tests techniques (analyse statique, sécurité et de performances, etc.). Ces tests peuvent être lancés automatiquement à chaque build ;

4. Des outils d'analyse statique du code permettent d'évaluer la qualité du code conçu et de s'assurer du respect des diverses règles de codage, ce qui permet une maintenabilité accrue des composants.

5. Des techniques de test de base (partitions d'équivalence, valeurs limites, etc.) sont connues et mises en œuvre par de plus en plus de testeurs certifiés ISTQB ;

6. L'implication des développeurs dans les activités de tests (cf. DevOps) justifie d'inclure dans leurs formations les techniques de tests pour permettre la détection précoce des défauts ;

Les IDE d'intégration continue permettent un lancement automatique d'outils d'analyse statique et de tests fonctionnels, permettent de limiter les régressions dans le code. Cela améliore la qualité, au dépend d'un processus plus rigoureux et d'une charge de travail plus importante, même si partiellement automatisée.

Encapsulation et regroupement

L'utilisation de Docker et de containers augmente le nombre de composants et impactera l'effort de gestion de configurations. Regrouper dans un même référentiel le niveau de confiance (p.ex. : hash des éléments constitutifs de la version) et tous les composants de la version permet d'inclure – d'encapsuler – dans la référence de version des informations qui en assureront le niveau de confiance. Tout changement (volontaire ou non) sera identifié en comparant le code hash calculé et le hash référencé. Des solutions de type blockchain peuvent assurer un niveau de confiance suffisant (p.ex. : cas de logiciels libres ou OpenSource).

Des techniques existent permettent d'assurer efficacement la gestion des tests, des versions de composants et la confiance que l'on peut leur accorder. •

Yvan Phélizot
Software Craftsman Arolla

Tester ce que l'on ne maîtrise pas

Les vacances de fin d'année ont été un peu courtes. Disons plutôt le week-end : étant parti le vendredi 28 décembre, me voilà déjà de retour au travail le mardi 2 janvier… On ouvre le backlog de la nouvelle année, on prend une nouvelle tâche, on l'analyse et, en bon praticien du TDD, on écrit un nouveau test. On lance les tests et BAM! C'est le drame du nouvel an ! Plusieurs tests sont en « carafe », pour une fois que ce n'est pas le vin ! Mais qui a bien pu casser les tests ? Je suis certain d'avoir tout laissé en bon état en partant. Pardon? Personne n'a commité depuis plusieurs jours ? C'est bizarre…

Je continue mes investigations et me décide à lancer un petit « git bissect » de derrière les fagots. Je choisis un commit que j'avais fait la semaine dernière, je lance… et BAM ! Voilà le responsable : mon commit. Comment est-ce possible ?! L'intégration continue aurait dû avoir planté depuis une semaine ! En cherchant un peu plus, je finis par me rendre compte que le test échoue en comparant 2018 à 2017. Fichtre, Diantre… C'est donc la nouvelle année qui fait planter le test… Par expérience, j'aurais pu m'en douter étant donné que c'est la troisième fois en moins d'un an (les deux autres fois étaient liées au changement d'heure). L'origine du problème était la présence de la date sous forme d'un appel dans le code et d'une date en dur dans le test. Mais, comment résoudre ce problème ?

La dépendance au temps

Pourquoi s'embêter à extraire cette dépendance ? Imaginons une fonction métier dont le but est d'augmenter l'année de 1. On pourrait tester cette fonction de la façon suivante :

```
@Test
public void should_give_the_good_value() {
  Year the_date = some_business_logic();
  Year expectedDate = Year.now().plusYears(1);
  assertThat(the_date).isEqualTo(expectedDate);
}
```

Le problème avec cette approche est que cela ne fonctionne pas de manière absolue. On pourrait même imaginer que le test s'exécute le 31 décembre à 23h59 et que la valeur attendue soit évaluée le lendemain à minuit (c'est un peu tiré par les cheveux, mais il suffit de remplacer les dates par des secondes pour voir que le problème est plausible). C'est une sorte de mini « **race condition** ». Ensuite, on se retrouve à avoir l'implémentation dans les tests, une forme de duplication assez commune. Enfin, on ne prédit plus le résultat attendu, ce qui est contraire à une approche TDD.

Pour résoudre ce problème, plusieurs approches sont possibles. On peut fournir la date en paramètre à l'appel de la fonction nous permettant ainsi d'obtenir une fonction pure, dont une des propriétés est que la sortie se déduit uniquement des entrées :

```
@Test
public void should_give_the_good_value() throws Exception {
  Year the_date = some_business_logic(Year.of(2017));
  assertThat(the_date).isEqualTo(Year.of(2018));
}
```

Si jamais, on a besoin d'évaluer régulièrement l'heure, il est possible de fournir le service sous forme d'une injection de dépendance classique :

```
public class Business {
  private ClockService clockService;
  public Business(ClockService clockService) {
    this.clockService = clockService;
  }
  Year some_business_logic() {
    Year now = clockService.now();
    return now.plusYears(1);
  }
}
```

Il devient alors facile d'injecter un **mock**, un **dummy** ou un **stub** (rayer la mention inutile) en tant qu'horloge. Sandro Mancuso présente cette approche dans la vidéo « **Outside-In TDD** » (https://youtu.be/XHnuMjah6ps).

Enfin, une autre façon de s'y prendre est de s'appuyer sur le framework **JodaTime** qui fournit des fonctions pour forcer la date et l'heure (via **setCurrentMillisFixed** par exemple). Une méthode qui est bien moins élégante que les précédentes.

Ceci dit, la date n'est pas la seule variable non prédictive que l'on peut rencontrer. On peut, par exemple, faire face à l'aléatoire lui-même.

Prédire l'aléatoire

Lors du « Global Day of Code Retreat » 2016 chez Arolla, avec les derniers participants restant à la fin de la journée, nous nous sommes lancé un petit défi : imaginons qu'au lieu d'avoir des cellules qui apparaissent ou disparaissent de manière prédictive, celles-ci vont mourir ou survivre de manière aléatoire, bien que toutes les conditions soient remplies. Et faisons-le en Haskell. Sacré défi ! Comment tester alors qu'on ne sait pas quel résultat nous allons avoir ?

Après quelques étapes, le problème ne nous est pas apparu plus compliqué que le problème classique du jeu de la vie. En effet, il

nous a suffi d'introduire la variable aléatoire sous la forme d'un paramètre que nous contrôlions. Ainsi, l'aléatoire dans le code se réduit au même problème que celui des dates. (cf. le paragraphe précédent pour retrouver les solutions).

Allons plus loin. Lors d'échanges avec un stagiaire de notre équipe, il nous a semblé intéressant de mettre en œuvre un test *a posteriori* sur une implémentation de la méthode de Monte-Carlo. Si vous ne connaissez pas cette méthode, le principe est de calculer de manière aléatoire un grand nombre de valeurs et de compter le nombre de valeurs respectant une condition. Ici, plutôt que d'avoir un jeu de test de milliers de valeurs préfixées, nous avons utilisé une propriété des générateurs de nombres aléatoires :

```
@Test
public void monte_carlo_test() {
  MonteCarlo monteCarlo = new MonteCarlo(new Random(42));
  double ratio = monteCarlo.evaluate();
  assertThat(ratio).isCloseTo(0.9, Offset.offset(0.01));
}
```

En fixant la « **graine** » à la création, nous obtenons un générateur aléatoire tout à fait prédictif, permettant de remplacer un fichier de valeurs de tests difficile à maintenir. Il ne restera plus qu'à remplacer la classe **Random** par son implémentation sécurisée dans le code en production : **SecureRandom**.

Le dernier point : en introduisant de l'aléatoire dans les résultats, on contrevient aux principes FIRST, notamment le R (pour **Repeatable**). Ce principe est issu du livre **Clean Code** de Robert C. Martin. Pour qu'il soit « repeatable », le test :

- Ne doit pas dépendre de son environnement ;
- Doit avoir le même résultat quel que soit l'instant ou l'endroit où il s'exécute.

Par exemple, un test reposant sur un identifiant généré aléatoirement ne peut pas être considéré comme aléatoire car, le résultat n'est pas identique dans le temps.

```
@Test
public void random_uuid() throws Exception {
  UUID uuid = UUID.randomUUID();
  Entity e = new Entity(uuid, "state1");
  e.process();
  assertThat(e).isEqualTo(new Entity(uuid, "state2"));
}
```

Mais, est-ce que l'on peut et doit toujours contrôler l'incontrôlable ?

Tout est sous contrôle ?

Il existe bien évidemment des cas où il n'est pas possible d'injecter la date telle que nous la voulons. Par exemple, le code peut dépendre d'un framework avec la date en dur sur laquelle nous n'avons pas forcément la maîtrise. Ensuite, il peut être dans la nature même du système que de dépendre du temps. Un exemple qui me vient en tête est le cas du framework Akka pour Scala :

```
class CheatyPlayerSpec extends TestKit(ActorSystem("test"))
 with WordSpecLike
 with ImplicitSender {
 implicit val timeout: Timeout = Timeout(1 second)
 "CheatyPlayer" should {
  "always cheat" in {
   val player = system.actorOf(Props[CheatyPlayer])
   player ! Play
   expectMsg(Cheat)
  }
 }
}
```

Les acteurs s'échangent des messages entre eux. Une durée (« **timeout** ») est définie et fait échouer le test si celui-ci dure plus qu'une certaine durée. Comme tout fonctionne de manière asynchrone, on espère globalement que tout se passera avant le déclenchement du timeout.

Inutile de dire que ce genre de test est à la fois lent et surtout instable (leur comportement est différent en fonction de la puissance de la machine, plus ou moins rapide).

Le principe **FIRST** ne s'applique pas à tous les tests. Dans le cas du property-based testing, les données sont générées aléatoirement. On vérifie si une propriété est toujours valide sur le résultat. On espère ainsi que le test soit toujours utile mais sans certitude. Les tests de plus haut niveau ne respectent pas non plus cette propriété. De nombreux paramètres peuvent intervenir, rendant souvent les tests fragiles et peu fiables (puissance des machines, réseau, …).

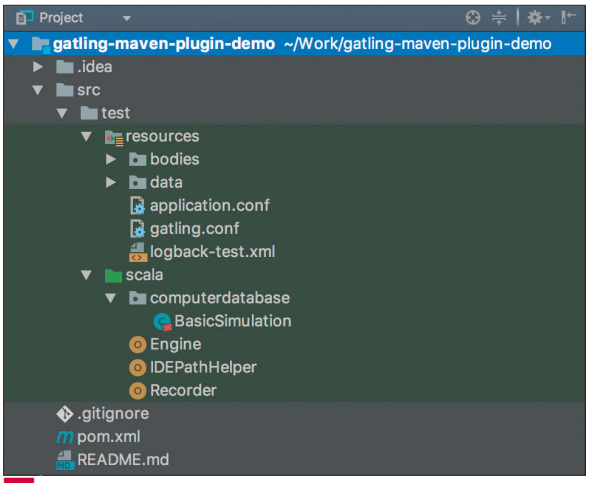
Les tests avec des dépendances non prédictives ont une fâcheuse tendance à échouer au moment où on s'y attend le moins. Cependant, une bonne hygiène de développement en essayant de s'abstraire le plus possible de ces contraintes permet d'obtenir un meilleur design et des tests robustes et fiables.

Ah… la nouvelle année… Cette année, je prends donc deux résolutions. La première : refaire un peu de sport. La seconde : ne plus avoir de tests qui dépendent du temps. Je me demande si j'arriverai à tenir toutes ces bonnes résolutions ou si je craquerai avant. •

Alexandre Chaouat
développeur Full-Stack chez
Gatling Corp

Mettre en œuvre et interpréter les différents types de tests de charge avec Gatling

Depuis sa création en 2011, Gatling s'est peu à peu imposé comme un incontournable du monde du test de charge pour applications web. Le test de charge est nécessaire pour prévoir le comportement d'une application soumise à une très forte charge d'utilisateurs, afin de garantir une future qualité de service. Gatling présente plusieurs avantages : c'est un outil open-source, et son mode d'utilisation tranche nettement avec les acteurs historiques du test de charge. Il se destine aux spécialistes, mais aussi aux développeurs qui peuvent désormais participer à la création et la maintenance des tests de charge de leur application, au même titre que leurs tests unitaires et d'intégration. Gatling est développé par Gatling Corp, qui édite également la version entreprise Gatling FrontLine.



1 *La structure du projet gatling-maven-plugin-demo*

En tant que développeur chez Gatling Corp, j'ai été amené à analyser de nombreuses applications, avec à chaque fois leurs spécificités et leurs contraintes. Je vais vous apporter quelques astuces et bonnes pratiques afin de rapidement mettre en place des tests de charge de qualité. L'analyse des rapports générés en fin de test n'est pas une mince affaire : certains indicateurs sont toutefois très instructifs. Nous verrons donc comment les analyser, notamment au regard des différents types de test de charge que l'on peut mettre en place.

Réaliser une simulation Gatling

Gatling a fait le choix dans son approche d'un parti pris très fort : être capable de s'intégrer aux outils déjà en place des applications. Un test Gatling peut ainsi être lancé depuis les outils de build du monde Java, comme Maven, que nous allons utiliser par la suite. Il est également possible d'utiliser un bundle Gatling afin de s'affranchir des outils de build, mais ça n'est pas la démarche que nous allons utiliser ici car elle ne simplifie pas l'automatisation et introduit un composant supplémentaire à vos outils de builds standards.
Pour commencer à réaliser un test Gatling, il faut :

- un JDK 8
- un IDE afin de simplifier le développement de nos tests.
- Un outil de build (Maven, SBT ou Gradle)

Il est possible de cloner le repository https://github.com/gatling/gatling-maven-plugin-demo afin d'avoir une base pour écrire un test HTTP. Gatling permet également d'écrire des tests JMS et WebSockets. 1
On trouve en premier lieu dans le dossier ressources, le dossier bodies qui permet de stocker le corps des requêtes qui seront générées. Le dossier data contiendra les jeux de données. On y trouve aussi des fichiers de configuration pour gérer le log, la librairie Akka, et Gatling lui-même.
C'est le dossier scala qui contiendra nos tests.
La classe Engine permet de lancer Gatling de manière programmatique, et la classe Recorder permet de lancer l'utilitaire permettant de capturer une navigation web et de la traduire en test Gatling.
BasicSimulation.scala est un exemple de simulation qui va nous permettre d'en décrire la structure.
Le fichier commence par le nom du package et les imports nécessaires

```
package computerdatabase
import io.gatling.core.Predef._
import io.gatling.http.Predef._
import scala.concurrent.duration._
```

Pour que Gatling considère votre classe comme un test, il faut qu'elle étende de la classe Simulation.
Une simulation Gatling se divise en plusieurs éléments qui peuvent être disséminés à l'intérieur de plusieurs classes avant d'être finalement assemblés au sein de la classe qui étend Simulation au moment du setup. Diviser son code a le même intérêt que dans un projet de développement normal : rendre le code plus facile à lire et plus maintenable.
Chacun des éléments que nous allons maintenant décrire est stocké dans une **val** (une référence immutable).
En premier lieu la configuration globale HTTP :

```
val httpConf = http
  .baseURL("http://computer-database.gatling.io")
  …
```

Celle-ci permet de définir la configuration qui sera appliquée par défaut sur l'ensemble des requêtes HTTP. On peut y définir une baseURL permettant de définir des url relatives pour les requêtes, le

user-agent à utiliser, un éventuel proxy, de l'authentification etc.
On retrouve ensuite la val scn contenant un scénario, c'est à dire une chaîne de requêtes, de pauses, éventuellement de traitement et de mise en place de critères de validation sur les requêtes.

```
val scn = scenario("Scenario Name")
.exec(http("request_1")
.get("/"))
.pause(7)
…
```

et enfin, le setup qui permet d'assembler ces éléments

```
setUp(scn.inject(atOnceUsers(1)).protocols(httpConf))
```

Cette fonction permet de définir les scénarios qui vont être utilisés, le profil d'injection utilisé, et d'appliquer la configuration HTTP via la méthode **protocols.**
Le profil d'injection correspond au nombre d'utilisateurs qui vont réaliser le scénario et à leur mode d'arrivée. La méthode atOnceUsers permet de définir un profil d'injection dans lequel tous les utilisateurs vont arriver au même moment, et elle prend en argument le nombre d'utilisateurs : ici 1 seul utilisateur sera injecté. Il existe de nombreuses fonctions de ce type correspondant chacune à des modes d'injection différents. Il est possible de chaîner les profils d'injection pour faire évoluer l'arrivée des utilisateurs au cours du test :

```
setUp(scn.inject(atOnceUsers(1), nothingFor(2 minutes), rampUsers(34) over (3 minutes))
.protocols(httpConf))
```

Pour lancer le test, il suffit de saisir la commande suivante :

```
mvn gatling:test -Dgatling.simulationClass=computerdatabase.BasicSimulation
```

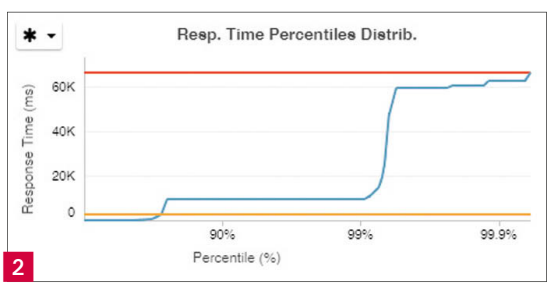
En composant ainsi ces différents profils d'injection, il est possible de mettre en œuvre les principaux types de test de charge. C'est ce que nous verrons dans la dernière partie de cet article.

Analyse et mise en œuvre d'un test de charge : quelques clefs

Moyenne vs percentiles

Une erreur assez courante dans l'analyse des tests de charge, est de considérer la moyenne des temps de réponse des utilisateurs comme la métrique pertinente, alors que les percentiles, contrairement à la moyenne, donnent une idée de la répartition des utilisateurs par rapport aux temps de réponse. Ils offrent ainsi la possibilité de distinguer les utilisateurs ayant eu de très mauvais temps de réponse par rapport à un seuil, qui ne se retrouvent plus masqués par l'éventuelle grande quantité de temps de réponse plus faibles au sein de la moyenne. Gatling vous présente dans son rapport de nombreux percentiles qui vous permettent d'analyser votre test correctement. 2

Distribution des temps de réponse, montrant à quel point la moyenne (en jaune) ne permet pas de distinguer les temps de réponse bien supérieurs.



2

Fréquence des tests

Les campagnes de test de charge sont souvent reléguées à la fin de la période de développement, juste avant la mise en production, comme une forme de validation de la capacité d'une application à assumer une charge donnée. C'est une façon de faire risquée, et qui peut se retourner contre vous en cas de problèmes, car cela implique une grande difficulté à en déterminer l'origine, et peut déboucher sur une remise en question des choix d'architecture ou des normes de développement.
La bonne pratique est d'adopter une démarche plus agile : tester en charge tout le long du cycle de développement. Cela vous permettra de traiter les problèmes en amont et de limiter les changements à analyser en cas de régression. Concrètement, se rendre compte d'une régression en réalisant des tests de charge une seule fois par mois oblige à analyser un mois de changements.

Automatisation

Pour maintenir la fréquence suggérée au paragraphe précédent, les tests de charge doivent s'intégrer avec votre usine logicielle. La fréquence peut être moins importante que celle des tests unitaires ou d'intégration, et peut être quotidienne plutôt que à chaque commit. Un plugin Gatling pour Jenkins est disponible pour simplifier cette automatisation et avoir accès au rapport Gatling directement dans l'interface Jenkins. L'intégration avec TeamCity et Bamboo est quant à elle disponible uniquement pour la version entreprise Gatling FrontLine.

Modèle ouvert vs modèle fermé

Pour tester votre application, il vous faut déterminer la nature du trafic auquel elle est confrontée. La plupart des applications web ont un modèle dit ouvert, c'est à dire que les utilisateurs arrivent dans l'application sans contrainte sur le nombre d'utilisateurs concurrents. C'est la métrique de l'arrivée des utilisateurs par unité de temps qui va ainsi être la plus à même de matérialiser votre flux d'utilisateurs. Au contraire, dans un modèle fermé, le système n'accepte qu'un nombre limité d'utilisateurs à la fois. C'est souvent le cas des billetteries en ligne, qui vont vous placer dans une file d'attente si jamais le site a atteint sa limite. Dans ce modèle, c'est le nombre d'utilisateurs concurrents qui est intéressant pour simuler votre trafic. Le modèle ouvert est le mode par défaut de Gatling, et il est possible de simuler le modèle fermé en faisant boucler ses utilisateurs et en vidant les caches à chaque tour de boucle. Néanmoins, cela reste très imparfait car les connections ne sont pas recréés à chaque tour de boucle. La gestion d'un vrai modèle fermé est prévue dans la version 3 de Gatling, et est d'ores et déjà disponible dans FrontLine.

Variabilisation

Il est essentiel lorsqu'on réalise un test de charge de faire en sorte de variabiliser les requêtes qui sont effectuées, c'est à dire de changer les paramètres de celles-ci. En effet, si vous ne le faites pas, vous risquez d'utiliser intensivement les caches de votre application (Varnish, cache de la base de données, etc), et ainsi d'avoir une fausse idée des performances réelles de votre application. C'est l'API feeder de Gatling qui vous permettra de gérer vos jeux de données, que ceux-ci soient des fichiers, ou générés à la volée par une fonction.

Confronter les résultats aux métriques systèmes et applicatives

Analyser un test de charge permet de mettre en évidence des problèmes applicatifs ou de composants sur votre système qu'il va falloir diagnostiquer. Le meilleur moyen d'y arriver est de mettre en corrélation les résultats des tests avec de la métrologie système et applicative venant de solutions comme Prometheus ou InfluxDB. En outre, une régression peut ne pas se manifester par une augmentation des temps de réponse, mais par celle des ressources nécessaires à assumer la même charge. L'intégration de FrontLine en tant que source de données pour Grafana permet de simplifier ce genre de mise en relation.

Les différents types de test de performance

Nous allons maintenant voir comment mettre en œuvre trois des principaux tests de charge : le test d'endurance, de capacité et le stress test.

Test d'endurance

Un test d'endurance consiste à injecter une quantité assez importante d'utilisateurs, constamment pendant une durée assez longue. Ce test permet de tester le vieillissement de l'application sur le long terme. Il est particulièrement utile pour déceler des problèmes de fuites mémoires ou de ressources non libérées.

Pour mettre en place ce type de test dans Gatling, nous allons paramétrer une première augmentation progressive des utilisateurs, afin de « chauffer » l'application, jusqu'à atteindre le palier qui sera maintenu pendant un long moment.

```
val rampStepDuration = 5 minutes
val stageDuration = 10 hours
val stageUserPerSecond = 100

setUp(scn.inject(
 rampUsersPerSec(0) to stageUserPerSecond during(rampStepDuration),
 constantUsersPerSec(stageUserPerSecond) during(stageDuration)
).protocols(httpConf))
```

Test de capacité

Un test de capacité consiste à injecter un nombre d'utilisateurs constant par palier, en augmentant l'arrivée des utilisateurs à chaque palier, afin de déterminer à quel palier l'application commence à rencontrer des anomalies. On pourra considérer que le test est un succès si le dernier palier atteint sans problèmes est au-dessus de la charge prévue sur l'application. Ce test donne des informations sur la/les ressources qui vont être limitantes dans l'application à un certain niveau (le CPU, la RAM, l'accès à la BDD, etc.). Voilà un exemple de ce type de test dans Gatling :

```
val numberOfSteps = 10
val userPerStage = 50
val stepDuration = 10 minutes
val rampDuration = 2 minutes

val injectionProfile = (1 to numberOfSteps).flatMap { step =>
 val ramp = rampUsersPerSec((step - 1) * userPerStage) to (step * userPerStage) during
rampDuration
 val stage = constantUsersPerSec(step * userPerStage) during stepDuration
 Seq(ramp, stage)
}

setUp(scn.inject(injectionProfile)).protocols(httpConf)
```

On aurait pu chaîner les rampUserPerSec et les constantUserPerSec manuellement, mais il est beaucoup plus intéressant de faire une boucle. Cela évite d'avoir à calculer le nombre d'utilisateurs par seconde à chaque étape, et changer le nombre d'étapes revient à changer la variable numberOfSteps.

On crée un intervalle allant de 1 au nombre d'étapes (step) que l'on souhaite réaliser, et nous utilisons la fonction flatMap pour générer pour chacun des steps le rampUserPerSec et le constantUserPerSec. Nous utilisons flatMap et pas map car pour chaque step nous renvoyons une séquence, et nous ne voulons pas nous retrouver à la fin avec une séquence de séquence.

Test de stress

Dans le cadre du stress test, on veut simuler un pic d'arrivée d'utilisateurs. La situation testée peut être une promotion sur un site de vente, une notification sur téléphone ou tout autre mécanisme qui peut créer une arrivée massive et inhabituelle sur une application web.

Pour se faire, nous allons simuler une rampe d'arrivée d'utilisateurs avant d'atteindre un léger plateau, puis créer un gros pic d'utilisateur et redescendre vers le plateau.

```
val rampUpDuration = 2 minutes
val stageUsers = 100
val stageBeforeDuration = 1 minute
val peakUsers = 2000
val peakUpDuration = 30 seconds
val peakDuration = 2 minutes
val peakDownDuration = 30 seconds
val stageAfterDuration = 2 minutes

setUp(scn.inject(
 rampUsersPerSec(0) to (stageUsers) during rampUpDuration, // On grimpe jusqu'au plateau
 constantUsersPerSec(stageUsers) during stageBeforeDuration, // On y reste un moment
 rampUsersPerSec(stageUsers) to peakUsers during peakUpDuration, // Le pic
 constantUsersPerSec(peakUsers) during peakDuration, // on reste au pic
 rampUsersPerSec(peakUsers) to (stageUsers) during(peakDownDuration), // on redescend
jusqu'au plateau
 constantUsersPerSec(stageUsers) during stageAfterDuration // on y reste un moment
)).protocols(httpConf)
```

Conclusion

Nous avons ainsi vu brièvement comment mettre en place un test Gatling en HTTP, avant de nous intéresser à quelques éléments nécessaires pour tester correctement une application, et présenter la mise en place de quelques types de test de charge. Gatling permet une myriade de possibilités pour customiser vos scénarios et vos profils d'injections. C'est l'avantage d'être orienté code : ce qui n'est pas présent par défaut, vous pouvez le coder vous-même. Bon courage pour la réalisation de vos tests de charge !

•

Christophe PICHAUD
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com |
www.windowscpp.net

Les tests en C++

NeOS-SDI makes IT work

Les outils de tests pour le C++ ne sont pas récents et ils n'ont rien à envier à ceux des langages managés. Bien sûr, la notion de Reflection n'existe pas en C++ mais les fondamentaux existent depuis presque 20 ans ; depuis 2000 exactement. Pour parler testing, on va aussi parler IDE car certains outils sont couplés à Visual Studio, notre IDE préféré.

CPPUnit

Place au pionnier, CPPUnit : c'est le kit outil historique pour C/C++. Il a été forgé plusieurs fois et c'est une valeur sûre. C'est le portage de JUnit, l'outil fait en Java.

Il est disponible ici : https://dev-www.libreoffice.org/src/cppunit/

CPPUnit n'est pas intégré à Visual Studio, mais il a l'avantage de marcher sur Linux et sur Windows. Son fonctionnement requiert de le compiler sous forme de lib ou de DLL et ensuite de lancer un Runner : un lanceur de tests de nos tests ! Compiler CPPUnit n'est pas très difficile, il suffit de charger la solution cppunit-1.13.2\src\CppUnitLibraries2010.sln qui est pour Visual Studio 2010 et de faire build sur les projets cppunit et cppunit_dll. Pour voir ce qu'est un programme de test, il suffit d'inclure cppunit-1.13.2\examples\simple\simple.vcxproj dans la solution. La solution requiert cppunitd.lib au link et c'est tout. Voyons à quoi ressemble l'exécution d'une série de tests en C++ : 1

On y trouve des succès et des échecs. On trouve aussi le numéro de la ligne où sont les erreurs. C'est simple et efficace. Maintenant, regardons au niveau du code comment on fait écrit des tests en C++. On commence par écrire une classe de test :

```
#include <cppunit/extensions/HelperMacros.h>

class ExampleTestCase : public CPPUNIT_NS::TestFixture
{
 CPPUNIT_TEST_SUITE( ExampleTestCase );
 CPPUNIT_TEST( example );
 CPPUNIT_TEST( anotherExample );
 CPPUNIT_TEST( testAdd );
 CPPUNIT_TEST( testEquals );
 CPPUNIT_TEST_SUITE_END();

protected:
 double m_value1;
 double m_value2;

public:
 void setUp();

protected:
 void example();
 void anotherExample();
 void testAdd();
 void testEquals();
};
```

La routine setUp() est lancée au démarrage et ici il n'y en a pas, mais la méthode tearDown() est exécutée à la fin. C'est la seule subtilité de cette classe. Pour le reste, on utilise les macros pour définir des tests.

```
Developer Command Prompt for VS 2017
D:\Dev\cppunit-1.13.2\examples\simple\Debug>simple
ExampleTestCase::example : assertion
ExampleTestCase::anotherExample : assertion
ExampleTestCase::testAdd : assertion
ExampleTestCase::testEquals : assertion
ExampleTestCase.cpp(8) : error : Assertion
Test name: ExampleTestCase::example
double equality assertion failed
- Expected: 1
- Actual  : 1.1
- Delta   : 0.05

ExampleTestCase.cpp(16) : error : Assertion
Test name: ExampleTestCase::anotherExample
assertion failed
- Expression: 1 == 2

ExampleTestCase.cpp(28) : error : Assertion
Test name: ExampleTestCase::testAdd
assertion failed
- Expression: result == 6.0

ExampleTestCase.cpp(45) : error : Assertion
Test name: ExampleTestCase::testEquals
equality assertion failed
- Expected: 12
- Actual  : 13

Failures !!!
Run: 4   Failure total: 4   Failures: 4   Errors: 0

D:\Dev\cppunit-1.13.2\examples\simple\Debug>
```

1

```
#include <cppunit/config/SourcePrefix.h>
#include "ExampleTestCase.h"

CPPUNIT_TEST_SUITE_REGISTRATION( ExampleTestCase );

void ExampleTestCase::example()
{
 CPPUNIT_ASSERT_DOUBLES_EQUAL( 1.0, 1.1, 0.05 );
 CPPUNIT_ASSERT( 1 == 0 );
 CPPUNIT_ASSERT( 1 == 1 );
}


void ExampleTestCase::anotherExample()
{
 CPPUNIT_ASSERT (1 == 2);
}

void ExampleTestCase::setUp()
{
 m_value1 = 2.0;
 m_value2 = 3.0;
}

void ExampleTestCase::testAdd()
```

```
{
 double result = m_value1 + m_value2;
 CPPUNIT_ASSERT( result == 6.0 );
}


void ExampleTestCase::testEquals()
{
 long* l1 = new long(12);
 long* l2 = new long(12);

 CPPUNIT_ASSERT_EQUAL( 12, 12 );
 CPPUNIT_ASSERT_EQUAL( 12L, 12L );
 CPPUNIT_ASSERT_EQUAL( *l1, *l2 );

 delete l1;
 delete l2;

 CPPUNIT_ASSERT( 12L == 12L );
 CPPUNIT_ASSERT_EQUAL( 12, 13 );
 CPPUNIT_ASSERT_DOUBLES_EQUAL( 12.0, 11.99, 0.5 );
}
```

Comme vous pouvez le distinguer sur le code source, on utilise des macros pour déclarer ce qui nous intéresse. La mécanique est de faire appel à des routines (macros) de type ASSERT avec des conditions à remplir sinon le test est en erreur. Pour exécuter le test, il faut un host CPPUnit, dans le même module que vos tests :

```
#include <cppunit/BriefTestProgressListener.h>
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/TestResult.h>
#include <cppunit/TestResultCollector.h>
#include <cppunit/TestRunner.h>


int main()
{
 // Create the event manager and test controller
 CPPUNIT_NS::TestResult controller;

 // Add a listener that colllects test result
 CPPUNIT_NS::TestResultCollector result;
 controller.addListener( &result );

 // Add a listener that print dots as test run.
 CPPUNIT_NS::BriefTestProgressListener progress;
 controller.addListener( &progress );

 // Add the top suite to the test runner
 CPPUNIT_NS::TestRunner runner;
 runner.addTest( CPPUNIT_NS::TestFactoryRegistry::getRegistry().makeTest() );
 runner.run( controller );

 // Print test in a compiler compatible format.
 CPPUNIT_NS::CompilerOutputter outputter( &result, CPPUNIT_NS::stdCOut() );
 outputter.write();

 return result.wasSuccessful() ? 0 : 1;
}
```

Le code est toujours le même donc on en fait un code partagé et le tour est joué… Vous allez me dire que le résultat est un peu spartiate. En effet, on utilise Visual Studio et cela sert, bien que les tests soient visualisables dans la fenêtre de Tests du Visual… Examinons une autre suite de tests.

Google Test Adapter

Vous allez tiquer… Google fait un add-in pour VS ? Eh oui ! Et même qu'il est open-source sous GitHub. Ici : https://github.com/csolten-born/GoogleTestAdapter . Il est présent aussi sur le marketplace Visual Studio. Cet add-in tire parti de la fenêtre « Test Explorer » dans Visual Studio. Par contre, il faut compiler une solution comme suit. Ouvrez la solution googletest-master\googletest\msvc\2010\gtest.sln. Il y a plusieurs projets mais le plus simple pour comprendre la philosophie Google Tests c'est gtest_unittest-md. Il contient un fichier énorme avec plein de cas de tests. Pour faire la liaison avec VS, il faut compiler le projet et afficher la fenêtre Test Explorer. Voici à quoi cela ressemble : 2

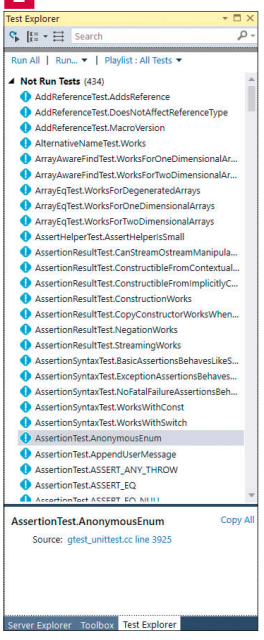
Au niveau du code, on va construire un ou plusieurs jeux de tests comme suit. Le fichier main ressemble à cela :

```
#include "gtest/gtest.h"

int main(int argc, char **argv) {
  ::testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

On ajoute le fichier gtest-all.cc dans notre projet comme ça, le projet contient le hosting de Google Test. Créons une classe de tests dans un fichier MyTests.h :

```
#pragma once

#include "gtest/gtest.h"

class MyTests : public testing::Test
{
public:
  MyTests() {}
  ~MyTests() {}

public:
  int Add(int i, int j) { return i + j; }

  virtual void SetUp()
  {
     printf("SetUp()...\n");
  }
```

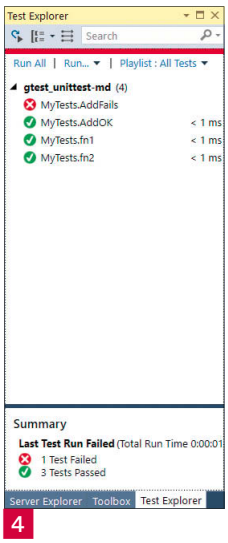
2



```
  virtual void TearDown()
  {
      printf("TearDown()...\n");
  }
};
```

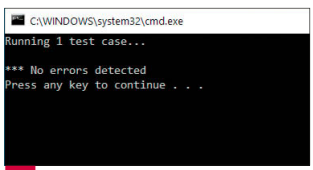
Dans le fichier MyTests.cpp, il y a cela :

```
#include "MyTests.h"

TEST_F(MyTests, fn1) { printf("fn1\n"); }
TEST_F(MyTests, fn2) { printf("fn2\n"); }

TEST_F(MyTests, AddOK)
{
  EXPECT_EQ(10, Add(5, 5));
}

TEST_F(MyTests, AddFails)
{
  EXPECT_EQ(10, Add(15, 5));
}
```



4

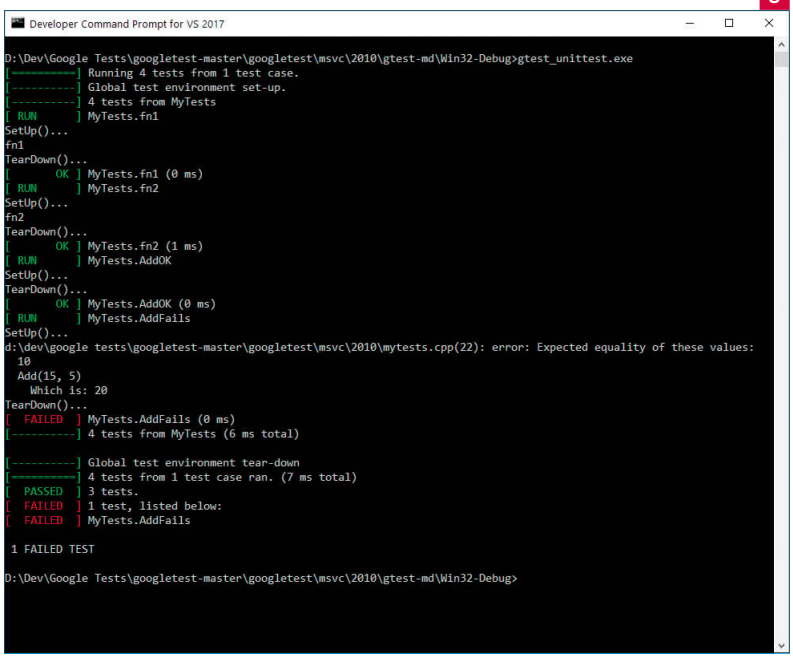
Vous pouvez constater que le code est très simple. On teste deux fonctions et deux méthodes. A chaque fois, la fonction SetUp() est appelée et à la fin la fonction TearDown() aussi.
Si je compile ce projet et que je le Run, la fenêtre Test Explorer de Visual Studio affiche cela : 3
Si vous n'avez pas Visual Studio, Google Test peut se lancer en ligne de commandes : 4



5

Le fichier gtest.cc contient de nombreux tests et de nombreuses macros. Il est possible de jouer avec les exceptions, les paramètres d'entrée et de nombreuses autres choses. C'est immense comme suite de test.

3



Boost.Test

La librairie Boost est particulièrement bien connue dans la communauté C++. De plus, elle possède une lib qui se nomme Test. La première chose à faire de builder Boost et ses librairies. Commençons par le début, il faut downloader Boost. Allez sur boost.org et téléchargez l'archive 7z pour Windows. Extrayez-la en local et ouvrez un VS command prompt. Tapez bootstrap.bat. Cela va compiler le tool de build. Ensuite tapez cela :

```
bjam toolset=msvc-14.1 variant=debug,release threading=multi link=shared
address-model=64
```

Cela prend un peu de temps et vous allez avoir dans le répertoire stage\lib toutes les libs… L'avantage de Boost, c'est que c'est bien fait et bien documenté. Le répertoire libs\test\example contient des jeux de tests avec différents types de ASSERT. Le principe est toujours le même, on fait des tests ou des suites de tests. Il est possible de faire des tests cases avec ou sans paramètres et sur des templates.
Voici comment faire un test minimal avec Boost.Test :

```
// ConsoleApplication1.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <boost/test/included/unit_test.hpp>
using namespace boost::unit_test;

void free_test_function()
{
  BOOST_TEST(true /* test assertion */);
}

test_suite* init_unit_test_suite(int /*argc*/, char* /*argv*/[])
{
  framework::master_test_suite().
     add(BOOST_TEST_CASE(&free_test_function));

  return 0;
}
```

La fonction init_unit_test_suite est requise. Chaque Framework possède son main, nous l'avons vu.
Si on lance ce programme console, on obtient ça : 5

Intégration continue

Que ce soit CPPUnit ou Google Tests, le mode console retourne un code qui permet de savoir si c'est PASSED ou FAILED. Il est ainsi possible de faire tourner les TU dans une chaîne d'intégration continue.

Conclusion

Ecrire des tests unitaires en C/C++ est très simple. On utilise les macros ASSERT de son framework de test préféré et le tour est joué. Quand on fait des TU, l'intelligence est dans les TU et non dans le framework de tests. Il existe d'autres framework comme catch, etc… Mais le principe est toujours le même.
A vous de jouer ! •

Emmanuel Roset
Ingénieur Produits Acquisition de Données
National Instruments
emmanuel.roset@ni.com

LabVIEW NXG : accélérer le développement et le déploiement des systèmes de test automatique

LabVIEW est un logiciel qui depuis 30 années est orienté sur l'automatisation des mesures. Au début, cette automatisation passait par le pilotage – depuis un PC – des appareils de laboratoire, sur table puis disposés en baies. Ces appareils utilisaient (et encore à ce jour) le bus IEEE488 dit GPIB (General Purpose Interface Bus). Puis progressivement les instruments au format PXI se sont imposés pour une automatisation encore plus rapide et des coûts encore plus faibles. Ces dernières années, National Instruments a tout mis en œuvre pour réarchitecturer et s'assurer qu'il maintienne des performances élevées tout en offrant la meilleure expérience utilisateur possible. En 2017, la première version de LabVIEW NXG a introduit de nouvelles fonctionnalités tant au niveau de l'interface que pour l'interactivité des mesures. LabVIEW NXG 2.0, est de nouveau axée sur le thème de l'automatisation des mesures pour les systèmes de test et de validation, du laboratoire à la production.

Parmi les nouveautés, nous découvrons un outil graphique de configuration et de documentation qui répertorie l'ensemble des matériels présents ou connectés au système, qu'ils soient de National Instruments ou de tiers. Appelé SystemDesigner, cet outil offre une vue schématisée des composants symbolisés par des blocs et des étiquettes qui sont interconnectés avec des flèches. L'insertion d'annotations textuelles sur les liaisons ou les objets et l'ajout d'images réelles (un circuit PCB à tester par exemple) sont également proposés. Une détection automatique est réalisée pour rafraîchir la vue globale non seulement du matériel mais également des logiciels, tels que les drivers installés ou non. Par ailleurs, des liens vers les téléchargements des logiciels drivers requis sont proposés lorsqu'ils ne sont pas déjà installés. Vous y trouvez également toutes les informations nécessaires pour les brochages des matériels vers les unités sous test (UUT) – ce qui vous évite d'avoir à parcourir la documentation papier – ainsi que sur l'état de l'étalonnage. Enfin, à partir de cette vue vous pouvez ouvrir pour chaque composant une fenêtre de mesure interactive et commencer à mettre au point vos séquences de tests. 1

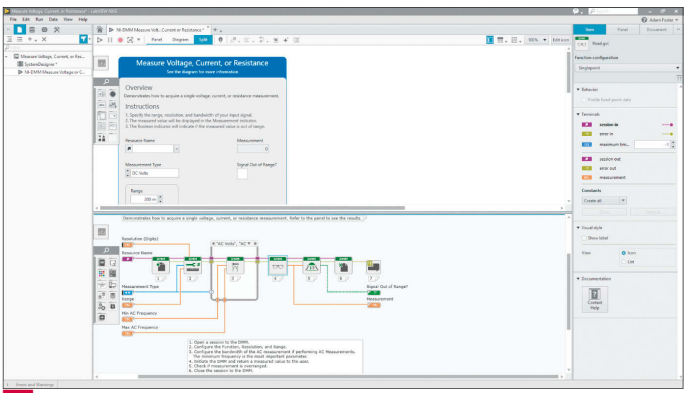
Après avoir vérifié que votre système fonctionne à l'aide des faces avant interactives, il est temps d'automatiser vos mesures en utilisant l'API LabVIEW NXG.

Il s'agit de blocs de fonctions logicielles pilotant les instruments. Plutôt que de démarrer votre conception par une page vide, les ingénieurs de R&D ont développé des modèles et programmes d'exemples pour tous les instruments modulaires NI, ainsi qu'une boîte spécifique pour les instruments de tiers. Vous pouvez ainsi réutiliser et combiner des exemples existants pour créer vos tests plus rapidement. 2

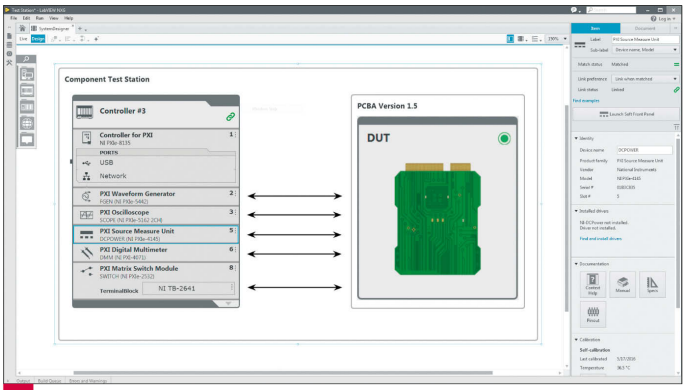
Dans cette optique de réutilisation, il est possible via le nœud C de l'outil d'inclure un code existant écrit en langage de programmation C (GNU89, bibliothèques ANSI-C) et même de développer de nouveaux algorithmes. Les fonctionnalités incluent l'affichage de menus déroulants d'éléments lorsque vous tapez au clavier pour vous aider à sélectionner une fonction. Une fenêtre vous permet de gérer les erreurs ou mises en garde de compilation. Un volet de sortie s'affiche si vous faites appel à la fonction cnode_printf.

En outre, si vous avez à disposition des IP exportées au format DLL, il est possible de référencer ces bibliothèques et d'appeler leurs fonctions. 3

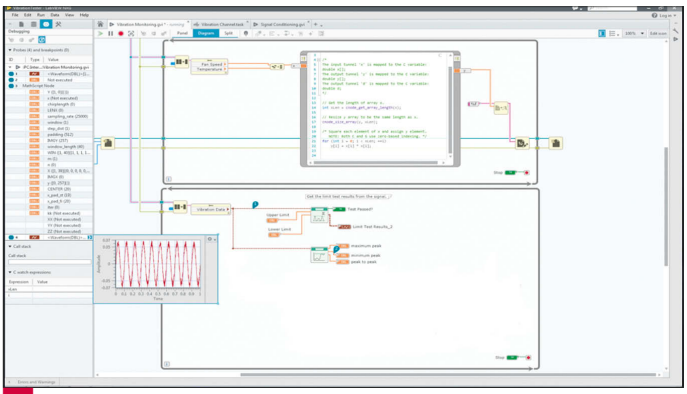
LabVIEW NXG 2.0 fournit une interface moderne encore plus abstraite et souple. De nombreux objets préconfigurés voient le jour, associés à des



2 *API LabVIEW NXG et exemples de programmes de mesures réutilisables*



1 *SystemDesigner pour une vue schématisée des composants du système*



3 *Nœud C de LabVIEW NXG pour inclure un code existant écrit en langage de programmation C*

guides dynamiques de placement ou d'alignement sur une grille et la modification de thèmes, taille et couleurs. Ces nouveautés reposent sur l'emploi de graphiques vectoriels qui permettent d'effectuer des zooms avant et arrière sans sacrifier la résolution ou la qualité. Elles concernent non seulement les objets mais également les données dans les graphes, permettant de créer des rapports de tests encore plus précis. 4

Pour la mise au point des tests unitaires, une méthode interactive est proposée. Dès que vous voyez des données (dans un graphe ou autre), il suffit de faire un clic droit pour en capturer une portion et la stocker dans votre projet pour un usage ultérieur (par exemple, pour y appliquer ensuite un filtrage numérique par une fenêtre interactive ou sélectionner une zone de limites manuellement sur la courbe). Après avoir manipulé et observé les variations des résultats en temps réel qui découlent des paramètres que vous avez sélectionnés, vous pouvez choisir de générer automatiquement un VI de fonction que vous pourrez déposer dans un diagramme. Celui-ci sera alors déjà préconfiguré avec les bons paramètres afin de réaliser au plus vite le pas de test. 5

Vous voilà à l'étape où votre application de test est réalisée. Il faut maintenant la partager et la déployer. LabVIEW NXG 2.0 permet de créer des paquets de distribution pour échanger facilement du code entre les systèmes de développement. L'outil NIPM (NI Package Manager) permet de trouver et télécharger tous les logiciels NI et les drivers matériels. Vous pouvez également créer des répertoires internes pour le maintien de configurations spécifiques à votre système. L'outil Application Builder vous permet de créer des applications autonomes et des bibliothèques. Ou simplement de retirer le code source éditable afin de protéger votre propriété intellectuelle ou vous préserver d'autres modifications involontaires.

Tous les outils nécessaires à la gestion d'applications modulaires sont regroupés dans une fenêtre pour en simplifier l'utilisation. 6

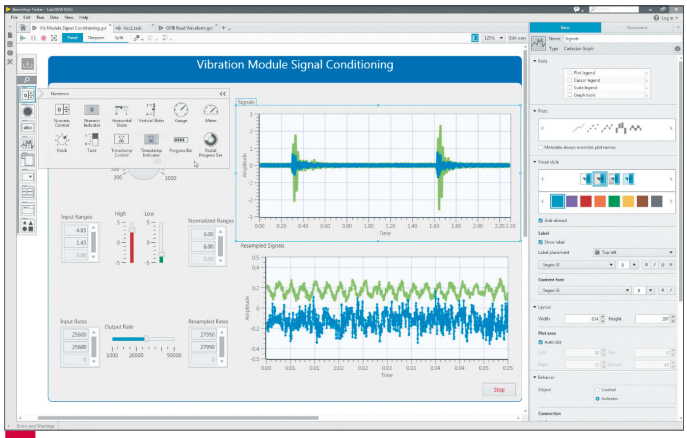
Parmi les applications déployées se trouvent les faces avant Web. Par le passé et depuis la version LabVIEW 6i sortie dans les années 2000, le partage des interfaces Web se faisait sans programmation par un simple clic. Mais ces interfaces nécessitaient l'utilisation d'un moteur d'exécution (Runtime) sur les machines cibles afin de les exécuter. L'outil permet de surmonter cet obstacle. Vous pouvez définir intuitivement votre interface de VI qui sera déployable directement et sans programmation sur tous les navigateurs, sans aucun plugin, téléchargement ou installation préalable. La nouvelle méthode permet la création et l'utilisation d'interfaces sur les mobiles pour du contrôle et de la surveillance de systèmes de tests déportés. Vous pouvez ainsi regrouper et présenter vos résultats de tests et les mesures avec des outils graphiques puissants. En accédant au code généré, vous pouvez étendre les interfaces avec du contenu HTML tel que des vidéos ou des PDF pour fournir de la documentation. Vos résultats peuvent ainsi être présentés et regroupés sur un tableau de bord à l'aspect professionnel à destination de vos clients ou utilisateurs. 7

Pour conclure, parmi les étapes de réalisation d'un système de test automatisé, qui débute avec la collecte des besoins du cahier des charges de l'unité sous test (DUT) et se termine par le déploiement et la maintenance du système de test, LabVIEW NXG facilite la conception à plusieurs égards. Son avantage principal réside dans la configuration intuitive du système de test grâce aux visualisations et à l'autodétection du système complet, et dans la création et la mise au point interactives des pas de test. Le logiciel permet également le pilotage de très nombreux instruments et fournit une visualisation via des interfaces Web, sans programmation ni installation nécessaire. Pour ceux qui souhaitent de l'automatisation encore plus complète, le séquenceur NI TestStand intègre de nouveaux pas. •

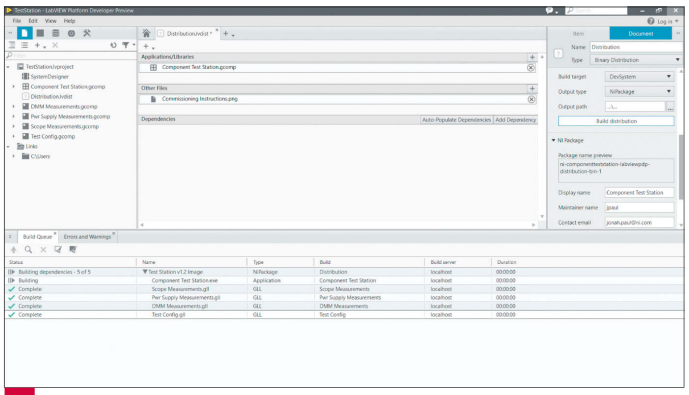
Liens

http://www.ni.com/fr-fr/shop/labview/how-do-i-use-labview-to-develop-production-test-systems.html

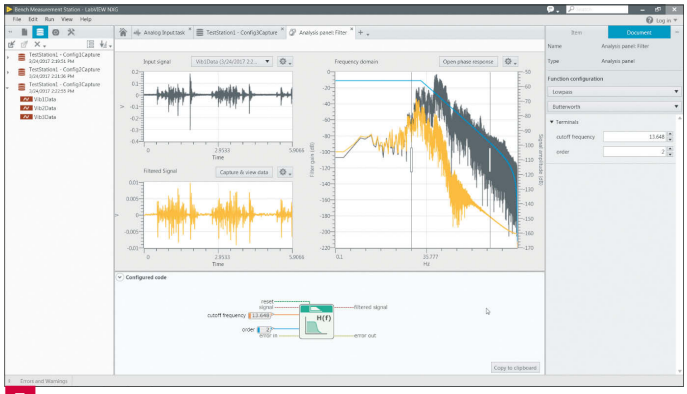
http://www.ni.com/fr-fr/shop/labview/labview-nxg.html



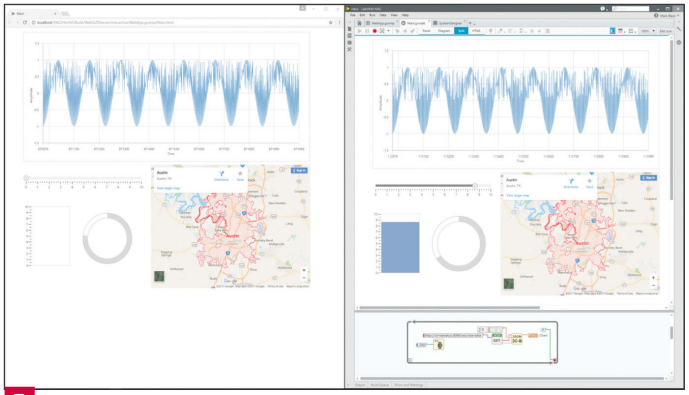
4 *Emploi de graphiques vectoriels pour des interfaces modernes et de haute résolution*



6 *Outil Application Builder pour créer des applications autonomes et des bibliothèques*



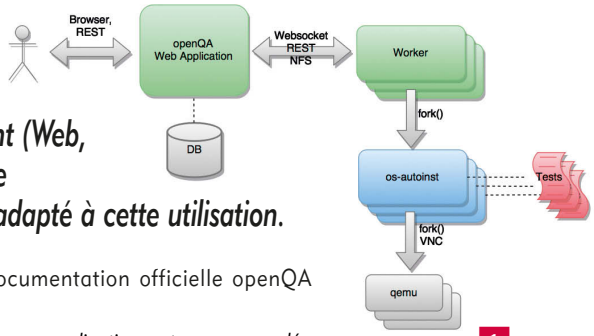
5 *Exemple de fenêtre d'analyse interactive et génération de VI associé*



7 *Face avant Web déployable sans programmation sur tous les navigateurs, sans aucun plugin*

Loïc Devulder,
Senior QA Engineer
chez **SUSE**

openQA, ou comment tester un système d'exploitation



1

Il existe des outils Open Source pour tester à-peu-près tout type de composant (Web, interface graphique ou ligne de commande), mais peu pour tester un système d'exploitation complet. Cet article se propose de présenter openQA, un outil adapté à cette utilisation.

Chaque éditeur de système d'exploitation (ou OS) a ses propres outils de qualité interne, mais ces outils sont souvent propriétaires (donc soumis à licence) ou mal adaptés (outils de QA "génériques" customisés). Bernhard M. Wiedemann, ingénieur SUSE n'aimant visiblement pas tout tester manuellement, développa, avec l'aide de Dominique Heidler, entre 2009 et 2010 un outil dans le but d'automatiser les tests de la distribution openSUSE (https://openqa.opensuse.org). openQA (http://open.qa) était né. Officiellement annoncé en 2011 et disponible sous licence GPL2, il n'a depuis cessé d'évoluer avec l'aide de nombreux contributeurs, et est même devenu le principal outil de QA de SUSE pour la distribution Linux entreprise SLE et ses dérivés.

openQA est également utilisé par la distribution Linux concurrente mais néanmoins amie Fedora (https://openqa.fedoraproject.org/), par Debian en test (http://openqa.debian.net/) et Gentoo l'étudie également de son côté (https://archives.gentoo.org/gentoo-dev/message/07dbdfb01366dfaee954924a3095bf6a). Même s'il a été pensé au départ pour Linux, openQA peut également tester d'autres OS comme Windows. A noter qu'openQA n'est pas le seul outil libre existant, AMD a également mis en 2011 sous licence Open Source son outil interne nommé Tapper (https://tapper.github.io/Tapper/), repris ensuite par Amazon mais dont le développement semble stagner depuis maintenant 2 ans. Taskotron (anciennement AutoQA), utilisé par Fedora, servait à tester l'OS avant openQA. Il est maintenant utilisé pour valider les packages RPM. On peut également citer Hydra (https://nixos.org/hydra/). Cette liste n'est pas exhaustive, il en existe certainement d'autres !

Prévu au départ pour du x86_64 et des tests mono-machine, openQA a beaucoup évolué au fil des années :

- support multi-architecture : x86_64, ppc64le, s390x et aarch64 ;
- test mono-machine ou multi-machine (pour les clusters HA ou HPC) grâce à l'utilisation de Open vSwitch ;
- utilisation de machine virtuelle (KVM, PowerVM, etc.) ou de machine physique (support de l'IPMI) ;
- possibilité de générer des images disques à partir des tests, utile pour rejouer manuellement les tests ou pour effectuer des tests chainés.

Alors (Michel !), comment ça marche ?

Comme vous l'aurez sans doute déjà tous compris, openQA sert à tester et valider un OS complet, de l'installation à l'utilisation des différents composants. Pour réaliser cela, openQA est architecturé autour de plusieurs composants, comme le résume le schéma suivant. 1

Traduction extraite de la documentation officielle openQA (http://open.qa/docs/) :

Le cœur du moteur de test est une application autonome appelée 'os-autoinst' (en bleu). Pour chaque exécution, cette application crée une machine virtuelle et l'utilise pour exécuter un ensemble de scripts de test (en rouge). 'os-autoinst' génère une vidéo, des captures d'écran et un fichier JSON avec les résultats détaillés.

D'autre part, 'openQA Web Application' (en vert) fournit une interface utilisateur Web et une infrastructure pour exécuter 'os-autoinst' de manière distribuée. L'interface Web fournit également une API de type REST basée sur JSON pour les scripts externes et pour une utilisation par le démon 'Worker'. Les 'Workers' récupèrent les données et les fichiers d'entrée pour 'os-autoinst' afin d'exécuter les tests. L'application 'openQA Web Application' se charge de distribuer les tâches parmi les 'Workers'. 'openQA Web Application' et les 'Workers' ne fonctionnent pas nécessairement sur la même machine mais peuvent être connectés via le réseau.

Après la définition des tests via l'interface Web, une machine virtuelle KVM instanciée par 'os-autoinst' se charge de les exécuter. Dans certains cas, cette machine virtuelle KVM peut être remplacée par un autre système de virtualisation (LPAR sur s390x, PowerVM sur ppc64le, etc.) ou une machine physique. Ce composant est appelé 'backend' dans la terminologie openQA. Une fois le test terminé, le résultat ainsi que les différents fichiers de logs sont remontés dans l'interface Web et peuvent donc être analysés. L'exécution du test peut également être suivie en temps réel mais également revue après grâce à une vidéo.

Mais comment développe-t-on les tests ?

Les tests, tout comme le moteur en lui-même, sont développés en langage Perl. Mais pas de panique pour ceux qui ne maîtrisent pas ce langage ! En effet, il n'est pas nécessaire d'être un expert Perl pour développer de nouveaux tests. Personnellement, je pratiquais assez peu ce langage avant de travailler sur openQA, et je n'ai pas connu de grosses difficultés pour coder des tests. Au fur et à mesure des années les développeurs openQA (à la fois les développeurs du produit en lui-même mais également les développeurs des tests - qui parfois sont les mêmes !) ont créé des bibliothèques de fonctions, ce qui fait que l'on utilise souvent les fonctions internes plutôt que du Perl "pur".

Exemple de test (disponible sur https://github.com/os-autoinst/os-autoinst-distri-opensuse) :

```
# SUSE's openQA tests
#
# Copyright (c) 2016-2018 SUSE LLC
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.

# Summary: Add node to existing cluster
# Maintainer: Loic Devulder <ldevulder@suse.com>

use base 'opensusebasetest';
use strict;
use testapi;
use lockapi;
use hacluster;

sub run {
  # Wait until cluster is initialized
  diag 'Wait until cluster is initialized...';
  barrier_wait("CLUSTER_INITIALIZED_$cluster_name");

  # Try to join the HA cluster through node HA_CLUSTER_JOIN
  assert_script_run 'ping -c1 ' . get_var('HA_CLUSTER_JOIN');
  type_string "ha-cluster-join -yc " . get_var('HA_CLUSTER_JOIN') . "\n";
  assert_screen 'ha-cluster-join-password';
  type_password;
  send_key 'ret';
  wait_still_screen;

  # Indicate that the other nodes have joined the cluster
  barrier_wait("NODE_JOINED_$cluster_name");

  # Do a check of the cluster with a screenshot
  save_state;
}

1;
# vim: set sw=4 et:
```

Mon test fonctionne, mais comment sont détectées les erreurs ?

Finalement la méthode utilisée est "assez" simple, openQA fait comme un utilisateur normal, c'est à dire qu'il peut interagir avec la VM via VNC ou une console série, et il peut "voir" les erreurs à l'écran grâce à l'utilisation de openCV et au concept de "needle".

Qu'est-ce qu'un "needle" ? Si l'on traduit littéralement il s'agit d'une aiguille, mais qu'est-ce que cela a à voir avec dans notre affaire me direz-vous? Eh bien un "needle" définit les zones de l'écran à détecter ainsi que des tags permettant de savoir quel "needle" doit être utilisé. Un "needle" est composé d'une copie d'écran au format PNG et d'un fichier JSON contenant les données qui sont la/les zone(s) à rechercher ainsi que les tags. Les fichiers PNG et JSON doivent obligatoirement avoir le même nom afin de pouvoir faire la correspondance entre l'image et les données. Les zones à rechercher peuvent être des zones d'inclusion ou d'exclusion.

Exemple de fichier JSON :

```
{
  "area": [
    {
      "xpos": 36,
      "ypos": 81,
      "width": 279,
      "height": 68,
      "type": "match"
    }
  ],
  "tags": [
    "yast2_ntp-client_synchronize_on_boot"
  ],
  "properties": []
}
```

Exemple d'un "needle" dans un test : 2

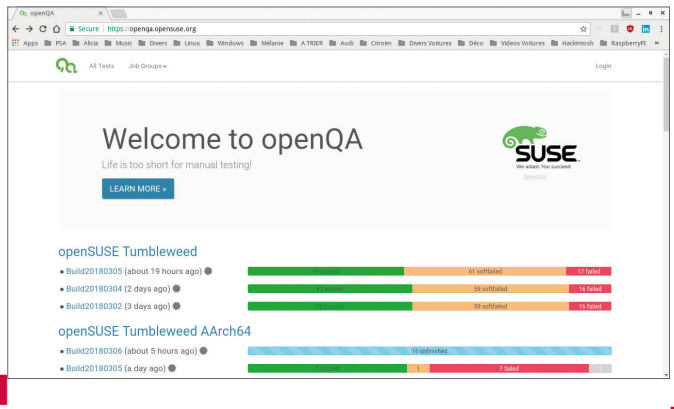
Pour les commandes retournant un code retour (généralement en mode "ligne de commande"), openQA fournit des fonctions permettant d'extraire le résultat renvoyé sur la console (standard ou série) : 3

openSUSE, un cas concret

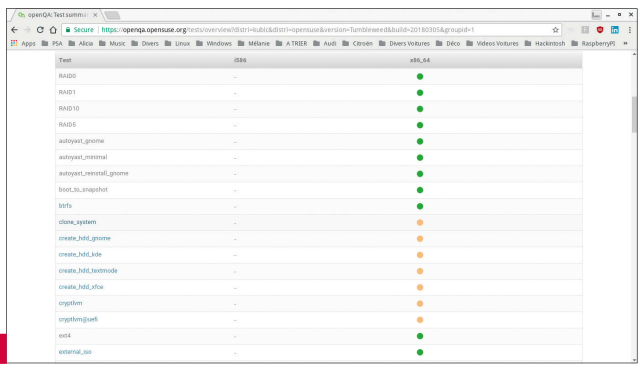
Bon la théorie c'est bien, mais voir concrètement comment ça fonctionne c'est mieux ! Pour cela, et grâce à la magie du Logiciel Libre, vous pouvez voir et analyser l'exécution des tests effectués pour la distribution Linux openSUSE.

Prenons en exemple Tumbleweed, la "rolling-release" de SUSE. Allons sur https://openqa.opensuse.org/ pour trouver la liste des derniers tests effectués. 4

Sélectionnez ensuite un numéro de version (ici Build20180305), vous verrez apparaître un écran résumant les différents tests qui ont été effectués. Vous pouvez trouver la liste des derniers build testés directement sur https://openqa.opensuse.org/group_overview/1. 5



2



3

Le code couleur est assez simple à comprendre :

- vert, tout s'est correctement déroulé ;
- orange, une ou plusieurs erreurs mineures sont survenues ;
- rouge, une ou plusieurs erreurs majeures/bloquantes sont survenues.

La notion d'erreur mineure/majeure est généralement définie par le développeur des tests. Par exemple, un OS qui ne "boot" pas est souvent une erreur majeure. A contrario, une erreur d'orthographe détectée sur un écran GNOME ou KDE sera considérée comme une erreur mineure.

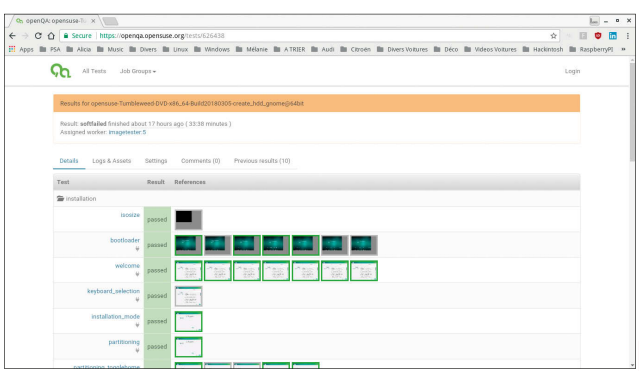
Mais revenons à notre exemple, si nous regardons de plus près le test nommé "create_hdd_gnome" en cliquant sur le rond orange à droite (en espérant qu'il soit vert ou orange quand vous lirez ces lignes !) vous pourrez voir les différents tests effectués ainsi que des copies d'écrans. 6

Si vous cliquez sur une copie d'écran vous pourrez voir plus d'informations sur le "needle", comme ses zones de recherches et ses tags. En cliquant sur un nom de test, par exemple "bootloader", vous pourrez voir son code source et ainsi étudier ce qui est fait. En revenant sur l'écran principal du test, vous pouvez également voir un menu "Log & Assets", c'est ici que vous trouverez les fichiers de logs, les images de l'OS (images ISO, QCOW2) ainsi que la vidéo du test.

D'accord, mais peut-on tester autre chose qu'un OS ?

Bien qu'initialement créé avec en tête l'objectif de tester un OS, openQA a également été pensé de façon modulaire afin de ne pas être lié à un système d'exploitation. Comme je l'ai déjà évoqué, vous pouvez tester Windows sans aucun souci. De la même façon, tester une application graphique ou en ligne de commande est également faisable. Par exemple, à l'instar du compilateur qui doit pouvoir se compiler lui-même, openQA est utilisé pour tester... openQA ! (https://openqa.opensuse.org/group_overview/24)

J'ai personnellement testé l'utilisation d'openQA pour autre chose qu'un composant OS. Dans le cadre des SUSE Expert Days 2018, nous avons mis en place avec deux collègues une démo montrant le processus d'intégration continue à l'aide des composants OBS (Open Build Service), CaaSP (Container as a Service Platform) et openQA. Mon travail de tous les jours consiste à tester les composants HA et SAP en partie à l'aide d'openQA, mais je n'avais encore jamais tenté de tester autre chose. Notre démo consistait en une application Android simple qui va effectuer une requête sur un Web Service hébergé sur CaaSP. Afin de montrer comment tester à la fois l'application Android et le Web Service j'ai réalisé un test "simple" exécutant l'émulateur Android dans une VM openSUSE. J'ai pu scripter toutes les actions effectuées sur l'application Android. openQA a donc ainsi permis de tester une application Android "simple". Certes, il existe certainement des outils mieux adaptés, mais cela a permis de montrer les possibilités de l'outil. La présentation ainsi que la démo sont disponibles sur BrightTalk. (https://www.brighttalk.com/webcast/11477/306839?utm_source=SUSE&utm_medium=brighttalk&utm_campaign=306839).



4

Pour aller plus loin

Cet article avait pour but de vous présenter succinctement l'outil openQA. Comme vous avez pu le voir malgré sa vocation première, tester un OS, openQA est tout à fait capable de tester d'autres applications. A partir du moment où une interaction clavier/souris/écran est possible, openQA peut le faire. Mais attention, ce n'est pas parce qu'il peut le faire que c'est l'outil le plus adapté ! Néanmoins il offre une grande flexibilité, et des capacités de reporting intéressantes et "sexy" :-). A vous de voir maintenant si cet outil répond à vos besoins !

Si certains se posent la question, openQA n'est pour le moment pas un produit commercial chez SUSE, c'est à dire qu'aucun support payant n'est proposé (mais cela peut changer !). Le support existant est celui de la communauté (principalement SUSE et Fedora) et les développeurs sont généralement assez réactifs.

Si cela vous a donné envie de l'essayer, voici quelques liens utiles vous permettant de tester et d'installer openQA. S'agissant d'un logiciel évoluant très rapidement, quelques erreurs dans la documentation peuvent apparaître, malgré le fait que ladite documentation soit mise à jour très fréquemment !

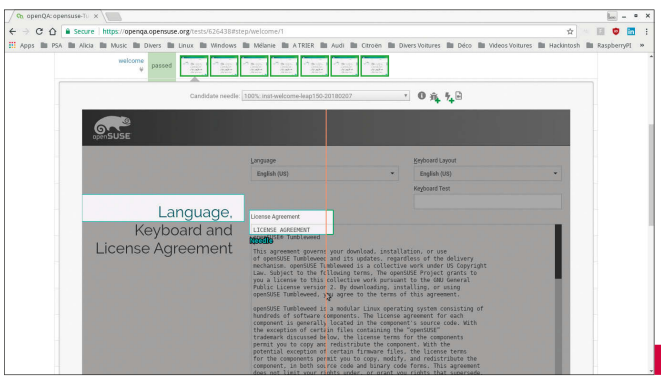
Installation : http://open.qa/downloads/ et https://github.com/os-autoinst/openQA/blob/master/docs/Installing.asciidoc

Documentation : http://open.qa/docs/

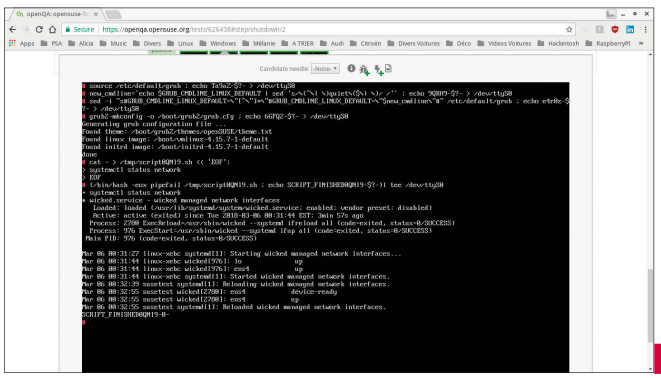
Dépôt GitHub : https://github.com/os-autoinst

Présentation vidéo SUSECON 2016 : https://www.youtube.com/watch?v=jrdCCefDfTM

C'est ici que s'arrête cette introduction à openQA. La prochaine partie mettra un peu plus les mains dans le cambouis avec un cas concret de test d'une application simple. D'ici là bonne lecture ! •



5

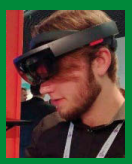


6


Julie Franel, *Software Engineering*, **Avanade**
julie.franel@avanade.com


Benoît Herbigniaux, *Software Engineering*, **Avanade**
benoit.herbigniaux@avanade.com


Stanislas Dolcini, *Software Engineering*, **Avanade**
stanislas.dolcini@avanade.com


Sébastien Maire, *Software Engineering*, **Avanade**
s.maire@avanade.com

avanade

Les moteurs 3D : outils indispensables à l'ère du numérique

Si les moteurs 3D existent depuis longtemps, avec leurs débuts dans les années 1990 et leur explosion dans les années 2000, ils sont en constante évolution. On peut affirmer qu'à l'ère de la réalité virtuelle et des images de synthèse, avoir des outils permettant de rendre des images 3D en temps réel et sans perte de performance est devenu indispensable en 2018.

Qu'il s'agisse des images de synthèse de nos films préférés ou de nos jeux vidéo favoris, qu'ils soient sur mobile, console ou web, nous sommes désormais tous sensibilisés, peut-être sans le savoir, à ce qu'est la 3D. Avec l'apparition récente de la réalité augmentée et de la réalité virtuelle, les outils possibles se sont multipliés, et choisir le moteur 3D le plus approprié pour une application devient dès lors un véritable travail de recherche.

Avec cette multiplication des moteurs, quelques grands noms se démarquent, comme Unity, Unreal Engine ou CryEngine : ce ne sont plus seulement des moteurs 3D mais des moteurs de jeu à part entière, ils incluent un moteur physique, un moteur de son, et bien plus encore.

Quelques définitions

Un **moteur 3D** est un composant logiciel proposant une couche haut niveau permettant au développeur de réaliser le rendu de scènes en 3D. Ces moteurs se basent sur des **bibliothèques bas niveau** qui leur fournissent la base de rendu ; parmi celles-ci, nous pouvons retrouver OpenGL, OpenGL ES, Direct3D ou encore plus récemment, Vulkan.

Ces bibliothèques se basent sur la technique dite de la « rastérisation » ; elle consiste à transformer le rendu de triangles 3D en triangles 2D pour ensuite les « rastériser », c'est-à-dire transformer ces triangles vectoriels en une forme matricielle (pixels) afin de permettre leur rendu sur un écran ou à l'impression.

Cette technique de rendu 3D est l'une des principales utilisées, mais elle n'est bien sûr pas la seule. Il est important de connaître les noms d'autres techniques, telles que le ray tracing ou encore le ray marching. Ces dernières ont d'autres avantages, mais un inconvénient important : l'absence de performance lors des calculs, ce qui est un point clé dans le développement de jeux vidéo ; en effet, si les scènes doivent mettre plusieurs heures à rendre, cela devient problématique.

Ces techniques restent tout de même très utilisées, notamment pour le rendu d'images de synthèse de très haute qualité pour les effets spéciaux et films d'animation, et de manière générale, pour tout ce qui n'a pas besoin d'être calculé en temps réel. Il est important de noter que ces techniques sont utilisées pour faire des rendus qui ne sont pas réalisables avec la technique de la rastérisation, comme par exemple le rendu de nuages volumétriques ou encore de fractales.

Lors du développement d'un jeu vidéo 3D, des modèles 3D sont utilisés ; il s'agit de fichiers contenant les milliers de triangles composant notre objet final. Ces modèles, souvent appelés "mesh", peuvent être stockés dans différents formats, chacun ayant sa spécificité, telle que la gestion de l'animation ou encore des formules paramétriques.

Ces modèles sont souvent associés à un matériau : c'est l'ensemble des textures qui vont être utilisées pour que l'apparence et l'interaction de la lumière correspondent aux choix de l'artiste. Un **matériau** est spécifique à un **shader**.

Un shader est un petit programme qui s'exécute sur la carte graphique, et qui est responsable des calculs de lumières et d'ombres. Différents langages existent pour programmer ces shaders, chacun étant dépendant de la bibliothèque bas niveau utilisée.

Au-delà d'un simple moteur 3D, un moteur de jeu se caractérise par la présence de tous ces éléments, mais également par l'intégration d'un moteur physique, d'un moteur son et de tout autre moteur permettant au développeur d'enrichir son application sans avoir à se soucier des dépendances. Le moteur physique est une des parties centrales d'un moteur de jeu ; c'est lui qui va réaliser les calculs des collisions entre les objets, intégrer la gravité, et ainsi permettre la simulation d'un environnement réaliste. Le moteur son, lui, va gérer toute la partie audio du jeu.

Choisir son moteur

Les moteurs 3D se divisent en plusieurs catégories. Les moteurs de jeu sont légion, mais on y trouve aussi des moteurs de rendu 3D et des moteurs intermédiaires qui proposent quelques outils supplémentaires au rendu 3D.

Les critères de sélection

Pour choisir le moteur le plus approprié à son application, il faut bien analyser la situation et se poser les bonnes questions. Veut-on réaliser un jeu, une application interactive ? Quelle est la cible de notre application ? Quels sont les moyens que l'on peut et veut y mettre ?

Le coût est un des principaux critères. Tous les moteurs 3D ne sont pas gratuits ; certains ne le sont qu'en partie, et selon les revenus générés par la future application, il pourra être demandé de reverser une partie des bénéfices en paiement ou de souscrire à un abonnement. Il faut donc, avant de porter son choix sur un moteur,

se renseigner sur son coût et sa licence. Il faut également envisager le coût du développeur : à moins d'être soi-même développeur et de travailler sur son temps libre, ou de réussir à trouver et convaincre un développeur de travailler bénévolement pour une bonne cause, le développement aura un coût qu'il faut intégrer à son étude de marché avant de se lancer dans le développement de l'application.

Les ressources graphiques et textures sont également importantes, non seulement en termes de critères de choix mais également en termes de coût. Certains moteurs disposent d'une boutique de ressources ; mais pour d'autres, il vous faudra fournir les vôtres, qu'elles soient libres d'utilisation ou payantes. Le choix du moteur est donc impacté par votre capacité à rassembler, ou non, les ressources nécessaires. Les plateformes à supporter sont le deuxième critère principal de choix. La future application doit-elle être portée uniquement sur mobile, donc iOS et Android ? Ou doit-elle également fonctionner sur le web ? Définir la cible du produit, et donc les plateformes supportées, en amont du choix du moteur, est très important car il est très difficile de revenir en arrière une fois le développement commencé.

Enfin, le dernier critère concerne, évidemment, le langage de programmation utilisé. Il s'agit ici plus d'une question de rapidité de développement et de confort : le développeur sera bien plus rapide s'il utilise un langage avec lequel il est déjà familiarisé et a des affinités ; *a contrario*, s'il a besoin de se former au langage avant, ou s'il n'est pas à l'aise avec, le temps de développement en sera allongé. Cela étant, tous les moteurs ne permettent pas nécessairement d'utiliser n'importe quel langage, il faut donc parfois faire des concessions et en tenir compte pour fixer les jalons importants du planning de développement.

Les principaux moteurs du marché

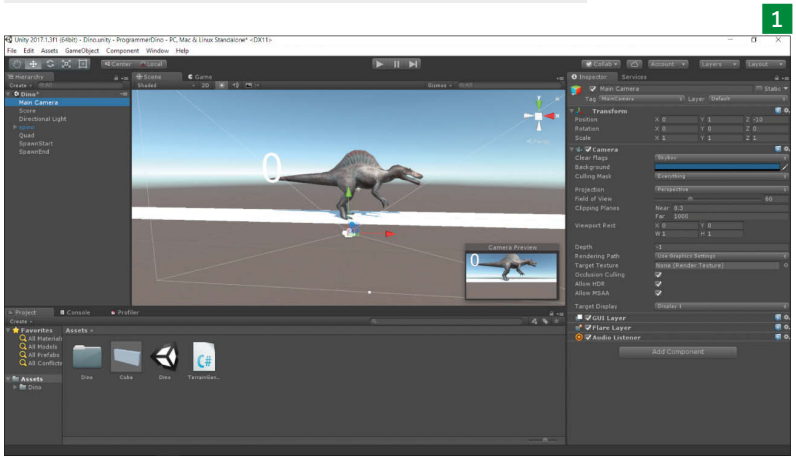
Comme il existe un certain nombre de moteurs 3D, ce dossier se concentrera sur les principaux connus et utilisés, à savoir : Unity, Unreal Engine et CryEngine pour les moteurs de jeu lourds ; Three.js et Babylon.js pour les moteurs 3D JavaScript ; et OGRE et Irrlicht pour les moteurs de rendu 3D plus légers.

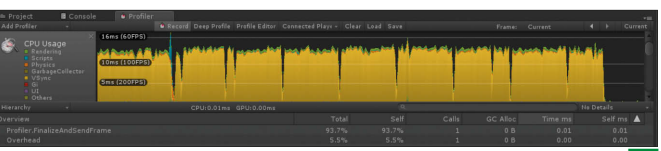
Unity 1

Créé en 2005 par Unity Technologies, Unity est l'un des moteurs de jeu les plus répandus et les plus populaires. Il dispose d'une grande communauté, de nombreux tutoriels et fonctionne sur des ordinateurs peu puissants, ce qui le rend très accessible ; via la gratuité de sa licence la plus basique et sa boutique de ressources très fournie, il est très plébiscité par les indépendants et les débutants. Ici, un exemple de jeu simple connu par nombre de personnes : le célèbre jeu du dinosaure de Google Chrome quand une page ne se charge pas. Une des spécificités d'Unity est son mécanisme de coroutine (la fonction SpawnWalls() dans l'exemple ci-dessous), qui permet de faire de la gestion asynchrone dans un contexte monothread. A noter que depuis les récentes versions d'Unity (2017) assurant une compatibilité avec le framework .NET 4.6.1, il est dorénavant possible de transformer ce code en asynchrone via *async* et *await*.

```
IEnumerator SpawnWalls()
  {
    while (true)
    {
      var wall = GetAvailableWall();
      wall.transform.position = new Vector3(StartPos.transform.position.x, wall.transform.
position.y, wall.transform.position.z);
      wall.SetActive(true);
      _currentWalls.Add(wall);
      yield return new WaitForSeconds(2.0f);
    }
  }

  // Update is called once per frame
  void Update()
  {
    if (_isAlive)
    {
      if (Input.GetKey(KeyCode.Space) && IsGrounded && !_isJumping)
      {
        _rigidbody.AddForce(Vector3.up * JumpForce, ForceMode.VelocityChange);
        _isJumping = true;
      }
      List<GameObject> toRemove = new List<GameObject>();
      foreach (var wall in _currentWalls)
      {
        wall.transform.position += _wallsMoveVector * Time.deltaTime * WallSpeed;
        if (wall.transform.position.x < EndPos.transform.position.x)
        {
          wall.SetActive(false);
          _wallsPool.Add(wall);
          toRemove.Add(wall);
        }
      }
      foreach (var wall in toRemove)
        _currentWalls.Remove(wall);
    }
    else if (Input.anyKey)
      StartGame();
  }
```

1



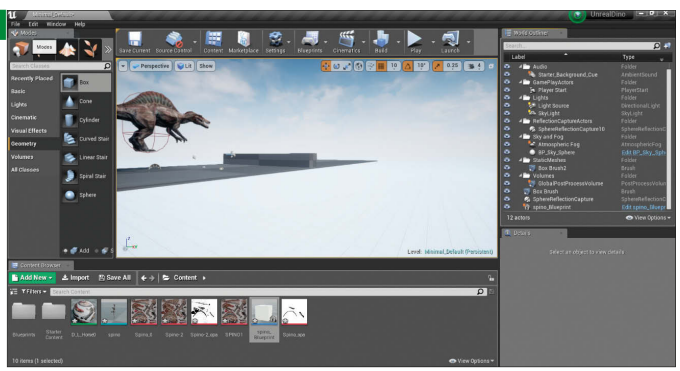


2

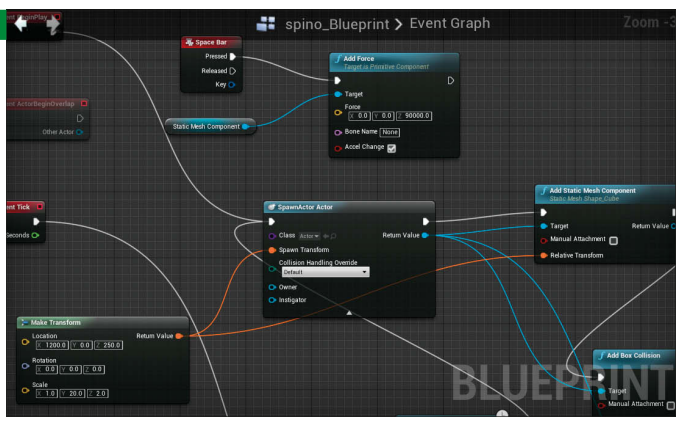
Le dépôt du projet est accessible depuis le lien suivant :
https://github.com/AvanadeProgrammez216/DinoUnity/. 2
Le profiler permet de voir où sont les goulots d'étranglement du mini-jeu. Ici, le jeu étant très simple, les performances sont plutôt bonnes sur la machine de test. Le profiler Unity, à la différence de beaucoup d'autres, permet d'afficher les temps par fonction en temps réel, il est donc facile de reproduire une situation qui provoque un freeze dans le jeu et de le constater sur le graphique du profiler. Les différentes documentations Unity aident ensuite à optimiser les parties identifiées grâce à cet outil.
Néanmoins, pour profiter de ces fonctionnalités il faudra tout de même prendre une licence payante ; dans le cas contraire, le profiler n'est pas fourni.

Critères	Unity
Type de moteur	Moteur de jeu 3D
Prix	4 licences dont une gratuite (Personal) Au-delà de 100 000 $ de revenus par an, obligation de passer à une licence payante
Langages	C#, JavaScript (voué à disparaître), C++ (présent dans des librairies natives)
Plateformes	Windows, OS X, Linux Xbox 360, Xbox One, Wii U, New 3DS, Nintendo Switch, PlayStation 4, PlayStation Vita, Windows Phone, iOS, Android, BlackBerry 10, Tizen, Unity Web Player, Windows Store, WebGL, Oculus Rift, Gear VR, Android TV, Samsung Smart TV
Projets notables	Hearthstone de Blizzard, Lara Croft : Relic Run de Square Enix
Store intégré	Oui
Editeur fourni	Oui
Mises à jour	Fréquentes (2017.3.0, 19 décembre 2017 / 2017.2.1,12 décembre 2017 / 2017.2.0, 12 octobre 2017) + patches

3



4

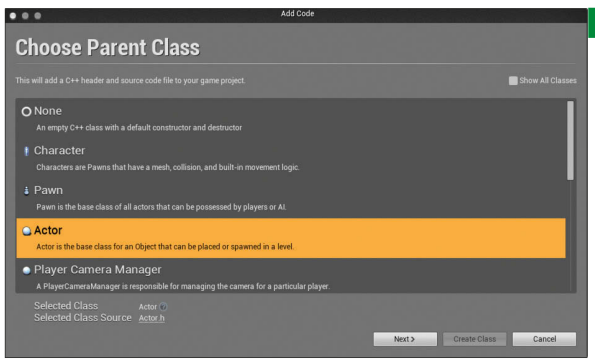


Unreal Engine 3

Principal concurrent de Unity avec le CryEngine, Unreal Engine est un moteur de jeu multiplateforme reconnu pour ses performances, notamment en termes de graphismes. Créé en 1998 par Epic Games, il a évolué jusqu'à atteindre sa version actuelle, la 4, laquelle apporte un changement de taille : jusqu'à la version 3 incluse, le langage utilisé était le UnrealScript, mais en raison de sa lenteur, il fut abandonné au profit du C++. Epic Games permet par ailleurs l'accès au code source du moteur sur leur Github privé, pour peu que l'on en fasse la demande. Avec le récent développement de sa communauté et de ses tutoriels, Unreal Engine est devenu plus accessible qu'il y a quelques années. Le moteur met par ailleurs à disposition Blueprint, un système de développement par Visual Scripting ; en d'autres termes, pas ou peu besoin de savoir développer, il est tout à fait possible d'utiliser pleinement les capacités du moteur au travers de ce système. 4
Cela présente de nombreux avantages, outre la facilité à mettre en place des systèmes, cela permet d'éviter des soucis lorsque l'on écrit du code à la main. On notera aussi que le moteur dispose de nombreux templates de scènes qui permettent de faire un jeu 2D en quelques clics. 5
Même lors de l'écriture de code en C++, le moteur propose un gestionnaire de template, qui permet de générer le code suivant :

```
#include "GameFramework/Actor.h"
#include "MyActor.generated.h"

UCLASS()
class AMyActor : public AActor
{
  GENERATED_BODY()

public:
  // Sets default values for this actor's properties
  AMyActor();

  // Called every frame
  virtual void Tick( float DeltaSeconds ) override;

  UPROPERTY(EditAnywhere, BlueprintReadWrite, Category="Damage")
  int32 TotalDamage;

  float DamagePerSecond;
protected:
  // Called when the game starts or when spawned
```

5

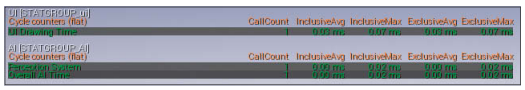


```
    virtual void BeginPlay() override;
};
```

On peut ainsi éditer les propriétés de l'objet directement dans l'éditeur, au travers des menus comme ci-dessous :

Une fois la classe créée, il est possible de l'exporter en tant qu'objet préfabriqué pour pouvoir la réutiliser dans de multiples projets. Les AActors sont la classe de base pour n'importe quel objet du jeu.

Comme sur Unity, les performances sont visibles sur un profiler. Il se présente sous forme d'une console et permet de visualiser les statistiques des ressources du jeu et de leur utilisation. L'exemple ci-dessous montre donc l'utilisation de la mémoire de l'interface utilisateur dans le jeu :



Critères	Unreal Engine
Type de moteur	Moteur de jeu 3D
Prix	5% du revenu brut à partir de 3000 $ de revenus par trimestre
Langage	C++, C# (support incomplet)
Plateformes	Windows PC, PlayStation 4, Xbox One, Nintendo Switch, OS X, iOS, Android, VR (incluant SteamVR/HTC Vive, Oculus Rift, PlayStation VR, Google VR/Daydream, OSVR and Samsung Gear VR), Linux, SteamOS, HTML5.
Projets notables	Final Fantasy VII (remake) de Square Enix, Tekken 7 de Bandai Namco Games
Store intégré	Oui
Éditeur fourni	Oui
Mises à jour	Trimestrielles (4.18 23 octobre 2017)

CryEngine 6

Dernier challenger sur le podium, CryEngine est un moteur de jeu développé par CryTek depuis 2005. Il est devenu libre de prix – c'est-à-dire que les utilisateurs choisissent la somme qu'ils veulent payer – et depuis la version 5, s'oriente de plus en plus vers la réalité virtuelle. Ce moteur offre beaucoup de possibilité et est puissant mais il est difficile à prendre en main par rapport aux autres moteurs. Il est accessible pour des développeurs déjà familiers avec le fonctionnement des moteurs de jeu et du développement 3D.

À noter que le moteur de jeu Amazon Lumberyard est un dérivé de CryEngine, suite à son rachat par Amazon en 2015.

Bien qu'on puisse scripter en C++, C# ou encore en Lua, CryEngine est fourni avec un éditeur de Visual Scripting semblable au Blueprint d'Unreal. Ces scripts sont appelés des "flowgraphs" et permettent de gérer les interactions entre objets, le déclenchement d'évènements et tout le level design. L'exemple ci-dessous, en montre l'utilisation. 7

Du côté du CryEngine, le profiler est embarqué dans le champ de vision du jeu en cours de développement. Beaucoup d'informations sont disponibles, aussi bien au niveau du rendering que des temps d'exécution processeur. CryEngine possède en effet plus d'une dizaine d'outils de profiling et de debugging, ce qui ravira les développeurs pointilleux visant à délivrer un jeu rapide et robuste. 8

Comme tout profiler moderne, celui de CryEngine propose d'évaluer les temps d'exécution par fonction ; ce type de profiler est indispensable pour identifier les causes de mauvaises performances ou encore les problèmes de stabilité de framerate pendant le jeu.

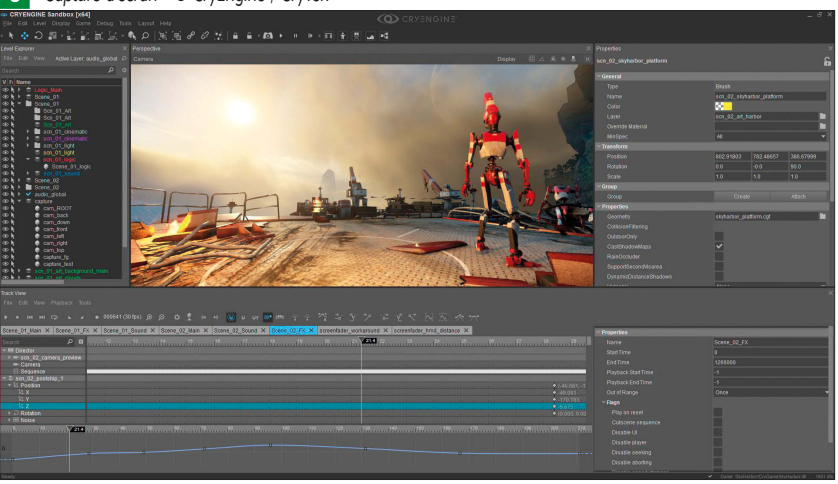
Critères	CryEngine
Type de moteur	Moteur de jeu 3D
Prix	Gratuit
Langage	C++, C#, Lua (déprécié)
Plateformes	Windows, Linux, PlayStation 4, Xbox One, Oculus Rift, OSVR, PSVR and HTC Vive
Projets notables	Crysis, Aion de NCSOFT, Far Cry de Ubisoft
Store intégré	Oui
Éditeur fourni	Oui
Mises à jour	Trimestrielles (5.4 21 septembre 2017)

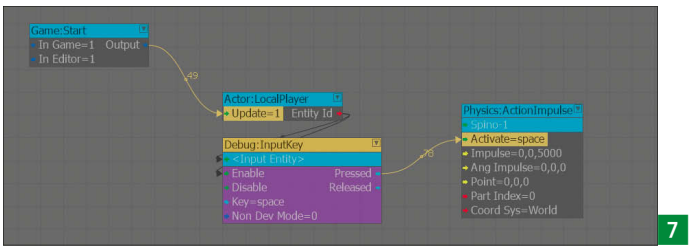
Three.js 9

Projet démarré en avril 2009, Three.js est une bibliothèque JavaScript open-source, qui fut à l'origine développée en ActionScript puis portée sous JavaScript. Elle utilise l'API WebGL, elle-même basée sur le standard OpenGL ES. Sa légèreté et son fonctionnement sur navigateur en font un outil de choix pour des projets web ou mobiles de petite ou moyenne envergure.

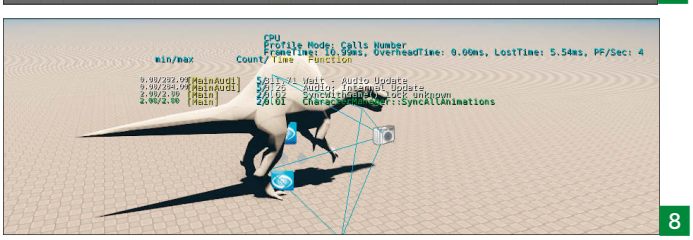
Lors de l'export web, le code généré par Unity est basé sur le moteur 3D de Three.js ; Three.js ne possédant pas de moteur

6 *Capture d'écran - © CryEngine / CryTek*





7



8

physique, il faut utiliser un module complémentaire, comme par exemple Physijs.

Au niveau de la gestion des performances, ces dernières sont directement gérées dans le profiler du navigateur ou bien via Three.js Inspector.

Le dépôt du projet d'exemple est accessible depuis le lien suivant : https://github.com/AvanadeProgrammez216/ThreejsDino

Critères	Three.js
Type de moteur	Moteur 3D
Prix	Gratuit sous licence MIT
Langage	JavaScript
Plateformes	HTML5, Windows, Linux, OSX, iOS, Android
Projets notables	Let's Play de Ouigo, React-VR de Facebook
Store intégré	Non
Éditeur fourni	Oui, en ligne : https://threejs.org/editor/
Mises à jour	Trimestrielles (r89 18 décembre 2017)

Babylon.js 10

Babylon.js est un moteur 3D temps réel open-source et multiplateforme, qui utilise lui aussi WebGL. Décrit comme moteur de jeu, il inclut en effet un moteur physique, oimo.js, ainsi que l'utilisation de l'API Web Audio pour le son.

Développé depuis 2013 par des employés de Microsoft, il a su se faire une place très rapidement au sein des moteurs 3D les plus plébiscités ces dernières années, en raison de sa légèreté et de ses performances.

Babylon.js et Three.js partagent la même approche du développement 3D. Cependant, si Babylon.js dispose d'un éditeur avec autocomplétion, son interface utilisateur est inexistante ; il faut donc avoir des notions de JavaScript et de développement 3D pour pouvoir développer rapidement et efficacement.

Ici, l'implémentation du mini-jeu d'exemple a été réalisée avec Babylon.js Playground, un éditeur en ligne qui permet de tester son code Babylon.js en ligne, de le sauvegarder et de le partager.

Le dépôt du projet est accessible via le lien suivant :

https://github.com/AvanadeProgrammez216/BabylonDino

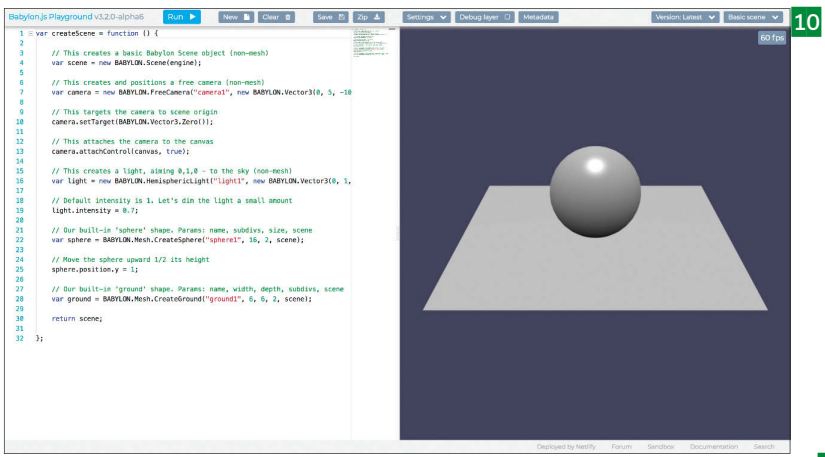
A la différence de Three.js, Babylon.js, bien qu'étant un moteur de rendu, intègre une couche de gestion de la physique. Dans ce cas précis, la gestion des collisions se fait de manière native et de même pour la gravité. Seul bémol, à la différence des autres moteurs comme CryEngine ou Unity, il n'existe pas de routine appelée à chaque collision sur un objet ; pour ceci, il faudra vérifier explicitement la collision entre les objets. 11

Pour les performances, il est possible d'utiliser Spector.js, un profiler WebGL, pour la partie rendering, ainsi que le profiler du navigateur pour le reste du programme. Spector.js peut être utilisé en tant qu'extension de navigateur (Chrome et Firefox), en tant que module ou encore en tant que script. A noter que Spector.js n'est pas spécifique à Babylon.js, il est possible de l'utiliser pour toute application utilisant WebGL.

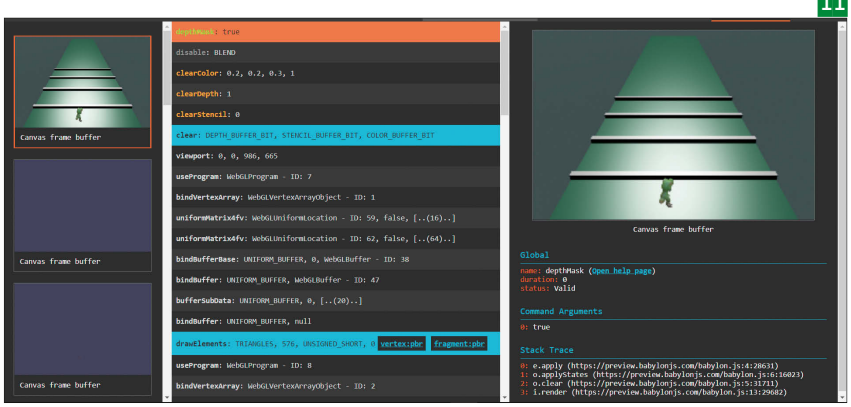
Critères	Babylon.js
Type de moteur	Moteur de jeu 3D
Prix	Gratuit sous licence Apache 2.0
Langage	JavaScript
Plateformes	IE11/MS Edge, Chrome, Firefox, Opera, Safari, iOS (iPad/iPhone), Android, Windows Phone 8.1/ Mobile 10, Firefox OS, Xbox One
Projets notables	XBOX Avatars et XBOX Design Lab de XBOX
Store intégré	Non
Éditeur fourni	Oui, en ligne : https://playground.babylonjs.com
Mises à jour	Semestrielles (3.1.0 13 décembre 2017)

OGRE 12

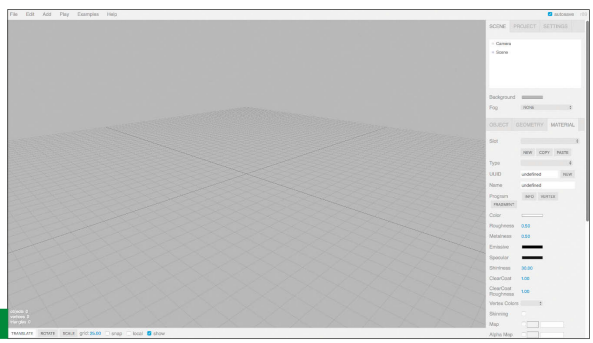
OGRE est l'acronyme de *Object-Oriented Graphics Rendering Engine*. Il s'agit d'un moteur de rendu 3D temps réel – c'est-à-dire qu'il se contente exclusivement de rendre des scènes et ce, juste



10



11


9

avant de les afficher. Toutefois, avoir un moteur de rendu détaché du reste permet aux développeurs de choisir les autres modules nécessaires à la réalisation d'un jeu.

OGRE est multiplateforme et il abstrait les bibliothèques bas niveau Direct3D et OpenGL, ce qui en fait un moteur à l'épreuve de notre époque où la compatibilité est un critère important. Il est par ailleurs open-source, sous licence MIT, on peut donc retrouver son code source sur Github.

OGRE gère la programmation de shaders en langage Cg, GLSL et HLSH et supporte tous les formats standards d'image utilisés dans les jeux vidéo (PNG, JPEG, BMP, TGA et DDS). Il est également équipé d'un moteur d'animation supportant les animations par pose et par morphing.

Il est aussi possible d'utiliser OGRE avec la plupart des éditeurs, puisque celui-ci est compatible avec MSVC (le compilateur de Microsoft), GCC et CLang.

Comme la plupart des moteurs graphiques de jeu, Ogre ne fait pas exception et propose un système de scènes qui permet de créer des niveaux. Une fois la configuration QT et CMake terminée pour l'affichage du jeu, il est possible de faire apparaître le premier modèle 3D dans l'interface et ainsi attaquer le vif du sujet.

```
void TutorialApplication::createScene(void)
{
// Set the scene's ambient light
mSceneMgr->setAmbientLight(Ogre::ColourValue(0.5f, 0.5f, 0.5f));
// Create an Entity
Ogre::Entity* ogreHead = mSceneMgr->createEntity("Head", "ogrehead.mesh");

// Create a SceneNode and attach the Entity to it
Ogre::SceneNode* headNode = mSceneMgr->getRootSceneNode()->createChildSceneNode
("HeadNode");
headNode->attachObject(ogreHead);

// Create a Light and set its position
Ogre::Light* light = mSceneMgr->createLight("MainLight");
light->setPosition(20.0f, 80.0f, 50.0f);
}
```

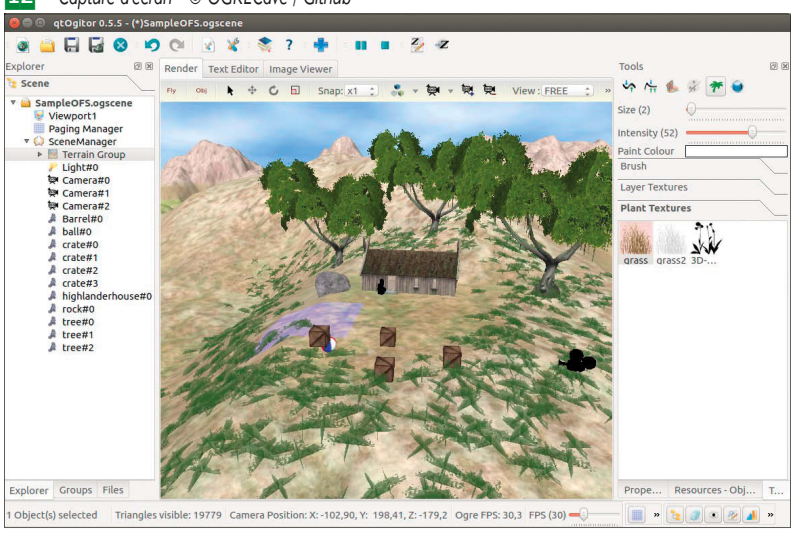
Critères	OGRE
Type de moteur	Moteur de rendu 3D
Prix	Gratuit sous licence MIT ou LGPL en fonction de la version, une licence OGRE "Unrestricted" est également disponible en version payante en cas de besoin d'éviter les conditions de la licence LGPL.
Langages	C++, C#, Python, Java, Lua, Ruby, Eiffel
Plateformes	Linux, Window, OSX, Windows Phone 8, iOS, Android
Projets notables	Pacific Storm, Ankh : Une aventure égyptienne
Store intégré	Non
Editeur fourni	Ogitor (officiel), mais plusieurs autres éditeurs sont disponibles et développés par la communauté (OGRE Studio, XgEditor)
Mises à jour	Peu fréquentes (1.10 23 Avril 2017, 2.1 9 février 2015)

Irrlicht 13

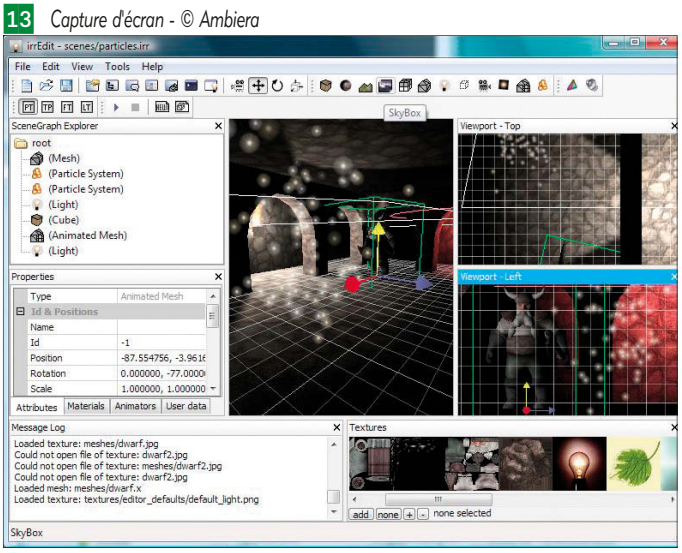
Le projet Irrlicht a débuté en 2002 et est maintenu par des développeurs indépendants. Irrlicht est un moteur temps réel open-source écrit en C++, multiplateforme également, qui abstrait OpenGL et DirectX 8, 9 et 11. Moins complet qu'OGRE, il sera plutôt utilisé pour des projets de moins grande envergure.

Comme les autres moteurs, Irrlicht offre la possibilité de rajouter des matériaux en programmant des shaders sur mesure que ce soit en HLSL ou en GLSL. Irrlicht est capable de faire des rendus d'eau animée, et supporte les formats standards d'animation par squelette. Bien qu'Irrlicht soit décrit comme un moteur de jeu, celui-ci est dépourvu d'un moteur son et d'un moteur physique. Pour intégrer de tels moteurs, il faudra se tourner du côté de PhysX ou Bullet pour la physique, et IrrKlang, OpenAL Fmod ou encore SFML-audio pour le son. Ici, IrrlichtLime, le wrapper .NET du moteur, a été utilisé. Le code ci-dessous affiche un des modèles d'exemple fournis avec IrrlichtLime et permet de faire sauter le personnage. Le moteur expose simplement les différentes fonctions de rendu d'interface et de scène qu'il suffit d'appeler dans une boucle, tout en prenant en compte le temps écoulé entre chaque frame pour garder un framerate constant.

12 *Capture d'écran - © OGRECave / Github*



13 *Capture d'écran - © Ambiera*



```
        while (device.Run())
        {
            // As the game is simple we will handle our simple physics ourselves
            uint now = device.Timer.RealTime;
            uint elapsed = now - then;
            float elapsedTimeSec = (float)elapsed / 1000.0f;
            then = now;

            if (elapsed < 17) // we target 58.8 FPS
                device.Sleep(17 - (int)elapsed);

            _playerVerticalSpeed += elapsedTimeSec * -(VerticalGravity * 10.0f);

            var calculatedNewPos = sydneyNode.Position - new Vector3Df(0.0f, _playerVertical
Speed * elapsedTimeSec, 0.0f);
            float offsetSydney = sydneyNode.BoundingBox.Extent.Y / 2.0f;

            _isGrounded = calculatedNewPos.Y <= (GroundAltitude + offsetSydney);
            if (_isGrounded)
            {
                _playerVerticalSpeed = 0.0f;
                calculatedNewPos.Y = GroundAltitude + offsetSydney;
            }
            else
                calculatedNewPos.Y = calculatedNewPos.Y;
            sydneyNode.Position = calculatedNewPos;

            driver.BeginScene(true, true, new Color(100, 101, 140));
            smgr.DrawAll();
            gui.DrawAll();
            driver.EndScene();
        }

        device.Drop();
```

Le dépôt du projet est accessible via le lien suivant :
https://github.com/AvanadeProgrammez216/IrrlichtDino

Irrlicht ne possède pas de profiler intégré contrairement aux autres moteurs. Il faut donc utiliser le profiler intégré à l'IDE utilisé pour analyser les performances.

Critères	Irrlicht
Type de moteur	Moteur de rendu 3D
Prix	Gratuit sous licence zlib
Langages	C++, C# (Irrlicht Lime)
Plateformes	Windows, Mac OS, Linux
Projets notables	Worlds
Store intégré	Non
Editeur fourni	IrrEditor
Mises à jour	1.8.4, 9 juillet 2016

Conclusion

Vous l'aurez constaté, il existe de nombreuses technologies de rendu 3D mobile et web, et faire son choix est donc assez difficile, surtout pour un non-initié ; les principaux critères qui permettent de choisir son moteur sont donc à étudier avec soin, car il est ardu de revenir en arrière.

Cependant, la prolifération et l'évolution constante de ces technologies les rendent de plus en plus accessibles, à un plus grand nombre de personnes : les outils sont beaucoup plus complets, certains sont même devenus de véritables moteurs de jeu plus que de simples moteurs 3D. De plus, les appareils gagnent largement en performance d'année en année, ce qui permet aujourd'hui de se diriger vers des moteurs plus lourds.

Par ailleurs, les moteurs 3D sont de plus en plus utilisés pour la réalité virtuelle et la réalité augmentée ; on peut citer les jeux mobiles Pokémon GO et Ingress, mais aussi des applications de tous les jours, telles que le scan de QR Code, les vidéos en 360°, etc. La réalité augmentée et la réalité virtuelle sur mobile sont très bien supportées ; quant au web, on peut noter que WebVR, bien qu'elle n'inclue pas encore les technologies de réalité augmentée, va très bientôt être supportée et elle promet un avenir très intéressant pour ce type d'application.

•

Michaël Bertocchi
Ingénieur développement
@dupot_org

Développer pour le mobile avec une technologie multiplateforme : développer avec Qt Framework

Partie 2

Lorsque vous souhaitez développer une application mobile multiplateforme performante, vous n'avez que deux choix : Xamarin ou Qt, nous allons découvrir l'environnement de travail avec son IDE phare : Qt Creator.

Pré-requis

Pour développer avec ce framework, vous aurez besoin d'un ordinateur tournant sur processeur x86. Qt et son IDE QtCreator sont disponibles sous Windows, macOS et GNU/Linux.

Installation

Je ne m'attarderai pas ici sur les détails d'installation, je vous présente les grandes lignes :

- allez sur le site de Qt https://www.qt.io/download-qt-for-application-development puis téléchargez l'installeur de Qt Creator (le site vous proposera celui compatible avec votre OS) ;
- ensuite sur le site https://www.java.com/fr/download/faq/develop.xml pour télécharger le JDK* ;
- puis sur le site d'Android pour télécharger le SDK (scrollez tout en bas pour les SDK**) https://developer.android.com/studio/index.html ;
- enfin sur le site d'Android pour le NDK*** https://developer.android.com/ndk/index.html.

Pour information, Qt compile en C++, ce qui n'est nativement pas le langage d'Android (Java pour rappel), mais les développeurs de Mountain View ont prévu ce cas et proposent justement NDK pour permettre à une application Java de pouvoir communiquer avec du code compilé en C/C++. Le JDK et le SDK sont bien sûr prévus pour pouvoir générer l'application Java principale (qui appelle le code C++ compilée grâce au NDK).

**JDK: Java Development Kit*
***SDK: android Software Development Kit*
****NDK : android Native Development Kit*

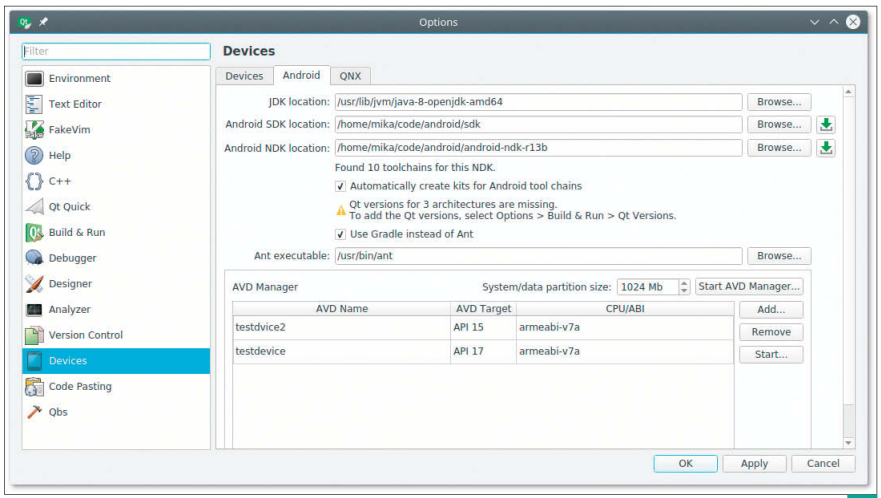
Paramétrage de l'IDE Qt Creator

Lorsque vous lancerez Qt Creator pour la première fois, il vous faudra paramétrer plusieurs choses: [figure 1]

- le JDK Java ;
- le SDK Android ;
- le NDK Android ;
- Ant (outil qui permet l'orchestration des générations, un équivalent de make en Java) ;
- créer au moins un AVD (Android Virtual Device: mobile virtuel). 1

Le choix d'un type de projet

Qt peut être utilisé pour développer différents types d'application, ainsi comme n'importe quel autre IDE, celui-ci va vous demander dès le départ de choisir quel type de projet afin de vous créer/configurer votre solution pour vous économiser du temps. A la création d'un projet, l'outil laisse le choix entre plusieurs templates organisés par catégorie :



1

- application ;
- librairie/bibliothèque ;
- autre projet ;
- non Qt Projet ;

Ce qui nous intéresse ici c'est la catégorie « Application ».
Celle-ci se décompose également en plusieurs sous choix :

- Qt Widget Application ;
- Qt Console Application ;
- Qt Quick Application ;
- Qt Quick Controls (2) Application ;
- Qt Canvas 3D Application.

Vous avez en effet le choix d'une part de créer des projets avec ou sans utiliser les bibliothèques Quick (QML) : par exemple pour un batch (Qt Console Application), un jeu 3D avec le template dédié ou une application sans utiliser QML.
Ou de l'autre de développer en utilisant la richesse du QML, ce qui est notre cas ici.
Il y a principalement deux alternatives : Quick et Quick Controls (1 et 2).
Le choix entre ces deux librairies permet juste d'avoir une entrée en matière différente mais vous pouvez créer un projet quick et utiliser des Quick controls dans des fichiers QML par la suite sans soucis c'est très flexible.

Parenthèse Qt Quick / Qt Quick Controls

Pour simplifier : Qt Quick c'est la base du QML, la boîte à outils pour démarrer, et Quick Controls vous propose des éléments/composants qui ont été écrit avec.

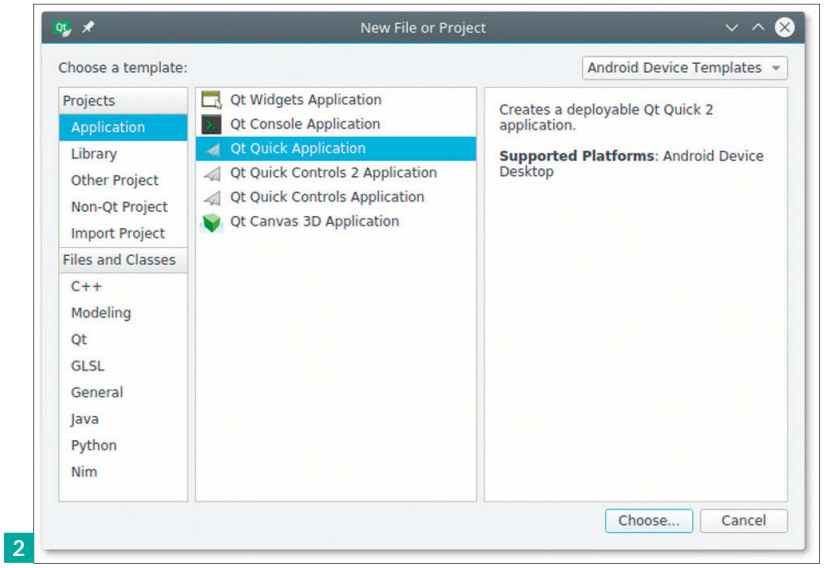
En Quick vous allez avoir la main sur les composants racines comme des rectangles, des zones actives… et en Quick Controls vous pourrez utiliser des boutons, menus, zones de texte…

Les deux ne s'opposent pas, ça dépend de votre besoin dans vos développements. Notez également que vous pouvez justement créer vos propres boutons (répondant au style de votre application), menus, zones de texte personnalisés si besoin et les réutiliser très simplement. Pour la différence entre Qt Quick Controls 1 et 2, vous pouvez trouver un tableau comparatif ici : https://doc.qt.io/qt-5.10/qt-quickcontrols2-differences.html

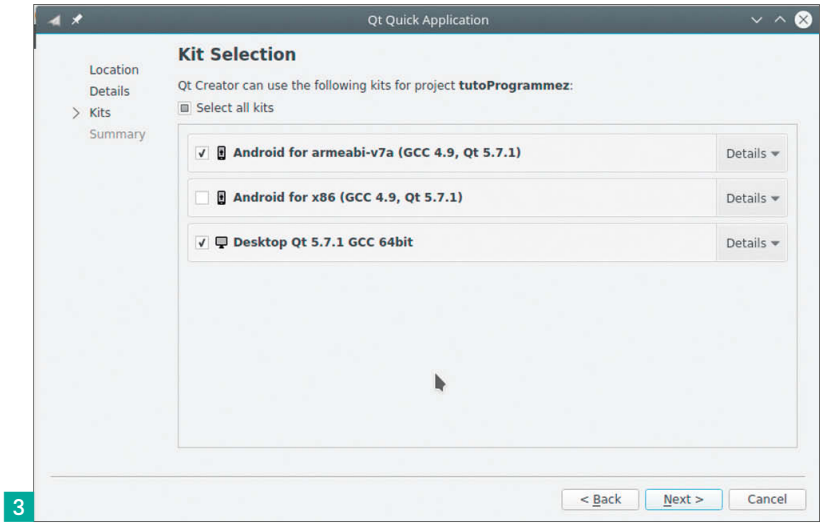
Créons notre projet

Créez un nouveau projet, sélectionnez template Application, puis Qt Quick Application 2

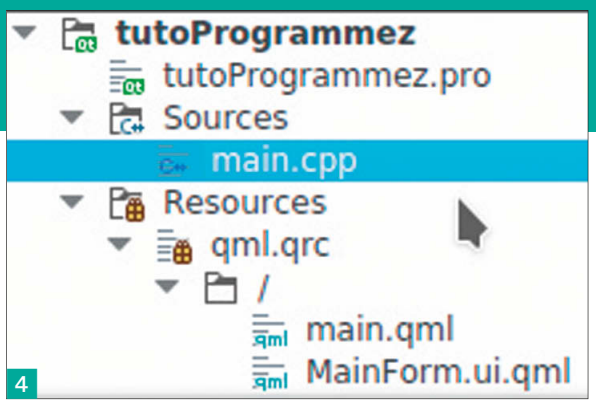
Entrez un nom pour votre application, puis Qt Creator vous demandera quels kits y associer à votre projet, ceci indiquera quelles cibles seront disponibles à la compilation : 3



2



3



4

L'IDE vous créera votre projet, dont l'arborescence ressemblera à ceci 4

Vous voyez :

- un fichier C++ "main.cpp" qui se charge d'initier l'application et de charger le ou les fichiers QML
- deux fichiers d'interface QML "main.qml" qui appelle "MainForm.ui.qml"

Observons les fichiers de notre projet

Regardons ensemble ces trois fichiers.

Le premier "main.cpp", comme son nom l'indique, est le fichier principal :

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>

int main(int argc, char *argv[])
{
  QGuiApplication app(argc, argv);

  QQmlApplicationEngine engine;
  engine.load(QUrl(QStringLiteral("qrc:/main.qml")));

  return app.exec();
}
```

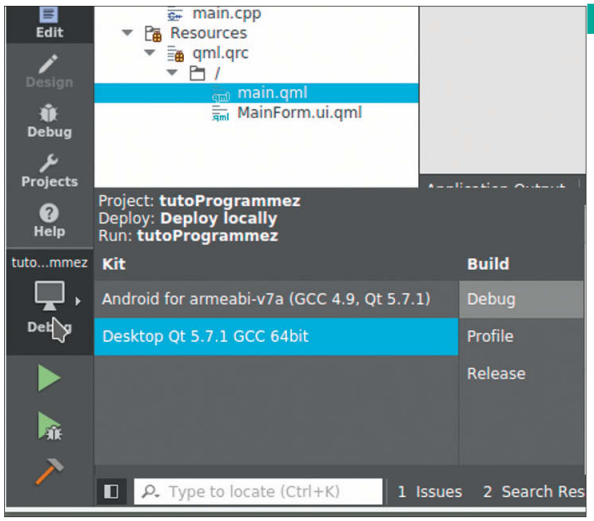
Comme vous pouvez le remarquer, le code est assez succinct, il consiste en la création de l'application, l'instanciation d'un objet "moteur QML" à qui on indique le fichier qml à charger. Enfin on demande à l'application de démarrer.

Voyons ensuite le premier fichier QML: "main.qml"

```
import QtQuick 2.5
import QtQuick.Window 2.2

Window {
  visible: true
  width: 640
  height: 480
  title: qsTr("Hello World")

  MainForm {
    anchors.fill: parent
    mouseArea.onClicked: {
      console.log(qsTr('Clicked on background. Text: "' + textEdit.text + '"'))
    }
  }
}
```

C'est l'occasion de parler un peu de ce format d'interface atypique qui ressemble un peu à l'HTML5 et au CSS. L'idée est simple : on crée des objets d'un type particulier (window,rectangle,button…) et on indique leurs propriétés.
Mais ce qui change de l'HTML5 (l'actuel, car les custom elements feront leur apparition prochainement dans les navigateurs modernes) c'est la possibilité de créer des objets personnels : par exemple autant "Window" est un type QML connu, autant "MainForm" est en revanche un objet propre à ce projet. En effet dans ce langage on peut simplement créer un fichier objet.qml puis l'appeler dans un autre uniquement avec son nom. Le bloc "MainForm" est donc une inclusion du fichier "MainForm.ui.qml" avec "à la volée" le paramétrage de son ancrage ainsi qu'un ajout d'un événement au clic sur cette zone.
C'est ainsi très pratique de pouvoir diviser le développement de ressources QML plus ou moins riches puis de les inclure dans vos interfaces.
Vous pouvez également noter que Qt Creator possède un designer d'interface : en effet vous avez le choix d'utiliser ou non un outil WYSIWYG* pour les créer.

* WYSIWYG What You See Is What You Get : éditeur graphique.



5

Note

Pour plus d'informations et d'aide sur l'utilisation de cet IDE, vous avez :

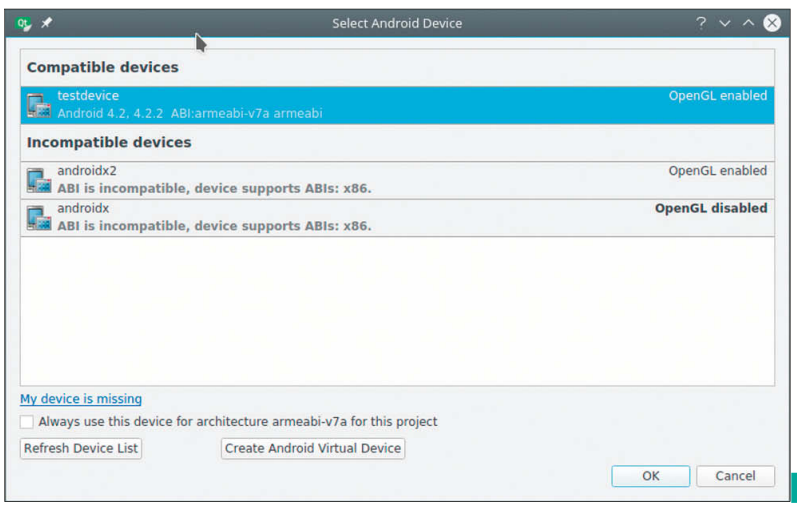
- *une documentation très bien expliquée directement dans l'IDE ;*
- *des tutoriaux sur le site officiel de Qt ;*
- *vous aurez l'occasion de voir une petite application développée avec cet IDE dans votre magazine préféré.*

Compiler un projet

Au moment de compiler vous avez le choix de la cible désirée, l'avantage de Qt c'est de pouvoir commencer à travailler en compilant sur votre ordinateur (cible ordinateur : compilation très rapide). Ensuite, pour vérifier les contraintes d'affichage et autres, vous pouvez compiler vers le mobile/tablette de votre choix en fonction de vos besoins (en connectant un cable usb, ou en utilisant l'émulateur android). 5

Vous avez d'ailleurs le choix. Soit de compiler uniquement avec l'icône de marteau : ainsi l'IDE vous génèrera les fichiers de l'application pour les cibles Windows/GNU/Linux/MacOs ou le fichier d'installation pour les plateformes mobiles (apk and co).
Soit de générer et lancer l'application avec le triangle vert : ceci sur ordinateur compilera puis exécutera l'application. Vous verrez ainsi une fenêtre s'ouvrir pour apprécier votre projet.
En revanche pour les plateformes mobiles, vous aurez un écran intermédiaire vous demandant sur quel périphériques exécuter la solution.
Par exemple sur Android, vous pourriez avoir ceci : 6

L'IDE vous proposera soit d'utiliser l'émulateur Android que vous aurez installé, soit d'utiliser un mobile/tablette connecté en USB.
Note : Votre périphérique Android propose un mode développeur qui vous permet de l'exécuter comme environnement de simulation, ainsi vous pouvez l'utiliser, connecté en USB.



6

Note

Activer le mode développeur sur Android : allez dans paramètres, cliquez ensuite sur à propos de l'appareil, enfin cliquez 7 fois sur le block « numéro de build », un message vous confirmera la manipulation.

Conclusion

Vous avez pu voir dans cet article l'environnement de développement confortable proposé par QtCreator.
J'utilise Qt depuis des années, à l'époque où il appartenait encore à Nokia, et je n'ai cessé d'être agréablement surpris par les améliorations apportées.
L'article suivant vous permettra de mieux vous rendre compte de la facilité de développement proposée par cet IDE. Vous verrez comment créer simplement une application mobile Qt en utilisant des langages simples à appréhender : Javascript et QML. •

Patrick Prémartin
MVP Embarcadero (Delphi), prestataire informatique freelance, auteur, formateur et occasionnellement conférencier. Utilisateur du Pascal depuis 1990 et de Delphi depuis sa sortie en 1995.

Delphi ? Pourquoi pas !

Dans cet article vous découvrirez les fonctionnalités principales de Delphi et de son langage de développement : le Pascal Objet. Depuis sa création Delphi a su évoluer et dépasser les tendances du marché des logiciels de développement. Voyons jusqu'où.

Avant Delphi, il y avait Turbo Pascal qui permettait de faire des développements pour MS-DOS. Borland y avait intégré la programmation objet et une hiérarchie de classes « visuelles » pour faire des interfaces professionnelles « de l'époque » nommée Turbo Vision. Même la souris était gérée, plus besoin de capturer l'interruption 33 des PC.

Quand Delphi est sorti en 1995 nous avons découvert la programmation d'interfaces graphiques par drag&drop. La VCL était une révolution par rapport à Turbo Vision. La programmation facile d'applications entièrement compilées pour Windows 3 était devenue facile. Plus besoin de se taper les kilomètres d'API de Microsoft pour afficher une fenêtre et gérer les actions de la souris. Plus besoin non plus de runtime à distribuer avec ses applications.

L'environnement de développement (EDI) de Delphi (qui sert aussi pour C++Builder) est développé sous Delphi avec la VCL. Peu de logiciels de développement peuvent en dire (ou en faire) autant.

Plusieurs versions

Quand on parle de Delphi on parle du logiciel de développement, mais aussi de son langage.

Je passe rapidement sur les licences car ça me semble nécessaire pour s'y retrouver dans les fonctionnalités fournies. L'outil de développement et ses frameworks (RTL, VCL, FMX, et d'autres) existent en plusieurs versions, dans plusieurs packagings. On trouve Delphi séparément ou accompagné de C++Builder dans le bundle RAD Studio. Delphi est disponible en édition Starter (gratuite pour un usage strictement personnel), Professional, Enterprise et Architect. Le choix de la version conditionne les composants fournis, les bases de données accessibles en standard, les compilateurs cibles et la licence de distribution des logiciels réalisés.

Pour tester Delphi, Embarcadero propose de télécharger une version d'évaluation dont la licence dure 60 jours. C'est une version Enterprise complète. Seule restriction : la licence ne vous autorise pas à développer et distribuer vos créations, mais vous pouvez jouer avec autant que vous voulez et tester tous les exemples.

L'environnement de développement

L'éditeur de développement intégré (EDI ou IDE en anglais) est la partie visible de Delphi. Il regroupe tous les outils pour créer un logiciel de sa conception à son déploiement.

Comme la plupart des outils de développement, l'EDI de Delphi est composé de panneaux qui s'adaptent en fonction de l'opération en cours. Il est entièrement paramétrable, peut s'afficher sous forme d'une seule fenêtre comme de fenêtres flottantes (ce qui est pratique si on travaille avec plusieurs écrans sur son ordinateur).

Et comme l'EDI est une application Delphi VCL, il est possible d'y accéder depuis un programme Delphi grâce à la Tools API. On peut ainsi y ajouter des fonctionnalités (options de menus, fonctionnalités dans l'éditeur de code, assistants …).

Les panels

L'IDE intègre de nombreux panneaux affichant des informations sur l'action en cours ou servant à modifier simplement ce que l'on fait. Voici les principaux.

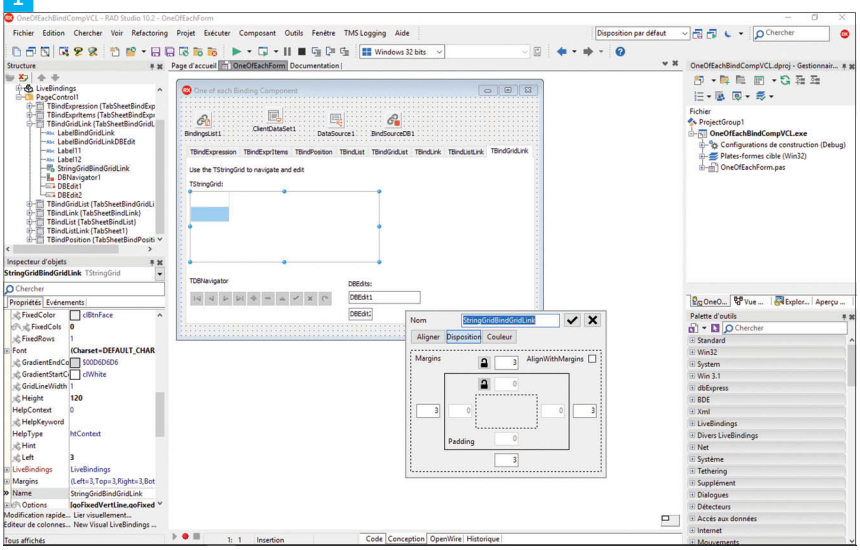
Le concepteur de fiches 1

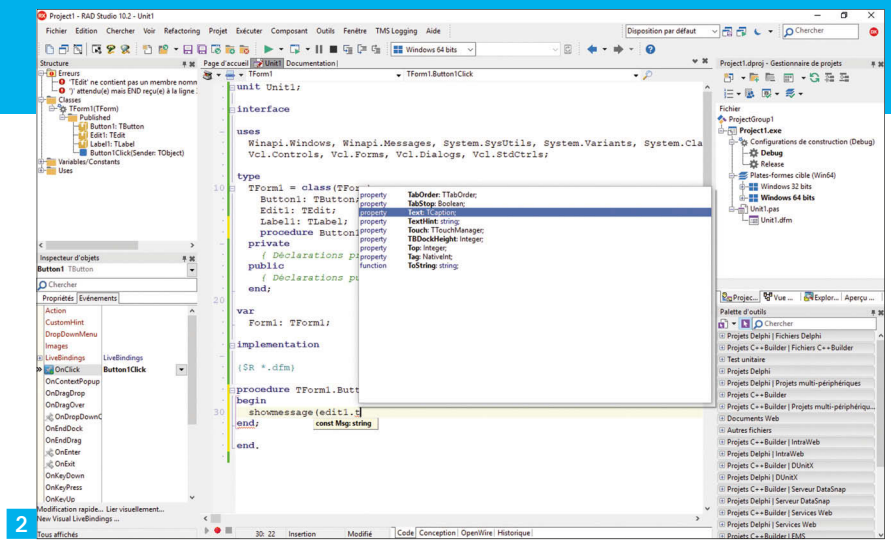
C'est la base des outils RAD pour la création d'interfaces utilisateurs graphiques. Le concepteur de fiches permet de gérer l'intégralité des modules visuels (ou non visuels) de chaque écran d'un logiciel Delphi ou C++Builder.

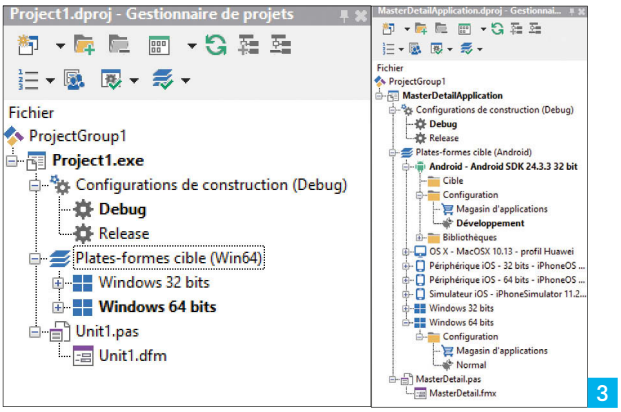
Il n'existe pas un mais plusieurs concepteurs de fiches : un pour la VCL, un pour Firemonkey et d'autres selon les types de projets réalisés (par exemple un concepteur pour créer ses pages web, un pour IntraWeb, …).

Chacun a ses fonctionnalités, mais ils ressemblent généralement à une grille sur laquelle on pose des composants. Ceux-ci se transformant en zones d'affichage ou de saisie pour les composants visuels et juste en logo pour les composants non visuels (stockage, accès aux bases de données, live binding, API, …).

1







L'éditeur de code 2

L'éditeur de code permet comme son nom l'indique de faire du code. On peut ainsi modifier les fichiers sources générés par l'assistant de création de projet ou de fichier choisi, adapté en temps réel par le concepteur de fiche lorsqu'il y en a un.

L'éditeur est un classique des outils de développement. Il propose une mise en forme automatique du texte avec une gestion de l'indentation, des raccourcis de code, des templates, une aide contextuelle et tout un tas de choses permettant d'accélérer notre travail quotidien de développeur.

Le gestionnaire de projets 3

Ce panneau permet de gérer les projets ouverts sous forme de groupe de projet et pour chacun d'eux la liste des fichiers, des dépendances, des options de configurations et des plateformes cibles. Selon les choix faits dedans (ou en raccourci dans la barre d'outils), on peut rapidement compiler un projet en 32 ou 64 bits, sous Windows, Mac, Linux ou n'importe lequel des systèmes d'exploitation gérés. On peut aussi déployer son projet ainsi compilé sur l'un des appareils connectés à l'ordinateur ou vers un ordinateur distant.

L'éditeur de propriétés 4

Accessible à tout moment, mais plus utile lors de la conception d'une fiche, l'éditeur de propriétés donne accès aux propriétés et événements des composants placés sur la fiche.

Les modifications faites dedans se répercutent en temps réel soit sur le code, soit dans la fiche selon que l'on modifie les valeurs des propriétés ou des événements.

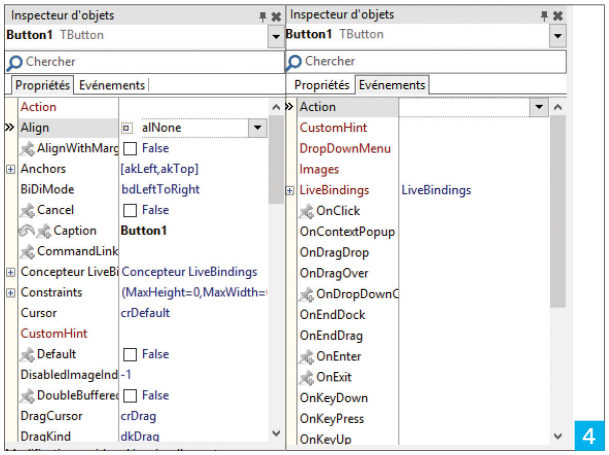
La vue structure 5

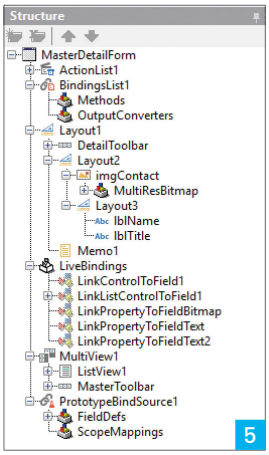
Grâce à la vue structure ou peut repérer directement l'arborescence des composants d'une fiche et accéder à l'un d'entre eux. On peut déplacer des composants dans la hiérarchie et les dupliquer. Ce panneau s'interface avec l'éditeur de propriétés.

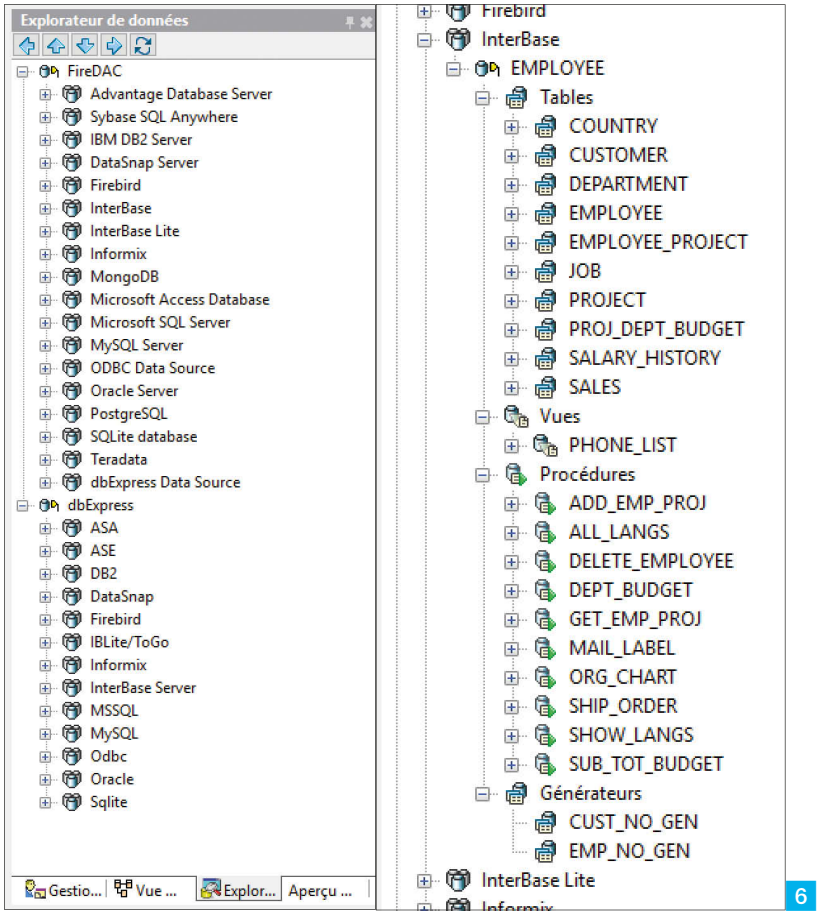
Avec la VCL et FMX il existe deux hiérarchies de composants : une relation de propriétaire / dépendant et une relation de type parent / fils. C'est la seconde qui est gérée par la vue structure : celle qui est visuellement représentée par le concepteur de fiche.

L'explorateur de bases de données 6

Pas forcément utilisé au quotidien ni systématiquement lors d'un développement, l'explorateur de bases de données est un assistant pratique pour consulter la structure d'une base de données et créer







les composants d'accès par un simple glisser/déposer. Il suffit en effet d'y ajouter une base de données, de la configurer puis de l'activer pour en afficher la liste des tables et de leurs champs. On peut alors déplacer un champ, une table ou la base elle-même vers la fiche. L'environnement crée automatiquement les composants nécessaires pour accéder à la base, la table ou au champ. Les composants d'accès aux bases permettent également de gérer un dictionnaire de données afin d'accélérer la mise en forme et la saisie des textes liés à chaque champ. Il suffit de poser ses composants de base de données sur une fiche de type « data module » pour que les informations liées soient accessibles depuis toutes les fiches de votre application.

La palette de composants 7

Le concepteur de fiches ne serait rien sans la palette de composants. Celle-ci se présente généralement sous forme de panneau en bas à droite de l'écran, mais il est toujours possible de l'utiliser « à l'ancienne », comme la barre d'outils connue des premiers utilisateurs de Delphi.

La palette de composants contient l'intégralité des composants visuels et non visuels disponibles pour la fiche en cours d'édition. Ils sont regroupés par catégorie ou éditeur. Chaque composant indique sur quelle plateforme il est utilisable en plaçant la souris dessus, comme ça pas de surprise à la compilation ni au déploiement contrairement à d'autres environnements multiplateformes imposant parfois un runtime pour que tout passe ou indiquant à l'exécution que telle ou telle commande n'est pas gérée.

7

Les possibilités

Delphi a beau avoir 23 ans, le Pascal près de 40 ans, ils n'en restent pas moins modernes. On peut les utiliser pour faire tout ce que l'on veut sous Windows, macOS, iOS, Android et Linux.

Si on n'a pas ce qu'il faut en standard on le trouve chez des éditeurs partenaires ou sur des sites comme GitHub ou Sourceforge. Et si on ne trouve pas là non plus, on peut toujours créer facilement ses propres composants et accéder aux API du système d'exploitation ou d'autres librairies.

Delphi n'a en réalité aucune limite.

Développement Windows

Historiquement Delphi a été créé pour faire du développement Windows et la VCL suit les évolutions visuelles du système d'exploitation de Microsoft. Vous pouvez donc créer des programmes en mode console, des applications plus graphiques (MDI, SDI ou même sans fenêtre), des modules pour le panneau de configuration, des DLL, des services Windows et des Active X. Les assistants sont fournis pour paramétrer vos programmes et le compilateur.

Pour les applications Windows, la VCL et Firemonkey sont utilisables. Ils permettent de compiler en 32 et 64 bits (sous réserve que vous ayez au moins la version Professional), de diffuser vos programmes en direct ou par le Windows Store. Delphi génère des fichiers EXE et APPX selon vos besoins. Notez que pour un programme qui n'a aucune raison de passer en multiplateforme un jour, l'utilisation de la VCL vous donnera accès aux fonctionnalités propres à Windows pas tout de suite reprises dans Firemonkey. Si vous pensez un jour avoir besoin d'une version Mac, ou autre, de votre logiciel, développez-le plutôt avec Firemonkey dès le départ.

La VCL n'a pas vocation à disparaître, vous pouvez continuer à l'utiliser. La création d'une application VCL commence par le choix du type d'application dans l'assistant de création de projet : une application SDI, MDI, Metropolis UI, pour le panneau de configuration, un service Windows ou même un bon vieil Active X. La fiche étant créée, on y place les composants dont on a besoin, par exemple ici un TLabel, un TEdit et un TBitBtn dont je définis le type à bkOk dans l'inspecteur d'objets. 8

L'EDI met à jour le fichier de description de l'écran et le fichier de l'unité dans lequel est décrite la classe correspondant à cette fenêtre.

```
type
 TForm1 = class(TForm)
  Edit1: TEdit;
  Label1: TLabel;
  BitBtn1: TBitBtn;
 private
  { Déclarations privées }
 public
  { Déclarations publiques }
 end;
```

En double cliquant sur le bouton on crée une méthode qui sera utilisée lorsque l'utilisateur cliquera dessus et on y met juste un appel de procédure permettant d'afficher une fenêtre de dialogue avec le contenu du champ de saisie.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
 showmessage(Edit1.Text);
end;
```

Plus qu'à exécuter le programme pour obtenir ceci en changeant le texte et en cliquant sur le bouton. 9

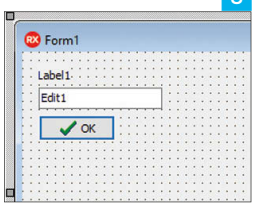
Delphi propose désormais des styles que l'on peut bien entendu personnaliser. Il suffit de cocher celui voulu dans les propriétés du projet et de l'activer par défaut ou par programmation. Pour cet exemple j'ai activé le style « Carbon ». 10

Un éditeur de styles permet bien entendu de créer ses propres déclinaisons. Delphi propose plusieurs librairies d'accès aux bases de données. La plus récente, FireDAC, est bien entendu celle que l'on recommande d'utiliser, mais ce n'est pas obligatoire.

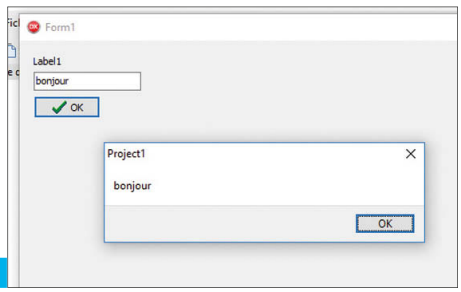
La VCL intègre des composants de saisie et d'affichage liés aux bases de données qui s'interfacent automatiquement avec un TDataSource ou ses descendants pour l'affichage et les mises à jour. N'ayant pas de base à vous proposer pour cet article, je vais passer en mode prototypage, comme je le ferais sur une application lors d'un hackaton ou d'un « startup week-end ».

Il existe un composant pour ça : TPrototypeBindSource. Il génère

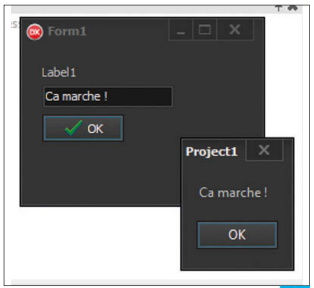
8



9



10



des données en mémoire sous forme de pseudo tables. Je l'ajoute à la fiche, le configure pour générer une table avec des noms et des dates.

Comme ce n'est pas une source de données exploitable telle quelle, on ne peut pas faire de requêtes dedans depuis l'éditeur. J'utilise donc la fonctionnalité de Live Binding entre ce composant de prototypage et les autres composants.

Je le relie à une grille que j'ajoute pour afficher le contenu de la table. Je relie le champ ContactName1 avec le Caption de Label1 et le champ DateField1 au Text du champ de saisie. 11

Live Binding synchronise les modifications faites par l'utilisateur :

- Si on clique sur une ligne de la grille, le label et le champ de saisie prennent automatiquement les bonnes valeurs.
- Si on modifie une date dans le champ de saisie, la modification se répercute sur la grille lorsqu'il perd le focus (en cliquant ailleurs ou par une tabulation).

Live Binding se greffe automatiquement au niveau des évènements onChange des composants.

Cette fonctionnalité de Live Binding permet de simplifier pas mal de choses sur les écrans ayant des informations redondantes ou pour alimenter des champs à partir de sources de données puisque ça ne se contente pas de fonctionner à partir du composant de prototypage, mais permet de lier tous les composants entre eux, visuels ou pas. Bien entendu ça fonctionne sur la VCL, mais aussi avec Firemonkey en multiplateforme.

J'en reste là pour la VCL, de nombreux exemples sont disponibles, notamment pour exploiter les fonctionnalités propres à Windows comme la barre des tâches, les notifications de Windows 10, l'accès aux capteurs présents sur les ordinateurs récents et la gestion des écrans tactiles.

Développement multiplateforme

L'un des gros avantages des versions récentes de Delphi est la possibilité de compiler le code vers plusieurs environnements cibles.

Il existe de nombreux outils pour faire du développement multiplateforme, plus ou moins simples d'utilisation, plus ou moins réellement multiplateforme, plus ou moins indépendants, plus ou moins à partir d'une réelle base de code unique. Je ne citerai personne, vous en avez probablement un ou deux en tête si vous vous êtes déjà penché sur la question.

Le plus ici reste la compilation native. On ne génère pas un exécutable sous forme de runtime qui s'amuse à interpréter ensuite le code du développeur. Ce n'est pas un serveur web local qui interprète du JavaScript pour créer des pages HTML encapsulées dans une application mobile. C'est encore moins une webview comme certains osent encore le proposer avec abonnement depuis des sites internet de « création facile d'applications mobiles ».

Delphi et C++ Builder créent un vrai exécutable comme le feraient un développeur iOS, un développeur macOS, un développeur Linux et/ou un développeur Android. Mais là, on n'a besoin que d'un développeur Delphi ou C++Builder sous Windows pour faire en une fois le travail que feraient les autres. L'exécution des programmes est native et dépend intégralement des performances de l'appareil sur lequel ils sont exécutés. Et en plus pas de surprise pour les utilisateurs : une application mobile iOS a les mêmes fonctionnalités que sa consœur sous Android. 12

La librairie Firemonkey et son concepteur de fiches permettent de cibler facilement plusieurs tailles d'écrans différentes sur plusieurs plateformes différentes. On peut même voir en temps réel comment ça se comportera une fois déployé grâce à FireUI dans l'EDI et LivePreview (disponible sur l'App Store et Google Play). Les sources de LivePreview sont fournies pour y ajouter nos propres composants ou ceux d'un éditeur tiers.

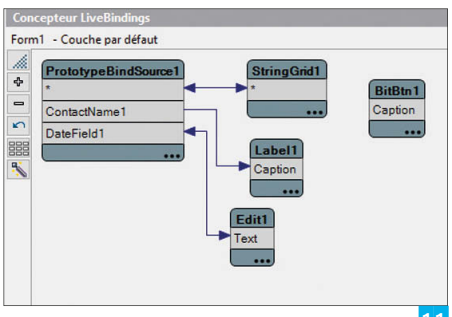
Firemonkey a été conçu dans une optique multiplateforme. C'est un framework entièrement graphique pour sa partie visuelle indépendante des API des systèmes d'exploitation cibles. Il se base sur des librairies comme OpenGL ce qui facilite les optimisations selon le matériel. Grâce à cette approche, on retrouve un système d'animations, des filtres et effets graphiques, des composants s'affichant en 2D et 3D. 13

Fonctionnant sur des types d'écrans variés, le framework gère également la profondeur des pixels. Les images peuvent lui être fournies en plusieurs tailles, il affichera celle qui est la plus adaptée selon l'appareil.

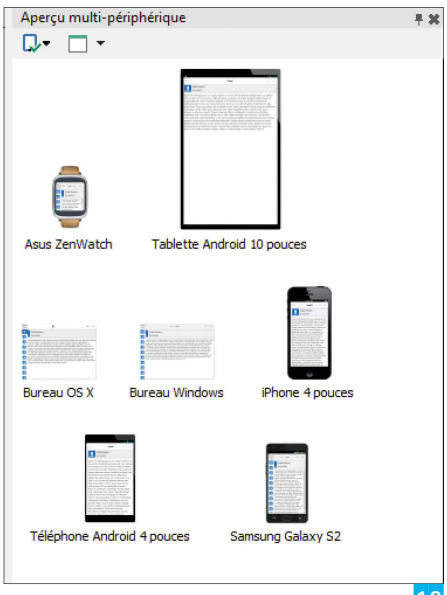
Embarcadero propose également un moteur physique et des fonctions de test de collisions. Idéal pour développer des jeux vidéo multiplateformes (plusieurs exemples sont téléchargeables avec GetIt) ou jouer avec les nerfs de ses utilisateurs le 1er avril.

On peut lancer un champ de saisie sur une barre d'outils ou le laisser tomber en changeant la gravité de certains composants et détecter la collision.

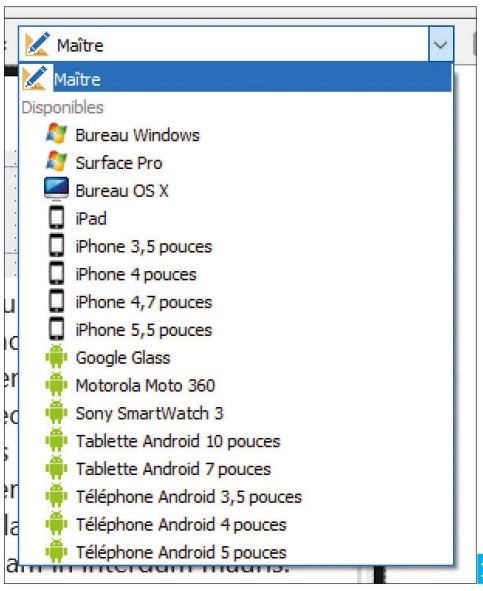
Outre le fait que tout composant puisse en accueillir d'autres, il existe une autre différence majeure entre Firemonkey et la VCL :



11



12



13

l'absence de composants de saisie ou d'affichage dédiés aux bases de données. En effet tous les composants peuvent servir à ça grâce à Live Binding ou par programmation, il n'est donc plus utile de dédier des composants à cet usage. En guise d'exemple pour illustrer ceci, je vous propose de faire un programme de saisie de mot de passe. En cas de saisie correcte, l'écran se referme pour laisser place au suivant. En cas de saisie erronée, le champ de saisie du mot de passe fera une jolie rotation à 360°.

La première chose à faire est de lancer l'assistant de création d'un projet multiplateforme et de choisir de créer une application vide.

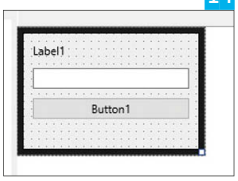
On y place un TLayout à l'intérieur duquel un label, un champ de saisie et un bouton. Les trois sont en alignement Top, avec une marge de 5px sur chaque côté. On met également un padding à 5px sur la fiche elle-même. 14

En programmation multiplateforme il est très important d'utiliser les alignements et les différents layouts. Cela permet de ne pas trop se poser de questions sur la taille des écrans des utilisateurs et de sortir des interfaces qui passent assez bien sur mobiles, tablettes et écrans d'ordinateur. On peut bien entendu personnaliser les choses en fonction d'une taille d'écran spécifique dans le concepteur de fiches et voir le résultat avec FireUI ou en exécutant le programme sur l'appareil concerné.

Je personnalise les textes, déclare le champ de saisie comme un champ mot de passe et active l'évènement onClick du bouton pour gérer le résultat de la saisie.

```
procedure TForm1.Button1Click(Sender: TObject);
var
 password: string;
begin
 password := Edit1.Text;
 if (password.ToLower = 'pass') then
  cestbon
 else
  cestpasbon;
end;
```

14



Reste à développer les deux méthodes appelées, mais avant ça il faut ajouter deux animations dans la fiche directement depuis l'inspecteur de propriétés. Pour cela, cliquez sur la valeur des propriétés et créez un TFloatAnimation.

- Le premier sur la propriété RotationAngle du champ de saisie en mettant 360 comme EndValue.
- Le second sur la propriété Height du layout en laissant 0 comme EndValue, mais en cochant la case StartFromCurrent. Comme ça on n'a pas besoin de connaître la hauteur de l'écran, ça s'adapte tout seul.

Et maintenant on code les deux méthodes à appeler en fonction de la valeur saisie comme mot de passe et les événements déclenchés en fin d'animation.

```
procedure TForm1.cestbon;
begin
 Layout1.Align := TAlignLayout.None;
 FloatAnimation2.Enabled := true;
end;

procedure TForm1.cestpasbon;
begin
 FloatAnimation1.Enabled := true;
end;

procedure TForm1.FloatAnimation1Finish(Sender: TObject);
begin
 Edit1.SetFocus;
 FloatAnimation1.Enabled := false;
end;

procedure TForm1.FloatAnimation2Finish(Sender: TObject);
begin
 FloatAnimation2.Enabled := false;
 freeandnil(Layout1);
end;
```

Il ne reste plus qu'à lancer le programme sous Windows ou sur un smartphone qui serait connecté à l'ordinateur.

Compilation

Via le gestionnaire de projets choisit la plateforme cible (Windows, macOS, iOS, Android ou Linux) de la compilation. On y retrouve la liste des appareils accessibles correspondant à ce système d'exploitation. Il suffit de choisir le bon et le programme se lance dessus.

C'est aussi dans le gestionnaire de projets que l'on décide de compiler en 32 ou 64 bits pour Windows et iOS, bientôt également pour macOS dont la compilation 64 bits est annoncée pour la version 10.13.

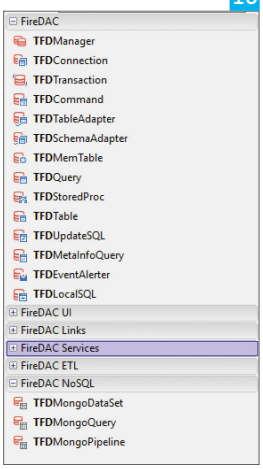
Delphi et C++Builder sont fournis avec une série de compilateurs permettant d'accéder aux plateformes choisies. Ils compilent soient directement, soit en passant par LLVM selon la plateforme.

Firemonkey étant destiné à fonctionner en multiplateforme, le framework contient des composants et des librairies gérant les caméras, la géolocalisation, l'accéléromètre, et d'une façon générale les capteurs habituels des smartphones ou tablettes.

Les bases de données

Depuis le début Delphi a été livré avec des composants permettant un accès aux bases de données du marché. On a commencé en 1995 avec Borland Database Engine (BDE) qui permettait entre autres de travailler avec des bases ODBC, Paradox et dBase. Il n'est plus fourni avec la distribution standard, mais peut toujours être téléchargé et installé sur les versions récentes de Delphi depuis l'espace de téléchargement des utilisateurs ayant une licence valide. ADO et dbExpress sont toujours disponibles dans les versions récentes et préinstallées. De nombreux projets les utilisent. Il est cependant recommandé de passer à FireDAC pour bénéficier de ses nombreuses fonctionnalités évoluées. 15

15



Avec FireDAC, en plus des fonctions habituelles, on peut créer des bases de données en mémoire, accéder à plusieurs bases / moteurs avec la même requête SQL, gérer un cache local, transférer des lots de mises à jour, gérer les coupures de connexion avec la base, …

Il suffit de regarder la liste des options du composant TFDConnection pour prendre conscience de l'évolution depuis le BDE. 16

De très nombreux exemples sont fournis pour arriver à prendre en main les subtilités de cette librairie. En plus de la documentation

de Delphi, je vous recommande également le livre de Cary Jensen sur le sujet. Et bien entendu FireDAC fonctionne sur tous les types de projets et vers toutes les cibles de compilation en mode console, en VCL, en Firemonkey ou d'autres comme IntraWeb ou web Broker. En guise d'exemple je vous propose un programme très simple : la création d'une base SQLite et son remplissage par une simple boucle. Je vais le faire en Firemonkey, par habitude, mais il fonctionnerait de la même façon avec la VCL. La première chose à faire est de placer un composant TFDConnection sur la fiche, puis un TFDQuery (pour l'affichage) et enfin un TFDPhysSQLiteDriver Link qui ne sert qu'à ajouter les bonnes unités pour gérer une base SQLite. On ajoute ensuite un bouton qui servira à remplir la table, une grille pour l'affichage du contenu et enfin deux labels en bas de fiche pour afficher le total des lignes. La grille est liée à la source de données par Live Binding, sans préciser de champ puisque l'EDI ne connaît pas cette information tant que la base n'existe pas. Du coup il affichera tout ce que nous mettrons dans le SELECT.
La base de données est créée lors de la première utilisation du programme et ouverte à la création de la fiche.

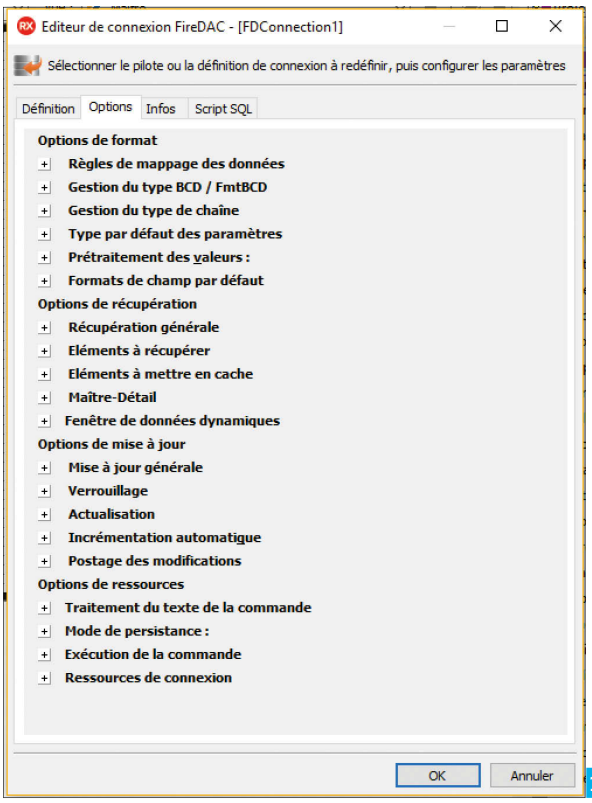
```
procedure TForm1.FormCreate(Sender: TObject);
var
 dbname: string;
 creerdb: boolean;
begin
{$IFDEF DEBUG}
 dbname := tpath.GetDocumentsPath + pathdelim + 'programmez-firedac-demo-DEBUG.db';
{$ELSE}
 dbname := tpath.GetDocumentsPath + pathdelim + 'programmez-firedac-demo.db';
{$ENDIF}
 creerdb := not tfile.Exists(dbname);
 FDConnection1.Params.Clear;
 FDConnection1.Params.Values['DriverID'] := 'SQLite';
 FDConnection1.Params.Values['Database'] := dbname;
 FDConnection1.Params.Values['OpenMode'] := 'CreateUTF16';
 FDConnection1.Params.Values['StringFormat'] := 'Unicode';
{$IFDEF DEBUG}
 FDConnection1.Params.Values['Password'] := '';
 FDConnection1.Params.Values['Encrypt'] := '';
 FDConnection1.Params.Values['LokingMode'] := 'Normal';
{$ELSE}
 FDConnection1.Params.Values['Password'] := 'motdepassepourlaprod';
 FDConnection1.Params.Values['Encrypt'] := 'aes-256';
 FDConnection1.Params.Values['LokingMode'] := 'Exclusive';
{$ENDIF}
 FDConnection1.LoginPrompt := false;
 FDConnection1.Connected := true;
 if creerdb then
 begin
  FDConnection1.ExecSQL('CREATE TABLE IF NOT EXISTS MaTable (ID INTEGER PRIMARY KEY,
Valeur INTEGER DEFAULT 0)');
 end;
 CalculerTotal;
 FDQuery1.Open('select * from MaTable');
end;
```

Je distingue toujours le mode RELEASE du mode DEBUG pour nommer mes fichiers afin d'éviter de jouer en développement sur des données de production. Pensez-y. Cela vous évitera de mauvaises surprises. La procédure CalculerTotal n'a rien de compliqué : une simple requête SQL de calcul de la somme d'une colonne de table. En revanche, comme elle est appelée alors que la table est potentiellement vide, la requête peut ne rien renvoyer. Il est donc important d'intercepter l'exception éventuelle.

```
procedure TForm1.CalculerTotal;
var
 total: integer;
begin
 try
  total := FDConnection1.ExecSQLScalar('select sum(Valeur) from MaTable');
 except
  total := 0;
 end;
 Label2.Text := total.ToString;
end;
```

Et pour finir voici le code lié au bouton : il ajoute des valeurs dans la table, rafraîchit la source de données actuelle et relance le calcul.

```
procedure TForm1.Button1Click(Sender: TObject);
var
 tab: TFDTable;
 i: integer;
begin
 tab := TFDTable.Create(Self);
 try
  tab.Connection := FDConnection1;
  tab.TableName := 'MaTable';
  tab.Open;
  for i := 1 to 10 do
```



16

```
  begin
    tab.Insert;
    tab.FieldByName('Valeur').AsInteger := random(500);
    tab.Post;
  end;
  tab.Close;
 finally
  freeandnil(tab);
 end;
 FDQuery1.Refresh;
 CalculerTotal;
end;
```

J'ai utilisé un composant TFDTable pour l'ajout, mais j'aurais pu le faire directement sur la requête déjà ouverte avec le TFDQuery, ce qui aurait évité de la rafraîchir. Comme vous le voyez, il est facile d'accéder à une base de données et de travailler dessus. Avec Live Binding il est même possible de faire une application simple de gestion de base de données sans une seule ligne de code (à part pour les éventuels boutons, menus et enchaînements d'écrans). FireDAC peut également donner accès à des sources de données qui ne sont pas des bases SQL. On peut s'en servir pour accéder à MongoDB dans les versions Enterprise et Architect de Delphi. On peut même accéder en SQL à des API web et de logiciels en passant par les composants vendus par CData.

Les autres librairies

Dans une optique d'interfaçage simple entre un programme Delphi et le reste du monde, Embarcadero livre des composants et librairies pour l'accès au web, à des API REST, pour travailler avec JSON et XML, convertir des données en base 64, gérer l'UTF8, l'UTF16 et Unicode. Concernant REST, Embarcadero propose le Débogueur REST, un logiciel pour Windows librement téléchargeable depuis leur site pour tester les API REST directement et préparer les composants à utiliser dans Delphi. Ce logiciel sert également à s'assurer qu'une API fonctionne correctement, quel que soit le langage utilisé côté consommateur ou côté serveur. On trouve aussi des composants pour gérer des communications en Bluetooth et Bluetooth LE. Les protocoles liés aux beacons (iBeacon d'Apple et EddyStone de Google) sont également de la partie. Les applications VCL et Firemonkey peuvent accéder aux capteurs présents sur les ordinateurs, smartphones et tablettes à l'aide de composants dédiés. On retrouve aussi l'App Tethering : des composants permettant de partager des services, des ressources et d'échanger des messages entre logiciels présents sur le même réseau local (ou en spécifiant l'IPv4 ou IPv6 des pairs) ou sur des appareils appairés en Bluetooth. On peut par exemple s'en servir pour étendre une application de bureau en la faisant communiquer avec une application mobile en Bluetooth ou sur le réseau local très facilement.

Très utiles dans les applications gourmandes en ressources ou sur les mobiles, il est possible de faire de la programmation asynchrone à l'aide des processus. Delphi est livré avec des classes gérant les threads classiques (TThread), mais aussi du parallélisme (TTask, TParallel). Et bien entendu tout ceci est disponible sur toutes les plateformes cibles (à condition bien entendu que le système d'exploitation et l'appareil puissent les gérer).

Développements web et client – serveur

Sur la partie serveur, Delphi permet de créer des services Windows, des programmes en console (exécutables sous forme de CGI), des extensions pour les serveurs web du marché (IIS, Apache) pour Windows et Linux, et des serveurs.

La possibilité de communiquer en mode web (http, https) et en direct avec des sockets permet de faire tout ce que l'on veut.

La compilation pour Linux ou Windows côté serveur permet d'accéder à quasiment tous les serveurs du marché. 17

Delphi est livré avec une version de base de IntraWeb qui permet de dessiner des applications web directement depuis l'éditeur. La compilation du projet crée un serveur web indépendant (pour Windows) ou une extension à des serveurs web selon la configuration faite à la création du projet.

web Broker et DataSnap permettent de générer des serveurs web ou extensions gérant soit directement des pages web et leur affichage de A à Z, soit des API REST. On a bien entendu accès en Pascal à l'intégralité des informations transmises par les navigateurs ou applications clientes afin d'adapter la réponse.

Enfin, RAD Server (anciennement appelé EMS) dont une licence est fournie avec les versions Enterprise et Architect de RAD Studio permet de développer des micro services et des applications complètes avec sa gestion intégrée des utilisateurs et des sessions.

Conclusion

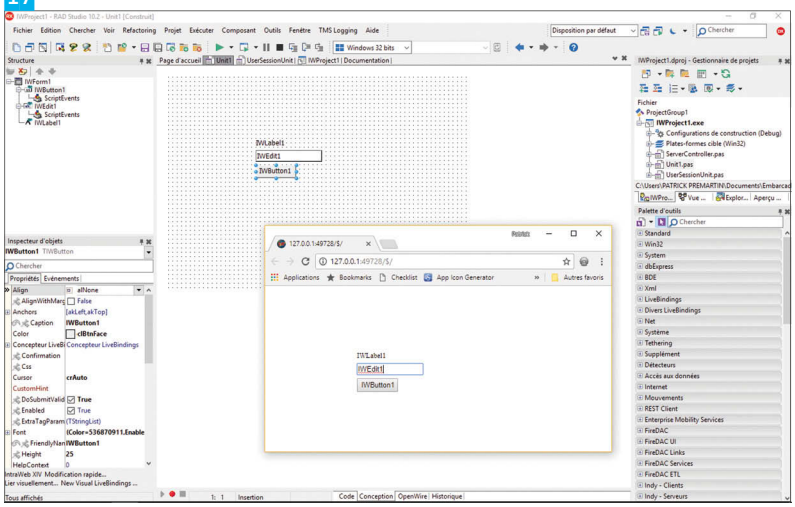
Difficile de tout dire et tout montrer sans écrire des milliers de pages illustrées de codes sources et de captures d'écrans. Si vous voulez la liste complète des fonctionnalités de Delphi vous pouvez vous rendre sur le site officiel pour télécharger la datasheet ou consulter la documentation sur docwiki.embarcadero.com.

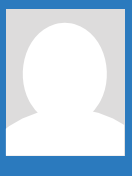
À travers cet article je voulais faire passer un message : Delphi n'est pas mort et a su évoluer avec les besoins des utilisateurs. On peut tout développer avec cet outil. J'espère que vous en êtes convaincu et n'hésiterez pas à l'avenir à le voir comme une alternative durable lors de vos choix d'outils de développement.

Et si vous voulez le voir en action jetez un coup d'œil sur les sites de partages de vidéos ou venez à l'une des réunions organisées régulièrement en France sous forme de présentations, de conférences ou de meetups. •

Pour télécharger les sources complets, rendez-vous sur http://vasur.fr/programmez01

17

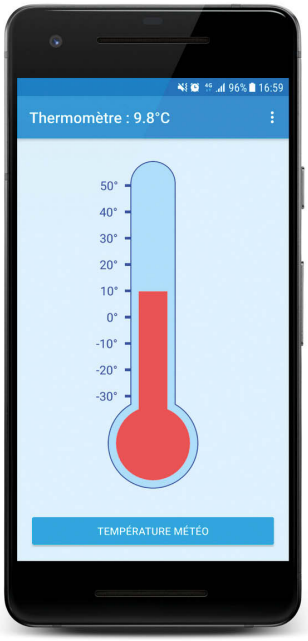




Sylvain SAUREL
Développeur Java / Android
https://www.ssaurel.com
sylvain.saurel@gmail.com

Créez une application Thermomètre sous Android

Les smartphones et tablettes Android sont de plus en plus puissants. Mieux encore, ils regorgent de capteurs permettant de transformer votre appareil Android en baromètre, magnétomètre ou encore en thermomètre. Dans cet article, nous vous proposons de découvrir comment créer une application Android de type thermomètre avec une interface graphique attrayante.



1 *Application Thermomètre*

Les capacités toujours plus grandes des smartphones et tablettes Android permettent aux développeurs d'imaginer toujours plus de cas d'utilisations pour ces appareils. De fait, il paraît naturel de mettre à profit le capteur de température ambiante proposé par certains appareils Android pour permettre aux utilisateurs d'utiliser leur appareil en tant que thermomètre. Malheureusement, certains appareils, parmi les plus récents, ne proposent plus de support pour ce capteur. L'application que nous allons réaliser devra en tenir compte et proposer d'autres sources de température aux utilisateurs. 1

Notre application Thermomètre va ainsi proposer la température aux utilisateurs via les 5 sources de données suivantes :

- Le capteur de température ambiante si l'appareil en possède un ;
- La température de la batterie ;
- La température du CPU ;
- La température ambiante extrapolée à partir de la température de la batterie de l'appareil ;
- La température météo qui donnera la température mesurée par la station météo la plus proche en utilisant la position GPS de l'appareil.

Récupération de la température ambiante

Fort logiquement, le capteur de température ambiante d'un appareil semble être le meilleur moyen de mesurer la température. Il faut cependant avoir la chance que son appareil Android en possède un, ce qui ne semble plus être la norme actuellement. Les derniers appareils les plus connus à bénéficier d'un tel capteur étaient le Samsung Galaxy S4, le Samsung Galaxy Note 3 ou encore le Motorola Moto X. Néanmoins, il est important de supporter cette fonctionnalité au sein de notre application dans le cas où les constructeurs se décideraient à équiper de nouveau leurs appareils avec ce capteur dans un futur proche.

Pour accéder au capteur de température ambiante, nous allons dans un premier temps récupérer une instance du *SensorManager* via un appel à la méthode *getSystemService* avec en entrée la constante *Context.SENSOR_SERVICE*. Ensuite, nous appelons la méthode *getDefaultSensor* de l'instance de *SensorManager* récupérée, avec, en entrée, la constante *Sensor.TYPE_AMBIENT_TEMPERATURE* . Cette dernière permet d'obtenir une instance de *Sensor* représentant le capteur de température ambiante de l'appareil.

Si l'appareil ne possède pas de capteur de ce type, la méthode *getDefaultSensor* retournera null et nous devrons afficher un message à l'utilisateur ; il sera averti d'utiliser une autre source de données pour la température. Dans le cas où l'appareil possède bien un tel capteur, nous allons nous abonner à ses changements de valeur en enregistrant une implémentation de l'interface *SensorEventListener* dans la méthode *registerListener* du *SensorManager*.

Les mises à jour de valeur mesurées par le capteur de température ambiante seront envoyées via un appel de la méthode *onSensorChanged*. Elle est issue de l'interface *SensorEventListener* que nous avons implémentée au sein de l'activité principale de notre application. Une fois la valeur de température ambiante en notre possession, nous réaliserons un appel à la méthode *setCurrentTemp* du composant graphique *Thermometer* afin de mettre à jour l'interface graphique de notre application. Ce composant *Thermometer* sera détaillé plus tard dans cet article.

Tout ceci nous donne le code suivant pour la gestion de la température ambiante :

```
public class MainActivity extends AppCompatActivity implements SensorEventListener{

  private SensorManager sensorManager;
  private Thermometer thermometer;
  // ...

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    thermometer = (Thermometer) findViewById(R.id.thermometer);
    sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    // ...
  }

  // ...

  @Override
  public void onSensorChanged(SensorEvent sensorEvent) {
    if (sensorEvent.values.length > 0) {
      temperature = sensorEvent.values[0];
      thermometer.setCurrentTemp(temperature);
```

```
  }
 }

 @Override
 public void onAccuracyChanged(Sensor sensor, int accuracy) {
 }

 private void loadAmbientTemperature() {
  Sensor sensor = sensorManager.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE);

  if (sensor != null) {
   sensorManager.registerListener(this, sensor, SensorManager.SENSOR_DELAY_FASTEST);
  } else {
   Toast.makeText(this, R.string.no_temperature_sensor, Toast.LENGTH_LONG).show();
  }
 }

// ...

}
```

Récupération de la température de la batterie

Certains utilisateurs apprécieront surement de pouvoir accéder directement à la température de la batterie de leur appareil via notre application. Il semble donc pertinent de proposer cette température comme une source de données supplémentaire pour notre application.

La récupération de la température de la batterie se fait en passant par un *BroadcastReceiver* sous Android. Nous créons donc un objet *IntentFilter* dédié avec une action de type *Intent.ACTION_BATTERY_CHANGE* stipulant que nous souhaitons être tenu informé des modifications liées à la batterie d'une manière générale. Nous appelons ensuite la méthode *registerReceiver* avec en paramètre notre *IntentFilter* et un objet de type *BroadcastReceiver* qui nous permettra d'être notifié des mises à jour de la batterie.

Au sein de notre instance de *BroadcastReceiver*, la méthode *onReceive* sera appelée lorsque la batterie subira une modification d'état. Dans le cas présent, nous souhaitons obtenir sa température. Nous allons donc accéder au paramètre *BatteryManager.EXTRA_TEMPERATURE* de l'*Intent* passé en entrée de la méthode *onReceive*. La valeur retournée par le capteur est de type entier et il est nécessaire de la diviser par 10 pour obtenir la valeur décimale réelle de la température. Une fois cette division faite, nous pouvons mettre à jour l'objet *Thermometer* en passant cette valeur en entrée de sa méthode *setCurrentTemp*.

Tout ceci nous donne le code suivant :

```
public class MainActivity extends AppCompatActivity implements SensorEventListener {

 // ...

 private BroadcastReceiver batteryInfoReceiver = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
   int temperature = intent.getIntExtra(BatteryManager.EXTRA_TEMPERATURE, 0);

   if (temperature > 0) {
    float temp = ((float) temperature) / 10f;
    thermometer.setCurrentTemp(temp);
   }
  }
 };

 // ...

 private void loadExtrapolatedTemperature() {
  IntentFilter intentFilter = new IntentFilter();
  intentFilter.addAction(Intent.ACTION_BATTERY_CHANGED);

  registerReceiver(extrapolatedInfoReceiver, intentFilter);
 }

 // ...
}
```

Récupération de la température du CPU

Dans la même veine que la température de la batterie, la température du CPU pourra intéresser les utilisateurs les plus Geeks de l'application Thermomètre. Il s'agit donc d'une source de données intéressante à prendre en compte dans l'application.

Le système d'exploitation Android est basé sur un noyau Linux modifié. De fait, la température du CPU peut être lue à partir d'un fichier système. Une fois ce fichier localisé, il suffira de lire la valeur qu'il stocke pour obtenir la température CPU. La principale difficulté ici réside dans le fait que le fichier système en question peut ne pas être le même suivant le type d'appareil hôte de l'application Thermomètre.

Le code présenté ici fonctionne cependant dans la majorité des cas. On lit ainsi le contenu des fichiers systèmes en question et on renvoie en sortie de méthode la valeur lue comme un entier :

```
public class CPU {

 public static int getCPUTemperature() {
  File fileTemp1 = new File("/sys/devices/virtual/thermal/thermal_zone0/temp");
  File fileTemp2 = new File("/sys/devices/platform/s5p-tmu/temperature");

  if (fileTemp1.exists()) {
   int temp = CPU.readSystemFileAsInt("/sys/devices/virtual/thermal/thermal_zone0/temp");

   if (temp > 3) {
    if (temp > 1000) {
     temp /= 1000;
    }

    return temp;
   }
  }

  if (fileTemp2.exists()) {
   int temp = CPU.readSystemFileAsInt("/sys/devices/platform/s5p-tmu/temperature");

   if (temp > 3) {
```

```
    if (temp > 1000) {
      temp /= 1000;
    }

    return temp;
   }
  }

  return -1;
 }
}
```

Il est bon de noter que sur certains appareils, la température est stockée avec des décimales. Dans ce cas-là, il est nécessaire de diviser la valeur par 1000 pour obtenir la température du CPU de l'appareil. Au niveau de la classe *MainActivity* de notre application, l'affichage de la température sur le composant *Thermometer* se fera comme suit :

```
private void loadCpuTemperature() {
 int cpuTemperature = CPU.getCPUTemperature();

 if (cpuTemperature >= 0) {
  thermometer.setCurrentTemp(cpuTemperature);
 } else {
  Toast.makeText(this, R.string.no_cpu_temperature, Toast.LENGTH_LONG).show();
 }
}
```

Calcul de la température ambiante extrapolée

Pour les smartphones et tablettes ne disposant pas de capteurs de température ambiante, il pourrait être intéressant de proposer une température ambiante extrapolée à partir de la température de la batterie. Dans cette optique, l'entreprise OpenSignal a publié une étude montrant une corrélation certaine entre la température ambiante et la température de la batterie d'un appareil. L'analyse d'un grand volume de données issues de smartphones et tablettes exécutant leur application a permis de mettre en lumière une formule qui doit être considérée comme expérimentale pour le moment :

*Température Ambiante = 2.55 * Température Batterie – 60.52*

Le rapport complet détaillant la mise en exergue de cette formule et des constantes en faisant partie est très instructif et nous vous invitons à le consulter à l'adresse suivante :

http://opensignal.com/reports/battery-temperature-weather/

L'implémentation de la température ambiante extrapolée au sein de notre application Thermomètre va se révéler assez simple puisque nous savons déjà comment obtenir la température de la batterie d'un appareil Android. Nous allons créer un nouveau *BroadcastReceiver* qui sera chargé d'appliquer la formule, que nous venons d'exposer, sur la température de la batterie :

```
private BroadcastReceiver extrapolatedInfoReceiver = new BroadcastReceiver() {
 @Override
 public void onReceive(Context context, Intent intent) {
  int temperature = intent.getIntExtra(BatteryManager.EXTRA_TEMPERATURE, 0);

  if (temperature > 0) {
   float temp = (((float) temperature) / 10f) * 2.55f - 60.52f;
   thermometer.setCurrentTemp(temp);
  }
 }
};
```

Récupération de la température météo

Finalement, notre application Thermomètre mettra à disposition des utilisateurs la température mesurée par la station météo la plus proche de sa position. Pour cela, nous allons devoir récupérer la position GPS de l'utilisateur. Une fois cette position obtenue, nous requêterons le Web Service météo gratuit proposé par OpenWeatherMap afin d'obtenir la température ambiante mesurée pour cette position donnée.

Récupération de la position GPS

Dans le but de récupérer la position de l'utilisateur de la manière la plus efficace possible, nous allons utiliser l'API Fused Location Provider de Google. Pour être notifié des mises à jour des données de position de l'utilisateur, notre classe *MainActivity* implémentera les interfaces *ConnectionCallbacks* et *OnConnectionFailedListener* de l'objet *GoogleApiClient*. A la création de notre activité, nous construire un objet *GoogleApiClient* auquel nous allons tenter de nous connecter en appelant sa méthode *connect* :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 // ...

 // Création de l'objet LocationRequest
 mLocationRequest = LocationRequest.create()
  .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
  .setInterval(10 * 1000)       // 10 seconds, in milliseconds
  .setFastestInterval(1 * 1000); // 1 second, in milliseconds

 if (mGoogleApiClient == null) {
  mGoogleApiClient = new GoogleApiClient.Builder(this)
   .addConnectionCallbacks(this)
   .addOnConnectionFailedListener(this)
   .addApi(LocationServices.API)
   .build();
 }

 mGoogleApiClient.connect();
}
```

Vous aurez également remarqué la création de l'objet *LocationRequest* utilisé pour configurer les données de localisation que nous voulons requêter auprès de l'API. Lorsque la connexion à l'API Fused Location Provider réussit, le callback *onConnected* est invoqué. Dans cette méthode, nous pouvons alors effectuer une requête pour obtenir la mise à jour de la dernière position connue de l'utilisateur en appelant la méthode statique *requestLocationUpdates* de l'objet *FusedLocationApi* comme suit :

```
@Override
public void onConnected(@Nullable Bundle bundle) {
 isConnected = true;

 LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient, mLocation
Request, new LocationListener() {
  @Override
  public void onLocationChanged(Location location) {
   mLastLocation = location;
   LocationServices.FusedLocationApi.removeLocationUpdates(mGoogleApiClient, this);

   if (mLastLocation != null) {
    processLoadingWeatherTemperature();
   } else {
    Toast.makeText(MainActivity.this, R.string.error_during_location_update, Toast.
LENGTH_SHORT).show();
   }
  }
 });
}
```

Appel de l'API OpenWeatherMap

Une fois la dernière position obtenue, nous la stockons dans notre activité puis nous appelons la méthode *processLoadingWeather-Temperature* en charge de la récupération de la température courante pour la station météo la plus proche de cette position. Cette récupération se faisant par un appel du Web Service météo d'OpenWeatherMap.

Pour appeler ce Web Service et récupérer la température, nous allons utiliser la bibliothèque open source OkHttp de Square. Nous construisons donc un objet *Request* avec l'URL de notre Web Service en paramètre puis créons un nouvel appel via la méthode *newCall* de l'instance d'objet *OkHttpClient* créée. Enfin, nous exécutons l'appel via la méthode *execute*. Ce dernier appel nous permet d'obtenir un objet *Response* représentant la réponse de l'appel au Web Service météo. La réponse renvoyée par OpenWeatherMap est au format JSON et il nous faut donc construire un objet JSONObject avec celle-ci. La suite consiste simplement à accéder à la valeur de la propriété temp correspondant à la température de la station météo la plus proche de la position GPS envoyée en paramètre du Web Service :

```
public class Weather {

 public static final String API_KEY; // à renseigner
 public static final String OPEN_WEATHER_URL = "http://api.openweathermap.org/
data/2.5/weather?lat=$lat$&lon=$lon$&appid=$appid$";
 private OkHttpClient okHttpClient;

 public Weather() {
  okHttpClient = new OkHttpClient();
 }

 public Double getTemperature(float lat, float lng) {
  Double temperature = null;
  String apiKey = apiKey();
  String url = OPEN_WEATHER_URL.replace("$lat$", lat + "")
   .replace("$lon$", lng + "")
   .replace("$appid$", API_KEY);

  Request request = new Request.Builder()
   .url(url)
   .build();

  try {
   Response response = okHttpClient.newCall(request).execute();
   String json = response.body().string();
   JSONObject jsonObject = new JSONObject(json);

   if (jsonObject.getInt("cod") == 200) {
    JSONObject mainObject = jsonObject.getJSONObject("main");

    if (mainObject != null) {
     double value = mainObject.getDouble("temp") - 273.15f;
     temperature = value;
    }
   }

  } catch (Exception e) { }

  return temperature;
 }
}
```

Au niveau de l'activité principale de notre application Thermomètre, il est bon de souligner que l'appel à la méthode *getTemperature* de l'objet *Weather* doit se faire au sein d'un Thread dédié. Une fois la température obtenue, il suffit d'appliquer la nouvelle valeur sur le composant graphique *Thermometer* via sa méthode *setCurrentTemp* comme vu précédemment. On obtient donc le code suivant pour la méthode *processLoadingWeathetTemperature* :

```
private void processLoadingWeatherTemperature() {
 final float lat = (float) mLastLocation.getLatitude();
 final float lng = (float) mLastLocation.getLongitude();

 progressDialog = new MaterialDialog.Builder(this)
  .title(R.string.loading_weather_temperature)
  .content(R.string.wait)
  .progress(true, 0)
  .show();

 new Thread(new Runnable() {
  @Override
  public void run() {

   Weather weather = new Weather();
   final Double temperature = weather.getTemperature(lat, lng);

   runOnUiThread(new Runnable() {
    @Override
    public void run() {
```

```
        Utils.closeDialog(progressDialog);

        if (temperature != null) {
          thermometer.setCurrentTemp(temperature.floatValue());
        } else {
        Toast.makeText(MainActivity.this, R.string.no_weather_temperature, Toast.LENGTH
_LONG).show();
        }
       }
     });
   }
 }).start();
}
```

Composant graphique Thermometer

Un affichage textuel de la température serait tout à fait convenable mais manquerait cependant d'un brin de folie pour les utilisateurs de notre application. Dans le but de rendre l'application plus attrayante visuellement parlant, nous avons expliqué précédemment que nous utilisions plutôt un composant graphique *Thermometer* offrant aux utilisateurs une vue graphique de la température mesurée via l'une des différentes sources proposées.

Nous avons fait le choix de créer un composant *Thermometer* dédié héritant de la classe *View*. Notre composant personnalisé propose une méthode *setCurrentTemp* permettant de lui passer en entrée la température à afficher. Dans cette méthode, on prendra soin de conserver la valeur de la température courante entre les bornes minimum et maximum de notre composant afin d'éviter d'avoir un dépassement de la couleur rouge (représentant le mercure) vers le bas ou vers le haut. Lorsque cette méthode est appelée, il est nécessaire de redessiner le composant en appelant sa méthode *invalidate*. Pour le reste, le gros du travail pour notre composant est réalisé au sein de sa méthode *onDraw*. C'est dans cette dernière que nous dessinons les différentes parties du thermomètre que nous souhaitons afficher.

Un travail loin d'être extraordinaire mais qui demande une précision certaine comme vous pouvez le constater dans le code ci-dessous :

```
public class Thermometer extends View {

 // ...

 public void setCurrentTemp(float currentTemp) {
  if (currentTemp > maxTemp) {
   this.currentTemp = maxTemp;
  } else if (currentTemp < minTemp) {
   this.currentTemp = minTemp;
  } else {
   this.currentTemp = currentTemp;
  }

  invalidate();
 }

 // ...

 @Override
 protected void onDraw(Canvas canvas) {
  super.onDraw(canvas);
  int height = getHeight();
  int width = getWidth();

  int CircleCenterX = width / 2;
  float CircleCenterY = height - outerCircleRadius;
  float outerStartY = 0;
  float middleStartY = outerStartY + 5;

  float innerEffectStartY = middleStartY + middleRectRadius + 10;
  float innerEffectEndY = CircleCenterY - outerCircleRadius - 10;
  float innerRectHeight = innerEffectEndY - innerEffectStartY;
  float innerStartY = innerEffectStartY + (maxTemp - currentTemp) / rangeTemp * inner
RectHeight;

  RectF outerRect = new RectF();
  outerRect.left = CircleCenterX - outerRectRadius;
  outerRect.top = outerStartY;
  outerRect.right=CircleCenterX + outerRectRadius;
  outerRect.bottom = CircleCenterY;

  canvas.drawRoundRect(outerRect, outerRectRadius, outerRectRadius, outerPaint);
  canvas.drawCircle(CircleCenterX, CircleCenterY, outerCircleRadius, outerPaint);

  RectF middleRect = new RectF();
  middleRect.left = CircleCenterX - middleRectRadius;
  middleRect.top = middleStartY;
  middleRect.right=CircleCenterX + middleRectRadius;
  middleRect.bottom = CircleCenterY;

  canvas.drawRoundRect(middleRect, middleRectRadius, middleRectRadius, middlePaint);
  canvas.drawCircle(CircleCenterX, CircleCenterY, middleCircleRadius, middlePaint);
  canvas.drawRect(CircleCenterX - innerRectRadius, innerStartY, CircleCenterX + innerRect
Radius, CircleCenterY, innerPaint);
  canvas.drawCircle(CircleCenterX, CircleCenterY, innerCircleRadius, innerPaint);

  // Traçage des graduations
  float tmp = innerEffectStartY;
  float startGraduation = maxTemp;
  float inc = rangeTemp / nbGraduations;

  while (tmp <= innerEffectEndY) {
   canvas.drawLine(CircleCenterX - outerRectRadius - DEGREE_WIDTH, tmp,
    CircleCenterX - outerRectRadius, tmp, degreePaint);
   String txt = ((int)startGraduation) + "°";
    graduationPaint.getTextBounds(txt, 0, txt.length(), rect);
   float textWidth = rect.width();
   float textHeight = rect.height();

   canvas.drawText(((int) startGraduation) + "°",
    CircleCenterX - outerRectRadius - DEGREE_WIDTH - textWidth - DEGREE_WIDTH * 1.5f,
     tmp + textHeight / 2, graduationPaint);
   tmp += (innerEffectEndY - innerEffectStartY) / nbGraduations;
```

```
    startGraduation -= inc;
  }
 }
}
```

Notre Thermomètre en action

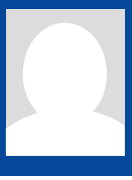
Le code de notre application terminé, il est temps de passer au test de notre Thermomètre pour Android. Une fois l'application lancée, il est tout d'abord demandé de choisir une source de données pour la température (2). Ensuite, nous pouvons prendre connaissance de la valeur de la température graphiquement avec le composant *Thermometer* créé sur mesure pour l'occasion (3).

Conclusion

La puissance toujours plus grande des appareils Android ainsi que le nombre important de capteurs mis à disposition offre aux développeurs toujours plus de possibilités pour créer des applications originales et utiles. Dans cet article, nous avons ainsi vu comme s'appuyer sur le capteur de température ambiante d'un appareil Android pour réaliser une application Thermomètre. Ce capteur devenant de plus en plus rare, nous avons vu comment proposer d'autres sources de température aux utilisateurs afin de conserver l'attrait de l'application. En outre, la partie présentant la récupération de la température météo constitue une introduction très instructive concernant la consommation de Web Services sous Android. Enfin, il est bon de signaler que l'application réalisée dans cet article peut être testée directement sur vos appareils Android en la téléchargeant sur le Google Play Store :
https://play.google.com/store/apps/details?id=com.ssaurel.thermometer .

2 *Choix d'une source de données*

3 *Température Météo*



Eugene Shen, Vincent Bechu,
François Le Corre et Sven Franck

Partie 2

OfficeJS : réaliser une Progressive Offline Web Application simple avec RenderJS et jIO

OfficeJS est une suite bureautique incluant plusieurs applications HTML5 : traitement de texte, présentation, tableur, illustration, traitement d'images, etc. La suite est compatible avec les principaux standards bureautiques et fonctionne en mode hors ligne (mode offline). L'appstore d'OfficeJS est aussi le cœur de la nouvelle interface d'ERP5, un ERP/CRM libre. Ces solutions sont publiées par Nexedi. Aujourd'hui, nous allons expliquer comment créer une application HTML5 avec OfficeJS, qui fonctionne sans connexion réseau en utilisant le concept du Progressive Offline Web Application (POWA).

jIO en action

Retournons tester ceci sur notre application. Remplaçons le code de addItem() dans le fichier index.js par celui ci-dessous :

```
.declareMethod("addItem", function (item) {
  var gadget = this;
  return gadget.getDeclaredGadget("model")
    .push(function (model_gadget) {
      return model_gadget.post({
        title: item,
        completed: false
      });
    })
    .push(function () {
      return gadget.changeState({update: true});
    });
})
```

et le code de onEvent par celui ci-dessous :

```
.onEvent("submit", function (event) {
  var item = event.target.elements[0].value.trim();
  event.target.elements[0].value = "";
  if (item) {
    return this.addItem(item);
  }
}, false, true);
```

Ainsi lorsque l'utilisateur va envoyer le formulaire, jIO, via le model_gadget, il va enregistrer le document avec la méthode post, qui crée l'id automatiquement. Mais si vous testez que cela fonctionne bien, vous trouverez, que Capacity "Post" is not implemented on "indexeddb". C'est dommage. Il est nécessaire de modifier la définition du stockage dans createJIO :

```
storage: jIO.createJIO({
  type: "query",
  sub_storage: {
    type: "uuid",
    sub_storage: {
      type: "indexeddb",
      database: "todos-renderjs"
    }
  }
})
```

Deux nouveau types de stockage jIO sont utilisés: "uuid" qui permet d'utiliser post() avec un id généré automatiquement, et "query" qui permet d'utiliser tout le potentiel de allDocs(). La fonction allDocs() peut prendre en option : sort_on: [["property et", "descending"]] qui permet de trier les résultats, limit: [1,5] qui permet de limiter le nombre de résultats, query: property et : %foo' qui permet de faire une recherche dans la base de données, select_list: ["property"] permettant de retrouver cette propriété dans data.row[i].value.property et include_docs: true pour retourner les fichiers complétés. Rafraîchissez votre navigateur et inspectez directement dans l'indexedDB nommé jio:todos-renderjs si les documents sont bien créés. Voyons maintenant une première utilisation de allDocs(), toujours dans index.js en remplaçant onStateChange() :

```
.onStateChange(function (modification_dict) {
  var gadget = this;
  return this.getDeclaredGadget("model")
    .push(function (model) {
      model_gadget = model;
      return model_gadget.allDocs();
    })
    .push(function (result_list) {
      var promise_list = [], i;
      for (i = 0; i < result_list.data.total_rows; i += 1) {
          promise_list.push(model_gadget.get(result_list.data.rows[i].id) );
      }
      return RSVP.all(promise_list);
    })
    .push(function (result_list) {
      var i, plural, li;
      gadget.element.querySelector("ul").innerHTML = "";
      for (i = 0; i < result_list.length; i += 1) {
        li = document.createElement("li");
        li.textContent = result_list[i].title;
        gadget.element.querySelector("ul").appendChild(li);
      }
      gadget.state.update = false;
    });
});
```

et remplaçant declareService par :

```
.declareService(function () {
  this.element.querySelector("p").textContent = "Hello, world!";
```

```
 return this.changeState({"update": true});
})
```

Ce bout de code permet à la fin d'initialiser la liste des items déjà présents dans la base de données de l'application. Ainsi on voit que allDocs() nous renvoie un object de la forme {data: {rows: [{id: id_du_document, value: {}}, ...], total_rows: le_nombre_de_document }} , cette méthode renvoie l'ensemble des ids des documents présents dans la base de données. Nous pouvons alors utiliser model_gadget.get() pour récupérer les documents : cette méthode renvoie une promesse que l'on stocke dans la liste promise_list. On utilise ensuite RSVP.all qui exécute une liste de promesses en parallèle puis renvoie le résultat une fois que toutes les promesses sont terminées.

Utiliser Handlebars

Après avoir fini d'implémenter les fonctionnalités basiques, il faut mettre à jour les feuilles de style pour avoir un visuel ressemblant à celui de ToDoMVC. Téléchargeons les feuilles Handlebars.js, base.css et index.css et remplacez index.html par le suivant :

```
<!doctype html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>OfficeJS App</title>
  <script src="rsvp.js"></script>
  <script src="renderjs.js"></script>
  <script src="jio.js"></script>
  <script src="index.js"></script>
  <script src="handlebars.js"></script>
  <link href="base.css" rel="stylesheet">
  <link href="index.css" rel="stylesheet">

  <script class="handlebars-template" type="text/x-handlebars-template">
  <section class="todoapp">
   <header class="header">
    <h1>todos</h1>
    <form>
     <input class="new-todo" placeholder="What needs to be done?" autofocus>
    </form>
   </header>
   <section class="main {{#unless todo_exists}}hidden{{/unless}}">
    <input class="toggle-all" type="checkbox" {{#if all_completed}}checked="true"{{/if}}>
    <label for="toggle-all" class="toggle-label">Mark all as complete</label>
    <ul class="todo-list">
     {{#each todo_list}}
      <li class="todo-item {{#if this.completed}}completed{{/if}} {{#if this.editing}}editing{{/if}}">
       <div class="view {{#if this.edit}}hidden{{/if}}">
        <input class="toggle" type="checkbox"{{#if this.completed}} checked="true"{{/if}}>
        <label class="todo-label">{{this.title}}</label>
        <button class="destroy"></button>
       </div>
       <input class="edit{{#unless this.editing}} hidden{{/unless}}">
      </li>
     {{/each}}
    </ul>
   </section>
   <footer class="footer {{#unless todo_exists}}hidden{{/unless}}">
    <span class="todo-count">{{todo_count}}</span>
    <div class="filters">
     <a href="#/" class="selected">All</a>
     <a href="#/active">Active</a>
     <a href="#/completed">Completed</a>
    </div>
    <button class="clear-completed">Clear completed</button>
   </footer>
  </section>
  <footer class="info">
   <p>Double-click to edit a todo</p>
  </footer>
 </script>
</head>
<body>
 <div data-gadget-url="gadget_model.html"
  data-gadget-scope="model"
  data-gadget-sandbox="public">
 </div>
 <main class="handlebars-anchor">
 </main>
</body>
</html>
```

Notez que la propriété id n'est jamais utilisée. Les gadgets doivent être réutilisables et encapsulables ; avoir deux gadgets ayant tous deux une balise avec un id commun empêcherait leur utilisation côte à côte. Déclarez var handlebars_template; comme variable globale entre "use strict" et rJS(window). Les templates d'Handlebars doivent être "compilés" avant toute modification du DOM dans declareService():

```
.declareService(function () {
 var gadget = this;
 handlebars_template = Handlebars.compile(    document.head.querySelector(".hand
lebars-template").innerHTML
 );
 return this.changeState({"update": true});
})
```

Effacez item_list: [],dans setState() et modifiez la dernière section de onStateChange()par :

```
.push(function (result_list) {
 var i, plural, item_list = [];
 for (i = 0; i < result_list.length; i += 1) {
  item_list.push({"title": result_list[i].title});
 }
 plural = item_list.length === 1 ? " item" : " items";
 gadget.element.querySelector(".handlebars-anchor").innerHTML = handlebars_template({
  todo_list: item_list,
  todo_exists: item_list.length >= 1,
  todo_count: item_list.length.toString() + plural,
  all_completed: false
 });
 gadget.state.update = false;
});
```

Rafraîchissez votre navigateur - les bases de l'application TodoMVC sont bien créées. L'app n'est pas complète, mais il est possible de définir et regarder des tâches.

Acquisition

Comme nous le savons maintenant les gadgets RenderJS doivent être auto-

nomes, mais il est possible de faire communiquer plusieurs gadgets, deux méthodes RenderJS sont alors utilisées :

```
allowPublicAcquisition("nomMéthode", function)
```

qui nous permet de rendre accessible une méthode aux gadgets enfants et

```
declareAcquiredMethod("nomMethodeFille", "nomMethodeMere")
```

qui elle, permet d'accéder à une méthode rendue accessible par un gadget parent. Pour notre application, dans le gadget index.js :

```
.allowPublicAcquisition("jio_allDocs", function (option) {
  return this.getDeclaredGadget("model")
    .push(function (model_gadget) {
      return model_gadget.allDocs(option);
    });
})
```

Ainsi dans n'importe quel sous-gadget de index.html, comme dans un routeur, nous pouvons utiliser la méthode jio_allDocs() comme .declareAcquiredMethod("jio_allDocs", "allDocs") et l'utiliser ainsi dans ce gadget avec gadget.allDocs();

Intégrer un gadget avec JavaScript

Il est possible d'intégrer un gadget dans un autre avec JavaScript seulement, et ce n'est pas plus compliqué qu'en HTML. Par exemple :

```
.declareService(function () {
  var gadget = this,
    template = gadget.element.querySelector(".handlebars-template"),
    div = document.createElement("div");

  gadget.element.appendChild(div);
  handlebars_template = Handlebars.compile(
    document.head.querySelector(".handlebars-template").innerHTML
  );
  return gadget.declareGadget("gadget_routeur.html", {
    scope: "router",
    sandbox: "public",
    element: div
  })
  .push(function (router) {
    console.log(router);
    // faire quelque chose avec le routeur
    return this.changeState({"update": true});
  });
})
```

Comme cela, il est possible d'ajouter des gadgets de façon dynamique.

Gadget Routeur

Le but du routeur est de contrôler la navigation dans l'application. Pour commencer il est possible d'écouter l'évènement "hashchange" sur l'objet window ; on ne peut pas utiliser le onEvent de RenderJS, puisqu'il se limite au gadget. Puis on peut utiliser le chargement dynamique des gadgets jIO pour changer de page ou de gadget. Et pour communiquer avec les autres gadgets, nous venons juste d'apprendre comment faire avec l'acquisition. Chargez le gadget_global.js et ajoutez dans index.html. Déclarez une deuxième classe service :

```
.declareService(function () {
  var listener = window.loopEventListener;
  function handleHash(event) {
    console.log(event)
  }
  return listener(window, "hashchange", false, handleHash);
})
```

et ajoutez #123 à la fin de l'url. A vous de jouer.

Application déconnectée

Maintenant que vous avez une application qui fonctionne correctement, il serait agréable de pouvoir en profiter même loin de tout réseau. Pour cela nous utiliserons un "service worker". Créons un fichier serviceworker.js avec le code suivant :

voir code sur www.programmez.com

Pour plus de détails sur les "service worker", regardez :

https://developer.mozilla.org/fr/docs/Web/API/Service_Worker_API/Using_Service_Workers

Pour l'installer dans votre application, il suffit d'ajouter dans index.js la fonction declareService:

```
.push(function () {
  if (navigator.serviceWorker) {
    return navigator.serviceWorker.register("serviceworker.js");
  }
})
```

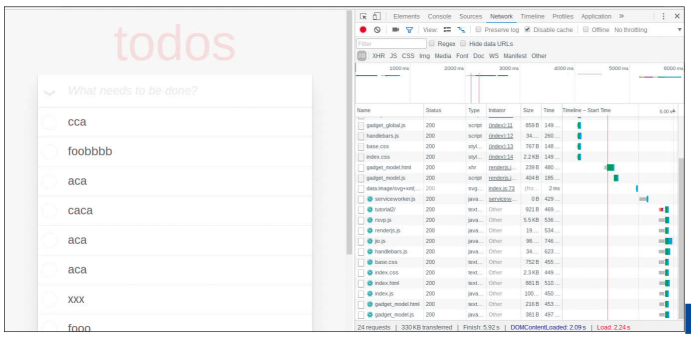
Maintenant vous pouvez recharger votre application, vous éloigner de votre réseau et continuer d'utiliser votre application. 3

De plus, en ajoutant un manifeste JSON, il est possible de transformer notre application en application web progressive, c'est à dire que notre application pourra être portée sur android comme une application native. Il vous faut aussi une image pour l'icône de l'application. Voici un exemple de fichier manifest.json :

```
{
  "short_name": "OfficeJS Todos",
  "name": "OfficeJS Todo List - Sample Tutorial Application",
  "icons": [{
    "src": "launcher_icon.png",
    "sizes": "any",
    "type": "image/png"
  }],
  "start_url": "./",
  "display": "standalone"
}
```

Conclusion

C'est fait! Félicitations, vous venez de construire une application web modulaire, réactive et progressive, qui peut être utilisée avec n'importe quel type de base de données et sans internet ! Vous pouvez jeter un oeil à notre implémentation sur https://github.com/tastejs/todomvc •



3

Estelle Auberix
Entrepreneure
Consultante Cloud & IoT
Speaker, MVP Azure

OpenFaas sur un Cluster Raspberry Pi

Partie 1

Cet article est destiné à une audience sans connaissances préalables sur Raspberry Pi, Docker, Kubernetes et OpenFaaS. Merci à Alex Ellis et aux contributeurs de la communauté d'OpenFaaS pour leur travail incroyable.

Depuis bientôt 5 ans, les projets de clusters de Raspberry Pi fleurissent mais dans quel but ? Outre l'aspect ludique du montage, on peut évoquer un intérêt au niveau du coût puisque ce type de montage permet d'envisager la construction d'un super computer, voire d'un super calculateur. Pour exemple, Joshua KIEPERT, alors étudiant à l'Université de Boise (Idaho), avait monté un super calculateur en 2013 basé sur 33 Raspberry Pi. Il obtenait ainsi une puissance de calcul de 10,13 GFlops pour moins de 2000 $ alors que le Cray-2, premier super calculateur en 1985, autorisait seulement 1,9 GFlops. 1

En novembre dernier, un cluster de 3 000 cœurs basé sur des Raspberry Pi a été créé dans le cadre du concours de BitScope Designs (Los Alamos) pour tester le code des chercheurs. Il s'agit d'un cluster modulaire qui rassemble 750 nœuds Raspberry Pi 3 Model B regroupés dans 5 modules montés en rack (150 cartes Raspberry Pi avec des commutateurs réseaux intégrés). A noter également un gain énergétique puisque cette émulation de super ordinateur basé sur ARM ne consomme que 2 à 3 watts par processeurs, soit environ 1 000 watts en mode inactif et 3 à 4 000 watts en mode normal.

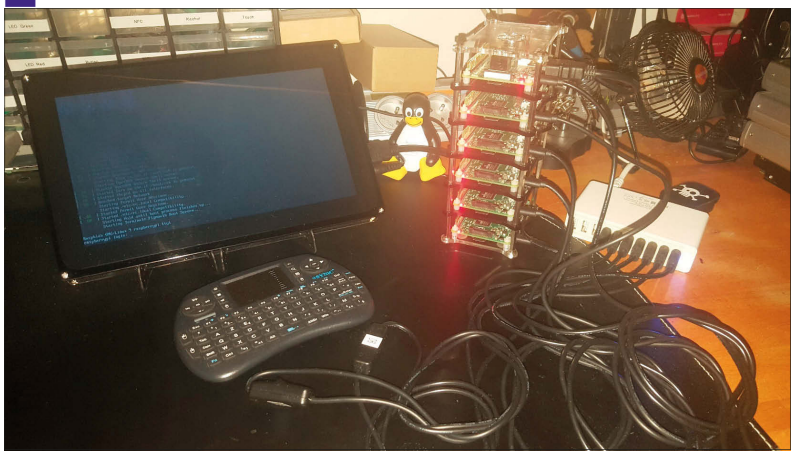
Dans le cadre de ce lab, nos ambitions seront de moindre importance mais le raisonnement reste similaire. Nous effectuerons un montage sur un cluster de 6 Raspberry Pi (1 RPi3 pour le master et 5 RPi2 avec Dongle pour les workers car c'est le matériel que j'avais sous la main).

Note

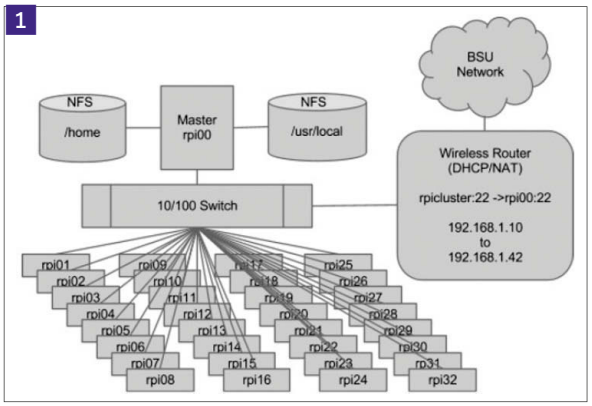
Par ailleurs, apprendre comment monter un Cluster de Raspberry Pi avec Kubernetes et OpenFaaS est un bon moyen d'avoir son propre environnement de Lab Serverless.

Avertissement : nous travaillons dans un environnement qui évolue très vite, donc ce lab est susceptible d'évoluer et d'être mis à jour en fonction des nouvelles versions des logiciels. Je vous invite à surveiller le blog d'Alex pour connaître les dernières avancées : https://www.alexellis.io/.

2



1



Prérequis

- 3 Raspberry Pi 2/3 ou plus ;
- 1 alimentation secteur par Raspberry Pi (USB ou secteur) ;
- 1 micro SD Card par Raspberry Pi (16Go min., 32Go si vous pouvez – Plus la carte est rapide, meilleur c'est !) ;
- 1 petit routeur Ethernet + autant de câbles RJ45 que de Raspberry Pi (optionnel si vous ne pouvez pas contrôler votre connexion WiFi) ;
- 1 écran HDMI + câble (optionnel mais fortement conseillé) ;
- 1 clavier USB (optionnel).

Coûts du matériel 2

6 Raspberry Pi 2/3	Environ 210 Euros
6 Câbles USB avec interrupteur	Environ 10 Euros (vendus par 10 sur AliExpress)
1 chargeur USB 10 ports	Environ 22 Euros (10 ports sur AliExpress)
6 cartes micro SD	Environ 60 Euros
1 écran HDMI + câble	Environ 100 Euros
1 mini clavier WiFi	Environ 15 Euros
1 rack 6 emplacements	Environ 30 Euros

Installation des Raspberry Pi

Paramétrage manuel

1 - Télécharger Raspbian Stretch Lite

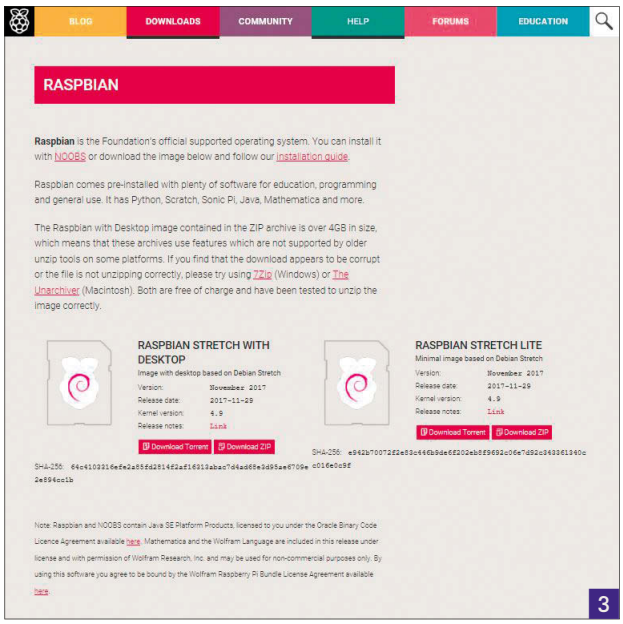
(https://www.raspberrypi.org/downloads/raspbian/) 3

2 – Installer Raspbian sur la micro SD Card

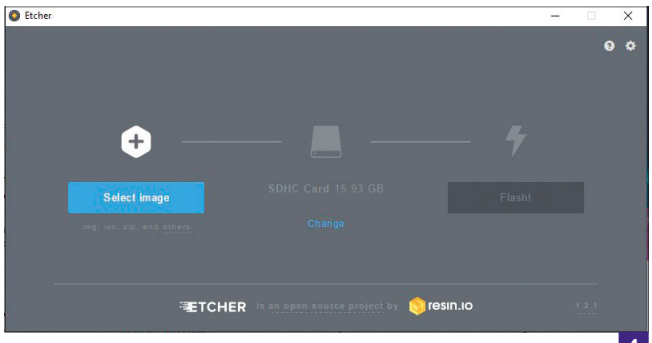
Comme préconisé sur le site de Raspberrypi.org, je vous conseille Etcher (https://etcher.io/) mais vous pouvez choisir dd command (Linux/Mac), ApplePi Baker (Mac) or Win32DiskImager (Windows) si vous préférez. 4 5 6

3 – Activez le SSH

Sans affichage écran, vous pouvez facilement activer le SSH en ajoutant un fichier nommé SSH sans extension, sur la partition 'boot' de la micro SD carte. 7



3



4

Avec affichage écran, connectez-vous avec les login/mot de passe par défaut (pi/raspberry) puis accédez au menu de configuration avec la commande suivante :

```
sudo raspi-config
```

Interfacing Options \ P2 SSH 8

4 – Téléchargez et installez Putty (https://www.putty.org/).

Paramétrage du réseau

Configuration de l'adresse IP

5 - Vérifiez votre adresse IP

Connectez votre RPi à votre routeur avec un câble réseau.

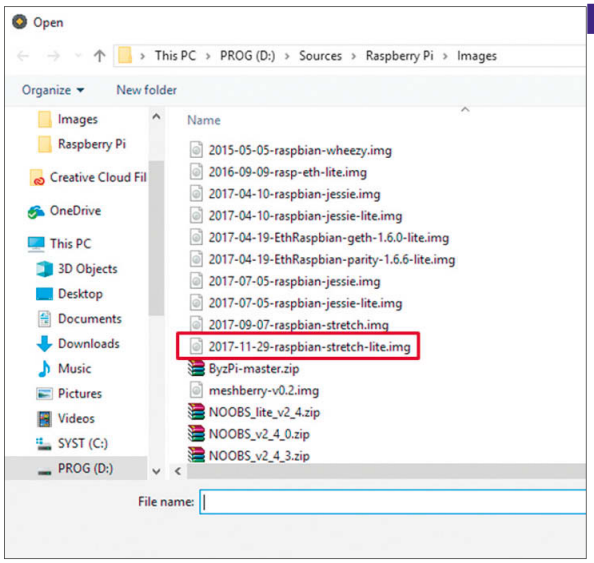
Sans affichage écran, vous pouvez trouver l'adresse IP de votre RPi en regardant la table d'allocation DHCP des adresses IP sur votre routeur (généralement accessible en 192.168.1.1 + mot de passe attribué par votre fournisseur internet) ou via un logiciel tel que IP scanner par exemple (http://www.advanced-ip-scanner.com/).

Avec un écran HDMI (TV ou autre)

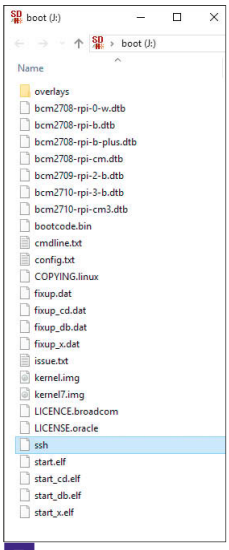
L'adresse IP est affichée quelques lignes au-dessus de l'invitation de login. Sinon, vous pouvez trouver votre IP avec la commande :

```
hostname -I
```

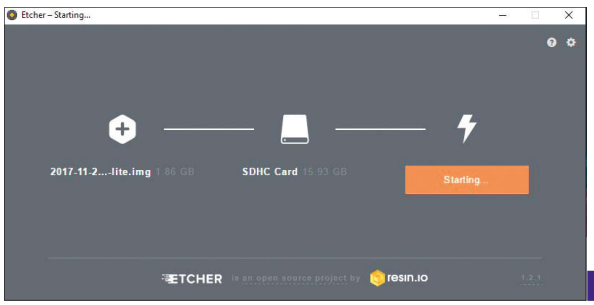
Si vous souhaitez effectuer toutes les étapes suivantes en tant que root, tapez



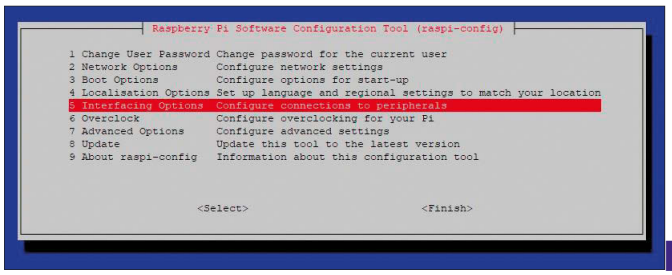
5



7



6



8

```
sudo -i
```

6 – Configuration réseau

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Ajouter à la fin de votre fichier (après avoir modifié SSID NAME et WIFI PASSWORD)

```
network={
 ssid="SSID NAME"
 psk="WIFI PASSWORD"
 key_mgmt=WPA-PSK
}
```

Redémarrez votre RPi puis vérifiez l'adresse IP

```
ifconfig wlan0
hostname -I
```

7 – Fixez l'adresse IP

Paramétrez le fichier dhcpcd.conf

```
sudo nano /etc/dhcpcd.conf
```

Après #interface eth0, collez le bloc suivant

```
interface wlan0
static ip_address=192.168.0.100/24
static routers=192.168.0.1
static domain_name_servers=8.8.8.8
```

⚠ Vous pouvez incrémenter les IPs statiques avec 100 puis 101, 102, 103 etc. Vous pouvez également faire une réservation d'IP sur la table DHCP de votre routeur afin que ces adresses ne soient pas attribuées à d'autres appareils dans votre réseau.

8 – Changez le hostname

Utilisez l'utilitaire raspi-config pour changer le hostname en k8s-master-001 ou autre, puis redémarrez.

Installation de Docker & Kubernetes

Rak8s (prononcez 'rackets' : https://rak8s.io/) est une automatisation de installation via Ansible mais l'intérêt premier de ce cluster étant d'apprendre et de comprendre les fondations d'OpenFaaS, voici la procédure manuelle d'installation.

9 – Mise à jour des paquets

Pour mettre à jour les paquets de vos RPis, utilisez les commandes suivantes :

```
sudo apt-get update && \
sudo apt-get upgrade
```

10 – Installez Docker

```
curl -sSL https://get.docker.com | sh
```

Vérifiez la version de Docker

docker --version

```
pi@master130:~ $ docker --version
Docker version 18.02.0-ce, build fc4de44
pi@master130:~ $
```

11 – Ajoutez la permission utilisateur à votre RPi pour les commandes de Docker

```
sudo usermod pi -aG docker
```

12 – Désactiver le swap

Pour Kubernetes 1.7 et supérieur, vous obtiendrez une erreur si l'espace swap est activé.

Arrêtez swap:

```
sudo dphys-swapfile swapoff
sudo dphys-swapfile uninstall && \
sudo update-rc.d dphys-swapfile remove
```

Vérifiez que vous n'avez plus d'entrées

```
sudo swapon --summary
```

13 – Changez cmdline.txt

```
sudo nano /boot/cmdline.txt
```

Ajoutez le texte suivant à la fin de la ligne, mais ne créez pas de nouvelle ligne :

```
cgroup_enable=cpuset cgroup_enable=memory cgroup_memory=1
```

14 – Redémarrez

```
sudo reboot
```

⚠ N'oubliez pas cette étape

15 – Vérifiez votre installation Docker

```
docker run armhf/hello-world
```

```
pi@master130:~ $ docker run armhf/hello-world
Unable to find image 'armhf/hello-world:latest' locally
latest: Pulling from armhf/hello-world
a0691bf12e4e: Pull complete
Digest: sha256:9701edc932223a66e49dd6c894a11db8c2cf4eccd1414f1ec105a623bf16b426
Status: Downloaded newer image for armhf/hello-world:latest

Hello from Docker on armhf!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
 https://hub.docker.com

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

16 – Installez Kubeadmin

kubeadm est un outil qui vous aide à démarrer votre Cluster et Kubernetes avec de bonnes pratiques. C'est facile, passablement sécurisé et évolutif. Vous pourrez suivre ce Lab même si vous n'avez aucune connaissance sur Kubernetes et Kubeadmin, mais je vous invite à approfondir vos connaissances sur le sujet. N'hésitez pas à lire la documentation de Kubernetes pour comprendre toutes les possibilités de cet outil (https://kubernetes.io/docs/).

Ajoutez la repo list

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add - && \
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.
list.d/kubernetes.list && \
sudo apt-get update -q && \
sudo apt-get install -qy kubeadm
```

17 – Redémarrez votre RPi

```
sudo reboot
```

18 – Initialisez le master Kubernetes

```
sudo nano kubeadm_conf.yaml
```

Copiez et insérez le texte suivant

```
apiVersion: kubeadm.k8s.io/v1alpha1
kind: MasterConfiguration
controllerManagerExtraArgs:
 horizontal-pod-autoscaler-use-rest-clients: "true"
 horizontal-pod-autoscaler-sync-period: "10s"
 pod-eviction-timeout: 10s
 node-monitor-grace-period: 10s
apiServerExtraArgs:
 runtime-config: "api/all=true"
```

Sauvegardez et lancez.

```
sudo kubeadm init --config kubeadm_conf.yaml
```

Cette étape prend un peu de temps (15 à 20' environ) donc vous avez le temps d'aller boire un petit café. Vous devez obtenir un

écran similaire à celui-ci :

```
Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each node
as root:

  kubeadm join --token e95460.9306aca90eb332a1 10.10.10.130:6443 --discovery-token-ca-cert-hash sha256:2e467de20b8427
00f5eb79befalfa8759fa1cafd235831d0279ad4b3a3ee5dbd

pi@master130:~ $
```

En sélectionnant le texte de kubeadmin join… jusqu'à la fin de la clé sha256, enregistrez la commande kubeadm join qui vous est présentée par kubeadm init. Vous en aurez besoin dans un moment. Pour faire fonctionner kubectl pour un utilisateur non-admin, vous devez utiliser ces commandes :

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Vous pouvez vérifier que votre nœud maître (master node) est monté et fonctionne

```
kubectl get nodes
```

```
pi@master130:~ $ kubectl get nodes
NAME        STATUS     ROLES     AGE       VERSION
master130   NotReady   master    3m        v1.9.3
pi@master130:~ $
```

Ne faites pas attention au statut 'NotReady' à ce stade de l'installation. Afin que ce nœud maître soit prêt, vous avez besoin d'installer un réseau pour les conteneurs.

19 – Paramétrez le réseau des conteneurs

Les nœuds ont besoin d'un réseau pour communiquer avec le master node, lancez la commande suivante :

```
kubectl apply -f https://git.io/weave-kube-1.6
```

20 – Paramétrez les noeuds worker

Ajoutons quelques noeuds supplémentaires au cluster.
Partant du principe que votre master node fonctionne correctement, vous pouvez lancer la commande suivante sur toutes vos autres RPi qui seront des nœuds worker :

```
sudo kubeadm join --token TOKEN 192.168.1.100:6443 --discovery-token-ca-cert-hash HASH
```

(reprendre la commande que vous avez copiée à l'étape 18)

```
pi@worker121:~ $ sudo kubeadm join --token e95460.9306aca90eb332a1 10.10.10.130:6443 --discovery-token-ca-cert-has
h sha256:2e467de20b842700f5eb74befalfa8759fa1cafd235831d0279ad4b3a3ee5dbd
[preflight] Running pre-flight checks.
        [WARNING SystemVerification]: docker version is greater than the most recently validated version. Docker v
ersion: 18.02.0-ce. Max validated version: 17.03
        [WARNING FileExisting-crictl]: crictl not found in system path
[discovery] Trying to connect to API Server "10.10.10.130:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://10.10.10.130:6443"
[discovery] Requesting info from "https://10.10.10.130:6443" again to validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will
 use API Server "10.10.10.130:6443"
[discovery] Successfully established connection with API Server "10.10.10.130:6443"

This node has joined the cluster:
* Certificate signing request was sent to master and a response
  was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.

pi@worker121:~ $
```

```
pi@master130:~ $ kubectl get nodes
NAME        STATUS    ROLES     AGE       VERSION
master130   Ready     master    1h        v1.9.3
worker121   Ready     <none>    54m       v1.9.3
worker122   Ready     <none>    29m       v1.9.3
worker123   Ready     <none>    26m       v1.9.3
worker124   Ready     <none>    16m       v1.9.3
worker125   Ready     <none>    6m        v1.9.3
pi@master130:~ $
```

Vérifiez que tout fonctionne correctement :

```
kubectl get pods --namespace=kube-system
```

```
pi@master130:~ $ kubectl get pods --namespace=kube-system
NAME                                READY     STATUS    RESTARTS   AGE
etcd-master130                      1/1       Running   0          1h
kube-apiserver-master130            1/1       Running   0          1h
kube-controller-manager-master130   1/1       Running   0          1h
kube-dns-7b6ff86f69-hzzcf           3/3       Running   0          1h
kube-proxy-5mltz                    1/1       Running   0          30m
kube-proxy-bgtrb                    1/1       Running   0          1h
kube-proxy-bkvql                    1/1       Running   0          7m
kube-proxy-p7vrr                    1/1       Running   0          55m
kube-proxy-q8khv                    1/1       Running   0          17m
kube-proxy-zvbn6                    1/1       Running   0          27m
kube-scheduler-master130            1/1       Running   0          1h
weave-net-9dhzz                     2/2       Running   0          55m
weave-net-cvtzt                     2/2       Running   1          27m
weave-net-hpgkz                     2/2       Running   0          58m
weave-net-lwvls                     2/2       Running   0          17m
weave-net-vgjt4                     2/2       Running   1          30m
weave-net-x7644                     2/2       Running   5          7m
pi@master130:~ $
```

22 – Testez la configuration du controller-manager

```
kubectl describe pod kube-controller-manager-master -n kube-system
```

Vous devez obtenir quelque chose du type :

23 – Vérifiez le statut des Pods :

```
kubectl get pods --all-namespaces
```

```
pi@master130:~ $ kubectl describe pod kube-controller-manager-master -n kube-system
Name:               kube-controller-manager-master130
Namespace:          kube-system
Node:               master130/10.10.10.130
Start Time:         Mon, 12 Feb 2018 07:20:48 +0000
Labels:             component=kube-controller-manager
                    tier=control-plane
Annotations:        kubernetes.io/config.hash=e9376f0ac214f3de57862d1721369fe7
                    kubernetes.io/config.mirror=e9376f0ac214f3de57862d1721369fe7
                    kubernetes.io/config.seen=2018-02-12T07:20:48.102046488Z
                    kubernetes.io/config.source=file
                    scheduler.alpha.kubernetes.io/critical-pod=
Status:             Running
IP:                 10.10.10.130
Containers:
  kube-controller-manager:
    Container ID:  docker://3459a8b59adef66795a03cf03fb9885e6770ac0d0ba33545ad92080bc0f540d5
    Image:         gcr.io/google_containers/kube-controller-manager-arm:v1.9.3
    Image ID:      docker-pullable://gcr.io/google_containers/kube-controller-manager-arm@sha256:67f4453e4f5968a6a767
fd6a8b7ae33fecc2d786c7bafb2ede40df285edb4324
    Port:          <none>
    Command:
      kube-controller-manager
      --controllers=*,bootstrapsigner,tokencleaner
      --kubeconfig=/etc/kubernetes/controller-manager.conf
      --cluster-signing-key-file=/etc/kubernetes/pki/ca.key
      --address=127.0.0.1
      --use-service-account-credentials=true
      --root-ca-file=/etc/kubernetes/pki/ca.crt
      --service-account-private-key-file=/etc/kubernetes/pki/sa.key
      --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt
      --leader-elect=true
    State:          Running
      Started:      Mon, 12 Feb 2018 07:23:47 +0000
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        200m
    Liveness:     http-get http://127.0.0.1:10252/healthz delay=15s timeout=15s period=10s #success=1 #failure=8
    Environment:  <none>
    Mounts:
      /etc/kubernetes/controller-manager.conf from kubeconfig (ro)
      /etc/kubernetes/pki from k8s-certs (ro)
      /etc/ssl/certs from ca-certs (ro)
      /usr/libexec/kubernetes/kubelet-plugins/volume/exec from flexvolume-dir (rw)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Volumes:
  k8s-certs:
    Type:          HostPath (bare host directory volume)
    Path:          /etc/kubernetes/pki
    HostPathType:  DirectoryOrCreate
  ca-certs:
    Type:          HostPath (bare host directory volume)
    Path:          /etc/ssl/certs
    HostPathType:  DirectoryOrCreate
  kubeconfig:
    Type:          HostPath (bare host directory volume)
    Path:          /etc/kubernetes/controller-manager.conf
    HostPathType:  FileOrCreate
  flexvolume-dir:
    Type:          HostPath (bare host directory volume)
    Path:          /usr/libexec/kubernetes/kubelet-plugins/volume/exec
    HostPathType:  DirectoryOrCreate
QoS Class:         Burstable
Node-Selectors:    <none>
Tolerations:       :NoExecute
Events:            <none>

pi@master130:~ $
```

```
pi@master130:~ $ kubectl get pods --all-namespaces
NAMESPACE     NAME                                  READY     STATUS    RESTARTS   AGE
kube-system   etcd-master130                        1/1       Running   0          1h
kube-system   kube-apiserver-master130              1/1       Running   0          1h
kube-system   kube-controller-manager-master130     1/1       Running   0          1h
kube-system   kube-dns-7b6ff86f69-hzzcf             3/3       Running   0          1h
kube-system   kube-proxy-5mltz                      1/1       Running   0          35m
kube-system   kube-proxy-bgtrb                      1/1       Running   0          1h
kube-system   kube-proxy-bkvql                      1/1       Running   0          12m
kube-system   kube-proxy-p7vrr                      1/1       Running   0          59m
kube-system   kube-proxy-q8khv                      1/1       Running   0          21m
kube-system   kube-proxy-zvbn6                      1/1       Running   0          32m
kube-system   kube-scheduler-master130              1/1       Running   0          1h
kube-system   weave-net-9dhzz                       2/2       Running   0          59m
kube-system   weave-net-cvtzt                       2/2       Running   1          32m
kube-system   weave-net-hpgkz                       2/2       Running   0          1h
kube-system   weave-net-lwvls                       2/2       Running   0          21m
kube-system   weave-net-vgjt4                       2/2       Running   1          35m
kube-system   weave-net-x7644                       2/2       Running   5          12m
pi@master130:~ $
```

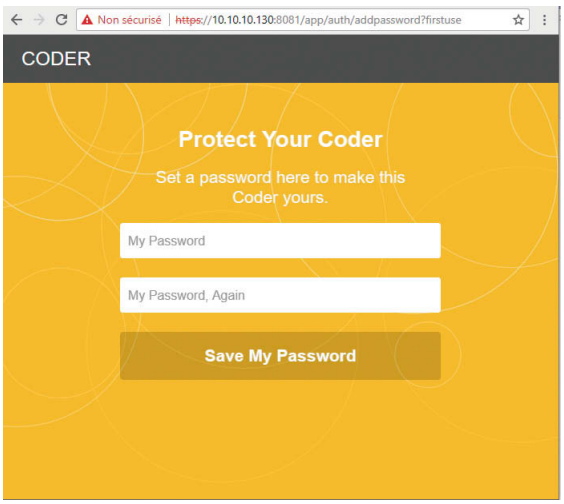
24 – Lancez un exemple d'application et confirmez que les instances de Docker sur votre RPi sont bien accessibles depuis n'importe où :

```
docker run -d -p 8081:8081 resin/rpi-google-coder
```

Une fois que le conteneur est poussé et démarré, confirmez qu'il fonctionne :

```
docker ps
```

Ouvrez un navigateur sur votre ordinateur et accédez à l'URL suivante : https://<rpi ip address here>:8081
Votre navigateur devrait charger l'application Coder en Node.js. Il est possible qu'un avertissement s'affiche pour le certificate SSL, vous pouvez l'accepter.



Installation d'OpenFaas

OpenFaaS est un framework pour Docker qui permet de lancer des fonctions pouvant être programmées dans des langages de code différents. Ce projet a été créé par Alex Ellis, capitaine Docker, lors d'un Hack au DockerCon 2017 et c'est le projet qui a obtenu le plus d'étoiles sur GitHub en 2017.
Vous pouvez voir une introduction et quelques demos faites par Alex lors d'un meetup Cloud Native à Londres : FaaS and Furious - 0 to Serverless in 60 seconds, anywhere (https://skillsmatter.com/skills-casts/10813-faas-and-furious-0-to-serverless-in-60-seconds-anywhere).

25 – Clonez le repository GitHub repository et utilisez Kubectl pour le déployer :

```
sudo apt install git
git clone https://github.com/openfaas/faas-netes && \
cd faas-netes
```

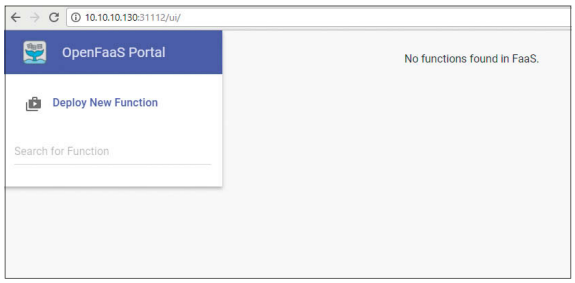
Puis :

```
kubectl apply -f ./namespaces.yml && \
kubectl apply -f ./yaml_armhf
```

Le contrôleur Kubernetes pour OpenFaaS est nommé faas-netes. Pour plus de détails, vous pouvez consulter le repository faas-netes (https://github.com/openfaas/faas-netes).

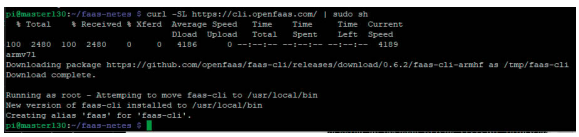
26 – Testez OpenFaaS

Maintenant, vous pouvez ouvrir l'interface graphique d'OpenFaaS dans votre navigateur (en utilisant l'adresse IP de votre nœud maître) : http://192.168.0.100:31112/ui/



27 – Installez OpenFaaS CLI :

```
curl -SL https://cli.openfaas.com/ | sudo sh
```



Vous pouvez obtenir de l'aide sur CLI à tout moment ou vérifier sa version avec :

```
faas-cli –help
faas-cli <command> --help
faas-cli version
```

Clonez les exemples de CLI samples et patchez les modèles pour ARM :

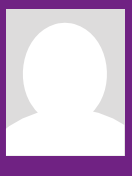
```
git clone https://github.com/alexellis/faas-cli && \
cd faas-cli
```

Dans certains tutoriels, vous pourrez voir les instructions suivantes, mais cela n'est plus utile car tout est maintenant directement construit pour ARM :

```
cp template/node-armhf/Dockerfile template/node/ && \
cp template/python-armhf/Dockerfile template/python/ && \
cp template/go-armhf/Dockerfile template/go/
```

La fonction va être chargée par Kubernetes et déployée sur un des nœuds de votre cluster.
Cela peut prendre quelques minutes la première fois car le réseau et les interfaces des RPi est plus lent que celui des ordinateurs.
Vous pouvez maintenant appeler votre fonction via l'interface graphique ou en CLI.

•



Pierre FICHEUX
pierre.ficheux@smile.fr
Directeur technique Smile ECS
Innovation @ Smile

Introduction à Yocto

Partie 1

Yocto est devenu un standard de l'industrie pour la technologie « Linux embarqué ». Dans cette courte série de deux articles nous décrirons tout d'abord les principes de cet outil, puis nous décrirons quelques exemples plus avancés dans un deuxième article.

De la distribution au « build system »

La plupart des utilisateurs et développeurs GNU/Linux utilisent des « distributions » (Debian, Ubuntu, Fedora, Red Hat, etc.). Une distribution est constituée d'un ensemble de composants rassemblés dans des « paquets » (packages). Ces paquets sont installés grâce à une procédure (graphique) la plus simple possible, l'utilisateur final n'étant pas forcément un développeur. Dans de nombreux cas on installe la distribution sur un PC/x86 afin de créer un poste de travail qui peut aller de la simple bureautique (Internet, LibreOffice) au développement C/C++, Python ou Java en intégrant des outils comme Eclipse. Un autre cas fréquent est l'installation d'un serveur. Grâce à une autre interface graphique on peut ajouter, retirer ou mettre à jour les paquets installés sur la distribution.

L'espace occupé par une distribution classique est relativement important car il est rare qu'un disque dur récent ait une capacité inférieure à 1 To (voire plus). Suivant les composants installés, la distribution utilisera donc plusieurs Go, voire plusieurs dizaines de Go sur le disque, et sera composée de milliers de paquets (plus de 3000 sur mon poste de travail sous Ubuntu).

Depuis plusieurs années GNU/Linux est utilisé pour des solutions industrielles, ou plus récemment pour des solutions « embarquées ». La principale différence correspondant à l'empreinte mémoire utilisée ainsi que la puissance du matériel (CPU, RAM, stockage). A titre d'exemple, la quasi-totalité des boîtiers « gateway » et décodeurs TV proposés par les fournisseurs d'accès à Internet (FAI) utilisent GNU/Linux dans une version adaptée. Il en est de même pour les décodeurs satellite. Ces produits étant largement diffusés (plusieurs millions d'exemplaires) il est indispensable de réduire le coût du matériel et donc en premier lieu l'empreinte mémoire du système d'exploitation donc la taille de la mémoire flash hébergeant ce système (ainsi que la RAM). En outre, ces produits doivent être d'une grande fiabilité. Il est donc indispensable de maîtriser la production du système ainsi que la mise à jour.

Pour toutes ces raisons, l'utilisation d'une distribution classique n'est pas recommandée et l'on créera l'image du système à partir des sources des composants en utilisant un outil dédié nommé « build system » que l'on peut traduire par « outil de construction ».

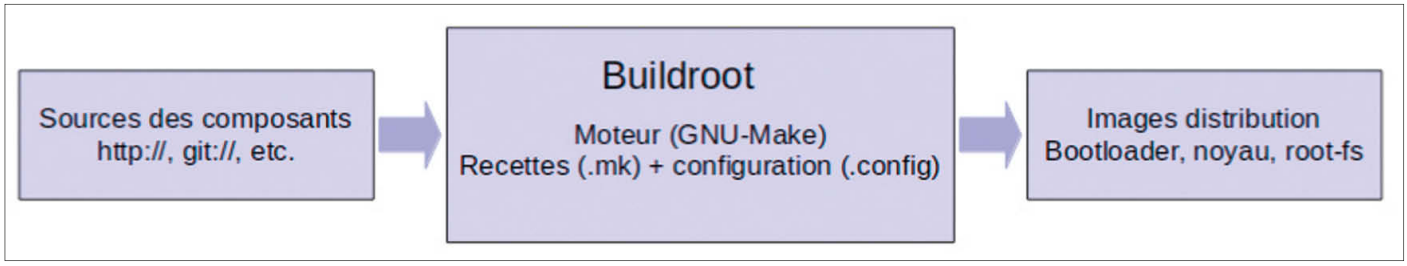
Les différents outils disponibles

Pendant plusieurs années il n'existait pas d'outils standards et ceux d'aujourd'hui ont été initialement créés en tant qu'outils internes pour d'autres projets. L'outil Buildroot [1] était au départ utilisé pour le projet uClibc (puis uClibc-ng). Buildroot a toujours à ce jour une approche « statique » ce qui signifie qu'il n'est pas possible d'ajouter/retirer/mettre à jour des composants sur la cible installée. De même, l'intégralité des recettes et des BSP (Board Support Package) disponibles sous Buildroot sont fournis dans le dépôt officiel (soit près de 1800 recettes et 190 cibles matérielles).

Son principal concurrent, OpenEmbedded [2] – à l'origine de Yocto - fut démarré dans le projet OpenZaurus, une version de GNU/Linux pour le PDA Zaurus de Sharp (sorti en 2002). OpenEmbedded permet de créer – si on le désire - une véritable « distribution » sur mesure bénéficiant d'un système de gestion de paquets similaire à celui des distributions classiques. Nous verrons également que son approche est beaucoup plus « dynamique » tant au niveau des recettes que des BSP disponibles.

Cette liste des outils n'est pas exhaustive car d'autres produits similaires sont encore utilisés, citons OpenWrt [4] initialement développé pour le routeur WRT54G de Linksys et toujours utilisé sur des passerelles grand public, PTXdist [5] créé par Pengutronix ou bien l'ancêtre LTIB [6] autrefois utilisé pour les BSP fournis par Freescale (à l'époque). Ces trois derniers outils sont très similaires à Buildroot malgré quelques différences comme le système de gestion de paquets disponible sur OpenWrt. Le principe de fonctionnement d'un build system est toujours le même :

- L'outil fournit des fichiers décrivant la manière de produire un composant binaire à partir de ses sources pour une cible matérielle donnée (on parle de « recette » similaire à une recette de cuisine). Il faut noter que l'outil ne fournit pas les sources des composants et qu'on les obtient à partir des dépôts des différents projets (kernel.org, busybox.net, github.com, etc.).
- A partir des composants sélectionnés, l'outil produit les images du système (bootloader, noyau Linux, root-filesystem) et souvent une image unique à écrire sur la mémoire flash de la cible (ou bien la Micro-SD).



Les outils similaires à Buildroot ont en commun la possibilité de définir le contenu de l'image à produire (i.e. les composants à intégrer) en utilisant un outil graphique identique à celui utilisé pour la configuration du noyau Linux. 1

En revanche OpenEmbedded (et donc Yocto) n'utilisent – presque - pas d'outil graphique et l'on définit le contenu de l'image par des fichiers de configuration. Cette approche est plus complexe pour l'utilisateur occasionnel mais permet de créer des configurations plus avancées dans le cas d'une approche industrielle.

Introduction à Yocto / OpenEmbedded

Comme nous l'avons dit OpenEmbedded est né suite aux premiers travaux sur OpenZaurus (qui utilisait Buildroot). Les limites de la version Buildroot de l'époque (2003) – et probablement son approche « statique » - conduisirent les trois principaux développeurs (Chris Larson, Michael Lauer, and Holger Schurig) à définir leur propre outil de construction constitué de deux projets indépendants.

- Un ensemble de recettes (OpenEmbedded-Core)
- Un outil de construction et d'exploitation des recettes (BitBake)

Le projet était à l'époque assez complexe à utiliser du fait d'un cruel manque de documentation. Son intégration en 2010 au projet Yocto (projet officiel de la fondation Linux) permit de rationaliser l'architecture et de rendre son usage plus accessible. Yocto est un projet « chapeau » intégrant différents projets libres comme OpenEmbedded, BitBake, Poky ainsi que les grands noms de l'industrie tant au niveau du matériel que du logiciel (Intel, TI, Xilinx , AMD, Mentor Graphics, Wind River, etc.). De ce fait, Yocto est aujourd'hui une référence industrielle et il est utilisé par les fabricants de matériel pour fournir leur BSP Linux. Il est également à la base des solutions d'éditeurs de logiciels comme celles de Wind River (Wind River Linux [7]). D'autres projets majeurs dans l'informatique embarquée dédiée à l'automobile (IVI pour In Vehicle Infotainment) comme GENIVI [8] ou AGL [9] sont également basés sur Yocto.

L'approche de Yocto est très différente de celle de Buildroot. En effet, Yocto est basé sur une architecture en couches (layers) permettant de construire l'image d'une distribution de référence nommée Poky et d'enrichir cette image en y ajoutant d'autres couches (donc de nouvelles recettes). Seules les couches indispensables (soit oe-core et meta-yocto) sont fournies par le projet Yocto. On peut ensuite ajouter des couches externes liées au matériel (BSP), à des systèmes graphiques comme Qt ou à des composants métier. Comme nous pouvons le constater dans la liste des layers répertoriés (disponible en [10]), un layer est le plus souvent nommé meta-<nom-du-layer> et correspond concrètement à une arborescence de sous-répertoires contenant des recettes. 2

Poky est la distribution de référence mais d'autres sont disponibles comme ELDK de DENX (meta-eldk), Arago de TI (meta-arago-distro), Angstrom (meta-angstrom). Rappelons que des produits commerciaux comme Wind River Linux ou MontaVista Carrier Grade utilisent Yocto comme système de construction.

Yocto produit systématiquement des paquets binaires au format RPM, IPK ou DEB. Basé sur le format DEB, le format IPK (Itsy Package Format) a l'avantage d'être très compact, bien plus que les deux autres formats (RPM et DEB) compatibles avec les distributions classiques. Le format IPK est également utilisé pour OpenWrt déjà cité en début d'article.

La base de données des paquets est le plus souvent installée sur la cible ce qui permet l'ajout/suppression/mise à jour comme sur une distribution classique. Cette fonctionnalité n'est cependant pas activée par défaut sur l'image la plus légère que nous testerons au paragraphe suivant.

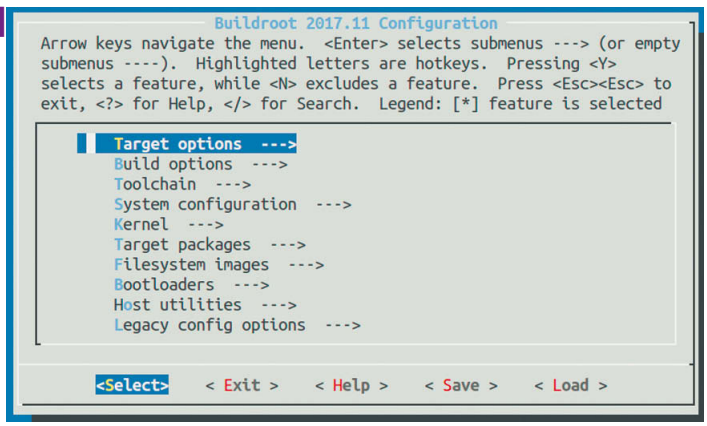
Un premier test de Yocto

Un des principes fondamentaux de Yocto est de maintenir une liste de cibles de référence constituée en majorité de cibles émulées par l'outil QEMU (plus quelques cibles génériques, soit 13 au total). Les noms des cibles sont *qemux86*, *qemuarm*, *qemumips*, etc. Si l'on désire construire l'image pour une cible réelle, il faudra obtenir le BSP sous forme de layer dans la liste citée en [10]. Le layer peut également être disponible auprès du fournisseur sans être référencé ni public (mais dans ce cas-là, la méfiance est de mise !).

REMARQUE : nous avons dit que le développement des layers externes (comme les BSP) n'est pas de la responsabilité du projet Yocto. Il est donc de la responsabilité des mainteneurs du BSP de suivre les évolutions de Yocto. Ce point est à prendre en compte lors du choix d'une cible matérielle car dans le cas contraire l'utilisateur sera limité à des anciennes versions de Yocto et donc limité dans l'utilisation d'autres layers.

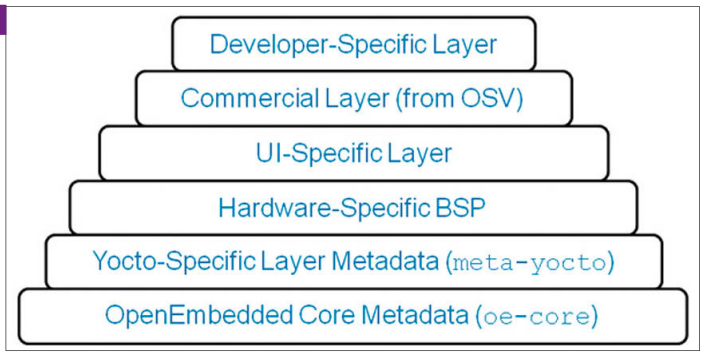
Dans la suite de l'article nous utiliserons le BSP Yocto pour la célèbre carte Raspberry Pi, soit meta-raspberrypi [11]. Paradoxalement, le support Yocto de cette carte « grand public » est de très bonne qualité, bien meilleur que celui de certaines cartes industrielles. Avant cela nous allons construire une image pour la cible par défaut soit qemux86.

1



Écran de configuration de Buildroot

2



Les couches (layers) de Yocto

Test pour QEMU/x86

La première étape consiste à récupérer les sources du projet Yocto. La branche correspond à la version choisie, chaque version ayant un nom de code (rocko pour 2.4, pyro pour 2.3, morty pour 2.2, etc.).

```
$ git clone -b <branche> git://git.yoctoproject.org/poky
```

On doit alors créer un répertoire de compilation en utilisant le script **oe-init-build-env**. Cette commande a pour effet de créer le répertoire choisi (soit **qemux86-build**) contenant les fichiers **local.conf** et **bblayers.conf** dans le sous-répertoire **conf**.

```
$ cd poky
$ source oe-init-build-env qemux86-build
```

La construction de l'image la plus simple nommée « core-image-minimal » correspond à la ligne suivante :

```
$ bitbake core-image-minimal
```

La création de cette image – bien que légère - est relativement longue. En effet, outre la création de l'image à installer sur la cible, la première compilation produit également les outils de compilation croisés. Sur une machine raisonnablement puissante (i7, 8 Go de RAM), le temps de compilation avoisine une heure même si l'utilisation d'un disque SSD améliore considérablement les performances. Une fois l'image produite, on peut la tester en utilisant la commande suivante :

```
$ runqemu qemux86
```

Cette dernière commande doit conduire à l'affichage de la fenêtre de l'émulateur indiquant le démarrage du système. 3

Test sur Raspberry Pi 3

Si l'on désire tester la même image sur une Raspberry Pi 3, il faut obtenir le layer correspondant et indiquer le type de cible en renseignant la variable d'environnement **MACHINE** dans le fichier **local.conf**. Il est important d'indiquer la même branche que pour les sources de Yocto lors du test précédent.

```
$ cd poky
$ git clone -b <branche> git://git.yoctoproject.org/meta-raspberrypi
```

On crée un nouveau répertoire de travail et l'on ajoute le nouveau layer au fichier **bblayers.conf** en utilisant la commande **bitbake-layers** (différente de **bitbake** !). On note que l'on peut utiliser plusieurs répertoires de compilation dans la même arborescence Yocto, ce qui n'est pas possible avec Buildroot.

```
$ source oe-init-build-env rpi3-build
$ bitbake-layers add-layer ../meta-raspberrypi
```

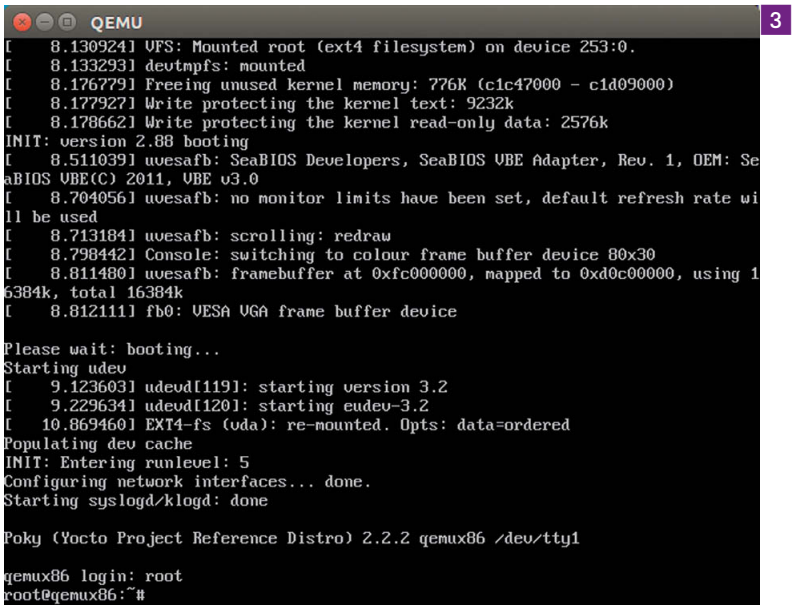
On définit ensuite le type de cible.

```
echo "MACHINE = \"raspberrypi3\"" >> conf/local.conf
```

On peut alors créer l'image.

```
$ bitbake core-image-minimal
```

Pour tester sur la Raspberry Pi 3, on copie l'image produite sur une Micro-SD.



3

Test de l'image Poky pour QEMU/x86

```
$ umount /dev/mmcblk0p*
$ sudo dd if=tmp/deploy/image/raspberrypi3/core-image-minimal-raspberrypi3.
rpi-sdimg of=/dev/mmcblk0
```

Lors du test sur la carte on obtient finalement les traces suivantes :

```
...
Poky (Yocto Project Reference Distro) 2.4.1 raspberrypi3 /dev/ttyS0

raspberrypi3 login: root
root@raspberrypi3:~# uname -a
Linux raspberrypi3 4.9.77 #1 SMP Sun Feb 11 02:20:40 CET 2018 armv7l GNU/Linux
# df -h
Filesystem      Size  Used Available Use% Mounted on
/dev/root       10.6M   5.5M   4.5M  55% /
```

On note la très faible empreinte mémoire du système (5,5 Mo) mais il est vrai que nous n'avons pas installé les systèmes de gestion de paquets (package management) sur l'image ni les modules du noyau. L'ajout du système de gestion de paquets IPK permet cependant de conserver une empreinte mémoire très raisonnable (8,5 Mo).

```
root@raspberrypi3:~# df -h
Filesystem      Size  Used Available Use% Mounted on
/dev/root       14.5M   8.5M   5.2M  62% /
```

Répertoires produits

Le test précédent nous a permis de produire une image à tester. Dans cette partie nous allons voir les éléments créés par Yocto (en fait par BitBake). Ces éléments sont tous situés dans des sous-répertoires du répertoire de construction, soit **rpi3-build** dans notre cas. Étrangement, les fichiers directement utilisables sur la cible (en particulier l'image à installer) sont localisés dans un sous-répertoire de **tmp** et dans cet article d'introduction, nous évoquerons uniquement **tmp/deploy** et **tmp/work**. Le plus important est certainement **tmp/deploy/images/raspberrypi3** puisqu'il contient tous les éléments nécessaires au démarrage de la cible (noyau Linux, modules,

fichiers « device tree », images du root-filesystem, etc.). Certains BSP, comme celui de la Raspberry Pi, produisent également une image directement utilisable sur une mémoire flash (cas de notre test).
Si l'on détaille un peu plus **tmp/deploy** on constate qu'il contient trois sous-répertoires.

```
tmp/deploy/
|—— images
|—— ipk
|—— licences
```

Le sous-répertoire **licences** contient les licences des composants produits lors de la création de l'image. Ce point est important lors d'une démarche industrielle car il est souvent nécessaire de disposer de cette liste. Si l'on détaille le sous-répertoire **ipk**, on constate qu'il contient lui-même trois sous-répertoires.

```
tmp/deploy/ipk/
|—— all
|—— cortexa7hf-neon-vfpv4
|—— raspberrypi3
```

Le but des sous-répertoires est de classer les paquets binaires produits par catégories. Le sous-répertoire **all** correspond à des paquets indépendants de l'architecture de la cible (scripts, fichiers de données). Le sous-répertoire **cortexa7hf-neon-vfpv4** correspond à l'architecture matérielle de la cible (ici un cœur Cortex A7). Une cible utilisant la même architecture devrait pouvoir utiliser ces paquets. Le sous-répertoire **raspberrypi3** correspond à des paquets spécifiques à la cible comme les pilotes ou les fichiers « device tree » de la Raspberry Pi 3. Notons que si le format de paquets choisi est RPM (valeur par défaut) ou DEB, nous verrons apparaître les sous-répertoires **rpm** et/ou **deb**.
Le répertoire **tmp/work** nous est moins utile dans un premier temps car il correspond à un répertoire de travail hébergeant la production de chaque paquet binaire à partir des recettes. Outre cela il peut contenir des informations intéressantes comme le contenu du root-filesystem ou les traces d'exécution des recettes (ainsi que les erreurs!). Il est donc destiné à un utilisateur avancé développant ses propres recettes comme nous le verrons dans le deuxième article.

Configuration avancée

Pour l'instant nous avons uniquement spécifié le type de cible par la variable **MACHINE** dans **local.conf** ainsi que l'ajout du layer meta-raspberry grâce à la commande **bitbake-layers** qui impacte le fichier **bblayers.conf**. Nous allons terminer ce premier article en indiquant quelques options de configurations que l'on peut mentionner dans **local.conf**. Une description complète des options est bien entendu disponible dans la documentation Yocto, en particulier le manuel de référence [12]. Lors du deuxième article nous décrirons plus précisément la notion de « feature » évoquée ci-après.

Réduction de l'espace utilisé

La production d'une image correspond à l'exécution par BitBake de plusieurs milliers de recettes. Par défaut chaque étape de construction est tracée dans **tmp/work** et les fichiers intermédiaires correspondant à la compilation des composants sont conservés, ce qui peut occuper plusieurs dizaines de Go sur la machine de développement. Pour réduire la taille de **tmp/work** au strict minimum (soit tout de même 1 Go) on peut ajouter la directive :

```
INHERIT += "rm_work"
```

Ajout du gestionnaire de paquets

Si l'on désire disposer d'un gestionnaire de paquets binaire sur la cible, on peut ajouter la ligne suivante :

```
EXTRA_IMAGE_FEATURES += "package-management"
```

On peut également spécifier le format de paquet utilisé qui par défaut est **package_rpm**.

```
PACKAGE_CLASSES = "package_ipk"
```

Ajout d'espace libre sur la cible

La taille du root-filesystem est calculée par Yocto au plus juste et nous avons vu lors de notre test qu'il restait à peine 5 Mo d'espace disponible. Si l'on veut augmenter l'espace libre on peut indiquer le volume à ajouter (en Ko).

```
IMAGE_ROOTFS_EXTRA_SPACE = "50000"
```

Ajout d'un format de root-filesystem

En fonction du BSP, Yocto construit les images du root-filesystem dans **tmp/deploy/images/<nom-de-cible>**. Si l'on désire ajouter un format d'image supplémentaire, on peut utiliser la variable **IMAGE_FSTYPES**. Dans l'exemple qui suit on produit en plus une image au format Initramfs (soit une archive CPIO compressée).

```
IMAGE_FSTYPES += "cpio.gz"
```

Construction d'une image en lecture seule

Dans de nombreux cas de figures, il est judicieux que le root-filesystem installé sur la cible en production soit en lecture seule. Pour cela il suffit d'ajouter la ligne suivante :

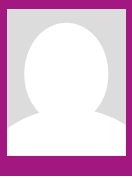
```
EXTRA_IMAGE_FEATURES += "read-only-rootfs"
```

Conclusion

Dans cette première partie nous avons pu voir les rudiments de l'utilisation de Yocto après une brève comparaison avec d'autres outils comme Buildroot. Dans un prochain article nous verrons comment construire nous-même des recettes afin d'enrichir notre image.

Bibliographie

[1] *Buildroot sur* https://buildroot.org
[2] *OpenEmbedded sur* http://www.openembedded.org/wiki/Main_Page
[3] *Projet Yocto sur* https://www.yoctoproject.org/
[4] *OpenWrt sur* https://openwrt.org/
[5] *PTXdist sur* https://www.pengutronix.de/en/software/ptxdist.html
[6] *LTIB sur* http://ltib.org/
[7] *Wind River Linux sur* https://www.windriver.com/products/linux/
[8] *Projet GENIVI sur* https://www.yoctoproject.org/product/genivi-baseline
[9] *Automotive Grade Linux sur* https://www.automotivelinux.org
[10] *Liste des layers officiels sur* http://layers.openembedded.org/layerindex/branch/master/layers
[11] *Layer du BSP Raspberry Pi sur* http://git.yoctoproject.org/cgit/cgit.cgi/meta-raspberrypi
[12] *Manuel de référence Yocto sur* http://www.yoctoproject.org/docs/2.4.1/ref-manual/ref-manual.html



Sébastien Vallée,
chef de projet JEE Osaxis
www.osaxis.fr

Lombok : gérer du code lisible

Lombok est une librairie sous licence MIT permettant de générer du code lors de la compilation des classes Java. A travers l'utilisation d'annotations, Lombok permet d'obtenir un code plus lisible et libère également le développeur de la longueur d'écriture des méthodes que l'on retrouve communément dans les classes objets et métiers d'un projet.

Comment installer Lombok

Lombok est actuellement disponible en version 1.16.18 sortie en juillet 2017. Les mises à jour régulières de la librairie témoignent de sa pérennité. Elle s'intègre facilement à un projet existant :

Utilisation avec Maven

```
<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.16.18</version>
  <scope>provided</scope>
</dependency>
```

Utilisation avec Gradle

```
// https://mvnrepository.com/artifact/org.projectlombok/lombok
provided group: 'org.projectlombok', name: 'lombok', version: '1.16.18'
```

Comment utiliser Lombok

Prenons une classe Java qui permet de gérer des informations simples d'un utilisateur : nom, prénom et âge.

```
/**
 * Classe utilisateur.
 */
public class UtilisateurBean {
  /** Age de l'utilisateur. */
  private Integer _age;

  /** Nom de l'utilisateur. */
  private String _nom;

  /** Prénom de l'utilisateur. */
  private String _prenom;

  /**
   * Constructeur vide.
   */
  public UtilisateurBean() {
  }

  /**
   * Constructeur avec initialisation des propriétés.
   * @param pNom
   *        Le nom.
   * @param pPrenom
   *        Le prénom.
   * @param pAge
   *        L'âge.
   */
  public UtilisateurBean(final String pNom, final String pPrenom, final Integer pAge) {
    this._nom = pNom;
    this._prenom = pPrenom;
    this._age = pAge;
  }

  /**
   * @return l'âge de l'utilisateur.
   */
  public Integer getAge() {
    return this._age;
  }

  /**
   * @return le nom de l'utilisateur.
   */
  public String getNom() {
    return this._nom;
  }

  /**
   * @return le prénom de l'utilisateur.
   */
  public String getPrenom() {
    return this._prenom;
  }

  /**
   * Affectation de l'âge.
   * @param pAge
   *        L'âge de l'utilisateur.
   */
  public void setAge(final Integer pAge) {
    this._age = pAge;
  }

  /**
   * Affectation du nom.
   * @param pNom
   *        Le nom de l'utilisateur.
   */
  public void setNom(final String pNom) {
```

```
    this._nom = pNom;
  }

  /**
   * Affectation du prénom.
   * @param pPrenom
   *        Le prénom de l'utilisateur.
   */
  public void setPrenom(final String pPrenom) {
    this._prenom = pPrenom;
  }
}
```

Difficile de faire une classe plus simple que celle-ci ! Pourtant, elle comporte déjà environ 80 lignes. En tant que développeur, nous écrivons souvent ce type de classe dont la majeure partie est systématiquement structurée de la même façon. C'est ici que la mise en place de Lombok prend tout son intérêt.

@Getter et @Setter

L'utilisation des annotations @Getter et @Setter permet de ne plus s'occuper de l'écriture des accesseurs de la classe. Nous obtenons déjà un objet plus lisible :

```
/**
 * Classe utilisateur.
 */
@Getter
@Setter
public class UtilisateurBean {
  /** Age de l'utilisateur. */
  private Integer _age;

  /** Nom de l'utilisateur. */
  private String _nom;

  /** Prénom de l'utilisateur. */
  private String _prenom;

  /**
   * Constructeur vide.
   */
  public UtilisateurBean() {
  }

  /**
   * Constructeur avec initialisation des propriétés.
   * @param pNom
   *        Le nom.
   * @param pPrenom
   *        Le prénom.
   * @param pAge
   *        L'age.
   */
  public UtilisateurBean(final String pNom, final String pPrenom, final Integer pAge) {
    this._nom = pNom;
    this._prenom = pPrenom;
    this._age = pAge;
  }
}
```

Il est possible de rendre le getter d'une propriété « lazy », c'est-à-dire d'exécuter son affectation uniquement lors du premier appel. Pour cela, il suffit d'ajouter @Getter(lazy=true) sur la propriété ; c'est utilisé principalement sur les initialisations coûteuses en mémoire ou en processeur.

```
@Getter(lazy=true)
private final float lazyPropriete = initialisationLazy();

private float initialisationLazy() {
  float resultat = 0;
  // ...
  // Calcul complexe.
  // ...
  return resultat;
}
```

@Accessors

A l'utilisation de la classe, on remarque alors que les getters et les setters sont légèrement différents de ceux que nous avions écrits au préalable. En effet, nous obtenons pour le prénom : get_prenom() au lieu de getPrenom(). Ceci est dû au caractère « _ » que nous avons ajouté devant chaque propriété. Pour rester cohérent avec les éventuelles normes de codage du projet, il suffit d'ajouter dans le cas présent l'annotation @Accessors(prefix = « _ ») qui permet d'obtenir le getter getPrenom() initialement écrit.

@NoArgsConstructor et @AllArgsConstructor

Il est également possible de s'abstraire de l'écriture des constructeurs de classe. Dans le cas présenté ici, nous allons gérer un constructeur sans arguments, et un autre constructeur contenant l'initialisation de toutes les propriétés. Avec l'utilisation supplémentaire de ces deux annotations, nous obtenons l'équivalent de la classe de départ avec seulement une vingtaine de lignes :

```
/**
 * Classe utilisateur.
 */
@Getter
@Setter
@Accessors(prefix = "_")
@NoArgsConstructor
@AllArgsConstructor
public class UtilisateurBean {
  /** Age de l'utilisateur. */
  private Integer _age;

  /** Nom de l'utilisateur. */
  private String _nom;

  /** Prénom de l'utilisateur. */
```

```
  private String _prenom;
}
```

A noter qu'il est possible de spécifier le niveau de visibilité du constructeur complet, en ajoutant à l'annotation l'information suivante :

```
@AllArgsConstructor(access = AccessLevel.PROTECTED).
```

Il peut arriver que les propriétés d'une classe soient définies en « static ». Dans ce cas, une erreur de compilation apparaît. Ceci est dû au fait qu'une propriété « static » doit être initialisée. Pour éviter l'erreur de compilation, il faut ajouter l'argument « force = true » à l'annotation @NoArgsConstructor :

```
@NoArgsConstructor( force = true )
```

Ceci a pour effet d'initialiser les propriétés avec la valeur 0, false ou null.

@RequiredArgsConstructor(staticName="of")

Pour réaliser un constructeur plus spécifique, il est possible de déterminer quelles propriétés doivent être initialisées lors de l'instanciation de la classe. Pour cela, il suffit de suivre deux étapes :

- ajouter l'annotation @NonNull aux propriétés concernées ;
- ajouter l'annotation @RequiredArgsConstructor(staticName= « of »).

Où « of » correspond au nom du nouveau constructeur. Exemple avec notre classe :

```
/**
 * Classe utilisateur.
 */
@Getter
@Setter
@Accessors(prefix = "_")
@RequiredArgsConstructor(staticName = "of")
public class UtilisateurBean {
  /** Age de l'utilisateur. */
  private Integer _age;

  /** Nom de l'utilisateur. */
  @NonNull
  private String _nom;

  /** Prénom de l'utilisateur. */
  private String _prenom;
}
```

Pour l'utiliser, on écrit un code semblable à :

```
UtilisateurBean vUtilisateur = UtilisateurBean.of(« admin ») ;
```

L'inconvénient est qu'il n'est pas possible avec cette méthode de définir plusieurs constructeurs particuliers. Pour cela, il reste toujours la solution de l'écriture manuelle.

@Builder

Cette annotation génère un « builder » permettant de ne plus être dépendant de l'ordre particulier des paramètres lors de l'instanciation d'une classe. Dans notre exemple, il est donc possible d'écrire le code suivant :

```
UtilisateurBean.builder().nom(«unNom»).prenom(«unPrenom»).build();
```

L'inconvénient est qu'aucune erreur de compilation n'est remontée si un paramètre est manquant. De plus, lors d'une revue de code par exemple, la recherche de la hiérarchie d'appel est plus contraignante à analyser.

@Equals @HashCode et @ToString

Ces annotations permettent à Lombok de générer respectivement les méthodes « equals », « hashcode » et « ToString ». L'intérêt d'utiliser ces annotations, en plus d'éviter de les coder, réside dans le paramétrage des attributs que l'on souhaite utiliser dans ces méthodes. Pour cela, il y a deux manières de faire : soit par sélection en utilisant le paramètre « of », soit par exclusion en utilisant le paramètre « exclude ». Il est possible ainsi de définir précisément le comportement souhaité de la classe. Voici deux exemples qui illustrent l'utilisation de ces annotations pour que les méthodes correspondantes ne prennent en compte que les propriétés « nom » et « prenom » :

Méthode ToString() :

```
@ToString(of = {"nom", "prenom"}) ou @ToString(exclude = {"age"})
```

Méthode HashCode() :

```
@HashCode(of = {"nom", "prenom"}) ou @HashCode(exclude = {"age"})
```

Méthode Equals() :

```
@Equals(of = {"nom", "prenom"}) ou @Equals(exclude = {"age"})
```

Conclusion

La plupart des environnements de développement proposent la réécriture de code permettant au développeur de ne pas coder manuellement les méthodes redondantes d'une classe métier Java. Un des principaux avantages qu'offre Lombok réside dans l'utilisation des annotations permettant d'améliorer la lisibilité du code, en ne générant les méthodes que dans les classes compilées. De plus, les classes gagnent en maintenabilité, par exemple lors de la modification d'un nom de variable qui ne se fait qu'au niveau de la propriété.

Il faut noter cependant quelques inconvénients. Une fois les annotations Lombok mises en place, la navigation dans la hiérarchie des appels (de getter et setter par exemple) n'est plus aussi directe. Sous Eclipse, une solution de contournement consiste à lancer les recherches depuis la vue « outlines ». De plus, la Javadoc est également très succincte. Il faut penser à écrire la documentation complète sur la propriété directement.

Un autre piège à éviter est celui de la surcharge d'annotation. En effet, cette solution populaire utilisée par de nombreux Frameworks comme Hibernate, Spring ou Lombok transforme nos classes en une pile d'annotations donnant un résultat parfois contraire à celui attendu, à savoir simplifier l'écriture et la compréhension.

Site internet : https://projectlombok.org/

Nicolas ROBERT
Technical Manager
chez Cellenza
@NicoRobPro

Coline Thomas
Data Engineer
chez Quantmetry
@colineto

Services Cognitifs : l'IA sur étagère… mais pas seulement ! Partie 2

IA, machine Learning et Big Data, de nombreux mots pour parler de nouvelles technologies que nous avons tous intérêt à connaître et à découvrir. Pour cela il faut alors rendre une technologie de pointe accessible à tous. Nous allons donc vous parler des outils qui mettent l'IA à votre disposition.

L'IA est au centre de toutes les attentions. Que ce soient les entreprises, les scientifiques, les développeurs ou tous les autres curieux, chacun veut faire de cette technologie son innovation. Si l'IA est devenue aussi accessible à tous, c'est qu'il existe de nombreux outils qui la mettent à disposition des différents niveaux techniques.

Dans l'épisode précédent nous vous avions parlé des services cognitifs « sur étagère », prêts à consommer et accessibles à tous sans quasiment aucun prérequis technique. Nous avions aussi vu les limites de ces outils lorsque les configurations et les besoins se font plus spécifiques.

Nous allons aujourd'hui présenter des solutions liées aux Cognitives Services, mais nécessitant du développement, offrant une réponse personnalisée en fonction des besoins.

Les services de Machine Learning de Microsoft

Aujourd'hui lorsque l'on veut faire du Machine Learning ou tout autre type d'apprentissage, de nombreuses bibliothèques sont à notre disposition, il existe aussi des API et des logiciels. Différents outils pour différents besoins. Sur ce marché, Microsoft propose trois offres.

Azure Machine Learning Studio, l'outil de Machine Learning

Azure Machine Learning Studio est un outil particulièrement visuel et facile à prendre en main, il s'adapte à de nombreux profils, pas forcément développeurs et surtout pas nécessairement Data Scientists.

Il permet d'associer entre elles des briques :

- de traitement des données ;
- d'opérations mathématiques ;
- de développement Python ou R ;
- d'apprentissage automatique.

Cognitive Toolkit, la librairie de Deep Learning

CNTK est une librairie open source de Deep Learning. Elle permet donc de créer des modèles et des solutions complètes d'apprentissage et de traitement des données. Elle est disponible via des API Python et C++ et de nombreux exemples et tutoriels. Cet outil nécessite tout de même quelques compétences techniques et est plutôt à destination des Data Scientists ou des développeurs qui souhaitent en apprendre plus sur ce domaine.

Microsoft Workbench, l'outil complet de Big Data

Cet outil à destination principale des Data Scientists et des Data Engineers permet de développer un projet de Big Data sur toute la chaîne. Les parties de préparation de la data, d'écriture de code, de développement d'algorithme et de visualisation sont lissées et liées entre elles pour fournir une expérience de développement améliorée. De plus, il intègre la majorité des librairies d'apprentissage ainsi qu'un outil performant et innovant de traitement de données capable de faire de l'apprentissage sur les colonnes à traiter.

Workbench permet de travailler en local, mais aussi dans le cloud directement, qui met à disposition de grandes capacités de calculs, mais aussi des clusters. Enfin, il possède aussi de nombreux exemples et des modèles préentraînés. 2

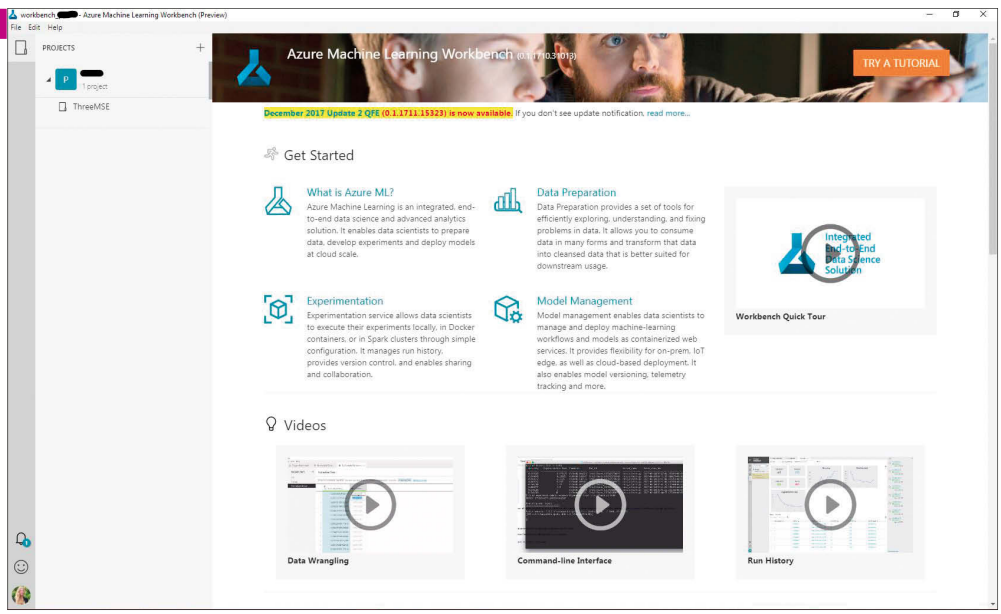
Les cas d'utilisation

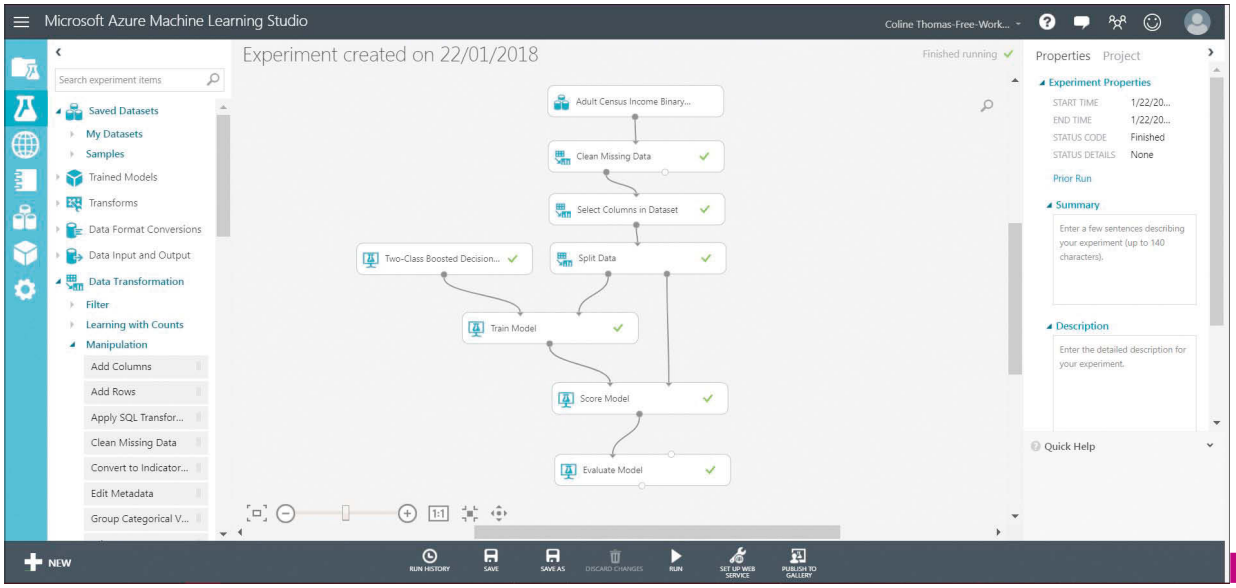
Chacun de ces outils s'utilise dans un contexte différent en fonctions des besoins, mais aussi de qui l'utilise.

Si l'on souhaite découvrir le Machine Learning et le traitement de la donnée sans forcément au préalable savoir coder il est très simple de prendre en main Azure ML Studio et de découvrir différents types d'apprentissage tels que la régression linéaire, la classification par réseaux de neurones, le clustering par voisins les plus proches ou encore l'analyse de texte ou d'images. Vous pourrez ensuite utiliser librement les modèles créés via une API dans toutes vos solutions.

Cet outil possède une version totalement gratuite accessible sans compte Azure mais disposant de peu de capacité de calcul. Si

2





3

vous souhaitez faire tourner de gros modèles, il vous faudra passer par le cloud Azure pour déployer un compte Azure ML payant (0,844 par heure d'expérimentation).

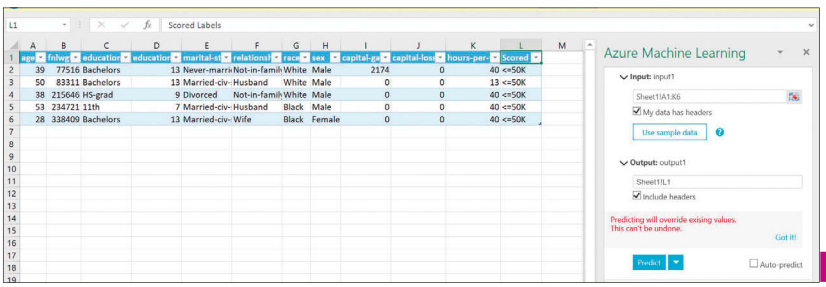
CNTK (Cognitive Toolkit) est, pour sa part, une librairie totalement open source à l'image de Tensorflow ou Scikit learn. Elle met donc à disposition des développeurs et autres profils techniques de nombreux outils pour développer soi-même ses modèles, de Deep Learning principalement.

CNTK, de même que Tensorflow ou Scikit, est intégré dans Workbench et peut ainsi être utilisée sans installation préalable. Utiliser l'ensemble des possibilités de traitement et de machine learning directement dans un seul outil tel que Workbench offre alors de nombreux avantages notamment celui de pouvoir être utilisé avec un simple compte Azure ML expérimentation, c'est-à-dire gratuit.

Exemple concret avec Azure Machine Learning Studio

Dans l'article précèdent nous avions montré comment utiliser les services clé en main de Microsoft. Une fois les limites de ces services atteintes il est assez simple de passer sur l'outil Azure ML Studio et de créer ses propres « cognitives services ». C'est ce que nous allons vous montrer ici.

Notre exemple se base sur des données recensant le salaire et des données plus personnelles telles que l'âge, l'origine, la situation maritale ou encore le parcours scolaire de nombreuses personnes. Ce set de données est directement disponible dans Azure ML, vous pourrez ainsi reproduire cet exemple si vous le souhaitez.



4

Commençons par nous connecter à Azure ML, https://studio.azureml.net. On ajoute ensuite différentes briques de traitement et d'apprentissage pour créer un modèle de classification qui nous permettra de déterminer en fonction de plusieurs critères si la personne à un salaire >50k ou <=50k.

3

Ce projet se compose :

- de notre set de données ;
- d'un split/découpage nous permettant d'obtenir un set d'apprentissage et un set de test ;
- d'un modèle de classification non entraîné (un arbre de décision sortant deux classes distinctes) ;
- d'un module d'entraînement ;
- d'un module de scoring qui nous fournit en résultat la classification du salaire ;
- d'un module d'évaluation nous permettant d'évaluer la qualité de notre modèle et de ses prédictions.

Nous lançons ensuite notre expérimentation qui va donc s'entraîner sur les données d'apprentissage puis s'évaluer grâce au set de test. Si le scoring final et l'évaluation nous conviennent, nous validons le modèle. Nous pouvons ensuite déployer notre expérimentation en tant que service Web.

Pour créer un service Web à partir d'un modèle d'apprentissage, nous passons d'abord par une étape intermédiaire : l'expérimentation prédictive. Cette étape nous permet de modifier les entrées ainsi que les sorties de la solution finale, on peut aussi rajouter des briques de traitement de données intermédiaires et ainsi garantir la qualité finale du modèle.

Une fois le service Web déployé, il suffit d'une requête et d'une clé pour y accéder et l'utiliser depuis n'importe quel service :

- un service Web ;
- un code quelconque ;
- Microsoft Excel ;

4

- et bien d'autres.

Conclusion

Ces services, majoritairement gratuits, permettent à tous les profils de faire évoluer leurs projets vers un niveau plus poussé : en utilisant différents outils, on peut alors traiter la majorité des besoins ! •

Denis Duplan
Sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Coder un sine scroll sur Amiga 500

Partie 5

Cet article est le 5e - et dernier !- article d'une série de cinq consacrés à la programmation d'un one pixel sine scroll sur Amiga, un effet très utilisé par les coders de démos et autres cracktros durant un temps. Par exemple, dans cette cracktro du groupe Angels (1).

Dans le premier article, nous avons vu comment installer un environnement de développement sur un Amiga émulé avec WinUAE et coder la Copper list de base pour afficher quelque chose à l'écran. Dans le deuxième article, nous avons vu comment préparer une police de caractères 16x16 pour en afficher facilement les colonnes de pixels des caractères, précalculer les valeurs du sinus requises pour déformer le texte en modifiant l'ordonnée des colonnes, et mettre en place un triple buffering pour alterner proprement les images à l'écran. Dans le troisième article, nous avons vu comment dessiner et animer le sine scroll, d'abord au CPU, puis au Blitter. Dans le quatrième article, nous avons vu comment enjoliver le sine scroll avec quelques effets peu coûteux en cycles assurés par le Copper, et rendre la main aussi proprement que possible à l'OS.

Dans ce cinquième et dernier article, nous allons optimiser le code afin d'être bien certain de tenir dans la trame, et nous protéger des lamers tentés de modifier le texte. Pour terminer, nous verrons s'il n'y a pas quelques leçons à tirer de cette immersion dans la programmation en assembleur du hardware de l'Amiga.

Vous pouvez télécharger l'archive contenant le code et les données du programme présenté ici à l'URL suivante :
http://www.programmez.com

Note

Cet article se lit mieux en écoutant l'excellent module composé par Nuke / Anarchy pour la partie magazine de Stolen Data #7, mais c'est affaire de goût personnel…

Cette archive contient plusieurs sources :

- sinescroll.s est la version de base dont il sera question jusqu'à ce que nous optimisions ;
- sinescroll_final.s est la version optimisée de la version de base ;
- sinescroll_star.s est la version enjolivée de la version optimisée.

Précalculer pour tenir dans la trame

Notre sine scroll est au pixel, ce qui est mieux que celui du groupe Falon. Mais il ne faut cependant pas oublier que nous l'exécutons sur Amiga 1200 et non sur Amiga 500, c'est-à-dire sur un ordinateur bien plus rapide ! Pour savoir si notre code est performant, nous devons le tester sur un Amiga 500.

Pour cela, nous allons mettre l'exécutable sur disquette, et booter à partir de cette dernière dans le contexte d'une émulation d'Amiga 500.

Dans ASM-One, utilisons les commandes en ligne A (Assemble) pour compiler, puis WO (Write Object) pour générer un exécutable et l'enregistrer dans SOURCES: sous le nom de sinescroll.exe.

1 *Sine scroll dans une cracktro du groupe Angels*
https://www.youtube.com/watch?v=luCI-soiN1E

Rendons-nous alors dans le Workbench. Double-cliquons sur l'icône du lecteur DH0, puis sur celle du dossier System et enfin sur celle du Shell.

Pressons F12 pour accéder à la configuration de WinUAE. Dans la rubrique Hardware, cliquons sur Floppy drives. Cliquons sur Create Standard Disk pour créer une disquette formatée au format ADF. Cliquons ensuite sur … à droite du lecteur DF0: et sélectionnons ce fichier pour simuler l'introduction de la disquette dans le lecteur. Cliquons enfin sur OK pour revenir au Workbench.

Dans le Shell, exécutons cette série de commandes pour commander l'exécution de sinescroll.exe lorsque nous booterons avec la disquette :

```
install df0:
copy sources:sinescroll.exe df0:
makedir df0:s
echo "sinescroll.exe" > df0:s/Startup-Sequence
```

L'archive mentionnée au début de cet article contient le fichier ADF qui correspond à la disquette ainsi préparée.

Créons alors une émulation d'Amiga 500 – nous aurons besoin du Kickstart 1.3. La chose faite, insérons la disquette dans le lecteur DF0: et démarrons la simulation en cliquant sur Reset. Le sine scroll se lance automatiquement.

Le résultat tourne tout juste dans la trame – pour ne pas être méchant en disant : pas dans la trame. Difficile de prétendre produire un sine scroll d'aussi belle hauteur que celui de Falon dans ces conditions ? Bah !, nous pourrions recourir à une astuce. Sans la documenter ici, elle consisterait à doubler les lignes à peu de frais, en demandant au Copper de modifier les modulos à chaque ligne

afin de répéter la ligne du dessus une ligne sur deux. Le résultat perdrait en finesse, mais il pourrait tromper son monde.
Il resterait toujours à optimiser le code pour tenir dans la trame. Ce dernier ayant été écrit sans réfléchir à la performance, il ne faudrait pas trop se creuser la tête pour trouver les moyens de réaliser de jolis gains de temps.
A cette fin, il faudrait commencer par se référer au *M68000 8-/16-/32-Bit Microprocessors User's Manual*, qui détaille le nombre de cycles d'horloge pris par une instruction selon la variante qui en est utilisée. Il faudrait aussi s'attarder sur l'*Amiga Hardware Reference Manual*, qui explique la manière dont le CPU et les différents coprocesseurs disposant d'accès DMA se partagent les cycles d'accès à la mémoire durant le tracé d'une ligne – la belle figure 6-9 du manuel.
Il faudrait ensuite travailler sur l'algorithme pour parvenir à un code performant au regard des consommations de cycles qui viennent d'être évoquées. Comme toujours, le premier réflexe devrait être de chercher à sortir de la boucle principale tout ce qui peut être précalculé. Ceci du moment que la mémoire pour stocker des précalculs est disponible.
Par exemple, il est possible de précalculer l'ordonnée de chaque colonne pour toutes les valeurs de l'angle variant entre 0 et 359 degrés. Ainsi, lors de l'affichage d'une colonne, le code exécuté à chaque itération de la boucle principale n'est plus… :

```
lea sinus,a6
move.w (a6,d0.w),d1
muls #(SCROLL_AMPLITUDE>>1),d1
swap d1
rol.l #2,d1
add.w #SCROLL_Y+(SCROLL_AMPLITUDE>>1),d1
move.w d1,d2
lsl.w #5,d1
lsl.w #3,d2
add.w d2,d1
add.w d6,d1
lea (a2,d1.w),a4
```

…mais :

```
move.w (a2,d0.w),d4
add.w d2,d4
lea (a0,d4.w),a4
```

Ou encore, il est possible d'analyser le texte avant la boucle pour créer une liste des colonnes auquelles ce texte correspond. Cette fois, c'est une vingtaine de lignes exécutées à chaque itération de la boucle principale qui sont d'un coup remplacées par les quelques suivantes :

```
cmp.l a1,a3
bne _nextColumnNoLoop
movea.l textColumns,a1
_nextColumnNoLoop:
```

Après avoir épuisé les précalculs, il est possible d'intervenir sur le code. Par exemple, pour supprimer le double test d'attente du Blitter… :

```
_waitBlitter0\@
btst #14,DMACONR(a5)
bne _waitBlitter0\@
_waitBlitter1\@
btst #14,DMACONR(a5)
bne _waitBlitter1\@
```

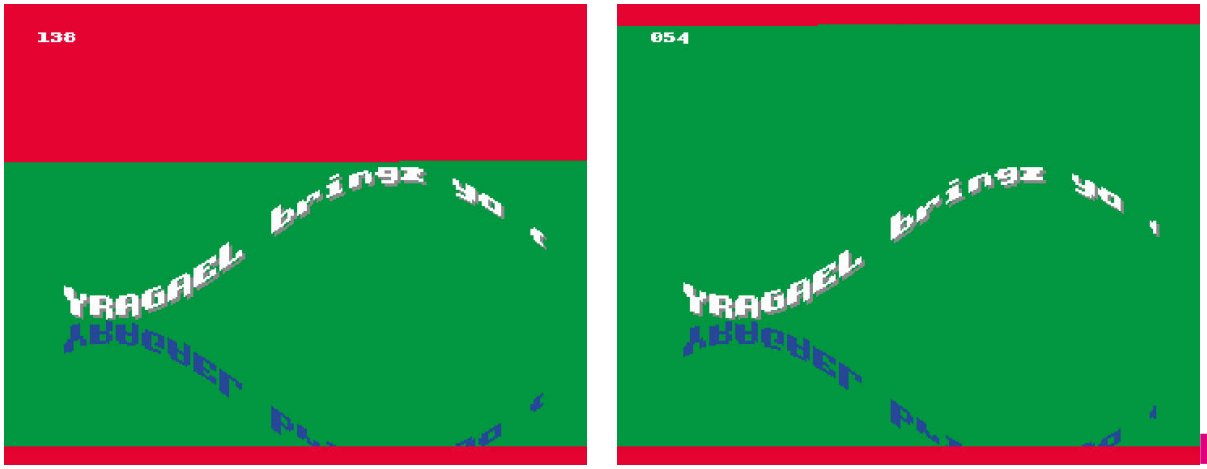
…ce qui donne :

```
_waitBlitter0\@
btst #14,DMACONR(a5)
bne _waitBlitter0\@
```

Ou encore, pour stocker à l'avance $0B4A dans le registre de données du CPU (ici, D3) utilisé pour alimenter BLTCON0 lorsqu'une colonne est tracée au Blitter… :

```
move.w d3,d7
ror.w #4,d7
or.w #$0B4A,d7
move.w d2,BLTCON0(a5)
```

…ce qui donne (pour passer au pixel suivant, ajouter $1000 à D3 et non plus 1, et tester le drapeau C du registre des conditions internes



2 *Temps par trame pris par la version optimisée sur Amiga 500 (à gauche) et Amiga 1200 (à droite).*

du CPU par BCC pour détecter un dépassement du 16ème pixel, lequel entraîne une réinitialisation D3 à la valeur voulue $0B4A qu'il est donc inutile de demander !) :

```
move.w d3,BLTCON0(a5)
```

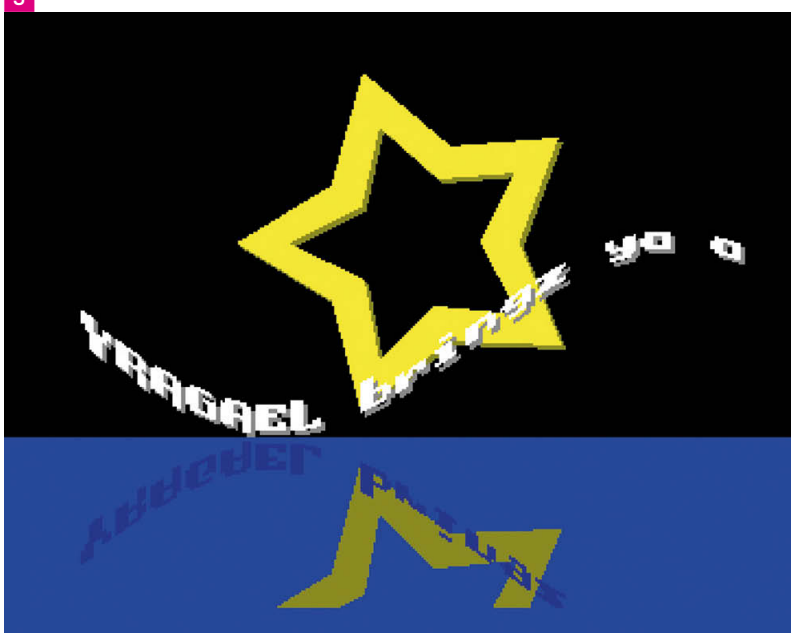
Le source de cette version optimisée correspond au fichier sinescroll_final.s qui se trouve dans l'archive mentionnée au début de cet article.
En bonus, ce source contient un code qui détermine le nombre de lignes parcourues par le faisceau d'électrons entre le début et la fin des calculs d'une trame. Ce code affiche ce nombre en décimal en haut à gauche – en PAL, c'est-à-dire à 50Hz, le faisceau d'électrons parcourt 313 lignes. Pour visualiser ce temps pris par les calculs, la couleur 0 est passée en rouge au début de cette période et en vert à sa fin.
Il est ainsi possible de constater que sur Amiga 500, il faut 138 lignes pour afficher le sine scroll dans la trame, alors que sur Amiga 1200, il en faut seulement 54 (2).

Le gain généré par cette optimisation est important, mais sans doute plus limité, sur Amiga 1200 où le nombre de lignes passe de 62 à 54, soit un gain de 13% . Pour information, le nombre de lignes d'une version où les colonnes sont tracées au CPU, et non au Blitter, passe de 183 à 127 lignes après optimisation, soit un gain de 31% !
Toute économie est toujours bonne à prendre, mais il ne faut pas perdre de vue qu'un précalcul immobilise toujours de la mémoire et génère une attente pour l'utilisateur si le résultat de ce précalcul n'a pas été stocké sous forme de données liées au code dans l'exécutable. En l'occurrence, précalculer les colonnes de la totalité du texte conduit à immobiliser 32 octets par caractère, soit 34 656 octets pour les 1 083 caractères de notre texte. Bon, cela reste raisonnable.
Ainsi, le sine scroll ne tenait pas dans la trame sur Amiga 500. Désormais, il reste largement assez de temps pour l'enjoliver ! Ne nous en privons pas, et sans qu'il soit question de détailler le code que cela implique – le source correspond au fichier sinescroll_star.s qui se trouve dans l'archive mentionnée au début de cet article –, rajoutons pour finir une étoile vectorielle qui tourne dans le fond, avec ombre projetée et reflet dans le miroir comme le sine scroll, ces effets ne coûtant pas plus. 3

Pour afficher le tout, il faut 219 lignes sur Amiga 500, et 103 sur Amiga 1200, sans aucune optimisation – en particulier, le remplissage n'est pas limité à la zone qu'occupe l'étoile, ce qui fait perdre beaucoup de temps sur Amiga 500. Nous pourrions facilement démultiplier la hauteur du sine scroll en répétant des lignes à l'aide d'un jeu sur le modulo au Copper, rajouter un starfield à base de sprites hardware répétés au Copper, agrémenter l'effet avec un beau module de Monty, etc. Mais c'est une autre histoire…

Avec une animation vectorielle, c'est mieux…

3



Se protéger des lamers

Un lamer pourrait ripper notre beau sine scroll ! En particulier, il pourrait utiliser un éditeur hexadécimal pour modifier le texte qui défile. Pour se protéger de ce lamer, adoptons une protection de base dont il fera les frais s'il entreprend de s'y attaquer.
Pour que le texte ne soit pas apparent, nous devons l'encoder. Contentons-nous de combiner les octets de ses caractères par XOR avec TEXT_XOR, un octet de valeur quelconque. Ainsi, les caractères n'apparaissent pas comme des caractères dans un éditeur hexadécimal.
Et si tout de même le lamer devait deviner l'opération – ce dont nous lui laissons délibérément la possibilité en exposant le texte encodé à une attaque fondée sur l'analyse de récurrences –, calculons TEXT_CHECKSUM, un checksum du texte encodé, et rajoutons ici et là des appels à un code qui vérifie que le texte n'a pas été altéré. Ce code calcule le checksum du texte courant et le remplace par « *You are a LAMER!* » (de checksum TEXT_CHECKSUM_LAMER) si ce checksum ne correspond à aucun des checksums de nos désormais deux textes originaux. Ne factorisons pas ce code, mais répétons-le, pour en demander l'exécution en divers endroits pour que le lamer ne puisse s'en débarrasser en substituant simplement un RTS à la première instruction de ce qui constituerait autrement son unique occurrence.
Attention au contexte dans lequel la macro est utilisée, car elle peut modifier la longueur du texte initial (qui doit être au moins aussi long que "You are a LAMER!" sous peine d'écraser des données) et donc embrouiller le code qui était en train de le parcourir.

```
CHECKTEXT: MACRO
    movem.l d0-d1/a0-a1,-(sp)
    lea text,a0
    clr.l d0
    clr.l d1
_checkTextLoop\@
    move.b (a0)+,d0
    add.l d0,d1
    eor.b #TEXT_XOR,d0
    bne _checkTextLoop\@
    cmp.l textChecksum,d1
    beq _checkTextOK\@
```

4 *La punition du lamer qui modifierait le texte du sine scroll.*

```
    move.l #TEXT_CHECKSUM_LAMER,textChecksum
    lea text,a0
    lea textLamer,a1
_checkTextLamerLoop\@
    move.b (a1)+,d0
    move.b d0,(a0)+
    eor.b #TEXT_XOR,d0
    bne _checkTextLamerLoop\@
_checkTextOK\@
    movem.l (sp)+,d0-d1/a0-a1
    ENDM
```

Très easter egg. On est jeune, on rigole. 4

Quelques mots pour conclure

Coder en assembleur 68000 est un travail exigeant. Le nombre important de registres disponibles et le souci permanent d'en optimiser l'usage conduit le codeur à empiler et dépiler dans sa propre mémoire l'usage qu'il en fait tandis qu'il progresse dans l'écriture du code. Dans mon souvenir, celui qui code en assembleur 80x86 est moins confronté à cette exigence, car le nombre des registres est si faible et leurs usages tellement contraints qu'il est indispensable de s'appuyer sans cesse sur la pile du CPU, pile dont il est plus facile de se souvenir du contenu que de celui de 13 registres.

Je n'étais pas parti dans l'idée de me remettre à coder sur Amiga lorsque j'ai entrepris de revisiter le code d'une cracktro dans la série d'articles précédents. C'est en relisant l'*Amiga Hardware Reference Manual* que je me suis rappelé que je n'avais jamais poussé bien loin l'étude de la manière dont le Blitter trace des lignes, fonctionnalité que je savais être utilisée pour produire notamment un sine scroll. Finalement, j'ai voulu clarifier les choses, et j'ai programmé à partir de rien cet effet.

Plus généralement, en feuilletant ce manuel mais aussi ceux du 68000, j'ai pu constater combien j'étais resté sur une vision très superficielle du fonctionnement du hardware et du CPU à l'époque. Si j'ai donc une leçon à formuler, c'est que chaque fois qu'on s'intéresse à une technologie, il faut se donner la peine de lire scrupuleusement l'intégralité de sa documentation de référence plutôt que de se contenter, par pure fainéantise, de s'en remettre à son intuition. C'est qu'à ce régime, on prend non seulement le risque de manquer des fonctionnalités importantes, mais de plus celui d'en mal comprendre certaines. Par exemple :

```
btst #14,$dff002
```

De prime abord, cette instruction teste le bit 14 du mot se trouvant à l'adresse $DFF002. En fait, la lecture de la description de BTST dans le *M68000 Family Programmer's User Manual* révèle que lorsque le premier opérande est N et le second est une adresse, c'est le bit N%8 (ie : N modulo 8) de l'octet se trouvant à l'adresse qui est testé. En l'espèce c'est donc le bit 14%8=6 de l'octet se trouvant à l'adresse $DFF002 qui est testé. Cela correspond bien au bit 14 de l'octet de poids fort du mot se trouvant à cette adresse, si bien que notre intuition se révèle pertinente. Toutefois, c'est par chance. Présumer ainsi de certains fonctionnements peut générer des erreurs d'autant plus difficiles à corriger qu'on est loin de soupçonner où elles se logent.

La lecture de la documentation de référence s'impose donc toujours comme un préalable difficilement contournable pour qui souhaite maîtriser véritablement une technologie. Et je dis bien la documentation de référence dans le texte, et non une de ses formes vulgarisées. C'est qu'au prétexte de rendre un savoir accessible, la vulgarisation prend trop souvent des libertés avec ce dernier, empruntant des raccourcis et faisant des impasses qui ne font que fourvoyer le talent et encourager la médiocrité. Une forme vulgarisée d'une documentation de référence ne doit jamais être considérée que comme un point d'entrée sur cette dernière. Elle ne saurait dispenser d'au moins en tenter la lecture, quand bien même cette entreprise peut se révéler ardue. C'est que la tendance est malheureusement plus aux exposés formels que didactiques – les auteurs des spécifications des technologies du Web auraient tout à gagner à lire l'*Amiga Hardware Reference Manual* !

Ce sera tout pour cette fois, et sans doute pour toujours en ce qui concerne la programmation en assembleur de l'Amiga – à laquelle je ne m'étais pas adonné depuis bientôt un quart de siècle. Je dédie ce travail à un vieux pote, Stormtrooper, sans la motivation duquel je n'aurais jamais entrepris de me mettre au metal bashing à l'époque, et à tous ceux dont les pseudos défilent dans les inévitables greetings que pourront lire les courageux qui compileront le source de ce sine scroll. « *Amiga rulez!* » •

Liens utiles

WinUAE : http://www.winuae.net/
Amiga Forever : https://www.amigaforever.com/
ASM-One : http://www.theflamearrows.info/documents/ftp.html
ReqTools : http://aminet.net/package/util/libs/ReqToolsUsr.lha
Amiga Hardware Reference Manual : http://amigadev.elowar.com/read/ADCD_2.1/Hardware_Manual_guide/node0000.html
M68000 8-/16-/32-Bit Microprocessors User's Manual : http://www.nxp.com/assets/documents/data/en/reference-manuals/M68000PRM.pdf
M68000 Family Programmer's User Manual : http://cache.freescale.com/files/32bit/doc/ref_manual/MC68000UM.pdf
Le manuel d'ASM-One : https://archive.org/details/AsmOne1.02Manual

Stagiaires & API

Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : S. Saurel
Nos experts techniques : O. Denoo, B. Legeard, G. Everitt, J. Paquet, B. Homes, Y. Phélizot, A. Chaouat, C. Pichaud, P. Prémartin, M. Bertocchi, J. Franel, S. Maire, B. Herbigniaux, S. Dolcini, D. Duplan, N. Robert, C. Thomas, S. Vallée, P. Ficheux, E. Auberix, E. Shen, V. BechuL. Devulder, E. Shen, V. Bechu, F. Le Corre, S. Franck, N. Root
Stagiaire : Maxime Ellerbach
Couverture : © D.R. - **Maquette** : Pierre Sandré.
Publicité : PC Presse, Tél.: 01 74 70 16 30, Fax : 01 40 90 70 81 - pub@programmez.com.
Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34

Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, mars 2018

Abonnement :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com*
Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30.

Tarifs
Abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter.

PDF
35 € (monde entier) souscription sur www.programmez.com

Sur abonnement ou en kiosque

Le magazine des pros de l'IT

Mais aussi sur le web

www.linformaticien.com

L'INFORMATICIEN

WINDEV® 23 Tech Tour

VOUS ÊTES INVITÉ

SÉMINAIRE
100%
TECHNIQUE

DSI
Développeurs
WebDesigners
Architectes
Ingénieurs
Start-ups...

10.000 places seulement. Réservation nécessaire

INSCRIVEZ-VOUS VITE !

MONTPELLIER	mardi 13 Mars
• GENÈVE	mardi 20 Mars
LYON	mercredi 21 Mars
STRASBOURG	jeudi 22 Mars
• BRUXELLES	mardi 27 Mars
LILLE	mercredi 28 Mars
PARIS	**jeudi 29 Mars**
NANTES	mardi 3 Avril
BORDEAUX	mercredi 4 Avril
TOULOUSE	jeudi 5 Avril
MARSEILLE	mardi 10 Avril
• MONTREAL	jeudi 12 Avril

GRATUIT ***13h45 à 17h45***

WINDEV TECH TOUR
SÉMINAIRE DÉVELOPPEMENT
WINDOWS WEB LINUX ANDROID IOS
DÉVELOPPEZ 10 FOIS PLUS VITE

PARMI LES 23 SUJETS:

BLOCKCHAIN: BITCOIN EN WINDEV • CHIFFREMENT •RGPD • IOT • RÉALITÉ VIRTUELLE • SITE WEB MOBILE FRIENDLY • SINGLE SIGN ON • UN MAGAZINE SUR MOBILE • ANALYSE DE DOCX ET DE XLSX • FRONT END BACK END LES BONNES PRATIQUES...

PC SOFT®
WWW.PCSOFT.FR