PROGRAMMEZ!

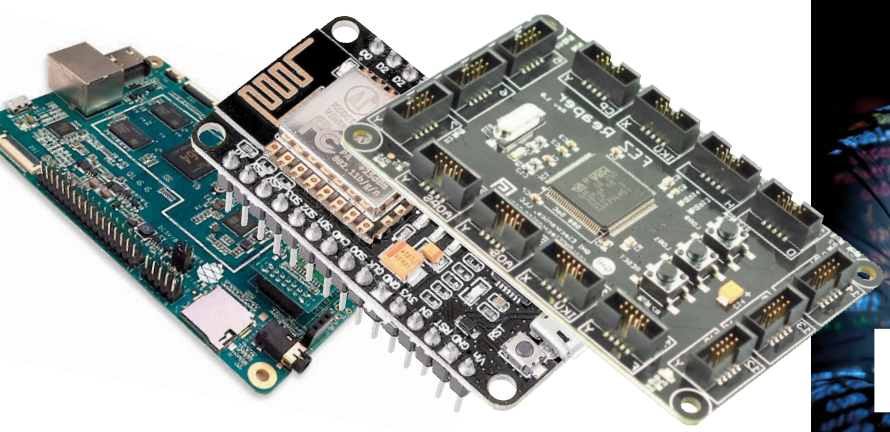
#206 - avril 2017

le magazine des développeurs

Visual Studio 2017

La nouvelle version de l'IDE de développement de Microsoft

Arduino, Raspberry Pi, ESP8266, C.H.I.P., Omega2, ODROID...



Notre guide complet

Et si on utilisait enfin les langages fonctionnels ?

Dossier tests

Pourquoi et comment utiliser les tests ?

© DragonImages

YouTube

100 % pratique

Embarquez des vidéos dans vos apps Android avec l'API YouTube

M 04319 - 206S - F: 6,50 € - RD

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

SERVEURS DÉDIÉS XEON®

AVEC ikoula HÉBERGEUR CLOUD

POUR LES LECTEURS DE L'INFORMATICIEN*

OFFRE SPÉCIALE -60 %

À PARTIR DE

11,99 €* HT/MOIS

~~29,99 €~~

CODE PROMO **XEINF17**

intel inside™ XEON®

Optez pour un serveur dédié dernière génération et bénéficiez d'un support technique expérimenté.

debian ubuntu CentOS Windows Server 2012

- Assistance technique en **24/7**
- Interface **Extranet** pour gérer vos prestations
- **KVM sur IP** pour garder l'accès
- Analyse et surveillance de **vos serveurs**
- **RAID matériel** sur tous nos serveurs
- Large choix d'**OS** Linux et Windows

* Offre spéciale -60 % valable sur la première période de souscription avec un engagement de 1 ou 3 mois. Offre valable jusqu'au 31 décembre 2017 23h59 pour une seule personne physique ou morale, et non cumulable avec d'autres remises. Prix TTC 14,39 €. Par défaut les prix TTC affichés incluent la TVA française en vigueur.

CHOISISSEZ VOTRE XEON®

https://express.ikoula.com/promoxeon-inf

ikoula HÉBERGEUR CLOUD

/ikoula @ikoula sales@ikoula.com 01 84 01 02 50

NOM DE DOMAINE | HÉBERGEMENT WEB | SERVEUR VPS | SERVEUR DÉDIÉ | CLOUD PUBLIC | MESSAGERIE | STOCKAGE | CERTIFICATS SSL

“Vous ne passerez pas”

Imaginez un instant Gandalf en testeur et le Balrog des mines de la Moria comme le démon des bugs. Que va faire Gandalf ? Il va l'affronter pour sauver les utilisateurs. Et il va se sacrifier pour empêcher le Balrog de passer et de compromettre la mission de la communauté de l'Anneau.

Les tests sont omniprésents dans notre vie quotidienne même si nous ne les voyons pas forcément. Ainsi pour homologuer de nouveaux produits, des voitures, des casques, des jouets, etc., des tests sont pratiqués pour accepter ou rejeter le produit. Les tests logiciels, c'est un peu la même chose. Mais comme dans la vraie vie, tout n'est pas parfait et parfois, des problèmes sont passés inaperçus, avec malheureusement de temps en temps des conséquences mortelles.

Déjà quand j'étais testeur dans les années 1990, on discutait beaucoup des tests, de la manière de les réaliser. Et je n'avais pas à ma disposition tous les outils que l'on a aujourd'hui, ni les méthodes de testing. Mais le test est l'étape obligatoire pour valider des développements et essayer de supprimer le plus grand nombre de bugs avant le déploiement de son application.

Oui, le test n'est pas le sujet le plus fun que l'on puisse avoir dans Programmez!. Cependant, impossible de ne pas en parler. Notre dossier spécial aborde les différents tests et quelques méthodes et outils. Vous verrez aussi que le test a été bouleversé par le Cloud, les méthodes agiles et le DevOps.

Programmez! fait appel à votre soutien pour notre nouveau projet : Programmez! Collectors. Le but est de publier des numéros spéciaux à tirage limité sur un thème unique. Le premier numéro sera 100 % vintage : les ordinateurs mythiques des années 80 et 90, la programmation à l'ancienne, le retrogaming et les retroconsoles. Pour concrétiser ce projet, nous avons lancé une campagne de financement participatif sur ulule qui se termine mi-avril.

Pour soutenir le projet :
https://fr.ulule.com/programmez-collectors/

Dans le prochain numéro !
Programmez! #207, dès le 29 avril 2017

Dossier cloud & DevOps
Amazon Alexa
L'art du debug

DevCon #3

CONFÉRENCE PROGRAMMEZ! LE 23 MAI

Programmez! organise sa 3e conférence technique sur 3 thèmes :

API, le Cloud Computing et les IoT.

- *2 keynotes*
- *4 sessions techniques*
- *1 pizza party*

+ un track spécial sur la plateforme IoT "Constellation"

Informations & inscriptions sur
www.programmez.com section DevCon

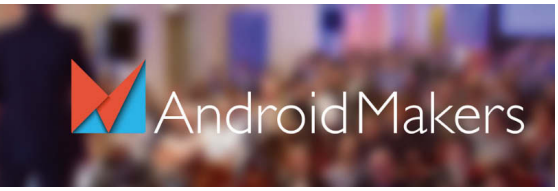
Où : la conférence se déroulera à Issy les Moulineaux.
A -5 minutes des RER C et T2.

Accueil à 13h30. Début des conférences à 14h. Fin à 19h30.

Avril

ANDROID MAKERS

10 & 11 avril / Paris



DroidCon Paris est mort, vive Android Makers. Nouveau format, nouveau nom pour parler Android. Android Makers se veut le plus grand rassemblement Android de France. Des rockstars d'Android seront au rendez-vous comme Lisa Wray, Ty Smith ou encore Cyril Mottier pour des talks, des workshops et des démos de makers triés sur le volet.
Pendant deux jours vous apprendrez tout sur l'Android Everywhere, les dernières tendances en Android Development et en UX/UI.
Inscription : http://androidmakers.fr

DEVOXX FRANCE

du 5 au 7 avril / Paris

La grand'messe Java revient pour une nouvelle année. La première journée est dédiée aux présentations longues et aux ateliers. Comme chaque année, vous aurez l'embarras du choix des sessions : +200 ! Toutes les dernières tendances du monde Java et des technologies en général seront présentées avec des keynotes toujours originales et surprenantes. Le village startup va grandir et accueillir 14 pousses.

JOURNÉE FRANÇAISE DES TESTS LOGICIELS

11 avril / Montrouge

Cette 9e édition se tiendra le 11 avril à Montrouge. 16 conférences seront organisées durant la journée. Le 10 avril se déroulera une journée complète de tutoriels. "Afin de permettre aux professionnels d'approfondir certaines problématiques du test au travers d'enseignements pratiques, le CFTL propose, dans une approche pédagogique, différents tutoriels d'une durée de 3 heures, dispensés par des praticiens expérimentés, sur les thèmes de l'Utilisabilité, du test en contexte agile, de l'automatisation pour le Web et le mobile, des processus métier et test de recette, et des environnements de test. » dixit les organisateurs.

REBUILD 2017

27 avril / Nantes

Les communautés Microsoft seront à l'honneur le 27 avril pour la grande journée Rebuild. Plus de 40 sessions seront jouées. Elles seront réparties en 3 tracks : décideurs, IT et développeurs. Une bonne occasion d'échanger et de parler technologies et tendances actuelles.

Mai

PHP TOUR 2017

18 et 19 mai / Nantes

Le PHP Tour, le cycle de conférences itinérant de l'AFUP pour toutes les communautés PHP, professionnelles et open-source, pose de nouveau cette année ses valises au C.C.O. de Nantes. Il sera question du futur ! PHP a célébré ses 20 ans en 2015, mais ne s'endort pas pour autant sur ses lauriers. Le langage a le regard obstinément tourné vers l'avenir et le programme du PHP Tour 2017 reflète cette vivacité : les conférences vont couvrir le large spectre de PHP.
Le site du PHP Tour 2017 à Nantes :
http://event.afup.org/
Le programme :
http://event.afup.org/phptournantes2017/programme/
La billetterie :
http://event.afup.org/phptournantes2017/tickets-inscriptions/

Juin

DEVFEST LILLE 2017

9 juin / Lille

Le Google Developer Group de Lille (@GDGLille) organisera la première édition du DevFest Lille. Au programme, des conférences et des codelabs autour des technologies que nous aimons tous : Web, Mobilité, Cloud, IoT. Les inscriptions et le CFP sont ouverts. Rendez-vous sur https://devfest.gdglille.org/.

MAKER FAIRE PARIS 2017

du 9 au 11 juin / Paris

Cette nouvelle édition du Maker Faire se déplace à la Villette et se tiendra en dehors de la foire de Paris comme ce fut le cas les années passées. Comme chaque année, ce sera l'occasion de découvrir des centaines de makers et de projets divers et variés. L'appel aux makers a été lancé il y a quelques jours.

[2017]

DÉVELOPPEZ 10 FOIS PLUS VITE

WINDEV®

WINDEV 22 : ATELIER DE DÉVELOPPEMENT PROFESSIONNEL, COMPLET EN STANDARD

Gestion du cycle de vie complet: Idée, Conception, Développement, Génération, Déploiement, Exploitation • Un code multi-plateformes Windows, Linux, Java, Internet, Mobiles • Environnement ALM complet • Toutes les bases de données sont supportées, Big Data • Inclus: HFSQL, base de données locale Client/Serveur, cluster, embarquée et cloud • Puissant RAD • Intégration continue • Tableau de bord de vos applications • Audit statique & dynamique • Générateur de fenêtres (UI) visuel & intuitif • Utilisation facile de charte graphique • Héritage et surcharge d'interface • Tous les champs (contrôles) sont très puissants et livrés en standard: Champ de saisie, Tableau croisé dynamique (cube), Champ Planning, Champ Diagramme de Gantt, Champ Tableau de bord, Champ Table, Champ Graphe, etc • FAA: chaque application bénéficie automatiquement de: Fonctionnalités Automatiques: export vers Excel, vers Word, envoi d'email, etc • Sécurité : Mot de passe de vos applications • Puissant générateur de rapports et codes-barres • Langage de 5ème génération: WLangage • Editeur de code intuitif avec puissant débogueur • Tests unitaires et tests automatiques • Versioning (GDS/SCM) • Webservices SOAP et Rest • Modélisation: Merise et UML • .NET, 3-Tier, MVP • Support de tous les standards: XML, USB, Bluetooth, NFC, J2EE, OLE, ActiveX, RPC, SaaS, SMTP, FTP, OPC, DLNA, IoT, Sockets, API, Webservices... • Lien avec Lotus Notes, SAP, Google, Outlook... • Multimédia, Domotique • Livré avec des centaines d'exemples et d'assistants • Génère le Dossier technique d'un clic • Télémétrie pour connaître l'utilisation réelle de vos applications • Générateur d'aide • Support de 64 langues étrangères par application • Générateur de procédures d'installation: local, CD, USB, Internet, Réseau, Push... • Robot de surveillance: surveillez vos applications • Gestion des suggestions et incidents utilisateurs • Support Technique Personnalisé Gratuit* • ...

CONSULTEZ PLUS DE 100 TÉMOIGNAGES DE PROFESSIONNELS SUR LE SITE PCSOFT.FR

Elu
«Langage le plus productif du marché»

VERSION EXPRESS GRATUITE
Téléchargez-la !

VU À LA TÉLÉ EN 2017
SUR **TF1**, SUR **BFMTV** ET SUR **M6**

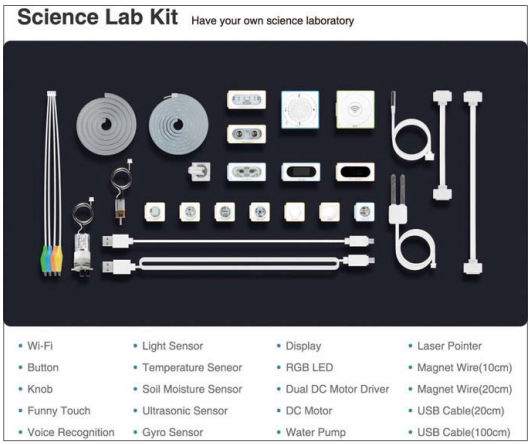
*Logiciel professionnel - * 15 requêtes gratuites sur la version en cours de commercialisation*

Tél Paris: 01 48 01 48 88 *Tél Montpellier: 04 67 032 032*

Plus de 100 témoignages
Dossier complet gratuit

WWW.PCSOFT.FR

MAKEBLOCK NEURON



Un autre projet super intéressant : des blocs intelligents avec intégration des capteurs et modules électroniques. On assemble, et les fonctions de base sont déjà implémentées. Ces blocs aimantés se programment avec un outil visuel. Les blocs se connectent très simplement. Une trentaine de blocs fonctionnels seront disponibles : réseau, contrôleurs, capteurs divers. L'outil sera disponible sur macOS et Windows (mBlock). L'app mobile le sera sur iPad et tablettes Android. Le SDK est open source et il sera simple de connecter des services de type MS Cognitive Services. La disponibilité est attendue pour juin prochain. Le kit le plus complet sera vendu 599 $. La campagne Kickstarter a fait exploser les contributions attendues.

UN NOUVEAU LANGAGE DE PROGRAMMATION : GRAVITY

Les outils et langages de programmation mobile sont nombreux. Le langage Gravity est le petit nouveau. Il propose un langage léger pour écrire des apps Android et iOS. Il est écrit en C et la syntaxe est proche de celle de Swift. Le maître mot est la légèreté : le compilateur et la machine virtuelle pèsent 200 Ko et fonctionnent en 64-bit. Projet disponible sur https://github.com/marcobambini/gravity

LE FUTUR EST-IL AU QUANTIQUE ?

En mars, l'ordinateur quantique a été un sujet très discuté, mais il est vite retombé. IBM, Google et D-Wave furent cités, tout comme la NASA. Mais ces acteurs ne sont pas nouveaux dans le quantique, ils y travaillent depuis de nombreuses années. Peu à peu, il va commencer à se déployer par exemple dans certaines entreprises comme Volkswagen ou encore via le service quantique disponible dans IBM Bluemix. Cependant, nous sommes très loin d'un ordinateur quantique produit en série et à un prix acceptable. Il faudra encore de nombreuses années avant de déployer un supercalculateur quantique. L'ordinateur quantique souffre de plusieurs problèmes : les usages, la puissance brute, le coût et le modèle de développement. Pour aider les développeurs, IBM propose des SDK et des API dédiés : le projet QUISkit. Le projet se compose de 3 éléments :

- API : en python pour se connecter et exécuter du code OpenQasm ;
- SDK : surtout du projet Quantum Experience (IBM). Il propose des exemples de codes ;
- OpenQasm : spécifications, exemples, outils et de documentations.

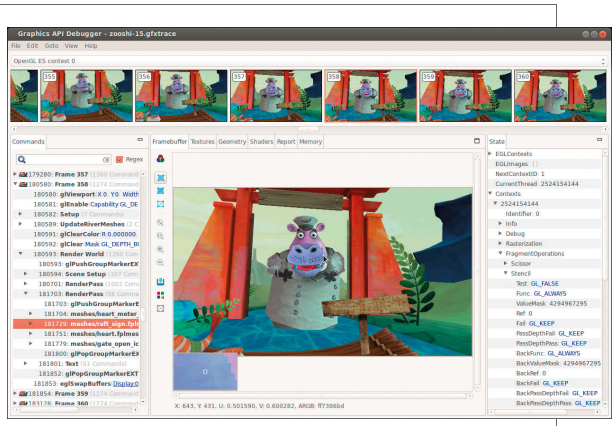
Pour en savoir plus :
https://developer.ibm.com/open/openprojects/qiskit/

SÉRIES & FILMS

L'agenda printemps / été est très chargé :

- 23 avril : saison 4 de Silicon Valley.
- 19 mai : Alien Convenant (ex-Prometheus 2) va pouvoir libérer les xénomorphes.
- 21 mai : Twin Peaks saison 3 ! Oui, enfin !
- 30 mai : House of Cards saison 5.
- 16 juillet : Game of Thrones saison 7 ! L'attente va être longue, très longue !
- Août (?) : dernière saison de Halt and catch fire.
- 7 octobre : Mr Robot saison 3.
- Septembre – octobre (?) : Star Trek Discovery. Enfin, la nouvelle série Star Trek arrive !

GAPID : L'ART DU DÉBUG 3D



Comment déboguer une app Android OpenGL ES ou Vulkan ? Le projet open source Gapid est fait pour vous ! Il permet de tracer les apps 3D depuis une interface fonctionnant sur macOS, Linux et Windows. Un outil à suivre ! Site : https://github.com/google/gapid

Des failles exploitées par la NSA ? WikiLeaks pourrait aider, mais rien n'est certain

La pub est bien arrivée dans Windows 10

SuSe complète ses gammes en rachetant les activités OpenStack et Cloud Foundry de HPE (ex-HP)

Quand tu fais un tar xzvf, penser à ne pas le faire depuis la racine d'un volume

PROJET TORINO : UN LANGAGE DE PROGRAMMATION PHYSIQUE

Microsoft a dévoilé un projet de recherche très intéressant : le projet Torino. Il s'agit de créer un langage de programmation physique pour faire comprendre la programmation et la logique de programmation aux enfants de 7 à 11 ans souffrant de déficiences visuelles. Avec Torino, les blocs de couleurs sont des éléments du langage que l'on assemble. Google avait dévoilé un projet un peu similaire : Project Bloks.

RETROGAMING : L'ÉVÈNEMENT AC 2017

C'est l'un des temps forts du retrogaming et du vintage computing : la convention AC organisée par l'association Retro-gaming Connexion. Elle aura lieu les 15 et 16 avril à Congis (77). L'AC est un rassemblement autour de l'informatique des années 70, 80 et 90, avec une place importante à la rétroprogrammation, le speed coding !

ikoula
HÉBERGEUR CLOUD

PRÉSENTE

CLOUD**IKOULA**ONE

Le succès est votre prochaine destination

MIAMI SINGAPOUR PARIS
AMSTERDAM FRANCFORT ___

*CLOUD**IKOULA**ONE* est une solution de Cloud public et privé qui vous permet de déployer en 1 clic et en moins de 30 secondes des machines virtuelles à travers le monde sur des infrastructures SSD haute performance.

www.ikoula.com sales@ikoula.com 01 84 01 02 50

ikoula HÉBERGEUR CLOUD

CLOUD | INFOGÉRANCE | SERVEUR DÉDIÉ | VPS | MESSAGERIE

Atanas Popov (Embarcadero - General Manager)
Le retour en force des ex-IDE Borland

Vous vous souvenez sans doute de Borland, co-fondée par Philippe Kahn en 1983. L'éditeur est rapidement devenu une référence pour les développeurs avec Turbo Pascal, C++ Builder ou encore Delphi, et sans oublier JBuilder. Borland se sépare des IDE en 2006 et dès 2008, Embarcadero rachète CodeGear, la filiale gérant ces outils. L'éditeur a dévoilé de grandes ambitions et une roadmap non moins ambitieuse. Atanas Popov, general manager of Application Development Tools Business, fait le point pour Programmez ! sur les nouveautés, les nouveaux marchés et le futur des outils.

Le marché des EDI a connu beaucoup de bouleversements. Borland n'a pas été épargné ni les différents outils rachetés par Embarcadero. Il n'existe finalement plus beaucoup de grands EDI. Comment analysez-vous le marché des EDI et existe-t-il toujours une place pour les EDI payants ?

Nous pensons que le marché des EDI reste porteur et les opportunités pour les EDI payants sont énormes, comme en témoigne le succès continu de RAD Studio. Même en considérant l'environnement Java, on note un léger déclin d'Eclipse et une croissance des outils payants. En outre, il existe de nouveaux EDI JS payants, notamment pour répondre aux besoins des entreprises. En matière de formation et pour les débutants, il est néanmoins important d'avoir une offre gratuite. C'est pour cela que nous proposons désormais une édition « Starter » gratuite pour donner une opportunité à tous d'essayer RAD Studio. De même, nous offrons des versions GRATUITES pour les établissements de formation. Le marché des EDI est en pleine croissance, et nous pensons contribuer à cet essor. Nous pensons qu'il existe un grand nombre de nouveaux utilisateurs potentiels qui peuvent être séduits par la vitesse et l'efficience de RAD Studio.

La France a longtemps occupé une position particulière vis-à-vis de Delphi et JBuilder. Quelles sont vos ambitions sur le marché français et comment pensez-vous vous adresser aux développeurs ?

RAD Studio est traduit en plusieurs langues, et notamment en français. Ceci pour apporter au marché français une expérience utilisateur complète et de qualité. Nous comptons investir beaucoup plus dans ce sens afin de nous rapprocher des développeurs français et de partager avec eux les avantages clés de RAD Studio : productivité, retour sur investissement et succès.

Vous proposez une ambitieuse feuille de route de développement. Où en êtes-vous ? Allez-vous tenir la cadence dans les prochaines années ? N'est-ce pas un rythme trop élevé pour les développeurs ?

Nous avons effectivement une « feuille de route » assez ambitieuse qui va de pair avec notre envie d'offrir toujours plus d'outils et de fonctionnalités. Nous pensons pouvoir atteindre ces objectifs et cette cadence pour satisfaire les attentes de nos clients.

Le développement mobile est l'une des fonctions clés pour les développeurs. Google propose Android Studio, Microsoft vient de racheter Xamarin. Quelle est votre offre dans ce domaine ? Quelles technologies (Xamarin, Cordova, autre) supportez-vous ?

Nous offrons la meilleure solution « source unique/IHM unique » du marché avec FireMonkey (FMX). Les développeurs peuvent construire une seule application dans la même IHM et la déployer ensuite sous Windows, MacOS, iOS et Android. Aucun autre outil ne permet cela. Nous proposons une solution simple et transparente supprimant toutes les contraintes de gestion de multiples systèmes d'exploitation. Nous croyons fortement au marché mobile.

Conteneurs, IoT, Fog Computing, Cloud Computing, DevOps, etc. Un EDI doit prendre en charge de plus en plus de technologies. Quelles sont vos priorités ?

Nous couvrons déjà l'environnement IoT avec FireMonkey et RAD Server. En effet, Embarcadero a remporté de multiples prix en 2016 dans ce domaine. Nous comptons développer nos solutions de Cloud Computing et d'intégration continue. Notre offre RAD Server permet un déploiement simplifié sur différents environnements Cloud.

Sur la partie compilation, les architectures ont beaucoup évolué. Microsoft propose l'approche très modulaire de Roslyn, et côté Open Source, nous disposons aussi du LLVM. Quelle est votre opinion sur ce sujet et ces architectures ?

Nous adhérons pleinement à la structure de compilation sous-jacente de LLVM et nous comptons l'utiliser à terme avec tous nos compilateurs. LLVM est extrêmement flexible et puissant, ce qui nous a permis d'offrir un support sur toutes les plateformes disponibles aujourd'hui

Les plus gros IDE posent des problèmes : installation très lourde (parfois +20 Go), lenteur d'exécution, exigences RAM, écrans toujours plus grands, problèmes de stabilité, etc. De quelle manière, prenez-vous en charge ces problématiques ?

Les EDI modernes sont très puissants et offrent de riches fonctionnalités. Nous avons l'avantage de développer RAD Studio de façon récursive. Ceci permet à nos développeurs d'anticiper tous les problèmes potentiels (aussi légers soient-ils) pour offrir aux clients finaux un outil plus puissant que jamais. Après tout, ils veulent utiliser les outils les plus rapides et les plus performants. RAD Studio accomplit exactement cela. L'année dernière, nous avons investi dans une nouvelle technologie dénommée Getit. Getit permet une expérience de téléchargement ultra rapide et permet aussi de distribuer les composants par le biais du processus de téléchargement. L'expérience utilisateur est primordiale pour réussir dans cette industrie et nous avons fait des progrès considérables. Il reste néanmoins beaucoup à faire dans ce sens. Embarcadero, est synonyme d'outils de développement. Nous ne faisons rien d'autre. Nous sommes extrêmement concentrés à rendre les développeurs heureux, productifs, et autonomes. Vous trouverez ici le lien de téléchargement, édition gratuit de Delphi Starter : https://www.barnsten.com/fr/development-tools/free-tools/delphi-starter

Abonnez-vous à 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

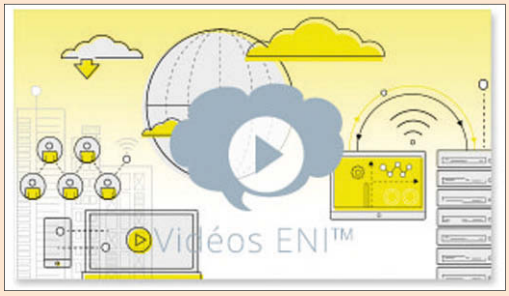
Souscription uniquement sur
www.programmez.com

Option :
accès aux archives 10€

Nos offres spéciales printemps 2017

1 an ..... **59€**
11 numéros + 1 vidéo ENI au choix :
- Big Data avec Hadoop
- Framework Spring

(Valeur de la vidéo de 29,99 à 59,99 €)



2 ans ..... **89€**
22 numéros + 1 vidéo ENI au choix :
- Big Data avec Hadoop
- Framework Spring

(Valeur de la vidéo de 29,99 à 59,99 €)

Offre limitée à la France métropolitaine

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 : Big Data avec Hadoop
☐ Vidéo : Framework Spring

PROG 206
Offre limitée, valable jusqu'au 30 avril 2017

☐ 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**

Mon cher ami,

Je t'écris suite au mail que tu m'as envoyé concernant l'agilité et l'arrivée de scrum dans ton équipe. Je sais aussi à quel point pour toi, diriger des hommes (et particulièrement les développeurs), contrôler, imposer jusque dans le détail une démarche long terme, participe à ta réalisation et ton équilibre personnel. Chose, comme je te l'ai déjà dit, que j'ai toujours su respecter.

Je voudrais te rassurer et te donner les éléments qui te permettront de conserver ce modèle qui te convient tant. Tu auras compris que lorsque je mets dans l'objet de mon email que scrum est un piège, je parle pour ceux qui y croient. De toute façon c'est la plateforme idéale pour corrompre l'agilité. C'est vrai, j'aurais été plus emprunté, si un autre framework agile comme XP, Kanban et plus particulièrement Crystal Clear, t'avait été imposé. Mais comme il n'en est rien, avançons.

Tout d'abord pour ce qui est du sprint, il est fondamental que tu n'ailles pas régulièrement en production. By the book, Scrum ne t'impose pas d'aller en production à la fin de chaque sprint. Utilise cela, ainsi tu pourras conserver tes deux releases par an. L'équipe aura l'impression de vivre des itérations agiles mais dans la pratique elle suivra tes deux tunnels annuels. Ne t'inquiète pas du feedback que peuvent te donner les utilisateurs lors de la démonstration, cela ne peut rester que superficiel, car seul l'exercice de l'outil en situation (c-à-d en production) peut donner un retour solide et changer fortement ta direction. Petite astuce, si tu le peux, fais en sorte que seul le product owner soit présent, tu pourras ainsi mieux maintenir les couches de séparation classique entre le développeur et l'utilisateur final.

Pour ce qui est de la durée du sprint, impose avec élégance la durée minimale qui est d'une semaine. Tu auras dans cette démarche le support complet du scrum master. Tu verras que chez ces gens-là, la virilité se mesure à la taille du sprint, et celle-ci est inversement proportionnelle aux règles communément appliquées dans la nature. Pour ce qui est du scrum master je t'en parlerai un peu plus loin. Pourquoi une semaine ? Tout simplement pour mettre une pression permanente sur l'équipe. De plus, une jeune équipe agile ne peut tenir ce rythme. Ajoute à cela et c'est très important, un deuxième mécanisme de contrôle : conserve tout simplement le suivi par projet. En effet, l'équipe devra souffrir à la fois du sprint court, mais aussi de la projection projet sur laquelle elle n'a aucune influence et ne s'est officiellement pas engagée. Bref, maintiens tout simplement tes anciens mécanismes de suivi.

Pour ce qui est de la rétrospective, ne laisse personne d'autre que toi la conduire. Assure-toi que toute proposition qui ne te convient pas soit actée durant ce temps. En effet, il arrive certaines fois que des équipes prennent des décisions en dehors de cette cérémonie, t'enlevant ainsi tout mécanisme de contrôle. Avec des sprints courts, toute nouvelle idée qui ne te convient pas aura du mal à faire ses preuves. Fais du bruit, disperse, multiplie les actions, invente des indicateurs inutiles ou mieux, faux, accentue les échecs, incrimine directement les personnes, en choisissant en priorité les plus faibles du groupe.

Je vais maintenant te parler du rôle le plus important dans scrum, à savoir le product owner. Dans l'idéal il faudrait que ce soit toi. Si cela n'est pas possible, assure-toi que celui-ci n'a aucune personnalité, ne porte aucune vision, ou mieux introduit le concept de proxy-product owner, une sorte d'usurpateur du PO, qui porte le pouvoir sans en avoir le mandat. Use de ruse, flatte-le, mais éloigne-le le plus possible de l'équipe. Un grand nombre de personnes n'associent encore à l'image du développeur qu'une sorte de grunge, frustré, reclus dans son garage à faire des blagues compréhensibles par sa seule communauté. Je sais comme toi que la grande majorité d'entre eux ont une vie sentimentale accomplie, sont socialement équilibrés, possèdent une culture générale solide qui ne se réduit pas à leur domaine, certains ont monté leur boîte ! Peu importe, je te conseille fortement de continuer à entretenir ce mythe de l'introverti associable.

Pour ce qui est du scrum master, joue-la comme Poutine ! Ils sont imbus d'un humanisme lâche. Convaincus que la nature humaine est bonne, ils pensent pouvoir régler tout conflit sous le format de la communication et du consensus. Ils ne croient ni à la punition, ni aux mesures de rétorsion. Profites-en ! L'idéal est de prendre un consultant externe, tenu par son « devoir de conseil », il subira sans rechigner.

Pour ce qui est du chiffrage, chiffre en story point mais assure-toi de conserver une conversion en jours hommes. En tout point, appuie-toi sur les pratiques agiles pour en dénaturer le sens. Autant que faire se peut, ne divise pas les story en plus petit item fonctionnel, et crée un maximum de stories n'apportant aucune valeur au business. Valide les étapes techniques en stories, et rends insécables les véritables stories métiers. Après tout, que peut dire un développeur sur une business value ?

Divise l'équipe en rôle, et cloisonne-les dans cela. Tu peux même être créatif en mettant une moitié de l'équipe dédiée au support et bug, et l'autre aux nouveaux développements. Divise, punis, humilie. A chaque difficulté, laisse l'équipe mettre du processus. Utilise pour cela la technique du « fait divers », c'est-à-dire base-toi sur un évènement de développement qui n'arrivera plus de toute évidence pour mettre en place un mécanisme de contrôle. Scrum nous donne la rétro pour cela.

Pour ce qui est du « done », laisse l'équipe délirer avec une abondance de critères. Plus ils se mettront des règles moins ils arriveront à les tenir. Plus ils se donneront des contraintes plus ils auront une impression de professionnalisme. Il y a ce sentiment de déclassement chez l'ingénieur informatique qui le pousse à vouloir reproduire au sein du génie logiciel ce qui est propre au génie civil.

N'appuie pas trop fort, on ne tue pas un prisonnier que l'on veut faire avouer, et un propriétaire n'est heureux que si sa terre est féconde. Mon ami, le temps me manque ici pour te parler de la mise en place du sur-processus, des jeux idiots, de la gestion des réunions infécondes, des indicateurs bancals.

Je t'embrasse avec toute mon affection, dans l'attente impatiente de connaître les fruits de mes conseils.

Ton Ami

PROGRAMMEZ! a besoin de vous !

Campagne de financement participatif sur ulule.

Programmez! lance son projet Programmez! Collectors.
Il s'agit de numéros spéciaux à tirages limités.

Le 1er numéro sera

100 % ordinateurs vintage, retrogaming & retroconsole

+ programmation à l'ancienne
+ focus sur les plus belles machines des années 1980 et 1990.

Un numéro unique.

Nous avons besoin de 7 000 € pour lancer le projet

La campagne s'arrête le 13 avril !

Soutenez Programmez! Collectors 100 % vintage dès aujourd'hui :
https://fr.ulule.com/programmez-collectors/

ulule

D.R.

PROGRAMMEZ!
le magazine des développeurs

Programmez! Collectors ne sera pas disponible en kiosques, ni sur abonnement.
Ce numéro comportera 100 pages, format magazine. Tirage maximum : 2 000 exemplaires. Publication prévue : juin 2017. Prix de vente facial : 9,99 €

Utilisation de la Google Cardboard sous Embarcadero Delphi

• Paul TOTH
contact@execute.fr
Expert Delphi
MVP Embarcadero

La réalité virtuelle est de plus en plus exploitée, tous les jours un nouveau projet apparaît autour de cette technologie. Les premiers téléphones compatibles Tango sont en vente et les constructeurs annoncent des lunettes toujours plus performantes. Dans cet article, je vous propose d'aborder le principe des Cardboards Google nativement sous Embarcadero Delphi.

Les Google Cardboards ont sans aucun doute permis de vulgariser l'accès à la réalité virtuelle. Loin des investissements parfois lourds des équipements existants sur le marché, Google propose avec une boîte en carton et une paire de lentilles de transformer votre Smarphone en casque de réalité virtuelle pour une poignée d'euros. Les développeurs Android ont à leur disposition un SDK pour exploiter cette technologie ; nous allons voir qu'il est cependant tout aussi facile de l'exploiter sous Embarcadero Delphi.

Vous pouvez retrouver les explications détaillées sur la mise en place d'un système de Cardboard dans la vidéo disponible sur le Blog de Barnsten, représentant officiel de Embarcadero pour la France (https://www.barnsten.com). Les codes source Delphi présentés dans la vidéo sont disponibles sur GitHub (https://github.com/tothpaul/Delphi).

La vision stéréoscopique

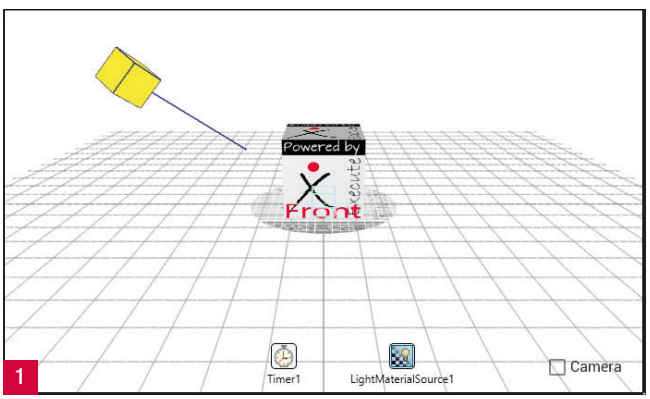
La première étape pour plonger dans la 3D est la partie vidéo. L'écran de votre smartphone doit afficher une image dédoublée de l'univers 3D avec pour chaque moitié de l'écran, le point de vue d'un œil. Les yeux sont en effets distants en moyenne de 63,73mm, c'est ce qui nous permet d'apprécier les distances et de voir en trois dimensions.

Embarcadero Delphi propose deux frameworks graphiques distincts ; *la Visual Component Library* (VCL) sous Windows, celle-ci est fortement liée à l'API Windows, et Firemonkey (FMX) qui est disponible sur l'ensemble des plateformes supportées, à savoir : Windows 32 et 64 Bits, OSX, iOS 32 et 64bits et Android.

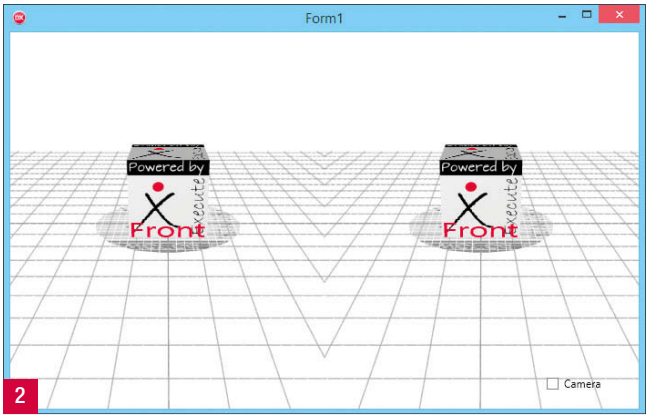
FMX propose nativement le support 2D et 3D de DirectX sous Windows et OpenGL sur les autres plateformes ; le développeur Delphi n'a donc pas à se soucier de l'API utilisée car Firemonkey propose un ensemble d'objets haut niveau qui masquent complètement l'aspect technique sans pour autant en interdire l'accès. Le rendu 3D est donc proposé nativement par l'environnement de développement, il reste à lui apporter la stéréo. [1]

C'est un travail moins compliqué qu'il n'y paraît puisque Firemonkey propose un objet Camera qui permet de fixer le point d'observation et que l'on peut facilement reproduire les objets d'une scène sur une autre portion de l'écran. [2]

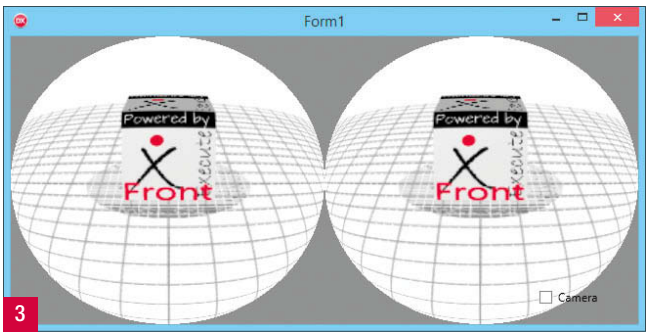
Ce rendu dédoublé n'est cependant pas suffisant pour obtenir l'effet désiré, mais là encore Firemonkey nous permet facilement d'effectuer ce rendu dans une texture que l'on pourra alors appliquer sur une forme hémisphérique nécessaire à l'observation de la scène au travers des lentilles de la Cardboard. [3]



Composition d'un univers 3D directement dans l'IDE Delphi



Dédoublement de la vue principale



Rendu de la scène sur une texture sphérique

La touche finale sera d'utiliser une texture plus grande que l'écran afin de limiter les zones grises de l'image sans sacrifier à la qualité de l'image. La mise en œuvre de tout cela est très simple avec le code proposé sur GitHub. Il suffit de référencer l'unité Execute.StereoViewPort dont le source est dans le répertoire Libs. Celle-ci surcharge la déclaration du composant TViewPort3D de Delphi pour lui ajouter les propriétés suivantes :

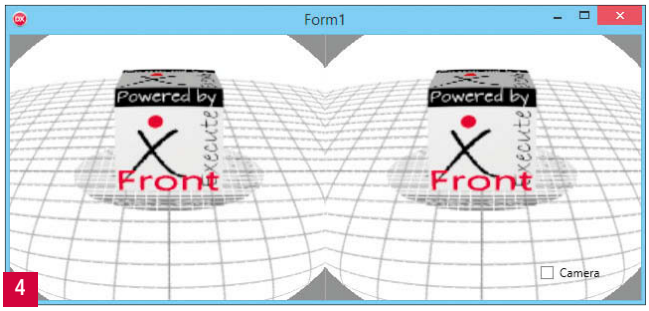
Stereo (boolean)	active le mode stéréo
EyeDistance (Single)	détermine la distance interoculaire elle est calculée automatiquement par défaut
Distorsion (boolean)	active la projection sphérique
Expand (single)	facteur d'agrandissement de la texture

Le HeadTracking

L'image stéréoscopique permet d'exploiter pleinement l'effet 3D des Cardboards, le décalage de point de vue entre les deux moitiés de l'écran suffit à notre cerveau pour apprécier les distances. Mais pour plonger pleinement dans la 3D il faut lui ajouter l'interactivité, l'image affichée doit en effet répondre aux mouvements de tête de l'utilisateur.

Cette technologie est indépendante de la précédente, on notera son usage en réalité augmentée où le smartphone vient ajouter des éléments virtuels sur l'image 2D produite par la caméra, mais c'est la combinaison des deux techniques qui donne l'effet saisissant de la réalité virtuelle.

Connaître l'orientation du smartphone n'est pas très compliqué en soit, cela consiste simplement à lire les valeurs de ses capteurs. Cependant pour avoir un HeadTracking efficace il faudra obligatoirement que le smartphone dispose d'un gyroscope. C'est surprenant de prime abord puisque le gyroscope n'est pas un capteur d'orientation mais un capteur de mouvement ; quand le smartphone ne bouge pas, le gyroscope n'a rien à vous dire. Son usage est cependant indispensable dans l'état actuel des technologies car c'est un capteur très précis, alors que le compas - qui permet lui de connaître l'orientation du smartphone – offre un signal fortement bruité. C'est en combinant les deux signaux que l'on obtiendra une orientation précise et stable. Mais pas de panique, Android propose déjà un capteur d'orientation qui s'occupe de tout, le HeadTracking peut au final se contenter de lire le vecteur d'orientation retourné par le système d'exploitation.



4

Optimisation de l'écran avec une texture étendue

Delphi propose un composant TSensor qui permet de lire les valeurs du capteur, mais celles-ci sont transformées en vecteur de rotation pour les besoins de Firemonkey ; qu'à cela ne tienne, on pourra très facilement accéder directement à l'API Android depuis Delphi pour prendre la valeur du capteur à la source.

La mise en œuvre est très simple puisque Delphi fournit l'unité Androidapi.Sensor qui déclare tout ce qui est nécessaire pour lire les capteurs natifs sous Android. Le code est alors une simple traduction en Pascal des exemples du NDK Android.

```
FSensorManager := ASensorManager_getInstance;
FNativeSensor := ASensorManager_getDefaultSensor(FSensorManager, ASENSOR
_TYPE_ROTATION_VECTOR);
if FNativeSensor <> nil then
begin
 FNativeEventQueue := ASensorManager_createEventQueue(FSensorManager,
 ALooper_forThread, LOOPER_ID_USER,
    nil, nil);
 ASensorEventQueue_setEventRate(FNativeEventQueue,FNativeSensor, 1000);
 AsensorEventQueue_enableSensor(FNativeEventQueue,FnativeSensor);
end;
```

QUI VEND LE PLUS DE CASQUES VR ?

Les casques VR se limitent principalement à Oculus, HTC Vive et Sony PlayStation VR. Un article du New York Times, paru en février dernier, a révélé des chiffres de ventes (non officiels) : 915 000 casques Sony vendus en 6 mois. Sony mise sur 2 éléments : un tarif abordable (399) et une utilisation avec la PlayStation 4. Ce couplage permet à Sony d'offrir un contenant et un contenu. Si le catalogue VR reste faible, la plateforme devrait voir de très nombreux titres disponibles dans les prochains mois. La concurrence est très loin derrière :

- Oculus Rift : + 240 000 exemplaires vendus
- HTV Vive : + 420 000 exemplaires vendus

Les deux modèles souffrent d'un prix élevé (699 & 899) et de contraintes de configuration. Là encore, le faible catalogue pèse sur l'usage. Et surtout, les deux constructeurs ne peuvent pas s'appuyer sur une plateforme matérielle comme Sony et l'utilisateur doit posséder un PC compatible VR.

L'arrivée dans les prochains mois des premiers matériels basés sur Windows Holographic, à des tarifs proches de ceux de Sony, vont redéfinir le marché. Même si ces casques seront moins avancés que le casque Hololens, Microsoft cherche à imposer son standard Windows Holographic.

CONCLUSION

Apporter la vision stéréoscopique à une application Delphi se fait en quelques dizaines de lignes de code. L'usage du framework multiplateformes Firemonkey rend la solution immédiatement compatible avec l'ensemble des OS supportés, il faudra simplement disposer d'un écran adapté aux dimensions des Cardboards.

Pour le HeadTracking il faudra disposer d'un périphérique disposant des capteurs nécessaires à son exploitation, ce qui nous réduit le choix aux mobiles Apple ainsi que les mobiles Android disposant d'un gyroscope.

Les projets exemple disponibles sur GitHub sont immédiatement utilisables sous Delphi Berlin, la version Starter gratuite permettra de tester le code sous Windows, mais pour tester sur mobile, il faudra une version Pro + Mobile Pack, Enterprise ou Academic (prochainement gratuite pour les établissements d'enseignement français) ou encore une version d'essai 30 jours. •

Spécial « tests logiciels »

Ah les tests ! On les aime ou on ne les aime pas. Ils ont souvent une mauvaise réputation auprès des développeurs. Le moindre bug ou problème et l'utilisateur va dire que le développeur n'a pas fait son travail !

Les tests sont importants et permettent de réduire les bugs, d'améliorer la qualité logicielle et de satisfaire les utilisateurs. Les tests recouvrent des réalités très différentes, car il n'existe pas un test, mais des tests : tests unitaires, tests fonctionnels, tests de performance, tests de charge, etc. Les méthodes agiles et le DevOps mettent en avant les tests, car ils contribuent aux cycles agiles et sont des étapes de ceux-ci.

Comment réaliser une intégration continue en nightly build si aucun test n'a été réalisé ?

Les tests furent régulièrement la variable d'ajustement des projets logiciels : pas assez de temps, trop chers, pas de testeurs attitrés, pas d'outils dédiés, etc. Les excuses étaient nombreuses. Mais aujourd'hui, avec les apps Web et mobiles, la pression est très forte pour sortir toujours plus vite une nouvelle version. Or, si l'app déployée est d'une qualité médiocre, les réactions ne se feront pas attendre.

L'automatisation de certains tests, la possibilité de rapidement déployer des environnements de tests complets en mode Cloud aident à avoir une méthodologie des tests et de mener des campagnes plus facilement. Il existe de très nombreuses solutions de testing (open source, freemium, payante), avec des niveaux fonctionnels variés, comme vous le verrez dans ce dossier spécial.

La Journée Française des Tests Logiciels, qui se tient chaque année, est un lieu unique pour échanger et voir les solutions. Cette année, l'événement se tiendra le 11 avril.

La rédaction

Le test logiciel : en pleine mutation au cœur de la transformation digitale.

Olivier Denoo
Président du CFTL
Governance Officer de l'ISTQB

Avec pour thème « Le test logiciel au cœur de la maîtrise des risques de la transformation digitale des entreprises» et une journée de tutoriels reprenant des sujets aussi variés que User eXperience, test agile, test des applications mobiles, processus métiers, automatisation des tests, tests exploratoires ou encore environnements de test, la Journée Française des Tests Logiciels se prépare à accueillir plus de 900 visiteurs les 10 et 11 avril prochains.

A cette occasion, il convient, plus que jamais, de se pencher sur le chemin parcouru, mais aussi et surtout sur l'avenir de ce secteur en pleine mutation.

Depuis la première édition de la JFTL, le Comité Français des Tests Logiciels n'a cessé d'œuvrer à mettre en relation les professionnels du test, à leur présenter le panel le plus large des outils et technologies de pointe en matière de qualité logicielle, à multiplier les initiatives et les contacts et à décrire les tendances fortes du secteur au travers d'analyses et d'observatoires ciblés.

Saluons au passage l'immense travail réalisé par ces volontaires passionnés qui font évoluer les standards (ISTQB, IREB, IQBBA, IBUQ…), la presse spécialisée qui très tôt s'est fait l'écho de nos valeurs, mais aussi ces nombreux organismes de formation accrédités qui ont permis à plus de 10 000 professionnels en France d'obtenir une certification qualifiante reconnue dans le monde entier.

Si vous nous faites l'honneur d'une visite à la JFTL, vous y rencontrerez, sans nul doute, tous les acteurs qui comptent dans ce secteur qui n'a plus de niche que le nom.

Autrefois négligé et confidentiel, le test logiciel est entré dans les mœurs et dans les pratiques des entreprises et plus personne aujourd'hui n'oserait contester son statut de métier à part entière.

Loin d'être figé ou dogmatique, le testeur se réinvente sans cesse au gré des besoins sociétaux et des tendances industrielles.

A la lueur de l'Agilité ou de l'intégration continue, le voici qu'il se rapproche des développeurs et du Métier, effectuant ainsi un grand écart en forme de trait d'union entre l'analyse et le code. Quand survient le DevOps, il se fait automaticien, spécialiste des performances et se plonge dans les arcanes du déploiement. Quand sonne l'heure de l'IA et de l'informatique de grande consommation, il devient « mobile », se fait ergonome, jonglant avec les flux et les données. Que viennent la robotique ou l'IOT et le voici qui se mue en spécialiste de la sécurité, des langages et protocoles propres à « l'embarqué ».

Si le testeur d'aujourd'hui doit être polymorphe, prêt à s'adapter à de nouvelles contraintes, à relever de nouveaux défis, à intégrer de nouvelles compétences ; le testeur de demain lui ressemblera en tous points, tant notre vision de l'informatique en constante évolution requiert une remise en question de chaque instant face à des métiers qui se réinventent sans cesse.

Ne nous y trompons pas, aujourd'hui, pour être performants, analystes, développeurs et testeurs se doivent de travailler de concert, en tant que codétenteurs du produit ou du service, des exigences, des besoins et de la qualité. Ils deviennent partenaires et les frontières entre leurs fonctions, autrefois rigides pour ne pas dire infranchissables, deviennent de plus en plus floues, permettant à chacun de donner le meilleur de soi-même.

Il est donc bien loin derrière nous – et c'est tant mieux – le temps où développeurs et testeurs étaient présentés comme des opposants voire des concurrents. Leurs fonctions se complètent et se rejoignent, le temps d'un sprint ou d'un projet, donnent la plénitude de leur valeur ajoutée, tout en gardant leur spécificité et leur nécessaire indépendance, ne fut-ce que le temps de jouer leur rôle ou d'endosser leur habit de circonstance.

De simple charge supplémentaire, qu'on ajoute à la liste déjà bien longue des frais généraux, à créateur de valeur, il y avait un pas immense que les métiers du test, aujourd'hui placés sous les meilleurs auspices, n'ont allègrement franchi qu'à force de résultats et de persuasion.

Pas étonnant dès lors que les analyses de Pierre Audouin Conseil, du World Quality Report ou de Technavio, pour ne citer qu'elles, leur prédisent une croissance soutenue – proche des deux chiffres pour les plus optimistes. Pas étonnant non plus que le marché ne s'y soit pas trompé, avec une explosion des demandes - tant de la part des industriels que des SSII - et une multiplication des offres des services et autres outils dédiés. La « machine-test » est en route et n'est pas prête de s'arrêter en si bon chemin, tant s'en faut !

D'ailleurs, les DSI qui gagnent ont déjà compris depuis longtemps que la qualité de leurs produits et services constitue - bien plus qu'un atout - une condition sine qua non de leur croissance. Sans surprise, le métier de testeur y est reconnu à sa juste valeur.

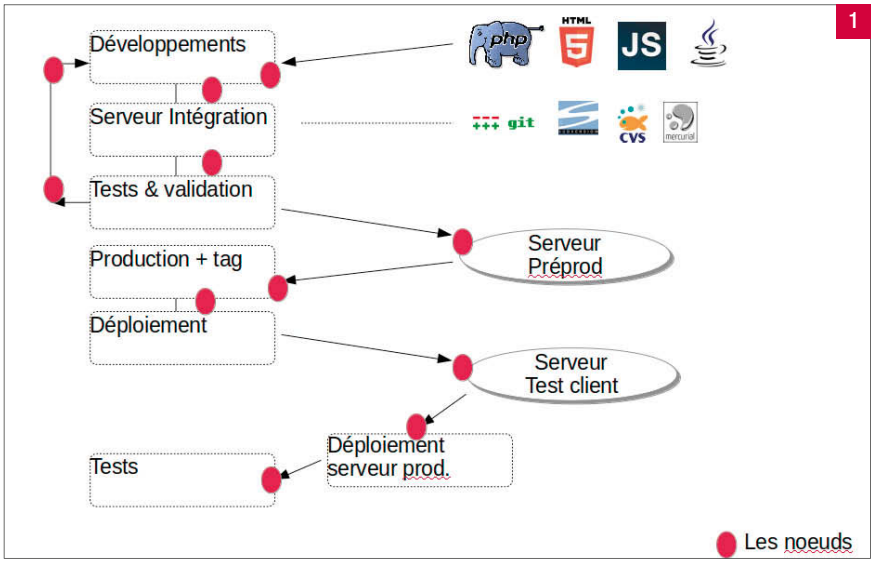
Différenciateur indispensable dans une société de consommation mondialisée où l'offre se banalise, le lien très fort entre les différents acteurs de leur écosystème centré sur la qualité constitue ainsi, sans nul doute, la pierre angulaire de leur réussite. Dans de telles structures, l'efficacité se mesure avant tout à l'aune de la collaboration, de la participation, de la passion et de l'engagement. Le reste suit…naturellement.

Un exemple à suivre ou un but à atteindre…

Les tests modernes

Un test est une procédure de vérification partielle d'un ou plusieurs éléments dans la réalisation d'une application ou d'un logiciel, jusqu'à sa mise en production. Son objectif est d'identifier un maximum d'anomalies éventuelles et d'améliorer la qualité. Un processus de tests est indispensable dans un développement moderne.

Christophe Villeneuve
Consultant IT pour Ausy, Mozilla Rep, auteur du livre "Drupal avancé" aux éditions Eyrolles et auteur aux Éditions ENI, PHPère des elePHPants PHP, membre des Teams DrupalFR, AFUP, LeMug.fr (MySQL/MariaDB User Group FR), Drupagora...



La notion de tests a toujours existé, mais depuis quelques années, de nouvelles méthodes sont apparues, provoquant une réorganisation des méthodes de développements. Cela s'est traduit par des cycles de déploiement beaucoup plus courts avec des évolutions (features) plus régulières. Cela a été facilité par l'utilisation de méthodologies comme le TDD (Test Driven Development), ou Extreme Programming (XP) dont la notion de tests est devenue un critère important dans le développement moderne. Les tests modernes parlent à toutes les étapes d'un projet, c'est pourquoi il est important de revenir sur les différentes familles que vous rencontrerez.

Les préjugés

Si vous parlez de tests autour de vous, vous allez être confronté à de nombreux préjugés et plus précisément à une méconnaissance des tests. Les premiers freins soulevés vont être le prix et la durée nécessaire. Mais du côté des développeurs, ce sera plus dans le genre : « moi je ne fais pas de bugs !!! ». Au niveau de la compréhension du terme et à quel moment ils sont déclenchés, on peut entendre d'un utilisateur, qu'il fait des tests unitaires tous les jours, juste en cliquant sur les liens pour ne pas obtenir des erreurs 404 à l'écran. [1]
Comme le montre l'illustration, chaque étape possède un noeud, qui doit impérativement passer par des tests.

Les tests unitaires

Les tests unitaires sont des tests réalisés par les architectes ou les développeurs dont la première préoccupation, c'est :"quoi tester ?". Si votre projet utilise un framework générique ou spécifique, une librairie..., il ne sera pas utile de refaire les tests. L'intérêt de les mettre en place, c'est de tester un module, une méthode, une classe que vous avez créée. Bien entendu, il faut tester ce qui est important à tester : ce qui risque de casser ou les parties que vous touchez souvent. Ce type de test est difficilement automatisable, car même s'il existe des frameworks unitaires pour vous faciliter l'opération, il faut quand même les écrire, car ils sont dépendants du code réalisé. Le but est d'écrire un test qui confronte une réalisation à sa spécification. Il détermine si celui-ci renvoie un succès ou un échec. Ainsi, le test unitaire permet de vérifier que la réalisation d'entrée/sortie est bel et bien réalisée. Ainsi, il permet d'identifier rapidement les erreurs éventuelles. Au niveau des outils, il existe un large choix de framework de tests unitaires disponibles pour chaque langage de développement par exemple JSUnit pour JavaScript, PHPUnit pour PHP, CUnit pour C, etc.
Du côté du fonctionnement, la procédure à mettre en place sera :

- Initialisation : une fonction setup() définit l'environnement de test qui sera reproductible ;
- Le test : le module à tester et à exécuter ;
- Vérification : comparaison entre les résultats obtenus et le résultat défini. Vous utiliserez les fonctions 'assert' ;
- Désactivation : permet de revenir à l'état initial du système pour éviter de ne pas perturber les tests suivants.

Pour écrire des tests de codes, cela se traduit comme ceci :

- Écrire une fonction de test pour appeler un code qui n'existe pas encore et donc échouer ;
- Écrire une fonction de test pour appeler un code qui existe et donc le test sera réussi ;
- Si le test est un succès, vous pouvez ajouter un autre test pour obtenir un résultat différent avec des points d'entrées différents ;
- Si le résultat retourne des erreurs, vous modifiez le code principal pour que les tests puissent passer ;
- Recommencer en lançant de nouveau les tests tant qu'il existe des erreurs.

Exemple

* Asserts (add(0,0)) == 0
Ce test compare si la valeur retournée est égale à Zéro

* Asserts (add(1,0)) == 1
Ce test compare si la valeur retournée est égale à Un

* Asserts (add(0,1)) == 1
Ce test compare si la valeur retournée est égale à Un

* Asserts (add(1+1)) == 2
Ce test compare si l'ajout des 2 nombres retourne une valeur 2

* Asserts (add(1+1)) == 3
Ce test compare si l'ajout des 2 nombres retourne une valeur 3

Enfin, un test unitaire doit être indépendant et reproductible unitairement et à tout moment. C'est pourquoi il est important de tester une caractéristique et une seule dans un test.

Envolez-vous
vers une meilleure qualité logicielle !

LES EXPERTS DU TEST LOGICIEL

- Maximisez votre qualité logicielle
- Réduisez vos coûts
- Minimisez vos risques
- Indépendance et réversibilité
- Des profils experts pour accompagner votre transformation digitale
- Mise en place de cellule de test sur mesure

Problèmes de qualité logicielle ? Transformation digitale difficile ? Absence de métriques ?

1 JOURNÉE D'AUDIT & CONSEIL GRATUITE

ps_testware
Votre Partenaire Qualité

ps_testware S.A.S (FRANCE)
Parc des Rouges Barres - 6, rue Marcel Dassault - Bât 3 - 59700 Marcq-en-Barœul
Tél : +33 (0) 3 20 04 97 39 - Email : infofr@pstestware.com - www.pstestware.fr

Les tests d'intégrations

Les tests d'intégrations doivent être effectués après les tests unitaires, car ils ne se limitent pas juste à tester manuellement les modules de votre application, mais d'effectuer des tests en les associant ensemble. Ils ont leurs places dans un développement en TDD (Test-Driven Development). Le but d'effectuer ce type de tests s'adapte suivant votre besoin, c'est pourquoi l'organisation de celui-ci est de déterminer une stratégie des actions couvertes et plus précisément les tests à réaliser.

Par exemple

L'exécution automatique des tests unitaires est déclenchée après chaque build, versionnée par GIT ou SVN, avec le code qu'ont réalisé les développeurs. L'opération permet de vérifier si les nouvelles lignes de codes ne provoqueront pas des régressions.

Au niveau des outils, qui se chargeront de lancer ces tâches pour analyser le code et par conséquent traiter le résultat obtenu, vous trouverez Jenkins, Redmine, Handson, qui sont les plus connus. Ces outils étudieront les résultats de vos tests et si le résultat est positif, vous passez à l'étape d'après. [2]

La mise en place de ce type de tests permet de garantir une certaine maintenabilité dans la vie de votre application. Ainsi, vous testerez une évolution avec éventuellement ces dépendances. Il existe plusieurs méthodes d'approches dont en voici quelques-unes :

- Top-down : le test s'effectue du haut vers le bas, c'est à dire à partir de la position 1 ;
- Bottom-up : le test s'effectue du bas vers le haut, c'est à dire à partir de la position 5 et 6 ;
- Sandwich : il s'agit des 2 méthodes précédentes, avec le but de se rejoindre au centre ;
- Big Bang : vous intégrez tous les modules d'un coup juste après les tests unitaires.

Au final, les tests d'intégrations ont évolué vers l'intégration continue, car les développeurs détiennent toute l'application sur son poste et ils peuvent faire l'intégration tout au long du développement. C'est pourquoi une partie des validations est réalisée directement dans un IDE.

Les tests de validation

Le test de validation est associé aux tests fonctionnels et correspond à des tests IHM (Interactions Homme-Machine), c'est-à-dire qu'ils jouent le rôle de validateur en vérifiant un cas d'utilisation donné. Il permet de répondre à la question "est ce que l'opération permet de faire ça ?" par rapport aux exigences demandées. Il s'agit de la méthode la plus efficace pour évaluer votre solution et l'opération sera principalement manuelle.

Par exemple un test de validation peut se représenter de la façon suivante :

Si vous lancez une calculatrice et vous tapez le chiffre '16', suivi de 'racine', puis '='.

L'application doit nous retourner la valeur '4'.

Si ce n'est pas le cas, le test de validation est cassé.

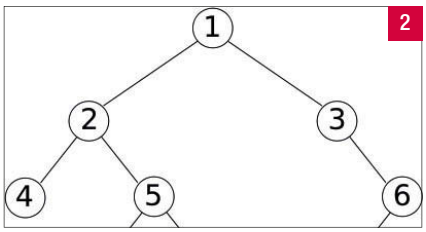
Comme le montre l'exemple, le test présenté est un cas d'utilisation et vous ne vous occupez pas de comment le code a été construit, exécuté ou appelé. Les tests de validation sont là pour éviter les régressions, identifier les bugs, les erreurs ou les défaillances dans le but de les réparer.

Après avoir enregistré les différentes valeurs nécessaires au bon fonctionnement, ils pourront être exécutés à volonté ultérieurement. Ainsi, vous pouvez énumérer tous les cas possibles. Toutefois, il est difficile de simuler toutes les combinaisons, car la réalisation d'un test exhaustif peut prendre beaucoup de temps. Vous devez quand même garder une bonne qualité de couverture tout en évitant d'avoir des combinaisons identiques. Lors de la mise en place de ces tests, il est impératif de se poser les bonnes questions, qui sont :

- L'écran fonctionne-t-il correctement ?
- Conformité aux spécifications ;
- Est-ce que cela correspond aux attentes du client ;
- Compatibilité avec les autres systèmes ou navigateurs ;
- Compréhensible pour être utilisé par un maximum d'utilisateurs ;
- Identifier les points d'amélioration ;
- Peut-on le déployer ?

```
# language: fr
Fonctionnalité: Consulter ses facture passées
  Afin de permettre aux utilisateurs de voir l'évolution de leur communication
  En tant qu'utilisateur connecté
  Je veux être capable de consulter l'ensemble des montants de mes factures

  Contexte:
    Soit un utilisateur connecté en tant que "vco" avec comme mot de passe "monkey"
    Et que ces factures existent au nom de "vco" :
      | Année | Mois     | Montant |
      | 2010  | Janvier  | 40      |
      | 2009  | Décembre | 35      |
      | 2009  | Novembre | 35      |

  Scénario: Consulter une facture en particulier
    Soit je suis sur la page de consultation de mes factures
    Lorsque je clique sur le lien "Facture de Novembre"
    Alors je devrais voir "Facture du mois de Novembre 2009"
    Et je devrais voir "Montant de votre facture: 35 euros"
```

3



2

En outillage, vous trouverez Selenium SE, Watir... qui permettent de rejouer les fonctionnalités. [3]

De plus, vous pouvez utiliser une approche récente, appelée Behavior Driven Development (BDD), qui consiste à écrire une fonctionnalité selon un formulaire "Given / When / Then".

Même si les scénarios sont en français, vous trouverez des interpréteurs associés à vos langages de développement.

Les tests automatisés

À la différence des tests fonctionnels ou unitaires, les tests automatisés ont pour but d'exécuter une ou plusieurs tâches répétitives, à différentes reprises, pendant le cycle de vie de votre projet. Toutefois, vous pouvez penser que les testeurs suffisent pour valider le bon fonctionnement de celui-ci, mais cette assertion ne résiste pas à une analyse un peu approfondie, dans un temps raisonnable.

Par exemple

Si votre projet est un site marchand, et qu'il possède des milliers de produits avec de nombreuses catégories, il est intéressant que la navigation entre les pages et les catégories puissent se faire facilement et puissent tester les milliers de cas de test.

Si vous vous référez à l'exemple, vous vous rendrez compte entre le moment où le code est écrit et le moment où il passe entre les mains des testeurs, plusieurs semaines peuvent s'écouler. La conséquence provoquera des aller-retours entre les développeurs et les testeurs avec des risques de régression. [4]

Au niveau de l'outillage, de nombreux logiciels sont disponibles pour effectuer ce type de tests et les plus connus sont Selenium, Quality Center, Silk test, iMacro... Chacun d'eux possède son propre langage avec l'identification par des clefs "ID" pour interpréter la gestion des conditions, la validation des critères de champs, les redirections...

Bien sûr, ils ont un rôle important lors de mises en place de micro-releases, c'est-à-dire la livrai-

Validez la performance en continu

Neotys

Test et monitoring de performance pour les équipes Agiles et DevOps

NeoLoad

Test de charge et de performance automatisé

- Testez 10x plus vite
- Réalisme inégalé
- Automatisé et intégré dans votre chaine d'outils

NeoSense

Monitoring d'application

- Monitoring 24h/24
- Monitoring niveau application et infrastructure
- Mise en œuvre simple

Pour en savoir plus et télécharger la version gratuite entièrement fonctionnelle de NeoLoad, rendez-vous sur www.neotys.fr

son d'une ou deux évolutions. Ainsi, les tests automatisés peuvent être joués rapidement ; pour un projet web, ils peuvent être faits sur un autre serveur en parallèle des autres réalisations, et, grâce à eux, les tests peuvent être joués sur les différents navigateurs du marché.

Les tests de charges

Les tests de charges sont souvent associés aux tests de performance. Ils mesurent le comportement d'une application en fonction du nombre d'utilisateurs connectés simultanément.

Leur but consiste à proposer un résultat qui permettra de définir un degré respectable pour rendre l'application utilisable et mesurera le temps nécessaire à afficher la page. Ainsi, il est possible de définir les limites, la résidence ou encore la fiabilité de celui-ci.

Exemple

La montée de charge est un facteur important, car le temps d'affichage sera un des premiers critères d'acceptation si votre projet est un jeu, or ce critère sera positionné à un niveau différent si c'est une application Web ou un logiciel.

Pour mettre en place ces tests, il est nécessaire de définir le périmètre de ce qu'il faut réaliser. C'est pourquoi les personnes s'occupant de cette partie doivent s'appuyer sur une documentation précise au niveau fonctionnel, et au niveau technique.

La méthodologie des tests de performance doit être mise en place le plus tôt possible dans le cycle de développement, c'est à dire :

- Posséder un environnement identique à l'environnement de production ;
- Tester de façon large, puis de façon approfondie les différents écrans ;
- Tester dans différents environnements.

Du côté de l'outillage, les logiciels les plus connus sont jMeter, Gatling... qui auront la capacité de simuler un nombre d'utilisateurs important. Ce critère vous aidera à répondre à 2 problèmes :

- La capacité à enchaîner une suite d'actions ;
- De déterminer ces actions sur les données.

Ces solutions simplifient et automatisent les tests. Ils s'appuient sur la création plus ou moins automatisée des scénarios de test, la configuration des scénarios avec ou sans script, et à simuler les utilisateurs virtuels pour collecter des mesures.

Enfin, dans le développement d'aujourd'hui, la génération automatique est quasiment faite pour produire les différentes transactions qui sont servir à valoriser les scripts de test.

Les tests d'utilisateurs

Les tests d'utilisateurs permettent d'évaluer un produit, en les faisant tester par des utilisateurs. Mais quand ces tests se portent dans le domaine de l'informatique, comme un logiciel ou un site Internet, nous parlerons beaucoup plus de tests d'ergonomies.

Les tests d'ergonomies ou utilisateurs, ne sont pas destinés à tout tester, vous devez lister les points bloquants, les évolutions. Ainsi, ils permettent de détecter les problèmes d'utilisation du projet et à identifier les difficultés que l'on peut rencontrer.

Au niveau de l'outillage, l'utilisateur sera le seul et unique moyen de réaliser ce test.

Le déroulement de ce test, consiste à sélectionner un ou plusieurs utilisateurs, pour obtenir un groupe de 6 à 10 personnes. Vous les placerez le plus proche possible de la version finale de l'application. Par conséquent, les fonctionnalités ne sont donc pas testées individuellement, mais globalement.

De plus, l'utilisateur devra suivre un ou plusieurs scénarios d'utilisations construits afin de valider les hypothèses identifiées précédemment. L'ensemble de ces scénarios sera écrit dans un guide de tests pour avoir :

- Une vue d'ensemble comprenant les impressions du site, un avis pour savoir avec quoi ils peuvent acheter ;
- Au niveau du scénario, vous pouvez établir la recherche d'information d'un produit ;
- Enfin des questions subjectives comme l'utilisation de certains termes utilisés, ou encore après l'utilisation, l'envie d'y retourner.

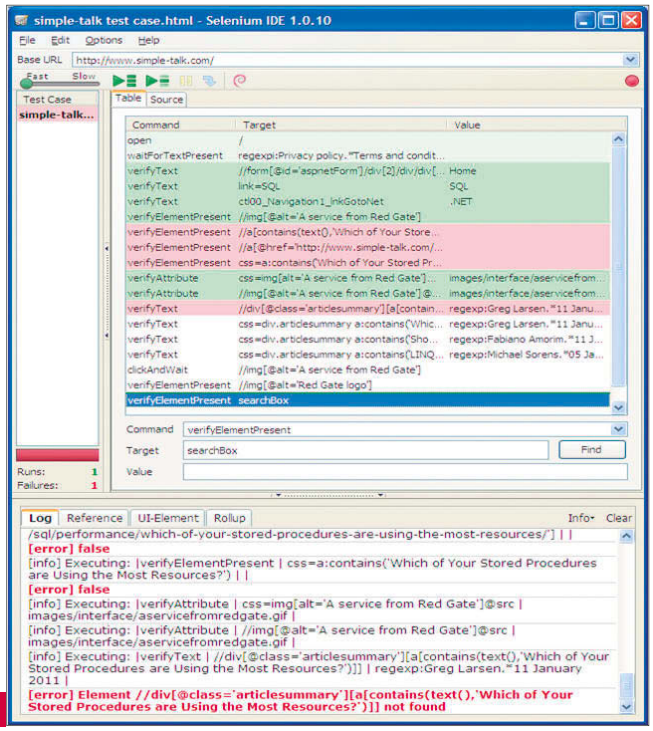
Un exemple de plan de test

Pour tester un nouveau site B2B (marchand), il est intéressant de voir le parcours d'une personne pour effectuer un achat jusqu'au paiement, la confirmation, en passant par l'utilisation du panier et la livraison.

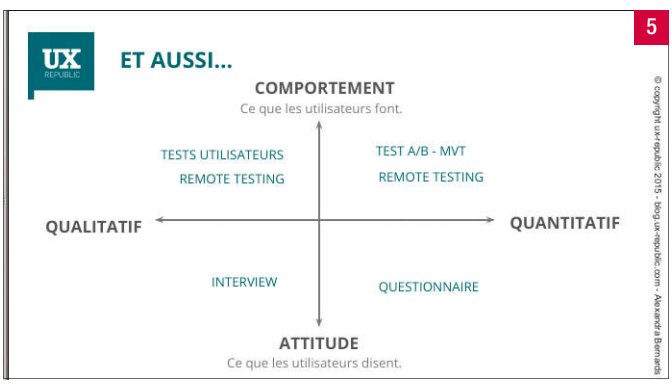
Lors de ces tests, vous devez rester en retrait pour éviter les interactions avec les utilisateurs et ne pas les influencer. De plus, il est indispensable d'écouter et de prendre des notes, d'analyser leurs attitudes... [5]

Lorsque le test est terminé, il faut analyser les résultats pour en lister les problèmes rencontrés, voir si les objectifs sont atteints. Par ailleurs, vous devez dégager les tendances pour en proposer des recommandations adaptées, en accompagnant de solutions visuelles, le tout dans un rapport de test. L'autre étape à réaliser concerne à traduire les valeurs en données pour sortir des indicateurs liés aux questions préparées en amont. Vous devez aussi lister les problèmes rencontrés, noter les bugs non corrigés en les classant par ordre de priorité.

Du côté de l'outillage, il en existe sous licence.



4



5

Cependant, il est très difficile actuellement de simuler le comportement d'une personne, il est préférable d'effectuer les tests manuellement.
Enfin, pour que ce test soit complètement réussi, vous avez besoin obligatoirement de personnes volontaires qui ne connaissent pas le projet afin de vérifier la cohérence et le bon enchaînement des questions. De plus, vous ne devez pas perdre l'idée, qu'un test utilisateur fait émerger des tendances et non des statistiques.

Les tests de sécurité

La sécurité est un élément clef et reste un critère souvent mal perçu par les conséquences indirectes du projet. Les conséquences sont désastreuses lors de la mise en production de votre projet, car les menaces sont présentes dès le début du déploiement, qui peut balayer par une simple attaque votre réalisation, malgré de nombreux mois passés à la développer. Le résultat vous obligera à fermer ou à arrêter temporairement votre projet le temps de réaliser les correctifs nécessaires.
Quelle que soit la méthode de développement, vous placez les tests à chaque instant et à chaque étape dans un projet. Pour cela, les tests de sécurité doivent aussi bien se porter sur le matériel, les périphériques, les smartphones, les logiciels, le Web, l'Internet des objets.
Au niveau outillage, il existe de nombreux logiciels, outils... que nous ne citerons pas, car si vous les utilisez, c'est que le travail préventif n'a pas été réalisé. Tout d'abord, il faut sensibiliser les développeurs à regarder ce que fait l'OWASP (Open Web Application Security Project) qui publie des rapports en détaillant les types de failles de sécurité que vous pouvez rencontrer au niveau :

- Des applications Internet ;
- Des applications mobiles ;
- Des objets connectés ;

Chaque point identifié aura une solution que vous devez utiliser pendant le développement.
Et de l'autre, vous pouvez ajouter des extensions, les librairies d'analyses de code dans les IDE et votre cycle de développement.
Mais le mieux, c'est de mettre en place un plan de sécurité. Pour cela, il faut :

- Identifier les API et les frameworks externes et les suivre au niveau des mises à jour ;
- Anticiper les problèmes réglementaires ;
- Planifier régulièrement des revues de codes par le biais d'analyses statiques.

Enfin, les tests de sécurité doivent se porter sur les tests d'intrusions, de résistance aux mots de passe, de l'utilisation des périphériques (WiFi, Bluetooth...), les captures de requêtes...

CAS PRATIQUE

Pour réaliser des tests unitaires, nous vous proposons de regarder notre calculatrice simplifiée réalisée avec le langage PHP. Il vous faudra au préalable avoir installé le langage PHP et le framework de tests PHPUnit.
Nous construisons une classe avec une fonction addition dans un fichier appelé calculs.class.php :

```
<?php
class calculs
{
 /**
   * @assert (0, 0) == 0
   * @assert (0, 1) == 1
   * @assert (1, 0) == 1
   * @assert (1, 1) == 2
   * @assert (1, 2) == 3
   */
  public function add($a, $b)
  {
    return $a + $b;
  }
}
?>
```

Au niveau de la programmation, nous développons de la méthode dans un fichier de votre choix par exemple : calculatrice.php :

```
<?php

require_once ('calculs.class.php');

$calculatrice = new calculs();
$resultat = $calculatrice->add(3, 4);
echo $resultat;
?>
```

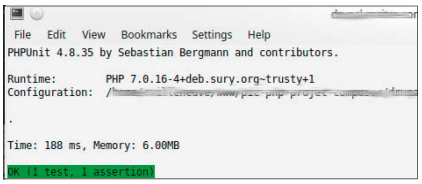
Pour exécuter la page calculatrice, nous pouvons l'exécuter à partir du navigateur ou en ligne de commande

Le résultat obtenu est 7 ce qui correspond à la somme des 2 valeurs (3 et 4), appelée avec la fonction add.
Au niveau des tests unitaires, nous concevons le fichier, appelé calculs.test.php :

```
<?php
require_once "calculs.class.php";
class base_test extends PHPUnit_Framework_TestCase
{
   /**
   * Generated from @assert (3, 4) == 7.
   */
   public function testAdd()
   {
     $this->assertEquals(
      7,
     Calculs::add(3, 4)
     );
    }
 }
?>
```

Le script montre que nous étendons la classe PHPUnit_Framework_TestCase pour utiliser la fonction assertEquals, qui correspond à vérifier l'utilisation du calcul additionnel et le résultat souhaité. Pour exécuter le script, l'opération s'effectue en ligne de commande :

```
$ phpunit calculs.test.php
```



L'image montre que le test s'est bien passé car un point est affiché. Si cela n'était pas le cas, nous aurions un E pour Erreur, F pour Faux...
Grâce à ce cas de test, celui-ci pourra être rejouer ultérieurement.

CONCLUSION

Quelle que soit la taille de votre projet, sa durée, il est indispensable d'effectuer des tests à moindres coûts.
Les tests doivent principalement être réalisés sur vos développements et réalisations, car il n'est pas utile de retester les librairies, framework, CMS venant de la communauté puisqu'ils ont déjà subi l'ensemble des tests. Il vous sera seulement nécessaire de suivre les mises à jour régulièrement.
Les librairies, extensions que proposent vos différents outils vous éviteront de nombreuses surprises. N'oubliez pas d'exploiter également la possibilité d'ajouter des actions qui se déclencheront régulièrement pour valider chaque étape de votre projet.
Au final, les tests garantissent une qualité de votre projet, qui n'aura pas de désagréable surprise lors de la livraison finale. •

Automatiser les tests avec Kalifast

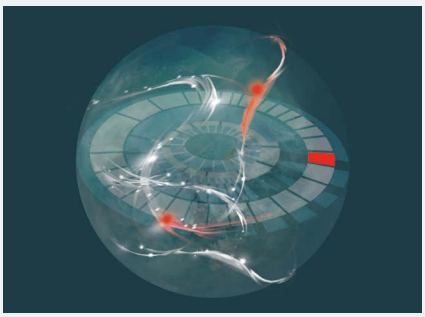
Kalifast a été créé en 2010 par des développeurs venant du monde des ERP. Incongru ? Pas vraiment, car ils ont une expertise technico-fonctionnelle et une vision métier de pointe propre aux progiciels de gestion d'entreprise.

L'aventure est partie de la découverte du robot open source Selenium en 2009 par son fondateur, alors qu'il était salarié d'Oracle. Une idée a rapidement émergé : rendre l'automatisation accessible à tous en levant les verrous du robot Selenium. Cela a permis à l'éditeur de Kalifast, EISGE, d'obtenir le label de J.E.I. validé par le Ministère de l'Enseignement Supérieur et de la Recherche.

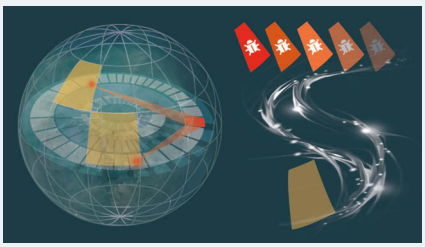
La difficile mise en œuvre de l'automatisation

Dans le monde du tests, l'automatisation va de soi mais n'est pas facile à mettre en place car un projet de développement est par définition mouvant et les équipes peuvent changer d'orientation. Par ailleurs, une application est devenue une jungle technologique : multiplication des technologies, des frameworks et des langages, Cloud computing, mobilité, API diverses, sources de données hétérogènes, etc. L'une des difficultés de la mouvance des projets et du développement est que l'automate doit s'adapter à la même vitesse que le code de l'application, notamment quand vous êtes en cycle agile / devops où l'on prône l'intégration continue, le nightly build (on intègre et on conçoit les codes chaque nuit). Sans parler des utilisateurs de plus en plus exigeants sur les délais...

Dans un tel contexte, les robots de test sont utilisés en fin de process, généralement en pré-production. Cela signifie que les tests automatisés sont joués juste avant le déploiement en production. Cette utilisation tardive a plusieurs conséquences : certes, le coût du scripting semble plus faible, mais l'équipe ne profite pas de la rentabilité des automates. Une telle approche ne permet qu'une détection tardive des dysfonctionnements et des bugs, sans parler de la mobilisation des ressources pour maintenir les automates et les scripts à un moment non optimisé. L'abandon des tests automatiques est donc fréquent.



Une application peut se représenter par une sphère qui est généré à partir d'un code source provenant d'un disque (dur). Un scénario représente la navigation d'un utilisateur sur l'application (la sphère). Un code impacte plusieurs endroits de la sphère et donc plusieurs scénarios.



Une fonction se représente par une zone de la sphère. Elle est utilisée dans plusieurs scénarios, et elle est impactée par plusieurs défauts. L'un des enjeux de l'automatisation est de trouver les impacts réalisés par la modification du code source.

Or, les tests automatisés peuvent être une aide précieuse aussi pour les développeurs. Ils permettent de contrôler la qualité du code tout au long du projet et de gagner du temps car toute opération que l'on automatise ne sera pas réalisée manuellement. Théoriquement, de par leurs compétences, les développeurs sont les personnes les plus à même d'écrire les scripts automatiques, d'autant plus qu'ils ont les moyens de modifier le code de l'application pour la rendre automatisable le cas échéant. De là est venue l'idée fondatrice de Kalifast : rendre l'automatisation accessible aux développeurs pendant les phases de développement sur des environnements les plus instables pour réaliser des économies substantielles lors du scripting des robots, mais aussi afin d'obtenir plus de gain en multipliant les cas d'utilisation.

Rentabiliser l'automatisation

Kalifast tient ses promesses en accélérant jusqu'à dix fois la vitesse du scripting. Le coût de maintenance est supprimé (il n'y a plus de gestion de fichiers), le taux d'utilisation des robots est multiplié par dix (partage des scripts sur l'ensemble des environnements des équipes), et les limites du robot sont levées (intégration des tests semi-automatiques en cas de scripting impossible ou trop coûteux). Qui plus est, les robots Kalifast produisent des reportings, explicites compréhensibles par tous, incluant des captures d'écran, exportables au format Word. Il serait dommage de s'en passer.

Au-delà de l'automatisation

Pour partager les scripts entre les environnements et pouvoir les faire évoluer au même rythme que le code, Kalifast organise les scripts du robot sous forme d'un arbre de fonctions. Les scénarios de tests sont simplifiés en un enchaînement de fonctions. Kalifast utilise ces fonctions pour introduire de nouvelles métriques : qui a travaillé sur ces fonctions ? Quand ont-elles été créées ? Combien de fois ont-elles été testées ? Combien de défauts les ont impactées ? Etc.

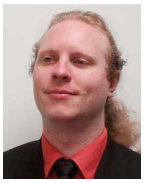
Vous gérez donc votre application en vous focalisant non plus uniquement sur le nombre de défauts corrigés, mais aussi sur la qualité des développements de vos fonctions et de votre application. Ainsi Kalifast vous permet de créer vos abaques en accord avec la théorie des points de fonction.

En résumé, Kalifast se focalise sur les axes suivants :

- La cartographie d'application
- La fourniture de statistiques de livraison sur cette cartographie
- L'assurance que les risques soient identifiés et testés
- La détection des risques d'une livraison
- Le déroulement des tests
- La maintenance facilitée des plans de test
- La capitalisation de l'information.

Bâtir et automatiser un environnement de dev et de tests pour son équipe avec DevTest Labs

À l'heure de l'agilité et du DevOps, avoir la possibilité de construire des environnements de tests aussi nombreux que nécessaires, avoir la possibilité de les construire à la demande, avoir la possibilité de créer rapidement un environnement de développement sont des prérequis indispensables à tout projet.



• Mick Philippon
Consultant Manager
@Cellenza

• Manon Pernin
Consultante Cloud
@Cellenza

cellenza
DOESITBETTER | Conseil - Expertise Microsoft & méthodes agiles

• Mikael Krief
Consultant Senior ALM
@Cellenza

Cependant, tout ceci se heurte rapidement à plusieurs écueils. Tout d'abord, un écueil lié à l'appartenance des environnements ainsi créés. À qui appartient l'environnement MonAppli_Test123 ? Sert-il encore ? Pourquoi utilise-t-il la plus grosse taille d'instance disponible ? Sans supervision, les coûts d'utilisation de l'infrastructure ont tendance à exploser dans le temps, car si on prévoit souvent la création automatisée d'environnements, il est plus rare d'entendre parler de suppression. Et c'est là le second écueil, la gestion de l'inventaire des environnements. Troisième écueil enfin, toujours lié au coût, celui de l'uptime. Un environnement de développement, par exemple, n'est très souvent pas utilisé entre 19h et 8h du matin, pas plus que le week-end. Un environnement de tests peut, lui, n'être utilisé que la nuit, entre 2h et 3h du matin.

Or, il est fréquent de voir ce type d'environnements allumés 24/24, 7/7, entraînant ainsi un surcoût non négligeable pour le projet qui les porte.

La gestion de ces différents sujets impose de mettre en place une couche de gouvernance et, dans la plupart des cas, il s'agira de scripts adaptés à la technologie utilisée. Néanmoins, la création de ces scripts peut être assez coûteuse en termes de temps. La plateforme Microsoft Azure a mis à disposition un outil pour faciliter la tâche : Azure DevTest Lab.

La brique Azure DevTest Lab permet d'adresser ces écueils et de naviguer à travers. Un DevTest Lab est un container pour machines virtuelles offrant un panel de services permettant de créer des recettes (ou formules) de machines facilement, de leur donner une date d'expiration, de programmer leur arrêt et leur démarrage et de leur attribuer un propriétaire clairement identifié. Seul bémol, Azure DevTest Lab ne permet, pour le moment du moins, de ne gérer que du IaaS.

Nous allons tout d'abord vous présenter la création et la configuration d'un DevTest Lab, puis les différentes manières d'automatiser la création de machines virtuelles au sein d'un DevTest Lab, le tout dans une logique d'automatisation complète de création d'environnements, qu'ils soient permanents ou éphémères.

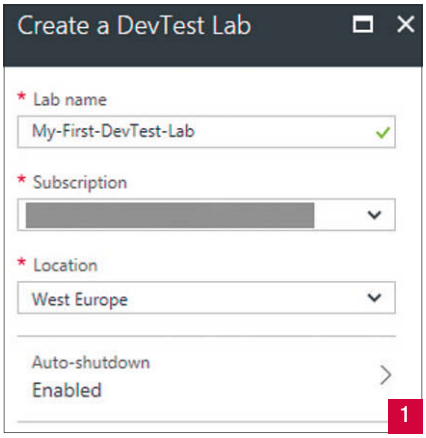
Création et configuration d'un DevTest Lab

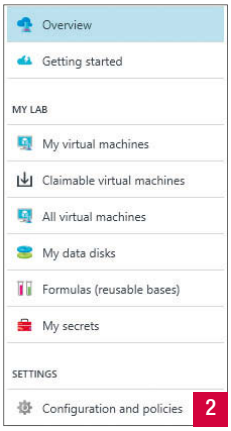
La création d'un DevTest Lab est d'une simplicité biblique. Il suffit de lui donner un nom et une région, et le tour est joué. Il est bien sûr possible d'utiliser un template ARM pour le créer, si nécessaire. [1]

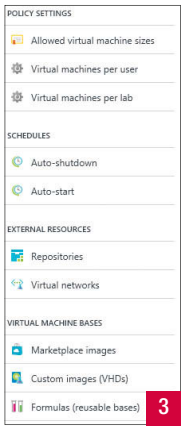
Une seule option mérite d'être détaillée : l'auto-shutdown. Il s'agit d'un mécanisme inhérent au DevTest Lab permettant d'éteindre automatiquement les machines virtuelles du Lab à une heure donnée. Nous parlions plus haut de maîtrise de l'uptime et des coûts, voici donc une partie de la réponse. Si toutes les VM d'un Lab s'éteignent automatiquement tous les soirs à 19h (par exemple), elles ne coûtent rien tant qu'elles n'ont pas été redémarrées. Ainsi, un environnement de tests qui n'est plus utilisé verra son coût réduit à celui de son stockage. Et même s'il est encore utilisé, il ne sera pas facturé la nuit (ni les week-ends). Grâce à cela, on peut passer d'un uptime 24/7 à un uptime 8/5, ce qui divise le coût de possession de l'environnement par 4. Une fois le DevTest Lab créé, outre les classiques Overview et Getting Started, on trouve une section configuration, que nous n'allons pas aborder dans cet article sinon pour un survol rapide, et une section 'My Lab' qui permet de créer des VM, de les réserver, de les gérer mais également d'accéder aux formules (qu'il est possible de comparer grossièrement à des templates de VM) et aux secrets, chaque DevTest Lab incluant une CMDB (Configuration Management DataBase), c'est-à-dire un coffre sécurisé où stocker mots de passe, clefs d'API, … [2]

Configurer un DevTest Lab

La partie de configuration d'un DevTest Lab va nous permettre de le piloter. Comme l'indique le 'My' dans 'My Lab', le DevTest Lab propose à chaque utilisateur une vue personnalisée de







son contenu. Cependant, il existe des éléments qui transcendent cette personnalisation.
Le premier élément concerne les limites pour les utilisateurs. Limites en termes de nombre de VM, total ou par utilisateur et limites en termes de tailles de VM autorisées dans le Lab. Tout ceci peut être fixé de manière à maîtriser la taille et le coût induit par un Lab.
Le second concerne l'auto-shutdown et l'auto-start. Comme mentionné, il est possible d'éteindre automatiquement toutes (toutes ? Non, il est possible de faire un opt-out) les ressources au sein d'un Lab automatiquement selon un agenda programmé. Il est également possible de démarrer automatiquement toutes (toutes ? Non, seules celles qui ont fait un opt-in) les ressources du Lab selon un second agenda. Par ailleurs, nous pouvons rattacher des réseaux virtuels à un DevTest Lab, permettant ainsi de créer les VM sur ce réseau, pour tirer parti, par exemple, d'une express route et d'un réseau étendu vers le réseau interne de l'entreprise. Nous pouvons également rattacher des repository privés (GitHub ou VSTS) à un DevTest Lab, qui contiennent des artefacts, permettant ainsi de proposer des artefacts et des templates ARM personnalisés. [3]

Note : *Un template ARM (Azure Resource Manager) est un fichier au format json qui indique de façon déclarative la liste et la configuration des ressources qui doivent être déployées. Le déploiement basé sur ces templates est idempotent.*

Enfin, le dernier élément de configuration possible est le choix des bases de VM disponibles pour les utilisateurs. Par défaut, un DevTest Lab propose toutes les bases disponibles sur Azure. A cela peuvent se rajouter celles provenant de repository privés, mais, pour des raisons de simplification, nous pouvons décider de limiter la liste (a-t-on vraiment besoin de SQL Server 2012, 2014 et 2016 ?).

Automatisation de la création de ressources

S'il est bien sûr possible de créer des ressources manuellement dans un DevTest Lab, tout l'intérêt réside dans l'automatisation de cette création. Personne n'a envie d'activer IIS à la main sur chacun des environnements créés ou de configurer pour la 30ème fois SQL Server. L'automatisation dans un DevTest Lab peut partir de trois bases différentes : les templates de VM disponibles sur le marketplace, des VHD préconfigurés ou des formules. Nous allons passer rapidement sur les deux premières, relativement classiques et connues, pour nous attarder sur l'intérêt principal du DevTest Lab : les formules.

Base de machine virtuelle : les templates

Créer une VM à partir d'un template est très simple, que ça soit de manière interactive ou à partir d'un template ARM. Le template peut être obtenu simplement en commençant une création interactive, puis en cliquant sur le menu 'View ARM Template'. Il suffit alors de le sauvegarder sur GitHub ou dans un BLOB pour pouvoir le réutiliser à l'envi. [4]

Base de machine virtuelle : les VHD

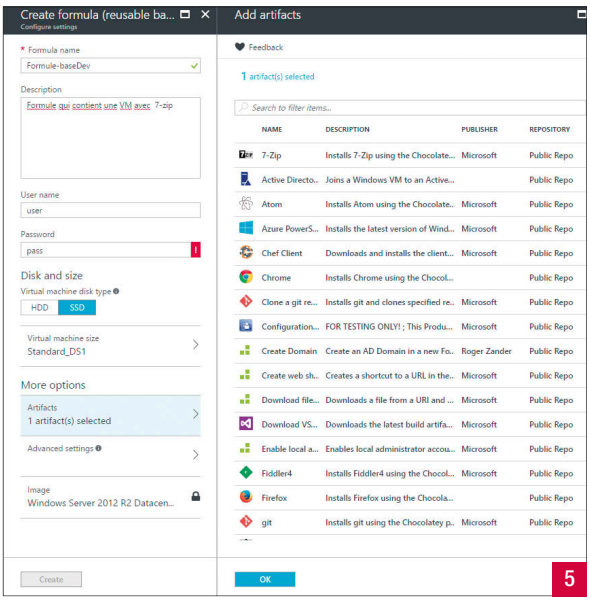
Les VHD disponibles sont mis à disposition via la partie configuration du DevTest Lab. Là encore, rien de plus simple : sélectionner le VHD, récupérer le template ARM, et le tour est joué. [5]

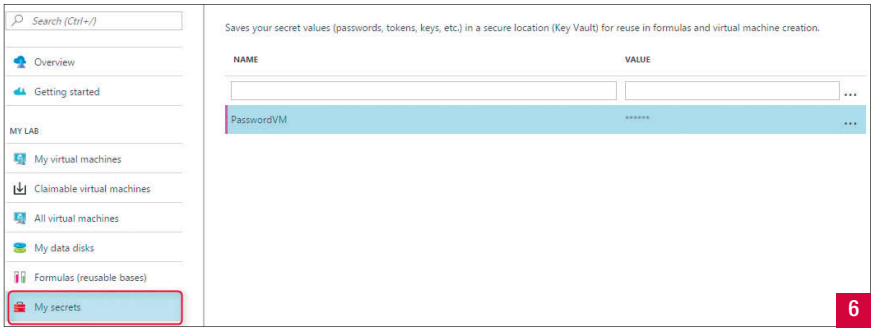
Base de machine virtuelle : les formules

C'est au travers des formules que DevTest Lab montre tout son potentiel. Une formule, un peu comme une recette de cuisine, indique une démarche à suivre pour obtenir une VM prête à l'emploi. Elle se compose :

- D'une base, soit un template de VM (Windows server 2016 nu, par exemple, ou alors Biztalk 2016 enterprise) pris parmi la liste des templates accessibles au Lab.
- D'éléments de configuration. Les VM créées par cette formule utilisent-elles des SSD ou des HDD classiques ? Quelles seront leurs tailles ? A quel réseau virtuel sont-elles rattachées ? Ont-elles une IP publique ? Sont-elles automatiquement attribuées à leur créateur ou sont-elles placées dans le pot commun ? Toutes ces questions trouvent une réponse dans ces éléments.
- D'artefacts. Un artefact est un élément de configuration qui sera appliqué sur la base. Il existe par exemple des artefacts pour installer des outils tels que 7zip ou Notepad++, d'autres pour exécuter un script PowerShell arbitraire, etc. Ces artefacts sont appliqués dans l'ordre indiqué, il est donc possible de se baser sur le résultat des précédents pour les enrichir (un exemple est de rejoindre un AD avant de configurer Sharepoint).

Une fois la base choisie, la configuration et les artefacts à appliquer, la formule est alors disponible à l'ensemble des utilisateurs du DevTest Lab, qui peuvent s'en servir pour créer des VM.





Gestion des machines virtuelles

L'avantage de DevTest Lab est que chaque utilisateur va en avoir une vue personnalisée. Cela grâce essentiellement à deux mécanismes : les claims et les secrets.

Les secrets

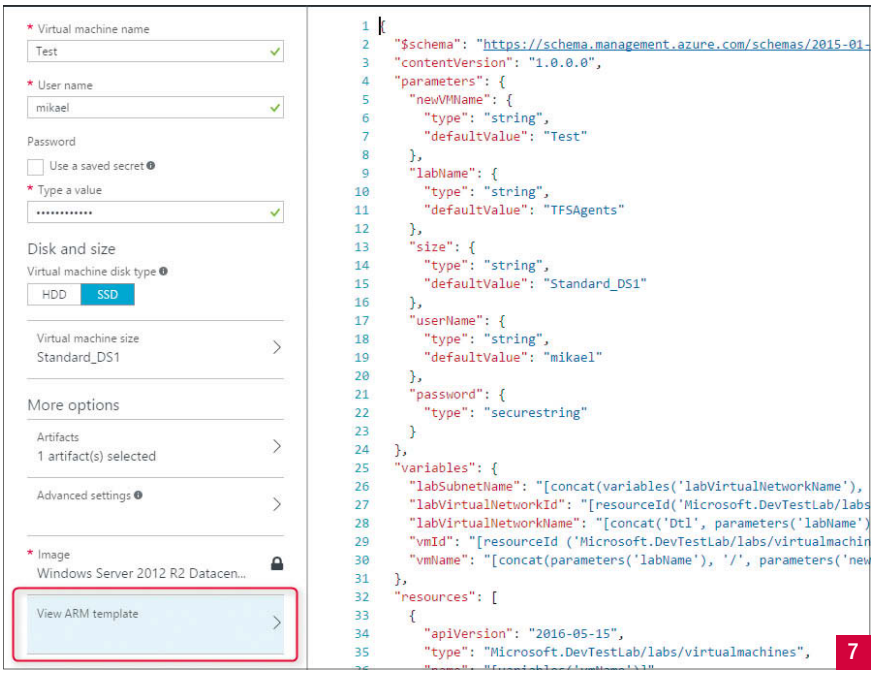
DevTest Lab est fourni avec un coffre-fort pour chaque utilisateur. Ce coffre-fort permet de stocker des mots de passe, des clefs SSH, des PAT (Personnal Access Token) pour l'accès à des ressources Azure ou à VSTS, etc. [6]

Chaque utilisateur y stocke ce qu'il souhaite. Une fois stocké dans le coffre-fort, un secret peut être utilisé lors de la création de VM ou de l'application d'artefacts. Si on est amené à créer de nombreuses fois des environnements utilisant l'artefact permettant de cloner un repository Git, par exemple, il peut être intéressant de stocker ses informations d'authentification dans le coffre-fort, pour éviter à chaque fois de devoir se rendre dans son gestionnaire de mot de passe pour faire un copier-coller.

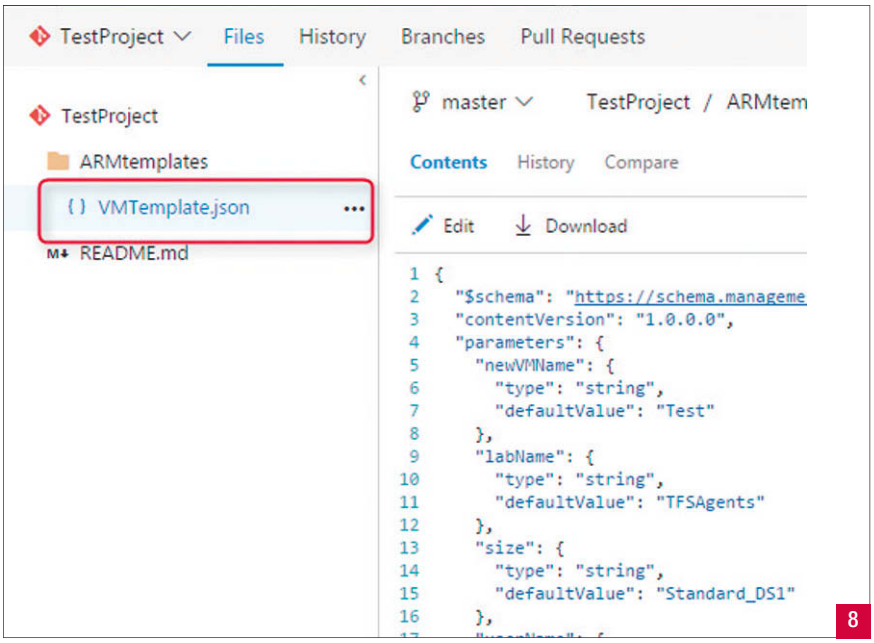
Les claims

Dans DevTest Lab, une machine virtuelle peut avoir deux états : claimable (attribuable) ou claimed (attribuée). Une VM claimable est en quelque sorte dans le pot commun. Tous les utilisateurs du DevTest Lab peuvent appliquer des artefacts supplémentaires sur cette VM, peuvent la démarrer, la stopper, la détruire, l'inscrire ou la désinscrire de l'auto-shutdown ou de l'auto-start, …

Et tous peuvent aussi se l'attribuer, la passant ainsi au statut claimed. Une VM dans ce statut n'est visible que par la personne à qui elle est attribuée (et de l'administrateur du Lab, bien sûr). Cela permet donc de réserver des ressources de manière à pouvoir créer des environnements de test isolés sans subir d'interférence d'autres utilisateurs du Lab.



7



8

Automatiser des tests dans DevTest Lab avec VSTS

La mise en œuvre de tous ces éléments nous permet d'automatiser entièrement l'exécution de tests d'application, comprenant la création d'un environnement de tests, le jeu des tests et la collecte des résultats, puis la suppression de cet environnement.

Ainsi nous allons pouvoir exécuter des tests de façon automatique sur des environnements différents (VM avec différentes configurations) et avoir un meilleur gain en termes de coût puisque ces VM ne seront seulement présentes que lors de l'exécution des tests.

L'outil utilisé pour automatiser cette exécution est VSTS (Visual Studio Team Services) qui possède tous les éléments pour s'intégrer avec DevTest Lab. Cette automatisation s'opère en plusieurs étapes :

Récupération du template ARM

Dans DevTest Lab créer une VM avec la configuration souhaitée et récupérer son template ARM [7]

Ici, le template ARM consiste en la description de la configuration de la VM au format Json. Cette description concerne les informations de base de la VM (taille, nom du lab), ses informations de connexion (login et mot de passe administrateur) et dans notre cas de DevTest Lab, la liste des artefacts sélectionnés. L'avantage de posséder ce Template est de pouvoir automatiser la création de ressources Azure avec des commandes PowerShell ou bien aussi avec VSTS.

Le contrôleur de sources

Ce template ARM doit être archivé dans le code source de votre projet de VSTS (à l'endroit où sont les tests) [8]

La génération du package

La definition de build compile le projet de tests et publie en artefacts les assemblies de tests ainsi que les Template ARM [9]

Automatiser la création de la VM dans le Lab et l'exécution les tests

C'est dans la partie Release de VSTS que nous allons créer une définition de release qui va automatiser de bout en bout la création de la VM ainsi que l'installation de tous ses artefacts, la copie des tests sur celle-ci, l'exécution des tests et pour finir supprimer les VM. [10]

Les tâches DevTest Lab sont disponibles dans le Visual Studio Marketplace https://marketplace.visualstudio.com/items?itemName=ms-azuredevtestlabs.tasks

Dépendances avec des briques PaaS

Comme indiqué précédemment, Azure DevTest Lab permet aujourd'hui de ne gérer que des VM. Or prenons le cas d'un site Web, hébergé sur une VM, lié à une base SQL Azure Database. Il peut être intéressant de faire des tests avec une base SQL Azure Database plutôt qu'avec une base sur la VM. Dans des cas comme celui-ci, il va falloir passer par de l'Infra as Code : c'est-à-dire mettre en place un script qui gère le déploiement des ressources souhaitées. [11]

Dans un contexte Azure, nous allons utiliser les fameux templates ARM pour définir les ressources qui sont nécessaires. Le déploiement et la configuration de ces ressources supplémentaires sera intégré à notre Release avant de procéder à l'étape de tests. On veillera à ajouter une étape de suppression à la fin de nos tests.

L'étape « Azure Resource Group Deployment » permettra de créer un groupe de ressources dans Azure à partir d'un template ARM. On pourra ainsi déployer une base SQL Azure. En ajoutant une étape « Azure SQL Database Deployment » on va pouvoir insérer un jeu de données de tests. [12]

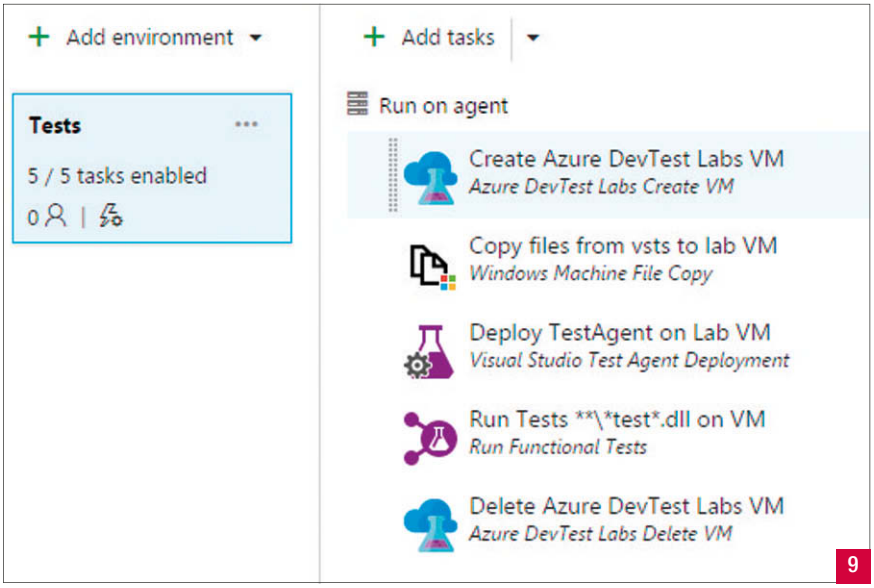
On passe ensuite aux étapes de tests comme expliqué précédemment et enfin nous pouvons supprimer nos ressources PaaS à l'aide d'une étape Azure Resource Group Deployment configurée pour supprimer notre groupe de ressources.

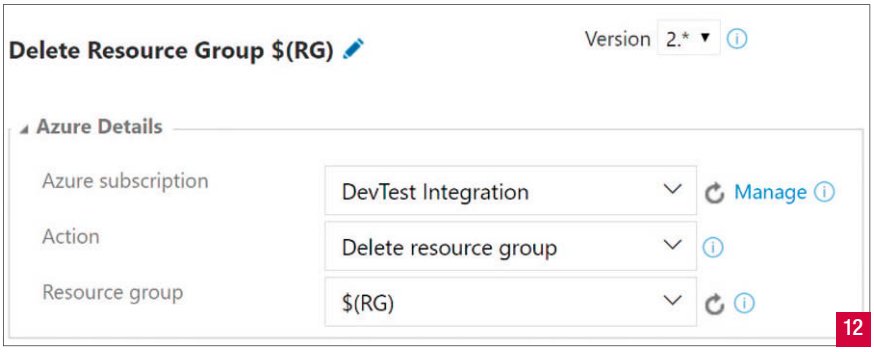
Que retenir de tout cela ?

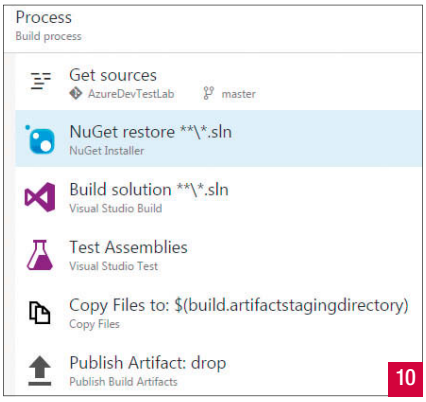
Le seul défaut d'un DevTest Lab est sa limitation à la création de VM. S'il est possible, à l'aide de VSTS, de passer outre cette limitation et de créer des environnements hybride PaaS/IaaS, force est de constater qu'en l'état, la partie PaaS ne bénéficie pas de tous les avantages induits par l'utilisation d'un DevTest Lab, tant pour les développeurs que pour les gestionnaires. A travers les vues personnalisées et la possibilité de réserver des ressources, il permet aux développeurs de s'assurer une exclusivité sur certaines ressources, évitant ainsi des interventions intempestives d'autres personnes ou équipes sur un environnement en cours d'utilisation pour une campagne de tests. Ce même mécanisme peut être utilisé pour déporter ses machines de développement dans Azure, offrant ainsi des machines de développement toujours à jour en termes de hardware sans devoir créer une catégorie spécifique aux développeurs dans la politique d'achats informatique de l'entreprise.

Pour les gestionnaires, Azure DevTest Lab offre la possibilité de maîtriser les coûts liés à l'infrastructure de développement et de tests, soit en s'assurant que toute ressource ainsi créée possède une date d'expiration, soit en maîtrisant les temps d'allumage des environnements.

Par ailleurs, la facilité de création des formules au sein d'un DevTest Lab permet de créer des environnements adaptés à un projet sans devoir se plonger dans les arcanes des API Azure et de manière extrêmement simple. •



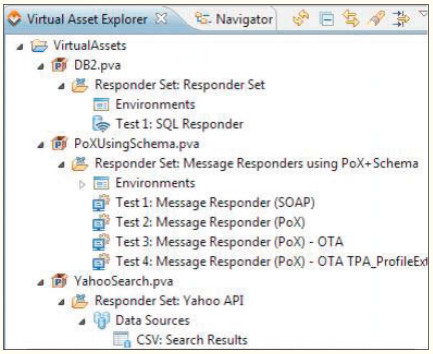




PARASOFT

La virtualisation de services : soutenir les tests

Les tests sont essentiels pour produire des applications de qualité. Pour réussir sa stratégie de tests, il faut les réaliser le plus tôt possible dans le cycle du projet puis à chaque itération, à chaque intégration continue, puis durant les phases de livraison et avant la production. Aujourd'hui, une app dépend de multiples composants et de sources de données, parfois difficilement accessibles durant la phase de développement. La virtualisation de services est une aide précieuse.



Jusqu'à présent, pour mettre en place des bouchons fonctionnels, il fallait installer des simulateurs ou des machines pour supporter ces services tiers, les bases de données ou se connecter aux ressources réelles, avec les contraintes que cela suppose. On définit le bouchon fonctionnel ainsi : services virtuels remplaçant le comportement des ressources/dépendances d'une application, comme des services Web (SOAP ou REST), des files de message, des bases de données, un mainframe,...

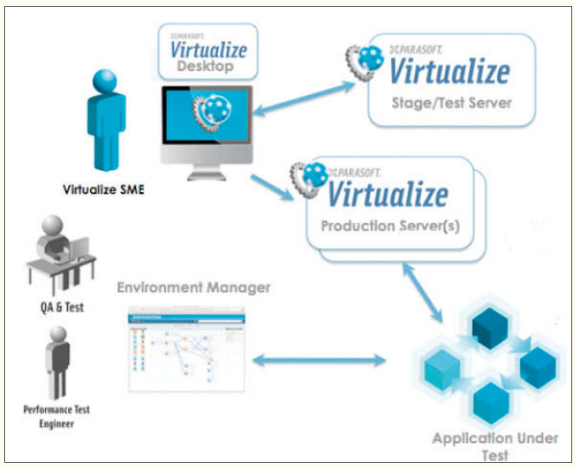
La virtualisation de services (ou bouchonnage fonctionnel) ne doit pas être confondue avec la virtualisation desktop, de serveurs ou d'infrastructure. Cette nouvelle technique apporte une grande flexibilité dans les tests d'environnements dépendants et complexes. On peut rapidement déployer des environnements complets, tout en gérant les données et les performances. Et on évite d'installer et de configurer de nouveaux matériels.

Parasoft a proposé son premier outil de virtualisation de services dès 2002 pour les Web services dans son outil de tests fonctionnels, SOAtest. En 2008, la solution Virtualize est arrivée sur le marché pour les entreprises et les équipes de développements et de tests.

La plateforme de création et de déploiement de Services Virtuels (bouchons) de Parasoft, permet de créer automatiquement des bouchons fonctionnels qui sont déployés sur des serveurs accessibles par les applications en test. Nous voyons ici les phases de création/modélisation à partir d'un proxy de capture et de déploiement sur un Serveur Virtualize. Le module Environment Manager permet ensuite de gérer facilement les bouchons et les accès depuis les applications sous tests.

Virtualize pour tous avec l'édition communautaire

De nombreux développeurs ou petites entreprises (éditeurs, PME, etc.), se limitent aux tests « traditionnels » (tests unitaires, tests fonctionnels) mais avec l'explosion du Cloud, des API et de la multiplication des sources de données, il était devenu parfois difficile de tester efficacement. Pour les aider et faire découvrir la virtualisation de services, Parasoft propose, depuis février 2017 : Virtualize Community Edition. Il s'agit d'une nouvelle offre de l'éditeur, gratuite et suffisamment riche en fonctionnalités pour démarrer rapidement. Comcast (premier câblo-opérateur américain) a constaté une réduction de 60 % du temps de mise en place de ses environnements de tests. Ignis Asset Management, une société financière londonienne indique avoir divisé par 20 le temps nécessaire au déroulement de ses campagnes de tests. Comparaison des différentes éditions : http://www.checking-tech.com/fr/products/virtualize.php



L'outil est simple à déployer :

- Des serveurs pour déployer et exécuter les bouchons ;
- La plateforme de développement : Virtualize Desktop ;
- Module de déploiement : Environment Manager, pour créer les bouchons.

4 usages

Plusieurs cas d'usage sont intéressants pour la virtualisation de services :

- La simulation d'interactions complexes entre services Web (souvent des services en appellent d'autres) ;
- Dans le cas de tests de charge, les temps de réponse des services tiers sont simulés de façon très consistante et répétitive, ce qui permet d'évaluer des situations extrêmes beaucoup plus facilement qu'avec une approche matérielle ;
- Pour le test des applications mobiles, la virtualisation est particulièrement bien adaptée pour simuler la géolocalisation, le débit, les cas d'erreurs, les protocoles de communication pour tout ce qui est SMS, appels JSON, appels REST, téléphonie.
- Pour les tests complets, de bout en bout, la virtualisation de services, de réseaux et des données permet de réaliser des tests qui n'étaient pas envisageables en raison du nombre et de la complexité des accès.

Parasoft propose aussi sa Marketplace (https://marketplace.parasoft.com) où l'on trouve, gratuitement, de nombreux bouchons préconfigurés, des extensions de protocoles ou des utilitaires comme un créateur de bouchons à partir d'un fichier Wireshark, ou encore un outil de création automatique d'un Environnement de Test à partir d'un fichier RAML ou Swagger ou soapUI,...

Téléchargement Virtualize Community Edition : http://www.checking-tech.com/fr/products/virtualize-community-edition.php

Les tests automatisés à chaque étape du DevOps

Paquet Judicaël
Coach Agile & Devops, Judicaël accompagne les entreprises comme Renault dans leur transformation avec Myagile Partner qu'il a créée en janvier 2017. **Myagile Partner**

Le Devops, qui est l'une des attentions particulières des DSI cette année, permet d'accélérer le time-to-market, mais aussi d'améliorer considérablement la qualité des codes livrés en production. Nous allons voir dans cet article les différents types de tests utilisés dans le monde du Devops qui sont assez nombreux et qui sont garants d'un bon fonctionnement applicatif.

Si le DevOps est un changement de culture d'entreprise et pas seulement des pratiques techniques, nous allons nous pencher sur un des aspects essentiels du DevOps : les tests automatisés pour garantir la qualité des livraisons.

Intégration continue, livraison continue et déploiement continu

Le Devops met en avant trois pratiques essentielles dans l'accélération des livraisons et la qualité de leurs contenus : l'intégration continue, la livraison continue et le déploiement continu.

L'intégration continue (CI) est une pratique en génie logiciel qui consiste à limiter au maximum les régressions grâce à de nombreux tests qui vérifient le bon fonctionnement de l'ensemble du code à chaque modification de celui-ci. Si ces pratiques s'étaient déployées de façon limitée dans le passé avec la mise en place de l'Extreme Programming (XP), elles sont en train de prendre une véritable ampleur avec la culture Devops. La livraison continue (continuous delivery) est une technique d'ingénierie informatique qui consiste à tester, préparer et déployer un changement de code. Une validation humaine finale sera à réaliser avant le déploiement final.

Le déploiement continu (continuous deployment) est une technique d'ingénierie informatique similaire à la livraison continue sauf que le déploiement s'automatise sans aucune validation humaine en amont. Le système est souvent capable de faire un rollback automatique si des erreurs bloquantes sont repérées (rollback humain possible). [1]

Mise en place de tests unitaires

Une pratique très appréciée dans le monde du Devops que nous trouvions d'ailleurs déjà dans l'XP, est la pratique du TDD (Test Driven Development). La TDD est une technique de développement logiciel qui consiste à écrire les tests unitaires d'une fonction avant d'écrire le contenu de cette fonction ; on parle de test unitaire quand on teste une fonction indépendamment des autres.

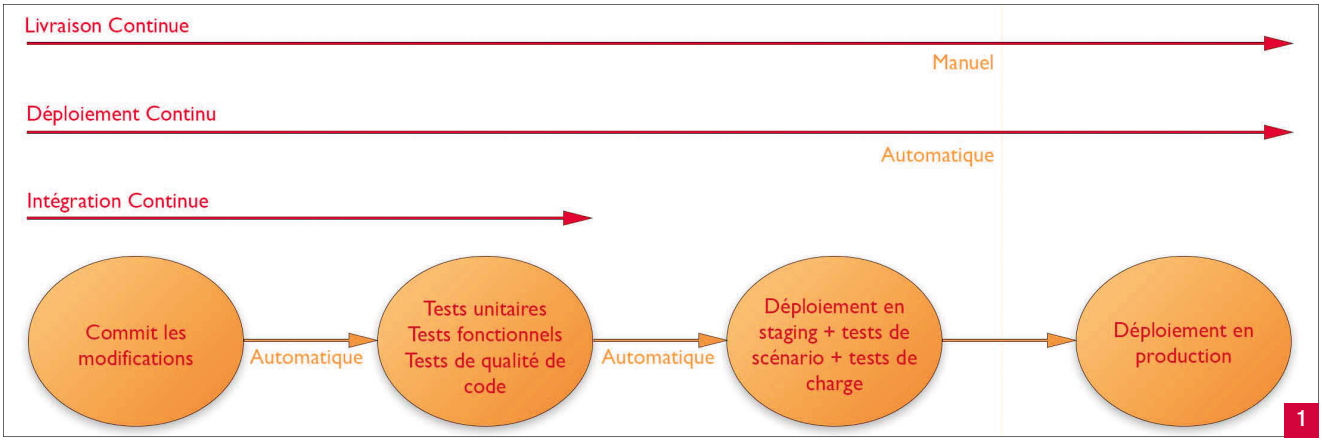
La TDD propose un cycle de travail aux développeurs pour obtenir une qualité optimale de la mise en place des tests unitaires :

- 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) ;

Chaque langage va avoir son outil spécialisé dans la mise en place de tests unitaires : PHP Unit pour PHP, Junit pour Java, Unittest pour Python… Tous les langages proposent leur propre outil de tests unitaires. C'est une démarche qui paraît contraignante au premier abord mais qui permettra de considérablement baisser le taux de bugs à l'avenir ; elle est un premier barrage de la non régression de l'application.

Installation de l'environnement

Afin de bien comprendre les principes de la TDD, nous allons installer un Symfony 3 de base sur lequel nous allons appliquer ces concepts de



1

TDD. Voici le minimum requis pour installer le nécessaire sur un Linux Ubuntu 16.04 :

```
sudo apt-get update
sudo apt-get install -y mysql-server apache2 php7.0 libapache2-mod-php7.0 php7.0
-mysql php7.0-cli php7.0-xml phpunit php7.0-curl composer php7.0-mbstring
sudo a2enmod rewrite
service apache restart
```

Ensuite nous devons installer notre Symfony 3 afin de pouvoir faire notre premier test unitaire :

```
sudo mkdir -p /usr/local/bin
sudo curl -LsS https://symfony.com/installer -o /usr/local/bin/symfony
sudo chmod a+x /usr/local/bin/symphony
sudo chmod -R 777 /var/www
sudo symfony new /var/www/mon_projet
```

Vous pourrez tester si Symfony est bien installé en lançant son serveur Web intégré comme ceci :

```
cd /var/www/mon_projet
bin/console server:run
```

Création du test unitaire

Nous allons ainsi créer notre premier fichier src/AppBundle/Util/Addition.php contenant une fonction très simple dans laquelle nous n'écrirons rien à cette étape ; pour rappel, la première étape consiste à écrire le test et à vérifier qu'il échoue bien (car la fonction à tester est vide) :

```
<?php
// src/AppBundle/Util/Addition.php
namespace AppBundle\Util;

class Addition
{
    public function addTwo($a)
    {
    }
}
```

Nous allons à présent écrire notre premier test unitaire tests/AppBundle/Util/AdditionTest.php qui va tester cette fonction :

```
<?php
// tests/AppBundle/Util/AdditionTest.php
namespace Tests\AppBundle\Util;

use AppBundle\Util\Addition;

class AdditionTest extends \PHPUnit_Framework_TestCase
{
    public function testAddTwo()
    {
        $calc = new Addition();
        $result = $calc->addTwo(30);
        // on vérfie qu'on a bien 2 en retour
        $this->assertEquals(32, $result);
    }
}
```

Si nous testons en faisant « phpunit » dans notre terminal, nous aurons un joli message d'erreur qui s'affiche :

```
FAILURES !
Tests : 2, Assertions : 3, Failures : 1.
```

L' étape suivante est d'écrire le contenu de notre fichier afin que le test unitaire soit cette fois complètement validé ; nous allons modifier le premier fichier pour avoir ceci :

```
<?php
// src/AppBundle/Util/Addition.php
namespace AppBundle\Util;

class Addition
{
    public function addTwo($a)
    {
        return $a + 2;
    }
}
```

En faisant notre « phpunit » dans notre terminal, nous aurons cette fois une validation de l'ensemble de nos tests comme ceci :

```
OK (2 tests, 3 assertions)
```

Afin d'automatiser ces tests, vous pourrez utiliser des outils comme Jenkins, Team City ou Travis ; ce dernier est très simple à utiliser car c'est un outil Saas qui se connecte automatiquement à Github. Il lancera par exemple vos tests unitaires à chaque push réalisé.
Travis a l'avantage d'être simple car un fichier .travis.yml à la racine du projet permettra de lancer vos tests automatiquement :

```
language: php
php:
  - '7.0'
before_script:
  - cd /home/travis/build/judicaelpaquet/monRepositoryTest
  - composer install
```

Judicaelpaquet/monRepositoryTest est votre utilisateur Github et le nom du repository. Vous pourrez voir tous les résultats sur le site de Travis ou directement sur votre espace Github.

Faire de la BDD

La 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 vivement le rapprochement des équipes techniques, des équipes fonctionnelles, voire des équipes métiers.
Cette pratique est beaucoup moins fréquente dans les entreprises que

celle des tests unitaires car elle implique beaucoup de changement au niveau des mentalités de l'entreprise.
Pour faire simple, la BDD a comme principes :

- La participation des équipes non techniques au projet pour écrire nos tests fonctionnels ;
- Des scénarios automatisés pour décrire des comportements afin de faire de la non régression ;
- Mieux comprendre le comportement attendu pour les équipes techniques.

Nous allons installer Behat dans Symfony en complétant le fichier composer.json à la racine du projet :

```
"require-dev": {
    "behat/behat": "^3.1",
    "behat/mink": "^1.7",
    "behat/mink-browserkit-driver": "^1.3",
    "behat/mink-extension": "^2.2",
    "behat/symfony2-extension": "^2.1",
    "behatch/contexts": "^2.5",
```

Ensuite, il suffit d'initialiser Behat qui va créer un dossier features avec un fichier features/bootstrap/FeatureContext.php :

```
sudo vendor/bin/behat --init
```

Nous allons écrire notre premier test avec le langage Gherkin qui est le langage compris par l'outil Behat que nous avons installé. Voici un exemple de tests dans le fichier features/AppBundle/check.feature

```
Feature: test
  Scenario: Test my homepage
    When I load http://localhost/
    Then I should see "Welcome to"
```

En lançant Behat avec la commande « sudo vendor/bin/behat », Behat nous fera un retour complet du type :

```
Feature: test

  Scenario: Test my homepage      # features/AppBundle/test.feature:2
    When I load "http://localhost/"
    Then I should see "Welcome to"

1 scénario (1 indéfinis)
2 étapes (2 indéfinis)
0m0.02s (7.08Mb)

 >> default suite has undefined steps. Please choose the context to generate snippets:

  [0] None
  [1] FeatureContext
 > 1

--- FeatureContext a des étapes manquantes. Définissez-les avec les modèles suivants :

    /**
     * @When I load :arg1
     */
    public function iLoad($arg1)
    {
        throw new PendingException();
    }

    /**
     * @Then I should see :arg1
     */
    public function iShouldSee($arg1)
    {
        throw new PendingException();
    }
```

Behat nous indique qu'il nous faudra écrire un minimum de code pour pouvoir avoir notre test fonctionnel. Nous allons donc écrire le contenu de notre test fonctionnel pour qu'il soit testable concrètement en remplissant le fichier features/bootstrap/FeatureContext.php qui a été créé lors de l'initialisation de Behat :

```
<?php

use Behat\Behat\Context\Context;
use Behat\Gherkin\Node\PyStringNode;
use Behat\Gherkin\Node\TableNode;

/**
 * Defines application features from the specific context.
 */
class FeatureContext implements Context
{
    private $_contentPage;

    /**
     * Initializes context.
     *
     * Every scenario gets its own context instance.
     * You can also pass arbitrary arguments to the
     * context constructor through behat.yml.
     */
    public function __construct()
    {
    }

    /**
     * @When I load :arg1
     */
    public function iLoad($arg1)
    {
        $curl = curl_init($arg1);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        $this->_contentPage = curl_exec($curl);
    }

    /**
     * @Then I should see :arg1
     */
```

```
    public function iShouldSee($arg1)
    {
        if (!strstr($this->_contentPage, $arg1)) {
            sprintf("Could not see '%s'", $arg1);
            throw new PendingException();
        }
    }
}
```

En lançant Behat, vous aurez une grosse erreur qui apparaît car aucun « Welcome to » n'est retourné sur la page d'accueil. Par défaut et sans avoir défini de VirtualHost, nous avons la page d'accueil d'Apache 2.
Si maintenant nous définissons notre VirtualHost pour qu'il pointe sur notre projet Symfony, vous verrez que le même test est valide (car la page contient le terme « Welcome to ») :

```
<VirtualHost *:80>
    ServerName localhost

    DocumentRoot /var/www/mon_projet/web/
    <Directory /var/www/mon_projet>
        AllowOverride All
        Order Allow,Deny
        Allow from All
    </Directory>
</VirtualHost>
```

Ce test est très simple mais il vous permet de comprendre comment faire des tests fonctionnels. Si ici, nous avons fait cela sur Symfony, la démarche reste relativement proche dans d'autres langages de développement. Dans Travis il faudra définir dans votre fichier le fait de vouloir lancer Behat de cette façon si vous désirez que tout votre projet soit testé fonctionnellement :

```
script:
  - vendor/bin/behat
```

Vous pourrez approfondir ce sujet en allant sur le site de Travis et le site de Behat.

Tester la qualité du code

Afin d'avoir toujours du code de qualité, il est essentiel de tester continuellement la qualité du code : les bonnes pratiques, les éventuelles failles de sécu, si les normes de codage sont respectées… Le but est de ne pas voir son code se dégrader avec le temps.
Pour des applications Symfony 3, deux outils se complètent plutôt bien dans ce domaine et se connectent parfaitement bien à l'univers Github : SensioLabsInsight et Scrutinizer.
Il suffira de connecter SensioLabsInsight et Github pour faire fonctionner celui-ci. Nous allons plutôt nous concentrer sur ce deuxième outil qui propose de nombreuses possibilités.

Scrutinizer

Scrutinizer est un outil Saas qui permet de vérifier l'ensemble de votre code sur les points déjà cités ci-dessus.
Il vous faudra faire un « webhook » sur Github pour profiter de Scrutinizer sur votre repository et mettre un fichier .scrutinizer.yml à la racine pour indiquer les tests qui seront à réaliser :

```
build:
    tests:
        override:
            -
                command: 'phpunit --coverage-clover=some-file'
                coverage:
                    file: 'some-file'
                    format: 'clover'
    environment:
        php:
            version: 7.0.7
```

Scrutinizer permet également de lancer du test unitaire comme vous le voyez dans l'exemple précédent. Si vous désirez ignorer certaines parties dans vos tests pour exclure par exemple les librairies extérieures, voire le framework, vous pouvez rajouter ceci :

```
filter:
    excluded_paths:
        - 'app/*'
        - 'bin/*'
        - 'features/*'
        - 'tests/*'
        - 'var/*'
        - 'vendor/*'
```

Je peux également indiquer à Scrutinizer ce que je considère comme un build invalide afin d'exiger une qualité minimale pour autoriser un déploiement. Voici un exemple simple de définition de la qualité exigée :

```
build_failure_conditions:
    # Aucune issue critique ne sera acceptée
    - 'issues.severity(= CRITICAL).exists'
    # Aucune nouvelle issue critiques ne sera acceptée
    - 'issues.severity(= CRITICAL).new.exists'
    # Ma classe n'a pas de test
    - 'classes.metric("php_code_coverage.coverage", = 0).exists'
    # Si la note est D ou pire
    - 'elements.rating(<= D).exists'
```

Sachez que ces outils sont gratuits sur des repository publiques.

Déploiement automatisé ?

Il existe de très nombreuses techniques pour livrer en production et nous n'aurons pas la place ici de toutes les citer.
Avec Github, il est possible de se connecter à AWS avec CodeDeploy qui propose d'ailleurs l'option Blue Green que nous verrons après.
Travis propose également de faire des livraisons directement en production si vous avez tous vos tests validés ; pour cela il suffira de compléter le fichier .travis.yml que nous avons vu en ajoutant cette nouvelle partie :

```
after_success:
  - [fonctions]
```

Le code se déploiera en cas de succès des tests.

Tester nos livraisons avec du Blue Green

Il existe des concepts de tests de plus en plus populaires pour faire de la livraison continue et du déploiement continu.

Le déploiement Blue Green devient un pattern de déploiement très répandu dans le monde du Devops qui a été initialement proposé par Jez Humble et David Farley dans leur livre Continuous Delivery en 2010. Cependant si ce pattern est devenu un classique des meetups Devops, on ne la rencontre qu'assez rarement dans les entreprises car on est vite confronté à un certain nombre de problèmes pas toujours simples à résoudre.

Dans le monde du Devops, on aime dire que nous pouvons déployer sans jamais interrompre les services. C'est forcément une notion très vendeur pour de nombreuses personnes au sein de nos entreprises. C'est le concept du Zero Downtime Deployment (ZDD) qui a pour but d'offrir la qualité de pouvoir livrer régulièrement de façon transparente et de façon quasi instantanée. Si cela peut paraître évident dans toutes les structures jeunes, il n'est pas rare de voir encore des déploiements prendre des heures dans les grandes entreprises qui ont encore beaucoup de Legacy.

De façon simplifiée, le déploiement Blue Green propose d'avoir un environnement de production identique à celui qui est utilisé par les utilisateurs qui recevra une nouvelle version. Il sera essentiel d'avoir un outil de routage qui saura passer d'un environnement de production à l'autre afin d'avoir la nouvelle version de façon totalement transparente. Aujourd'hui le load balancer HAProxy de Willy Tarreau est devenu le plus populaire pour faire ce type de routage. [2]

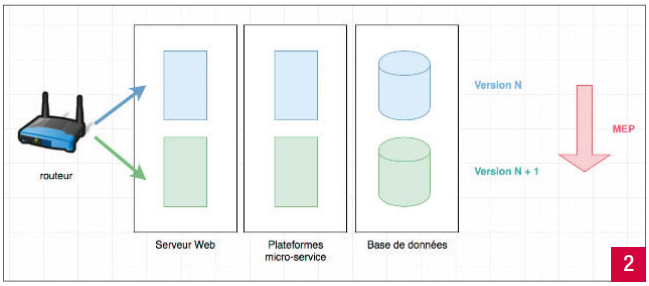
SI certaines entreprises utilisaient déjà des principes similaires il y a 10 ans de façon moins formalisée, cela avait un véritable coût non négligeable car on parlait de serveurs physiques ; aujourd'hui les Cloud permettent de mettre ce type de pratiques à des coûts raisonnables.

Quand tous les contrôles nécessaires sont réalisés, vous pourrez dire à votre routeur d'amener les utilisateurs vers l'environnement Green. Avec cette méthode, le rollback est relativement simple à mettre en place, car il suffira de dire au routeur de revenir sur la version Blue. Ces actions de routage sont instantanées pour les utilisateurs.

Pour ceux qui sont imaginatifs, ce concept pourrait même vous permettre de faire de l'A/B Testing car il suffira de router % de vos utilisateurs sur le Green et % de vos utilisateurs vers le Bleue.

Faire du Canary Release

Le Canary Release est un pattern associé qui permet de mettre une petite tranche de population sur la version N + 1 en laissant temporairement



2

les autres sur la version N. Cela permet de tester si tout va bien avant de déployer sur l'ensemble des utilisateurs.

Facebook utilise ce pattern pour déployer un jour avant une nouvelle version de l'application aux employés de l'entreprise.

Faire du Dark launch

Le Dark Launch est également un pattern associé au déploiement Blue Green avec comme concept de migrer progressivement la population de la version N à la version N+1. Cela permet de tester progressivement notre nouvelle version qui pourrait par exemple rapidement montrer des soucis de performance.

Tests de performances ?

Avant de router vos utilisateurs vers le Green, il est possible de réaliser des tests de performances des plateformes.

Vous pourrez par exemple lancer Apache Bench automatiquement et catcher les résultats de celui-ci afin de savoir si votre site sera capable de tenir la charge. Voici un exemple simple d'utilisation possible sur votre environnement de dev :

```
ab -n 10000 -c 1000 http://localhost/
```

Vous pourrez également utiliser un APM (Application Performance Management) en production comme New Relic en Saas qui pourra vous alerter si certaines choses sur votre production ne fonctionnent pas ; un rollback automatique est envisageable.

CONCLUSION

Il me faudrait des centaines de pages pour faire le tour de toutes les possibilités de tests réalisables dans l'univers DevOps mais cet article vous permet cependant de vous familiariser avec ces notions.

Le **test de performance** en continu dans un environnement agile

• Henrik Rexed
Ingénieur Performance chez **Neotys**

Avec le mouvement de transformation digitale, nous assistons à une véritable évolution dans le monde du numérique : toute entreprise devient un éditeur logiciel. Cette transformation est impulsée par la pression concurrentielle à laquelle sont soumises les entreprises et qui les poussent à accélérer le lancement de nouveaux services digitaux toujours plus innovants.

Une étude Forrester prédit que d'ici 2020, 47% de l'ensemble des ventes (mondiales) sera influencé par le digital ou se fera à travers l'utilisation d'objets connectés. Et quand on parle de services digitaux Web ou mobiles, l'expérience utilisateur est primordiale pour assurer l'adoption et le succès des initiatives. Dans ce contexte il est primordial pour une entreprise de valider la robustesse et la rapidité de ses applications en situation de charge réaliste.

La rapidité de la livraison logicielle est dimension clé de l'avantage concurrentiel sur un marché. Ce besoin de produire plus vite de nouveaux services ou des nouvelles fonctionnalités pousse les équipes à adopter de nouvelles méthodes de développement Agile et Devops.

Pour maintenir un rythme rapide d'innovation, il est important que le test de performance ne devienne pas un goulot d'étranglement. Le test de performance reste une discipline complexe qui ne peut pas être totalement automatisée. Pour répondre à la fois aux contraintes de rapidité et aux contraintes de qualité, les méthodes de test de performance changent, et c'est le métier du test de performance qui évolue vers l'ingénierie de la performance.

Chaque entreprise n'est peut-être pas encore cent pour cent Agile ou Devops ; mais toutes adoptent massivement ces nouveaux concepts pour leurs projets de développement existants ou pour leurs nouveaux projets. Lorsque vous commencez à devenir plus « agile », les développeurs sont capables de générer des codes à un rythme plus rapide. Et nombreux sont les testeurs qui ont des difficultés à se maintenir au même rythme. De plus, les testeurs, au sein des équipes agiles, sont en charge des tests automatisés, des tests unitaires, des tests de non régression, mais aussi des tests de charge et de performance. Dans ce contexte, il faut être capable de maintenir la rapidité de développement tout en répondant aux attentes élevées de qualité.

Le test de performance dans un environnement agile est bien plus puissant que dans un environnement de développement standard dans la mesure où il est possible :

- D'éviter l'identification tardive de problème de performance : lorsque les tests de charge et de performance sont repoussés à la fin du cycle de développement, les développeurs n'ont que très peu de temps, voire pas le temps du tout, d'effectuer des modifications. Cela peut obliger les équipes à repousser les dates de lancement et retarder la sortie de nouvelles fonctionnalités souhaitées par les clients.
- D'effectuer des modifications plus tôt, ce qui signifie quand elles sont les moins coûteuses : en incluant les tests de charge et de performance dans un processus de test en intégration continue, les entreprises sont capables d'identifier un problème de performance plus tôt avant que celui-ci ne devienne trop cher à résoudre. C'est particulièrement vrai pour les équipes Agile, pour lesquelles la découverte d'un problème de performance après plusieurs semaines signifie que plusieurs builds sont concernées, ce qui rend l'identification de son origine cauchemardesque.

Les ingénieurs de performance qui travaillent en mode Agile/Devops commencent à disposer d'une expérience suffisamment importante pour faire émerger les meilleures pratiques qui leur permettent de tester efficacement et rapidement.

Inscrivez les SLA de l'application au tableau de tâches

Chaque application a son propre niveau de qualité de service (SLA) minimum. Mais les équipes Agiles sont souvent plus concentrées sur la conception de modules et de fonctionnalités que sur l'optimisation de la performance applicative. Les récits utilisateurs sont généralement rédigés d'un point de vue fonctionnel, sans préciser les critères de performance de l'application. La mesure de performance doit être inscrite clairement sur le tableau de tâches pour que l'équipe y prête vraiment attention.

Travaillez avec les développeurs pour anticiper les modifications

Un des avantages du test en environnement Agile est d'être informé des mises à jour des tâches de développement lors des réunions quotidiennes.

Afin de tirer le meilleur parti de ce niveau de collaboration, les testeurs doivent constamment s'interroger sur la manière dont les récits utilisateurs en cours de développement devront être testés. De nouveaux tests en charge seront-ils nécessaires ? Le nouveau code va-t-il induire des modifications dans les scénarios des tests actuels ? Puis-je m'en tirer avec une simple adaptation des scénarios de tests actuels sans risque pour la suite ? La plupart du temps, il s'agit de modifications mineures, et les testeurs peuvent garder une longueur d'avance s'ils sont constamment et fortement impliqués.

Intégrez les tests en charge à votre serveur de build

Même si vous n'avez pas encore complètement fait le pari de l'Agilité, vous avez sans doute un serveur de builds (par exemple Jenkins) qui déclenche des tests automatisés, des tests unitaires, des « smoke tests », des tests de non-régression, etc.

De la même manière que les objectifs de performance doivent être ajoutés au tableau de tâches, chaque build doit être accompagnée de tests de performance. Cela peut être aussi simple que de définir le lancement automatique sur le serveur de builds pour démarrer le test. Mais en fonction de la sophistication de l'intégration, cela peut aussi inclure l'affichage de résultats de test dans l'outil de build. Idéalement, la personne qui lance la build doit

pouvoir consulter immédiatement les résultats des tests et connaître les modifications apportées à la build pour les corriger en cas de problème de performance.

Les spécificités du test de performance d'applications mobiles

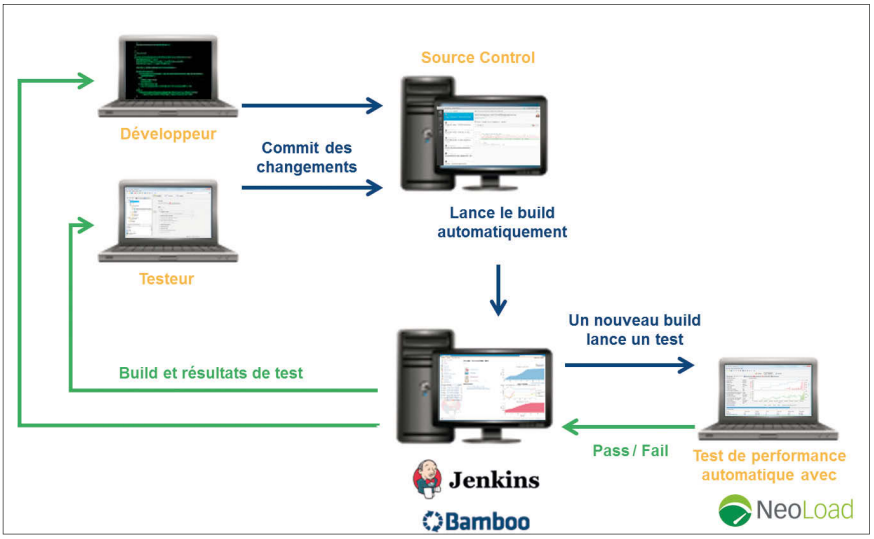
Qu'il s'agisse d'une application mobile native, ou d'une application Web utilisée depuis un terminal mobile, le test de performance mobile comporte trois spécificités principales :

- 1 La simulation des réseaux sans fil. Avec les protocoles 4/5G, un terminal mobile bénéficie d'une connexion Internet plus lente et moins régulière qu'un terminal fixe. Les conséquences sur les temps de réponse du terminal et sur le serveur lui-même doivent être prises en compte au moment où vous configurez vos tests et quand vous analysez les résultats.
- 2 L'enregistrement sur un terminal mobile. Une application mobile est conçue pour fonctionner sur un terminal mobile, ce qui peut représenter une difficulté pour enregistrer un scénario de test, surtout dans le cas d'une application HTTPS sécurisée.
- 3 Les multiples terminaux mobiles. Le grand nombre de terminaux mobiles sur le marché a contraint les concepteurs d'applications Web à adapter leur contenu aux limitations des plateformes clientes, ce qui complique l'enregistrement et le rejeu de tests. Chaque terminal mobile a des spécifications (processeur, mémoire), une taille de cache et une manière de gérer les requêtes différentes.

Les conditions réseau et les temps de réponse

La bande passante est directement en rapport avec le temps de téléchargement de données depuis le serveur. Plus la bande passante est basse, plus le temps de réponse est élevé. Un serveur peut fournir de bons temps de réponse à des utilisateurs sédentaires dotés d'une ligne DSL ou d'un autre service de haut débit, mais le service peut paraître désastreux à des utilisateurs nomades avec une bande passante médiocre.

De plus, les réseaux mobiles ont une latence élevée comparée au WIFI ou au LAN. Comme la latence est un délai ajouté à chaque requête et que les pages Web sont composées de plusieurs sous-requêtes, le temps requis pour charger une page Web sur un appareil mobile dépend beaucoup de la latence. Il est important de valider vos niveaux de qualité de service



(Service Level Agreements) et vos objectifs de performance avec des tests qui utilisent les mêmes conditions de réseau que vos utilisateurs pour éviter d'avoir des résultats de tests trompeurs. Vos tests doivent permettre de simuler les conditions réseau. Cela consiste à ralentir artificiellement le trafic pendant un test pour simuler une connexion plus lente et ajouter de la latence et de la perte de paquets.

Intégration Continue + builds quotidiennes + test en charge de fin de sprint

Les différences entre les builds d'Intégration Continue, les builds quotidiennes, et les builds générées en fin de sprint peuvent être énormes. Dans le premier cas, il s'agit des modifications relatives à un « commit » sur un serveur de contrôle de version, dans le second cas ce sont toutes les modifications publiées dans une journée, et dans le troisième cas ce sont toutes les modifications exécutées lors d'un sprint. Dans cette optique, il faut ajuster vos tests en charge selon le type de build exécuté.

La meilleure pratique dans ce cas consiste à commencer petit, et en interne. Pour les builds d'Intégration Continue exécutées à chaque fois qu'un développeur publie une modification, les tests doivent être lancés immédiatement pour fournir au développeur des informations sur l'impact de ses modifications sur le système. Envisagez d'exécuter un test de performance simple sur les scénarios les plus courants avec une charge normale, générée par vos propres injecteurs de charges. Pour les builds quotidiennes, intensifiez ce test pour inclure plusieurs scénarios plus complexes et augmenter la charge pour atteindre un volume correspondant à vos périodes de pointe, et vérifiez qu'aucun problème de performance n'est passé au travers de vos tests d'Intégration Continue. À la fin du sprint, il faut faire le test ultime : envisagez alors de générer la charge depuis le Cloud pour analyser le comportement de l'application lorsque les utilisateurs se connectent à travers le pare feu. Vérifiez que chaque SLA de la liste des contraintes est satisfaisant et que chaque récit utilisateur écrit pendant le sprint peut être marqué comme « validé ».

Le développement en mode Agile permet d'augmenter la productivité des équipes ainsi que la qualité des applications qui sont lancées. Lorsque les tests de charge et de performance sont ajoutés à cet environnement, une planification minutieuse doit être mise en place afin de s'assurer que la performance est une priorité dans chaque itération. Afin de s'assurer que vous tirez au mieux avantage de cette combinaison entre les méthodes Agile et le test de performance, il vous est conseillé :

- De vous assurer que les niveaux de qualité de service ont une place de choix sur vos tableaux des tâches de manière à garantir que le code associé à chacune des tâches obtienne de bons résultats avant que la tâche ne soit validée.
- De collaborer avec les développeurs afin d'anticiper le moment où un code va nécessiter de modifier les scénarios des tests de performance.
- De lancer de manière automatique des tests de performance pour chaque nouvelle build et effectuer un suivi de la performance d'une build à une autre.

Langages fonctionnels : des langages et des paradigmes

- Aurore De Amaral, *Développeur Scala*
- Mathieu Dulac, *Développeur Scala*
- Ilja Kempf, *Développeur Scala*
- Fabian Gutierrez, *Développeur*
- Charles Dufour, *Développeur Scala / Scrum master*
- Jonathan Raffre, *Ingénieur système*

Xebia

Depuis l'apparition des appareils de calcul, construits d'abord avec des pièces mécaniques puis avec des circuits électroniques, l'être humain a toujours eu besoin de les programmer afin de pouvoir les utiliser pour différentes tâches. Le besoin de documenter est devenu de plus en plus nécessaire au fur et à mesure que le travail se densifiait et que la tâche se popularisait, obligeant un partage des connaissances.

Une fois les premiers ordinateurs apparus, la programmation évolue pour se transformer en une activité principalement écrite. On s'abstrait de plus en plus du fonctionnement de la machine pour se concentrer sur l'algorithme. La première génération de langages de programmation est née. Mais l'écriture implique un problème supplémentaire : *il faut être autant compréhensible par la machine que pour les autres humains, voir même être lisible par toutes les machines.*

Ces langages ont dû se développer dans un contexte technologique très contraignant, d'abord par les fortes limitations de mémoire de l'époque puis le manque de connaissances concernant la mise au point de langages de programmation (tel que les grammaires formelles ①).

De par la popularisation de l'activité de programmation émerge un besoin de faire face à la complexité grandissante des logiciels de plus en plus conséquents. On définit des paradigmes de travail de plus en plus divergents dans leur approche. Deux d'entre elles vont être présentées dans la suite de ce chapitre qui ont tous deux inspiré de nombreux langages : le paradigme impératif et le paradigme fonctionnel.

Le paradigme impératif basé sur la machine de Turing

À partir de l'invention de la machine d'Alan Turing ② dans les années 40, un nouveau domaine d'activité est ouvert aux ingénieurs. Ce modèle introduit une machine conceptuelle basée sur deux idées :

- Un état modifiable et infini (en taille),
- Des instructions permettant de changer cet état.

Les langages qui suivent les règles définies par le modèle de Turing sont appelés "impératifs". Le qualificatif "impératif" a été choisi car il est synonyme de commande. Dans le modèle de Turing, la machine est commandée au travers d'*instructions* qui ordonnent les changements d'*état* de cette dernière. Le modèle d'Alan Turing a profondément influencé le travail d'un grand nombre d'ingénieurs, dont John Von Neumann. Ce dernier propose l'architecture d'un ordinateur contenant un CPU et une mémoire centrale. Par ailleurs, Von Neumann et J. Presper Eckert proposent (de manière indépendante) que les instructions d'un langage impératif soient similaires aux instructions d'un ordinateur.

Jusqu'à la fin des années 60, l'activité de programmation évoluera de façon exploratoire. Cette situation provoque alors une discussion initiée par Dijkstra pour éviter de produire du code dit "spaghetti".

Le modèle de Turing est omniprésent mais il est important de noter que d'autres approches sont développées en parallèle, notamment le paradigme fonctionnel.

Le paradigme fonctionnel basé sur le lambda-calcul

Le lambda calcul (noté λ-calcul) est un système formel décrit dans les années 30 par Alonzo Church et qui introduit la notion de *fonction calculable*. À travers le λ-calcul, Church souhaitait proposer un nouveau fondement aux mathématiques, une alternative à la théorie des ensembles, plus simple et orientée autour de cette notion de *fonction*. À l'origine simple, le lambda calcul ne permettait pas d'exprimer toutes les fonctions calculables dont on a besoin en mathématiques, il ne possédait pas la même expressivité que la théorie des ensembles. Avec le temps, il s'enrichira. Des extensions permettant de considérer les types comme des valeurs de première classe seront créées, comme par exemple le *calcul des constructions*, ces extensions appartenant à ce que l'on appelle les *lambda-calculs typés d'ordres supérieurs*. Cette catégorie de lambda calcul, aussi expressive que les machines de Turing, devient alors parfaite candidate pour inspirer un nouveau paradigme de programmation. Dans ce formalisme, tout est fonction. Une fonction se définit principalement à partir de trois concepts ③ :

- **La variable :** notée par exemple *x*.
- **L'application :** on peut appliquer l'expression *A* qui décrit une fonction à une expression *B* qui décrit un argument de la fonction *A* (B peut être par exemple une autre fonction). On obtient alors une nouvelle expression notée *AB*.
- **L'abstraction :** il s'agit d'un mécanisme permettant de "fabriquer" des fonctions. Prenons par exemple une expression *E* (dépendante ou non d'une variable *x*), on peut alors créer une fonction qui à *x* associe *E*, on la note $\lambda x.E$.

À partir de ces simples définitions, Church a défini un ensemble de puissants outils, tels que l'α-conversion, un mécanisme de substitution de variables, et la β-contraction, un mécanisme de réduction des lambda expressions. Très théoriques en apparence, ces mécaniques ont permis de mettre au point des stratégies de traitement de langages (telles que la minification de Javascript).

LE PARADIGME FONCTIONNEL

La fonction, le coeur du paradigme fonctionnel

La programmation fonctionnelle tire son nom de la fonction. Cette dernière est la seule unité, l'unique élément structurant, du paradigme qui décrit tout calcul sous forme d'évaluation d'une fonction.

① *Backus–Naur Form (BNF)*, http://www.cs.man.ac.uk/~pjj/bnf/bnf.html
② https://plato.stanford.edu/entries/turing-machine/
③ http://www.lsv.ens-cachan.fr/~goubault/Lambda/lambda.pdf

Une fonction est une structure composite ④, elle peut être simple comme la fonction identité ⑤ (f(x) = x) ou être la composition récursive d'autres fonctions (f(x) = g ∘ h). Une fonction est une transformation de valeurs d'un domaine d'entrée vers les valeurs d'un domaine de sortie (codomaine). Pour exemple, f(x) = x + 1, est la fonction qui à tout x numérique associe la valeur x + 1. La transformation décrite par la fonction f peut aussi être exprimée par une lambda que l'on notera λx. (x+1) ⑥.
Si beaucoup de langages offrent la possibilité d'exprimer des fonctions, ces dernières n'honorent pas forcément le contrat du paradigme fonctionnel (la décision est donc laissée au développeur). En effet, peu de langages imposent le respect de certaines propriétés pourtant nécessaires à l'approche fonctionnelle ⑦. Ce chapitre détaillera ces propriétés et leurs raisons d'être.

Totale

Toute fonction est définie sur un domaine. Ce domaine représente l'ensemble des valeurs pour lesquelles la fonction est applicable. Si ce domaine d'application est détaillé en mathématique de la façon suivante : f(x) = x + 1, ∀x ∈ R (la fonction f définie pour tout réel) il est aussi exprimable dans beaucoup de langages au travers de *types*. En Scala, cette même fonction f ressemblerait à cela `def f(x: Double): Double = x + 1` (le type `Double` étant un sous-ensemble des réels qu'on admettra suffisant pour cette fonction). La restriction et la description du domaine de la fonction est important car l'application f("a") est un non-sens et doit être empêchée (en mathématiques comme en programmation).
Prenons maintenant un cas moins évident, soit g(x) = 1/x, ∀x ∈ R, x ≠ 0 (la fonction g définie pour tout réel différent de zéro). On souhaite maintenant exprimer la même chose dans notre code, malheureusement le type "réel différent de zéro" n'existe pas, on devra donc se contenter d'une approximation en utilisant par exemple le type `Double`. Or, d'après ce domaine d'application, g(0) est une application valide ; comment traiter ce cas particulier dans la définition de la fonction ? Plusieurs choix sont possibles lors de l'appel de g(0) :

- Lancer une exception ;
- Retourner une valeur "spéciale" (null, undefined, 0, -1 , etc.) ;
- Retourner une valeur éventuelle (explication ci-dessous).

La première approche ne convient pas car la fonction serait partielle, autrement dit, elle ne retourne pas toujours une valeur. En effet, une exception lancée est un effet de bord, ce n'est pas une valeur manipulable au même titre qu'un retour de fonction :

```
def inverse(x: Double): Double = {
  if (x != 0) {
    1 / x
  } else {
    throw new IllegalArgumentException("Can't divide by zero")
  }
}
```

La deuxième approche n'est pas satisfaisante car elle retourne soit une valeur difficilement interprétable par l'appelant (-1, 0) soit une "non-valeur" (null, undefined) à l'origine d'innombrables bugs ⑧ :

```
def inverse(x: Double): Double = {
  if (x != 0) {
    1 / x
  } else {
    -1
  }
}
```

La troisième alternative est celle retenue par le paradigme fonctionnel. Il s'agit de retourner une valeur éventuelle au travers d'un type `Option`, présent en Haskell ⑨, OCaml ⑩, Scala ⑪ ou encore Java 8 ⑫. Le type `Option` enveloppe ou non une valeur et apporte des garanties (quant au bon fonctionnement du programme) à la compilation ce qui n'est pas possible avec les deux approches précédentes.

```
def inverse(x: Double): Option[Double] = {
  if (x != 0) {
    Some(1 / x)
  } else {
    None
  }
}
```

Les fonctions totales fournissent au développeur des garanties concernant leur bon fonctionnement. Comme elles sont définies pour toutes les valeurs satisfaisant leur signature, leur utilisation ne présente aucun risque pour la bonne exécution du programme. Cela permet un raisonnement de plus haut niveau basé sur le contrat (la signature) de la fonction sans devoir se soucier de l'implémentation effective et des cas particuliers de cette dernière.

Pure

Afin de pouvoir raisonner autour d'une fonction, de la réutiliser et/ou de la composer formellement et automatiquement, elle doit être *pure*. Cela signifie qu'elle ne doit dépendre que de ses arguments et en aucun cas faire usage de ressources externes.
Par exemple, admettons la fonction *impure* suivante :

```
def incrementCount(): Int = Vars.count + 1
```

Dans cet exemple, la fonction `incrementCount` accède à la variable globale `count`. Il est possible d'écrire une fonction avec le même comportement qui dépend exclusivement de ses paramètres :

```
def incrementCount(currentCount: Int) = currentCount + 1
```

Afin d'être pure, une fonction doit respecter une seconde contrainte, ne pas avoir *d'effet de bord*. Une action qui se traduit par une interaction *observable* par le monde extérieur (tout ce qui n'est pas dans le scope

④ *Patron composite*, http://www.dofactory.com/net/composite-design-pattern
⑤ http://mathworld.wolfram.com/IdentityFunction.html
⑥ *Alonzo CHURCH, The calculi of lambda-conversion. Annals of Mathematical Studies, Nr 6. vol. 22 ½ x 15 de pp. Princeton University Press et Humphrey Milford, London, 1941*
⑦ *P. ex. Haskell et Idris*
⑧ *The Billion Dollar Mistake*, https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare
⑨ https://hackage.haskell.org/package/base-4.9.1.0/docs/Data-Maybe.html
⑩ https://ocaml.org/learn/tutorials/error_handling.html#Resulttype
⑪ http://www.scala-lang.org/api/2.12.x/scala/Option.html
⑫ https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html

lexical de ladite fonction) est un effet de bord, ainsi la lecture/écriture d'un fichier qui peut être lu/modifié par un autre programme est observable et par conséquent un effet de bord.
Quelques exemples classiques d'effets de bord sont la modification de variables globales, la mutation des paramètres de la fonction ou encore la lecture/écriture sur la sortie standard.
Les fonctions pures permettent de réfléchir en termes d'évaluation : puisqu'une fonction pure retourne un même résultat pour des paramètres donnés, nous pouvons remplacer les appels de fonctions par leurs résultats sans modifier le programme.
Cette propriété se nomme *transparence référentielle* et permet notamment d'accroître le raisonnement autour d'un programme. Il existe un certain nombre de fonctions pouvant difficilement être pures telles que la génération de *nombre aléatoire* ou la récupération de l'*heure courante* qui dépendent respectivement d'une graine et de l'horloge de la machine. On reparlera de ces fonctions de nature à priori *impure* par la suite.

Curryfiée

Nous venons d'apprendre que les fonctions du paradigme fonctionnel sont totales et pures, ce qui permet de raisonner formellement à leur sujet, c'est-à-dire de façon certaine et systématique. On a aussi évoqué le besoin de composabilité et réutilisabilité de nos fonctions. Pour maximiser ces deux derniers points, nous allons introduire le concept de curryfication. La curryfication consiste à transformer une fonction à n paramètres en n fonctions à 1 paramètre.
Un exemple avec une fonction à deux paramètres :

```
def add(a: Int, b: Int): Int = a + b
```

Une fois curryfiée la fonction add devient :

```
def addCurr: Int => Int => Int = a => b => a + b
```

Cette signature de fonction peut être lue comme : "une fonction qui prend un entier qui retourne une fonction qui prend un entier qui retourne un entier". C'est assez acrobatique mais sachez que le résultat de add(3,5) est identique à addCurr(3)(5).
L'avantage des fonctions curryfiées est qu'elles ont toujours la même allure/signature, ce qui autorise leur composition de façon systématique :

```
def add: Int => Int => Int = a => b => a + b
def div: Int => Int => Int = a => b => b / a
def add6div2: Int => Int = add(6).andThen(div(2))
```

Les fonctions curryfiées sont aussi plus flexibles que leurs homologues car elles sont *partiellement applicables* (seule une partie des arguments est fournie) :

```
List(4, 1).map(add(2)) // List(6, 3)
List(4, 1).map(add) // List((b) => 4 + b, (b) => 1 + b)
```

En respectant ces trois propriétés vous serez en capacité d'écrire des fonctions qui laissent transparaître leur entièreté dans leur signature, qui peuvent être réutilisées en toutes circonstances et être composées de façon automatisée.

Tout est expression

L'*expression*, au même titre que l'*instruction* (*statement*), est un élément de syntaxe d'un langage. Voyons par exemple une instruction :

```
if (a < b) { a = b + 1 } else { b = a + 1 }
```

Les opérations effectuées dans les branches sont de simples affectations, et ne résultent pas en une valeur. Il est donc impossible d'écrire :

```
val res = if (a < b) { a = b + 1 } else { b = a + 1 }
```

De nombreux langages impératifs considèrent cette affectation comme syntaxiquement incorrecte. Il s'agit donc d'un ensemble d'instructions décrivant le déroulement d'un programme, en modifiant son état au travers d'affectations. De plus, le paradigme impératif, à base d'instructions, se concentre sur la description du *"Comment"* un traitement doit être effectué. L'exemple de la somme des entiers contenus dans une liste illustre la description du processus de calcul :

```
var s = 0
val tab = [1, 2, 3, 4, 5]

for (i <- tab) {
  s = s + i
}
```

Au contraire, une expression est un élément de syntaxe d'un langage, qui se concentre sur le *"Quoi"* d'un traitement, et dont l'évaluation résulte en une valeur. Il sera donc possible d'écrire :

```
val res = if (a < b) { b + 1 } else { a + 1 }
```

La valeur ***res***, résultat de l'évaluation de l'expression, sera de type numérique. L'exemple du calcul de la somme d'entiers serait :

```
val s = sum([1, 2, 3, 4, 5])
```

On décrit seulement l'intention, mais pas l'implémentation, grâce à des primitives de plus haut niveau.

L'immutabilité

Dans la grande majorité des langages, la durée de vie d'une variable court de l'instant de sa déclaration jusqu'à la fin de l'exécution de son bloc (classe, fonction, méthode, condition ou tout autre bloc de code). Avec un langage impératif, la variable peut être réassignée pendant sa durée de vie, c'est-à-dire qu'à tout instant de l'exécution du programme sa valeur peut être changée. La valeur de la variable est alors définie en fonction de l'ordre des instructions. C'est ce qu'on appelle une variable *mutable*. La mutabilité des variables possède de nombreux points négatifs, parmi eux :

- La difficulté de raisonner à travers les changements d'état, à la lecture du code. En effet, les instructions opérant sur une même variable au travers d'effets de bord sont couplées temporellement entre elles.
- La difficulté d'orchestrer l'accès à une variable mutable lorsqu'elle est partagée entre plusieurs files d'exécution (threads).
- La possibilité de déclarer une variable sans l'initialiser (l'initialisation peut se faire dans un second temps). On note que certains langages forcent l'initialisation de toute variable, même mutable.

À l'inverse, une variable peut être immuable, et c'est l'une des caractéristiques du paradigme fonctionnel. Une variable immuable est une variable dont la valeur restera inchangée de son initialisation (la déclaration et l'initialisation se font en une seule fois) jusqu'à la fin de sa durée de vie. Cette caractéristique peut aussi se retrouver dans certains langages impératifs (Java et ses variables final par exemple).
Les variables immuables permettent aux développeurs de raisonner beaucoup plus facilement sur leur code, et de ne plus se soucier de la notion d'ordre d'exécution.

LES OUTILS DES LANGAGES FONCTIONNELS

Ce chapitre présente quelques fonctionnalités fréquemment présentes dans les langages fonctionnels. Si ces outils ne sont pas fondamentalement *fonctionnels,* au moins une partie émerge naturellement autour de la fonction *totale* et *pure*.

La typeclass

Typiquement, les utilisateurs d'un langage sont confrontés aux situations dans lesquelles le code doit être extensible. Par exemple, le code en Scala pour trier un ensemble de points par rapport à sa distance ⑬ vers un point d'origine peut ressembler à :

```
def distance(x: List[Double])(y: List[Double]) : Double = {
  // distance euclidienne
}

val origin = List(0.0, 0.0, 0.0)
val elements = List(
    List(1.0, 10.0, 20.0),
    List(5.0, 3.0, 5.0),
    List(1.0, 1.0, 1.0))
 .sortWith((a,b) => distance(origin)(a) < distance(origin)(b))
```

Imaginons que l'on veuille réutiliser la même fonctionnalité de tri (sortWith) mais avec d'autres types de données, pour exemple avec des LocalDateTime. Considérons rapidement quelques alternatives :

- Utiliser un type commun entre List[Double] et le type souhaité ; dans le cas de LocalDateTime cela n'est pas possible car c'est une classe fermée du SDK.
- Implémenter le patron de conception Adaptateur ⑭ et donc, construire une classe qui fasse la liaison entre les deux types à chaque fois.
- La Typeclass.

La Typeclass est un ensemble de fonctions imposées pour identifier un type T_1 comme un membre du type T_2. Mais, à la différence des interfaces (comme ceux de Java), il n'est pas nécessaire de créer un lien hiérarchique entre les deux types. Cela permet de découpler un type de son implémentation d'une typeclass en plus de pouvoir fournir des implémentations de typeclass de façon rétroactive. C'est ce que nous allons faire dans l'exemple ci-dessous qui agit sur les types Double et LocalDateTime tous deux tirés de la librairie standard Scala.
Afin de créer la Typeclass, il faut généraliser le calcul de la distance. Nous représentons les opérations de notre Typeclass au travers d'un trait, qui dans ce contexte est identique à une interface :

```
trait DistanceLike[T] {
  def distance(x: T)(y: T): Double
}
```

Ensuite, nous implémentons la Typeclass sur la distance entres instances de List[Double]:

```
implicit object DistanceLikeLocalDateTime extends
  DistanceLike[LocalDateTime] {
   def distance(x: LocalDateTime)(y: LocalDateTime): Double = {
     // distance en jour entre deux dates
   }
}
```

Remplaçons ensuite l'implémentation initiale de la fonction de comparaison par le code suivant applicable à tous les types disposant d'une instance de la typeclass DistanceLike. La définition de "tri ascendant" est donc définie une unique fois tout en étant découplée des types ciblés.

```
import DistanceLike._
def ascendingOrder[T](origin: T)(x: T, y: T)(implicit d: DistanceLike[T]) =
  d.distance(origin)(x) < d.distance(origin)(y)
```

Vous remarquerez dans la signature ci-dessus un argument particulier avec le modifier implicit à destination du compilateur. Au moment de la compilation, la bonne implémentation de la distance est choisie par le compilateur pour un type donné T (List[Double], String, LocalDateTime, etc). Si l'implémentation pour un type n'existe pas, le compilateur empêche la compilation.
Mais, pourquoi fait-on tout cela ? Parce qu'ajouter des implémentations pour d'autres types est très simple : il suffit de créer une autre implémentation de la Typeclass DistanceLike pour le type souhaité. Finalement, la définition de la distance pour le type LocalDateTime peut ressembler à :

```
implicit object DistanceLikeLocalDateTime extends
  DistanceLike[LocalDateTime] {
   def distance(x: LocalDateTime)(y: LocalDateTime): Double = {
     Math.abs(ChronoUnit.DAYS.between(x, y))
   }
}
```

Finalement, la logique de tri ascendant, c'est-à-dire la fonction ascendingOrder peut être utilisée pour tous les types capables de fournir une instance de la typeclass DistanceLike.

```
val originDate = LocalDateTime.of(2013, DECEMBER, 21, 10, 25)
val sortedDates = List(
  LocalDateTime.of(2014, JULY, 26, 17, 0),
  LocalDateTime.of(1985, APRIL, 11, 5, 25),
  LocalDateTime.of(2011, AUGUST, 6, 21, 13))
  .sortWith(ascendingOrder(originDate))

val originPoint = List(0.0, 0.0, 0.0)
val sortedPoints = List(
  List(1.0, 10.0, 20.0),
  List(5.0, 3.0, 5.0),
  List(1.0, 1.0, 1.0))
  .sortWith(ascendingOrder(originPoint))
```

⑬ https://en.wikipedia.org/wiki/Euclidean_distance
⑭ https://fr.wikibooks.org/wiki/Patrons_de_conception/Adaptateur

Les types de données algébriques

Un type de données algébriques (noté ADT par la suite) est un type de données composites, il est l'agrégation de plusieurs types simples ⑮.

Type produit

Le type produit de deux types A et B est, dans la théorie des ensembles, le produit cartésien, soit A x B. D'un point de vue programmation, cela correspond communément au couple (tuple ou n-uplet pour n types). Il représente ainsi un type composé de plusieurs types.

```
case class Product(s: String, i: Int)
val p = Product("réponse", 42)
```

Type somme

Le type somme de deux types A et B correspond, lui, à l'union disjointe de la théorie des ensembles. Il représente une valeur étant soit du type A, soit du type B.

```
sealed trait Color
case object Blue extends Color
case object Red extends Color
```

Un ADT est l'alliance des types produit et somme, ainsi que de la récursivité. Il est donc une combinaison arbitraire de sommes et de produits et peut se définir avec lui-même. Un bon exemple d'ADT utilisant ces 3 notions la structure d'arbre binaire :

```
sealed trait Tree
case object Empty extends Tree
case class Node(v: Int, left: Tree, right: Tree) extends Tree
```

On définit ainsi un arbre, celui-ci étant soit vide, soit un noeud contenant une valeur de type entier et deux fils de type arbre. Une feuille est donc un noeud dont les deux fils sont vides.
Les ADT sont de puissants types car ils peuvent être facilement décomposés et explorés au travers du pattern matching.

Le filtrage par motif

Le filtrage par motif (ou pattern matching en anglais) est une technique utilisée pour vérifier la valeur d'une expression. Sa caractéristique la plus importante est sa capacité de décomposer un ADT.
Voici un exemple en Scala, qui reprend l'exemple d'arbre discuté dans la partie précédente. Nous analysons la structure de l'arbre (*pattern matching*) pour distinguer le cas d'un arbre vide et d'un arbre non vide. Cela nous permet par exemple d'additionner les valeurs des noeuds de l'arbre.

```
def sum(tree: Tree): Int = {
 tree match {
  case Empty => 0
  case Node(value, l, r) => value + sum(l) + sum(r)
 }
}
```

⑮ https://fr.wikipedia.org/wiki/Type_alg%C3%A9brique_de_donn%C3%A9es
⑯ http://blog.higher-order.com/blog/2009/08/21/structural-pattern-matching-in-java/
⑰ https://en.wikipedia.org/wiki/Visitor_pattern

Le filtrage par motif est une fonctionnalité très répandue parmi les langages fonctionnels. Peu de langages à faible inspiration fonctionnelle implémentent ce concept, comme en Java, où il n'est pas présent ⑯.
Si le code ci-dessus peut ressembler à un simple switch case, il n'en est rien. La technique du filtrage par motif permet d'éviter le cast nécessaire lors d'un switch et surtout nous garantit que toute valeur sera reconnue par au moins un des motifs.
Un moyen d'obtenir une fonctionnalité similaire dans un langage objet est le design pattern Visitor ⑰. Ce dernier garantit que l'ensemble des cas de l'ADT sera couvert mais son défaut est d'être verbeux à mettre en place et les types en question (Tree, Empty, Node) sont pollués par le code du Visitor. Ce couplage est gênant surtout si notre ADT fait l'objet de nombreux traitements différents.
Dernier point, mais non des moindres : le filtrage par motif peut servir pour décomposer des structures imbriquées là où le visitor pattern est limité au premier niveau. Un exemple de filtrage par motif complexe :

```
tree match {
 case Node(55, Empty, Node(v1, Node(v2, Empty, Empty), Empty))
  => true
 case _ => false
}
```

Monade

Nous allons introduire un concept abondamment présent en programmation fonctionnelle, celui de la Monade. C'est une notion qu'il n'est pas facile d'appréhender car c'est une abstraction qui tire ses racines des mathématiques et est par conséquent, pour la majorité, non intuitivement compréhensible.
Sachez qu'un *type monadique* est un type qui cherche à embellir/augmenter/envelopper un (et un seul) autre type ; le type résultant a ainsi une sémantique supplémentaire. Nous allons introduire l'abstraction au travers de deux exemples concrets en mettant en évidence les éléments faisant partie intégrante de l'abstraction.

Option

Le type Option encapsule une valeur éventuelle. Il est utilisé comme type de retour d'une fonction dans le cas où l'aptitude de cette dernière à fournir une valeur est incertaine.
Un exemple concret est le calcul de la racine carrée. Si nous appelons la fonction avec un nombre négatif, le résultat ne peut pas être calculé. Présent dans les librairies standards de quasiment tous les langages (Optional pour Java, Maybe pour Haskell, Option pour Scala/OcamlI/F#), le type Option est parfois même intégré à la syntaxe du langage (Swift, Kotlin, C#).
Observons les fonctionnalités clés du type Optional :

- Un constructeur **None** pour exprimer l'absence de valeur.
- Un constructeur **Some(t:T)** pour "élever" une valeur simple vers une valeur optionnelle.
- Une fonction **map** pour manipuler la valeur optionnelle.
- Une fonction **flatMap** pour aplatir/fusionner deux valeurs optionnelles en une unique valeur optionnelle.

Un exemple d'utilisation du type Option et de flatMap :

```
def computePrice(qty: Int, price: Int): Option[Int] = {
 if (qty >= 0)
  Some(qty * price)
```

```
  else
    None
}

val maybeQty: Option[Int] = Some(5)
maybeQty.flatMap(computePrice) // Some(50)
```

Future

Tout comme le type Option, le type Future est très répandu (ComputableFuture en Java, Task en C#, Promise en Javascript, Future en Scala/Haskell). C'est un outil puissant pour gérer la complexité liée à l'asynchronisme.
Les principales fonctionnalités du type Future sont :

- Une fonction pour compléter la future (en cas de succès ou d'erreur du traitement asynchrone).
- Un constructeur **Success(t:T)** pour envelopper une valeur simple dans une Future.
- Une fonction **map** pour manipuler la valeur éventuelle.
- Une fonction **flatMap** pour combiner deux Futures.

Un exemple d'utilisation du type Future et de flatMap :

```
def fetchUser(id: String): Future[User] =  // Retrieve user
def saveUser(user: User): Future[User] =  // Persist user

fetchUser("user-id")
  .map(user => user.copy(name = "new-name"))
  .flatMap(saveUser)
```

La partie commune

Vous remarquerez que l'Option et la Future partagent certaines propriétés :

- Enveloppent un type (Option[**T**], Future[**T**]).
- Permettent d'élever une valeur simple vers un *contexte monadique* (**Some**, **Success**).
- Possèdent une fonction **flatMap** pour combiner des monades.
- Respectent certaines lois qu'on ignore pour ce dossier (identité à droite, identité à gauche et associativité).

L'objectif

Le but de la Monade est le même que celui de toutes les autres abstractions : le partage de code. Il existe de nombreux patterns basés sur cette abstraction tels que *Sequence*, *Traverse*, *Monad combinators* ou encore *Foldable*, etc. Ces codes sont capables d'opérer sur toutes les Monades y compris les vôtres.
Le partage de code basé sur les Monades n'est cependant possible que dans certains langages possédant un système de type puissant avec pour fonctionnalité clé le support des types d'ordre supérieur.

Type d'ordre supérieur

Nous avons précédemment évoqué plusieurs types intéressants tels que Option ou Future communément présents dans les langages ou librairies fonctionnelles mais sachez qu'il y en a beaucoup plus tels que List, Either, State, etc. Malgré des sémantiques très différentes, ils peuvent satisfaire l'interface de la Monade. Comme toujours en programmation, nous allons partager un maximum de code entre différents types qui partagent une abstraction commune.
Prenons l'exemple suivant : pour toute Monade de type Monade[Int], nous souhaitons incrémenter ce qu'elle referme. Naïvement, nous pourrions commencer par écrire les fonctions suivantes en Scala :

```
def add(opt: Option[Int], value: Int): Option[Int] =
  opt.map(_ + value)

def add(list: List[Int], value: Int): List[Int] =
  list.map(_ + value)

def add(list: Future[Int], value: Int): Future[Int] =
  list.map(_ + value)
```

Nous remarquons que ces fonctions sont identiques mis à part le type du premier argument. Afin de partager ce code nous souhaitons donc paramétriser la fonction add. Cependant dans le cas présent, la paramétrisation ne se fait pas sur un type dit de premier-ordre (ex : add[T]) mais un type dit d'ordre supérieur (ex : add[M[Int]]). Si vous utilisez C# ou Java vous ne pourrez tirer profit de l'abstraction commune (la Monade), cependant, Scala, Haskell ou Ocaml supportent les types d'ordre supérieur ce qui va nous permettre de retravailler le code précédent afin de maximiser le partage de code. La fonction paramétrée ressemblera à cela :

```
def add[M[_]](f: M[Int], value: Int)(m: Monad[M]): M[Int] =
  m.map(f)(_ + value)
```

Ne vous souciez pas trop du troisième argument, c'est une instance qui vous sera usuellement fournie par le langage ou la libraire et qui atteste que le type (Option, Future et autres) satisfait bel et bien l'interface Monade.
Cette fonction *paramétrique* (ou plus communément *générique*) peut être utilisée pour tout type qui implémente l'abstraction de la monade tels que List ou Option.

```
add(List(5, 8), 2)(listMonad) // List(7, 10)
add(Option(3), 1)(optionMonad) // Option(4)
```

Le support des types d'ordre supérieur est donc essentiel si l'on veut aller plus loin dans l'abstraction et la généralisation de types. L'objectif étant, bien sûr, le partage de code.
Sachez aussi qu'il existe des outils pour faciliter voire rendre transparente l'utilisation de monades, ces derniers dépendent entre autres du support de types d'ordre supérieur par le langage hôte. C'est une des raisons qui fait que l'utilisation de monades est plus abondante et aisée en Scala qu'en Java par exemple.

LE FONCTIONNEL DANS LA PRATIQUE

Effets de bord

La fonction du paradigme fonctionnel permet d'exprimer tout calcul au même titre que le paradigme impératif. Cependant, si une fonction pure peut exprimer un calcul, il n'y a aucun moyen d'obtenir le résultat car l'effet de bord (sortie standard, communication http, persistance, etc.) est interdit. La grande majorité des langages, même ceux fortement inspirés par le paradigme fonctionnel, ignorent simplement la contrainte de pureté. En Scala ou Ocaml rien ne vous contraint à écrire du code pur, ainsi, vous pouvez écrire sur la console ou persister vos données sans boilerplate ou syntaxe particulière.
D'autres langages comme Haskell n'autorisent les effets de bord que dans

certains contextes, à défaut d'être dans ce contexte vous pouvez être certain de manipuler du code pur.
Au delà, vous trouverez des langages formels ou des assistants de preuves qui ne vous serviront pas dans votre activité de programmation.
Nous recommandons de limiter le code à effet de bord aux couches extrêmes de votre application, c'est-à-dire, les parties qui interagissent avec le monde extérieur. Cela exige bien évidemment une architecture par responsabilité de votre code, ce qui est une bonne chose en tout contexte.

Les performances

Une des questions les plus régulièrement posées aux langages de programmation est leur utilisation à grande échelle dans l'industrie. Nous avons souvent l'habitude de comparer les performances entre les différents algorithmes et leur implémentation dans les langages.
Plusieurs expérimentations scientifiques ont tenté d'évaluer les performances des langages fonctionnels [18]. Une de ces expérimentations, sur plusieurs langages fonctionnels se portait sur un algorithme de calcul d'une structure de molécule et sa nucléotide.
Les compilateurs supportant les fonctions d'ordre supérieur ne sont pas un critère déterminant sur la rapidité de l'exécution. Globalement, une bonne partie des langages et leur compilateur associé approche grandement des vitesses d'exécutions en C (mention notable pour le langage SISAL et Haskell).
D'autres recherches [19] sur le fonctionnement de la mémoire en Java et Scala, deux langages utilisant la JVM, montrent un certain nombre de disparités, notamment dans le nombre conséquent d'objets créés dans les programmes Scala. D'autres comportements de perte de performance ont été soulignés pour le langage Scala dans l'article. La dernière version de Scala, 2.12.0 à l'écriture de cet article, a augmenté l'interopérabilité [20] avec Java 8, ce qui peut augmenter les performances du langage fonctionnel.
Enfin, nous pouvons aussi citer le Great Benchmark Game [21], qui teste la rapidité des langages sur un même ordinateur sur différents algorithmes (recherche d'arbres binaires, calcul d'un mandelbrot, nombre de k-nucléotides, etc.) et qui montre, dans beaucoup d'exemples, que la rapidité du programme C dépasse celle des langages fonctionnels, mais que la version en Erlang ou Scala n'est pas plus lente que leur homologue en Java.
Concrètement, il peut y avoir un intérêt, même pour les performances, d'utiliser des langages fonctionnels, dans une utilisation multi-threadée notamment.

APPLICATIONS DANS L'INDUSTRIE

Les écosystèmes fonctionnels

Moins répandus que d'autres paradigmes dans l'industrie, les langages fonctionnels ont néanmoins su trouver leur place, parfois dans des marchés bien spécifiques.

Scala

Scala est aujourd'hui un des langages fonctionnels les plus présents. S'exécutant sur la JVM, il bénéficie d'une excellente interopérabilité avec Java, ce qui a contribué à son succès. Concrètement, cela signifie qu'une bibliothèque Java peut être utilisée en Scala et vice versa [22].
Mais Scala a également su se démarquer de Java en développant son propre écosystème. Dans le monde du Web, Play framework en est un remarquable exemple. Pour la création d'architectures distribuées, concurrentes et réactives, Lightbend (anciennement Typesafe) a joué un grand rôle en poussant Akka sur le devant de la scène et plus récemment, Lagom [23].
Enfin, nous n'oublierons pas de citer Apache Spark, framework de calcul distribué désormais incontournable en data science.

Ocaml

Plus discret, Ocaml est cependant aujourd'hui bien ancré dans certains domaines spécifiques, tels que la preuve formelle (p. ex. COQ un assistant de preuve) ou l'écriture de compilateurs. Poussé par l'INRIA, l'écosystème est assez peu développé. Nous pouvons néanmoins citer OPAM, le gestionnaire de paquets Ocaml, qui permet de gérer facilement ses dépendances, Ocsigen, un environnement complet pour le développement Web ou encore Core, une extension de la bibliothèque standard développée par Jane Street.
De manière générale, les champs d'application d'Ocaml utilisent peu de frameworks. En effet, pour l'écriture de compilateurs, Ocaml dispose de deux puissants outils, ocamllex et ocamlyacc pour le lexing et le parsing de fichiers sources. De plus, le design du langage, à travers la création et la manipulation de types, permet de rendre la tâche plus aisée.
Ocaml a également la particularité d'être assez bas niveau et de bien s'interfacer avec le langage C, cela lui permettant d'avoir accès à un ensemble de bibliothèques écrites en C.

Haskell

Haskell est un langage qui s'offre un regain d'intérêt depuis la récente popularité des langages fonctionnels. C'est un excellent choix pour apprendre la programmation fonctionnelle pure et son écosystème, et est aussi mature pour une utilisation en production. De nombreux compilateurs ont été écrits en Haskell, tels que Elm [24], Purescript [25] ou encore Idris [26].
La syntaxe du langage est très concise et expressive. Pour en apprécier une introduction, nous vous conseillons la lecture du très bon livre Learn You A Haskell For Great Good [27].
L'écosystème d'Haskell est essentiellement constitué d'un système de build, Cabal, d'un gestionnaire de packages, Stack [28], ainsi qu'un repository central de packages nommé Hackage [29].
Les framework Web Snap [30] et Yesod [31] vous permettront de développer rapidement des applications Web.
Le moteur de recherche Hoogle [32] permet de facilement retrouver le nom d'une fonction à partir de sa signature, et vice versa.

[18] *Pieter H. Hartel et al., Benchmarking Implementations of Functional Languages with "Pseudoknot", a Float-Intensive Benchmark*
[19] *Andreas Sewe et al., new Scala() instanceof Java, A Comparison of the Memory Behaviour of Java and Scala Programs*
[20] http://www.scala-lang.org/news/2.12.0
[21] http://benchmarksgame.alioth.debian.org/
[22] http://www.scala-lang.org/old/faq/4
[23] https://www.lightbend.com/lagom
[24] http://elm-lang.org/
[25] http://www.purescript.org/
[26] http://www.idris-lang.org/
[27] http://learnyouahaskell.com/
[28] https://docs.haskellstack.org/en/stable/README/
[29] https://hackage.haskell.org/
[30] http://snapframework.com/
[31] http://www.yesodweb.com/
[32] https://www.haskell.org/hoogle/

Pour finir, sachez que Gabriel Gonzalez a rédigé un document sur l'état de l'art de l'écosystème d'Haskell ㉝, mentionnant pléthore d'outils.

Retour d'expérience

Maintenant que nous avons vu ce qu'est un langage fonctionnel et les écosystèmes existants à l'heure actuelle, le tout est de savoir s'ils sont utilisés aujourd'hui dans les entreprises en France.

En juin de l'année dernière, Nicolas Martignole avait fait un sondage pour connaître les entreprises qui utilisaient Scala au quotidien ㉞.

Nous pouvons citer comme exemple la solution BI d'Ingenico, Merchant Service Hub, développée en Scala ㉟ sur laquelle travaille une partie des auteurs, la société Criteo sur sa plateforme big data, l'application reporting de Captain Dash, et bien-sûr l'équipe R&D de Twitter ㊱.

D'autres langages fonctionnels entrent dans le domaine industriel, comme le service de détection de spams de Facebook, Sigma, en Haskell ㊲ mais aussi le système de trading de Jane Street Capital en Ocaml ㊳. •

Comparaison des langages

	Haskell	Ocaml	F#	Scala	Java	JavaScript
Maturité	+++	+++	+	++	++++	+++
Accessibilité	+	++	++	+++	+++	++++
Communauté	++	+	++	+++	++++	++++
Présence dans l'industrie	+	+	+	++	++++	++++
Paradigmes	Fonctionnel	Fonctionnel Impératif	Fonctionnel Objet Impératif	Fonctionnel Objet Impératif	Fonctionnel Objet Impératif	Fonctionnel Objet ㊴ Impératif
Typeclasses	Oui	Oui	Non	Oui	Non	Non
ADT	Oui	Oui	Oui	Oui ㊵	Non	Non
Pattern matching	Oui	Oui	Oui	Oui	Non	Non

1. Maturité : Plus un langage est stable et possède de fonctionnalités, plus celui-ci est mature.
2. Accessibilité : Présence de tutoriels, livres et cours permettant de prendre en main le langage facilement.
3. Communauté : Personnes participant activement au développement du langage et de son écosystème.

㉝ https://github.com/Gabriel439/post-rfc/blob/master/sotu.md
㉞ http://www.touilleur-express.fr/2016/07/08/sondage-scala-dans-les-entreprises-en-france/
㉟ https://www.ingenico.com/press-and-publications/press-releases/all/2016/11/ingenico-to-introduce-its-bi-solution-for-small-and-medium-size-merchants-at-trustech.html
㊱ http://www.journaldunet.com/solutions/cloud-computing/coulisses-techniques-de-twitter/scala.shtml
㊲ http://cufp.org/2015/fighting-spam-with-haskell-at-facebook.html
㊳ https://blogs.janestreet.com/category/ocaml/
㊴ *Plus spécifiquement: "programmation orientée prototype"*
㊵ *Les ADT sont émulés au travers d'une hiérarchie d'objets*

Guides des **cartes makers**, IoT et pour mini-PC

François Tonic

Aujourd'hui des dizaines de cartes, pour prototyper ou créer des mini-PC, sont disponibles. Les makers, utilisateurs et développeurs n'ont que l'embarras du choix. Mais comment choisir ? Quelle carte pour quel usage ? Arduino et Raspberry Pi demeurent les maîtres étalons du marché, mais en réalité, ils sont dépassés par des cartes plus puissantes, moins chères, bref, il ne faut plus se focaliser uniquement sur ces cartes. La rédaction fait le point.

Avertissement : *cette sélection de cartes est forcément subjective même si nous avons essayé d'être le plus ouvert possible et proposer des alternatives malheureusement peu connues en France. Et le marché des boards bouge beaucoup depuis.*

Chaque mois, il sort de nouvelles cartes, la plupart du temps, non disponibles en France, ou avec un décalage de plusieurs mois. Plusieurs cartes, particulièrement prometteuses, ont du mal à trouver les chemins de la France, d'autres apparaissent très timidement. Par exemple, les cartes ODROID commencent seulement à être visibles, en cherchant un peu. Les cartes VoCore ne sont pas trouvables, ni la C.H.I.P., et là, Internet sera votre meilleur ami.

Il n'est pas simple créer des catégories ou des usages types. Certaines cartes sont très généralistes et peuvent s'adapter à tous les usages ou presque. Certaines seront présentes dans toutes les catégories.

Nous avons, arbitrairement, établi, 3 catégories :

- Prototypage, DIY (Faire soi-même), maker ;
- Mini-PC, mini-serveur ;
- IoT finalisé, industrialisation et wearable.

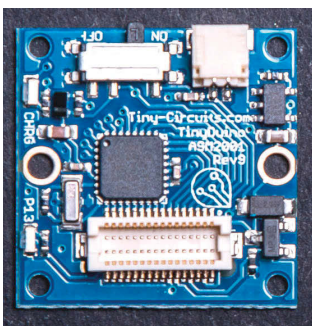
LES CARTES DE PROTOTYPAGE RAPIDE/MAKER

Pour créer rapidement un prototype, un objet pour la maison, le bureau, pour s'amuser, il existe de nombreuses cartes. Les cartes les plus courantes seront les Arduino. Il existe de nombreuses formes d'Arduino, avec des caractéristiques différentes et des tarifs très variés. Les clones existent et à des prix particulièrement attractifs.

En modèle premium, la Raspberry Pi reste la référence. Ces cartes, la Pi et les concurrents, proposent une puissance très supérieure à l'Arduino, mais peuvent aussi servir à créer des projets, des objets nécessitant de la puissance. Dans le même esprit, nous pouvons citer les cartes Gadgeteer ou Beaglebone. Nous aurions pu y mettre les ESP8266, mais ces cartes nécessitent une meilleure maîtrise technique. Comment choisir la bonne carte ? De nombreux critères peuvent être utilisés :

- Le prix ;
- La notion d'open hardware de la carte ;
- La communauté ;
- Les outils et le modèle de développement ;
- Disponibilité en France ;
- Les tutoriels et projets facilement trouvables ;
- La facilité à connecter et à coder les capteurs ;
- Consommation, dimension, poids.

Pour des montages découvertes ou pour des objets du quotidien (automatiser un jardin, une lumière, etc.), dans la plupart des cas, une carte Arduino suffira. Vous pourrez, si besoin, utiliser une carte plus puissante telle qu'une Raspberry Pi en serveur pour récupérer les données ou manipuler les fonctions de votre objet.



tinyduino

arduino (clone)

La gamme Arduino

La gamme Arduino est très large : de la simple Uno à la 101 en passant par la Nano, Pro Mini, MKR1000, Zero, Due, ou encore la Mega et la Yun. Sans oublier toutes les cartes compatibles telles que la Tinyduino, la Gemma, la Flora, LilyPad, etc. Toutes ces cartes ne sont pas forcément adaptées à tous les projets. Les Nano, Pro Mini, Tinyduino sont idéales pour des objets à dimensions réduites. Elles permettent d'optimiser la taille de votre montage électronique, mais vous aurez des contraintes avec des ressources très limitées et un nombre de broches (GPIO) réduit.

	Uno	101	Nano	Mega	Tinyduino	LilyPad
Microcontrôleur/CPU	Atmega328p	Intel Curie	Atmega128 Atmega328p	Atmega2560	Atmega328p	Atmega128v
Vitesse	16 MHz	32 MHz	16 MHz	16 MHz	8 MHz	8 MHz
GPIO analogiques/digitales	6/14	6/14	8/14	16/54	6/14	6/14
Réseau	Non	BLE	Non	Non	Non	Non
Flash	32 ko	196 ko	16 ou 32 ko	256 ko	32 ko	16 ko
USB	USB	USB	Micro	USB	Micro	-
Voltage (volt)	5/12	3,3/12	5/9	5/12	2,7/5,5	2,7/5,5
Programmation	C	C	C	C	C	C
Prix	7-9 *	40	-2 *	-10 *	14,95 $	3 *
Notre avis**	*****	**	****	*****	****	***

** Tarif clone et non de la carte officielle*
*** Un avis est toujours subjectif. La notation sur 5* se base sur notre expérience avec la carte.*

Mais Arduino souffre de plusieurs faiblesses et limitations fortes :

- Une consommation élevée selon la carte sauf à utiliser des boards dédiées wearable comme la LilyPad ;
- Un code fonctionnant en boucle et 1 seul code peut s'exécuter ;
- Une puissance très limitée du microcontrôleur.

Si vous cherchez un modèle de développement de type JavaScript, Python ou Java, mieux vaut chercher vers des cartes de type C.H.I.P.,

Raspberry. Mais l'énorme avantage d'Arduino est un modèle de développement simple et clair : C et Arduino IDE. L'outil supporte des dizaines de boards et des centaines de librairies sont disponibles pour tout et n'importe quoi. Facile à utiliser. Certaines cartes demandent un peu de manipulation pour les installer, mais dans la plupart des cas, c'est une histoire de 3-4 clics de souris. Et le code est en général facilement repris d'une carte à une autre. Parfois, il faut tout de même modifier le code ou changer de librairies.

Il existe des systèmes très simples pour connecter les capteurs. Seeed Studio est le fabricant le plus connu avec sa plateforme Grove. Il suffit de connecter le shield Grove. Les capteurs se branchent en Plug&Play. Tout est intégré et sans soudure ! Grove est un peu cher (un kit complet est vendu env. 40-50), mais vous gagnez en simplicité.

Arduino a l'avantage d'avoir une gamme de boards très large, vous trouverez sans aucun doute votre bonheur. Et, la plupart des modèles sont open source et donc Open Hardware ; cela signifie que les schémas électroniques sont ouverts à tout le monde. Cependant, Arduino n'est pas fait pour passer à un objet en production, éventuellement en petite série, mais avec prudence. Enfin, Arduino, étant open source, il existe de très nombreux clones pour les modèles les plus courants (Uno, nano, mega notamment) et à des tarifs particulièrement agressifs, -2 pour un clone Nano, avec câble USB… La qualité est parfois variable et il faut installer des pilotes dédiés, mais honnêtement, la qualité de ces clones s'est beaucoup améliorée et des projets originaux sortent des studios.

Les autres boards

Arduino n'est pas l'unique possibilité, vous pouvez utiliser bien d'autres boards des plus puissantes telles que les Raspberry Pi à la minuscule ESP8266. Tout dépend de vos envies et exigences de ressources. Pour notre part, si une Arduino suffit pour notre projet, nous choisirons une Arduino. Si vous avez besoin d'un peu plus de puissance ou tout simplement de pouvoir exécuter plusieurs programmes ou utiliser d'autres langages, vous devrez utiliser d'autres plateformes.

	Pi Zero*	C.H.I.P.	ESP8266	Gadgeteer FEZ Reaper
Microcontrôleur/CPU	ARM	ARM	RISC	ARM
Vitesse	1 GHz	1 GHz	80 MHz**	180 MHz
GPIO analogiques/digitales	40 en tout	80 en tout	variable	82 en tout
Réseau	WiFi Bluetooth	Bluetooth	WiFi	Ethernet
Flash	-	4 Go	512 ko à 16 Go**	256 Ko
USB	2x micro	USB	Micro/rien**	USB
Voltage (volt)	3,3/5	3/5	3,3**	-
Programmation	modèle Pi	très ouvert	très ouvert	C#
OS par défaut	Linux	Linux	-	Micro .Net Framework
Prix	10 $	9 $	-2 à 10 $**	40-45
Notre avis	***	***	*****	**

** Nouvelle version de la Pi Zero uniquement*
*** Selon les modèles*

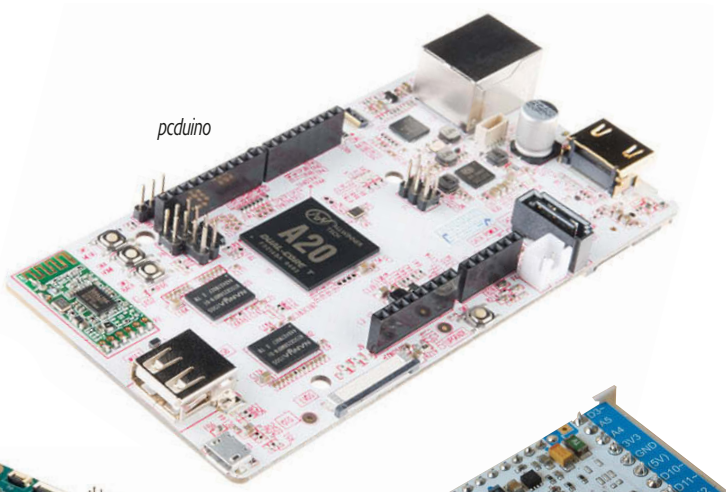
La Pi Zero est une version minimaliste de la Rapsberry Pi. Elle peut être une bonne alternative pour un objet ou un projet un peu complexe. La version la plus récente embarque Wifi et Bluetooth, mais le tarif est à doubler. La C.H.I.P est plus complète et offre un excellent ratio prix/ressources. Elle est encore très jeune, mais elle est déjà une alternative très crédible.

Les ESP8266 sont des nano board WiFi créées par Espressif System. Aujourd'hui, il en existe de très nombreux modèles. Les prix sont particulièrement intéressants. Le domaine de développement est très souple et une multitude de firmwares sont disponibles. Pour nous, il s'agit aujourd'hui de la meilleure alternative aux Arduino et surtout, les ESP sont taillées pour la production en masse. Les défauts des ESP pour un maker sont principalement le manque de GPIO, même si on peut trouver des solutions et un voltage de la carte pouvant être handicapant : 3,3 V. Vous serez capable d'optimiser les dimensions de votre projet et la consommation sera elle aussi largement améliorée. Et surtout, la connectivité WiFi par défaut est un gros avantage.

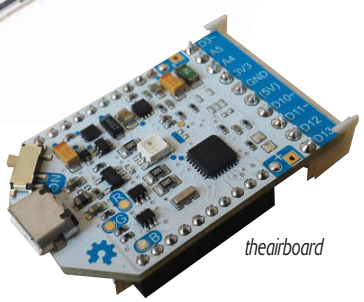
Nous aborderons les ESP un peu plus loin.

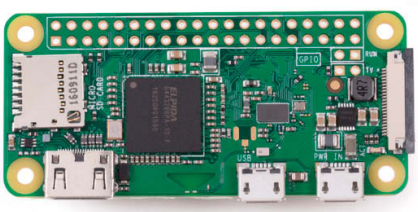
La plateforme gadgeteer tourne sur une édition allégée du framework .Net, dédié à l'embarqué. La carte se développe en C#. Quelques constructeurs ont lancé des cartes telles que GHI Electronics avec la gamme FEZ. Malgré la qualité des cartes, l'écosystème n'a pas pu se développer et le tarif des cartes et des capteurs n'a pas aidé. Dommage.

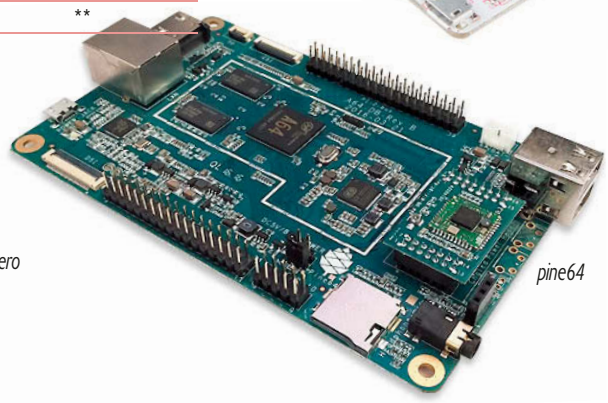
La limifrog est un beau projet français. Cette carte ultracompacte et très légère intégrait 7 capteurs, une batterie LiPo, un écran OLED de grande qualité, 64 Mo de stockage et un modèle de développement très large. La carte est toujours disponible à 75 H.T. Pour une découverte très facile et intuitive des IoT et de l'électronique, nous avions très récemment testé la carte micro:bit, carte anglaise. Certes, elle est plus limitée qu'une Arduino ou une ESP, mais tellement sympathique. Autre kit intuitif, Wio Link. Il s'agit d'une plateforme Plug&Play basée sur une ESP8266. En 5 minutes vous pouvez interagir entre la carte, des composants et une application mobile. La programmation est très ouverte : JavaScript, Java, Python, etc. La carte proprement dite est vendue 14 $.



pcduino


theairboard


Pi zero


pine64

PROJETS AVANCÉS, WEARABLE, IOT INDUSTRIEL, BORNES

Sauf à créer sa propre plateforme matérielle, ce qui est toujours possible et pas forcément aussi cher qu'on le pense, même en série limitée, quand vous souhaitez créer des objets de dimensions réduites, wearables ou un IoT en série ou des bornes interactives, le choix ne manque pas !

	Petite série	Wearable	Borne	Domotique	IoT grande série
Intel Edison	***	***	**	**	*****
Intel Curie	***	***	**	**	*****
Les ESP8266	*****	****	*	***	*****
Raspberry Pi, ODROID, C.H.I.P., etc.	****	*	*****	*****	**
LilyPad, TinyDuino, Xadow, etc.	****	****	**	**	**
AirBoard	****	***	***	****	****

Pour des projets IoT, de bornes ou de domotique, le choix n'est pas forcément très simple. Pour une borne ou au cœur d'un distributeur, les Raspberry Pi et équivalents, sont bien adaptés. Car il faut de la ressource matérielle, un système d'exploitation, une connectivité, sans pour autant alourdir la facture. Même si ces cartes sont assez volumineuses, la taille des bornes permet leur usage. En IoT, au sens large, la taille de l'objet, son interactivité, sa connectivité, sa consommation, son prix seront des critères de choix. Des cartes de type Intel Edison ou Curie sont taillées pour les IoT et pour la phase industrielle, pour de petites séries, pas forcément, à cause des contraintes techniques ni du prix de la carte. Un ESP8266 pourra s'installer à peu près partout, à condition de ne pas surestimer sa puissance. Pour des objets de domotique, des IoT du quotidien, les ESP sont parfaits. Pour des projets et objets nécessitant un peu plus de ressources, par exemple en pins de branchement, des cartes comme la AirBoard sont intéressantes. Un avantage est sa compatibilité avec les infrastructures réseau de type SigFox.

Pour des projets en grande série, vous pouvez aussi créer votre propre carte ou vous tourner vers des fournisseurs industriels tels que NXP.

Pour des projets wearable ou domotiques, AirBoard, Xadow, TinyDuino, LilyPad, ESP, etc. conviennent très bien. Là, tout dépend de vos affinités. Si vous êtes habitué à Arduino et que la plateforme vous suffit, les variantes embarquées sont nombreuses. Au-delà, des plateformes plus puissantes comme ESP ou Onion Omega sont des solutions à considérer. Un des projets les plus récents est l'Omega2 d'Onion. Cette carte, qui peut se comparer facilement à un C.H.I.P., fonctionne sur un processeur MIPS, embarque 64 Mo de RAM, 16 Mo de stockage, de l'USB 2, du WiFi, des GPIO. Comme la WeMos D1 Mini (ESP8266), de nombreuses extensions sont disponibles. L'Omega2 fonctionne sur OpenWRT. Et pour un tarif annoncé particulièrement agressif : 5 $! Si le projet est en phase de production, les livraisons ont pris plusieurs mois de retard. Mais cette carte a un potentiel énorme.

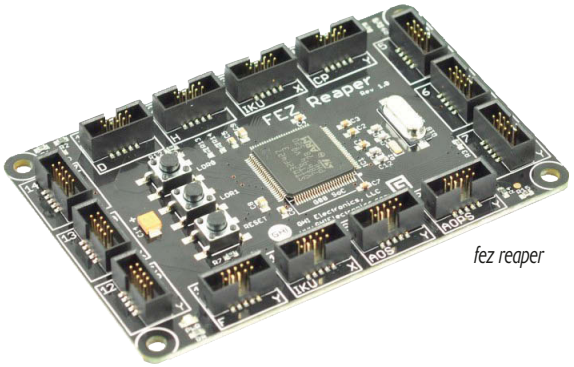
CONSTRUIRE UN MINI OU NANO-PC

On parle beaucoup de mini-PC, de barebone, de fracture numérique, de rétroconsole, de nanoserveur, etc. Le point commun est des cartes de type Raspberry, ODROID, C.H.I.P., pcDuino, etc. Elles coûtent généralement entre 40 et 80 et proposent une puissance suffisante pour beaucoup de fonctions pour une utilisation non professionnelle et hors gaming. Oui beaucoup d'utilisateurs n'ont pas besoin d'un PC ou d'un Mac coûtant 600 ou 700 minimum et dont la puissance sera sous-exploitée. Éventuellement des ordinateurs de type ChromeBook, mais il faut une connexion réseau et le tarif débute à 250 – 290 . Avec des cartes, ayant un excellent ratio prix/puissance, on peut construire un PC à -100 ! Même pour la partie serveur, aujourd'hui, des solutions particulièrement excitantes existent et nous impressionnent par leur puissance et surtout par un tarif défiant toute concurrence !

	Raspberry Pi 3	ODROID-C2	Banana Pro	Pine64
CPU	Cortex A53 1,2 GHz	Cortex A53 2 GHz	Cortex A7 1 GHz	Cortex A53 1,2 GHz
CPU 64 bits	oui	oui	non	oui
GPU	VideoCore IV	Penta Core	Mali400MP2	Mali400MP2
Mémoire vive	1 Go	2 Go	1 Go	1 Go(1)
Stockage	SD	SD eMMC	SD	SD
Port SATA	non	non	oui (v2)	non
Ethernet	10/100	1 Gb	1 Gb	1 Gb(2)
Sans-fil	Bluetooth WiFi	non	WiFi(3)	non(4)
USB	4	4 Mi-USB OTG	2	2
Vidéo	HDMI 1.4	HDMI 2.0(5)	HDMI 1.4	HDMI 1.4
OS supportés	Linux Windows 10 IoT Android	Linux Android	Linux Android	Linux Android
écosystème/ communauté en France	****	*	**	*
Tarif (carte seule)	40-45	env. 59	45-50	19 $

	pcDuino3B	Beaglebone black wireless	C.H.I.P.	VoCore2 Ultimate6
CPU	ARM Cortex A7 1 GHz	ARM Cortex A8 1 GHz + 2 PRU + Neon SIMD	ARMv7 1Ghz	MT7628AN MIPS
CPU 64 bits	Non	Non	Non	oui
GPU	Mali 400 Dual Core	SGX530	Mali 400	aucun
Mémoire vive	1 Go	512 Mo	512 Mo	128 Mo
Stockage	4 Go	4 Go	4 /8 Go	16 Mo
Port SATA	Oui	Non	non	non
Ethernet	1 Gb	Non	non	100 mb
Sans-fil	non	WiFi Bluetooth	WiFi Bluetooth	WiFi
USB	2	1	2	micro USB
Vidéo	HDMI 1.4	HDMI	non	non
OS supportés	Linux Android	Linux Android	Linux	Linux OpenWRT
écosystème/ communauté en France	*	**	*	*
Tarif (carte seule)	60	env. 70	9 $	44,99 $

(1) Pine64 propose de 512 Mo à 2 Go, selon le modèle choisi
(2) Pas sur toutes les versions de la board
(3) Bluetooth en option
(4) Module Wifi + Bluetooth en option, prix : 10,99 $
(5) Attention aux écrans compatibles ou non
(6) Edition complète avec boîtier



fez reaper

Le choix ne manque pas pour créer un vrai PC. Raspberry Pi 3, ou une version antérieure sont un bon choix. On joue la sécurité sur une carte désormais mature, facilement trouvable en France et avec une communauté active. Mais aujourd'hui, nous pouvons trouver des alternatives particulièrement intéressantes. Les cartes ODROID sont très attractives sur plusieurs points : mémoire vive, processeur, port eMMC, Ethernet 1 Gb et un CPU un peu plus véloce. Oui, il est plus cher que la Pi3, mais pour construire un mini serveur 64 bits, le choix est parfait.

Si vous souhaitez uniquement créer un barebone/mini-pc pour des besoins basiques, la Banana Pro n'est pas un mauvais choix, même si nous la trouvons poussive. La Pine64 est bien intéressante et se compare facilement à la Pi 3, excepté sur la partie WiFi. Mais la Pine64 est moitié moins chère par des performances très honnêtes. La Beaglebone black peut être un choix intéressant même si la version Wireless n'est pas forcément la plus intéressante et dommage qu'elle soit si chère, malgré des avantages non négligeables sur partie CPU et les unités de calculs supplémentaires.

La C.H.I.P. pourrait éventuellement être un choix à considérer pour un mini-PC, à condition de ne pas être trop exigeant sur les performances. Pour une connexion VGA ou HDMI vous devrez rajouter un shield vidéo, alourdissant la facture de 15 $. Mais la carte est proposée à 9 $. Un tarif très agressif. Les cartes pcDuino peuvent être un autre choix même si nous la considérerons plus pour des montages et bornes, mais pourquoi pas ? La version 3B propose des ressources intéressantes, un Ethernet 1Gb.

Toutes ces cartes fonctionnent sur des processeurs ARM, donc il faudra utiliser des systèmes et des logiciels ARM et non x86. Mais l'offre se développe rapidement, notamment sur la partie serveur. Pour la bureautique, Internet, vous n'aurez aucune difficulté pour trouver des logiciels adaptés même si l'offre est plus limitée et tous les logiciels ne sont pas disponibles en version ARM.

Côté cartes mini-PC x86, malheureusement, le choix est très restreint, voire, inexistant en France. Nous étions très intéressés par la JaguarBoard supportant Linux et Windows, mais nos tests ont montré une instabilité matérielle, un support Windows perfectible et un prix disproportionné.

Comment faire son choix ? Plusieurs critères :

- Le stockage par défaut ;
- L'ajout de stockage et les performances de celui-ci ;
- La connectivité video ;
- La connectivité réseau ;
- Processeur video ;
- Mémoire vive et processeur ;
- Les ports USB disponibles ;
- Le système d'exploitation.

Le stockage est bien entendu un élément essentiel pour un PC. Quelques cartes proposent un stockage directement sur la carte, mais la plupart nécessitent un rajout. Ce point est intéressant : est-ce uniquement une carte SD ou une autre connectivité ? La carte SD, malgré les améliorations réalisées, reste une unité de stockage lente et toutes les cartes ne supporteront pas plus 32 Go. Un port eMMC ou Sata sera le bienvenu. Mais n'espérez pas obtenir les performances d'un PC standard. Vous pouvez ajouter un disque dur via le port USB, mais, là-encore, les performances seront médiocres, comme nous l'avons constaté avec le PiDrive sur une Raspberry Pi. Malgré tout, pour un usage basique, le stockage ne sera pas le critère bloquant. Par contre, la connectivité vidéo et la puce graphique seront à regarder surtout si vous souhaitez avoir un usage en rétroconsole ou en mediacenter. Les Pi3 et ODROID s'en sortent plutôt bien.

La connectivité réseau est bien entendu un autre point essentiel. Le WiFi peut être le critère discriminant dans votre choix. Le port Ethernet est une interface classique et largement présente sur les cartes. Honnêtement, un port 10/100 suffira sauf si vous avez de gros débits à faire passer ou en mode serveur, dans ce cas, opter obligatoirement pour un Ethernet 1 Gb.

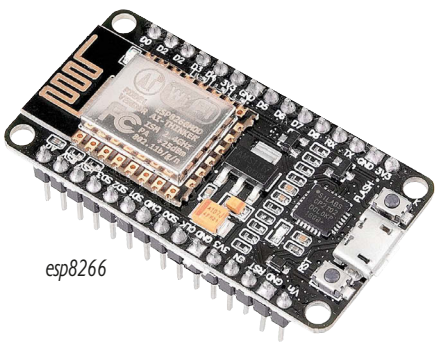
Pour l'alimentation, les claviers, l'écran, le boîtier, aucun souci à avoir ! Rien que du standard ! Soyez vigilant sur la version HDMI supportée par la carte choisie qui peut poser des soucis car tous les moniteurs ne supportent pas la v2.

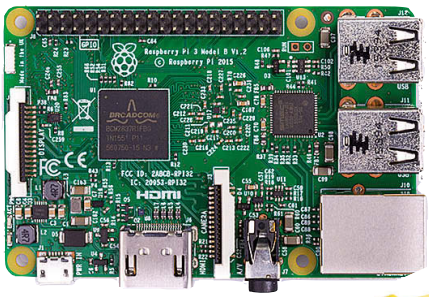
Il existe des kits complets pour monter un ordinateur de bureau ou un portable. L'un des plus connus est le Pi-Top. Il inclut un châssis, un écran 13,3'', un clavier. Une version Tout-en-un desktop est disponible, à un "tarif intéressant" (-160).

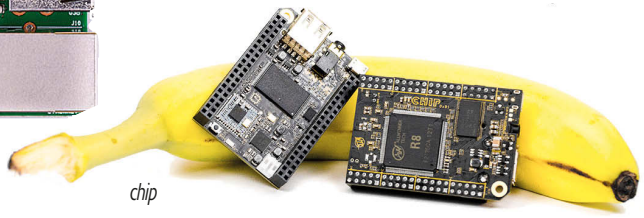
Si vous souhaitez créer un serveur, pour une petite entreprise ou personnelle, c'est possible. Nous vous conseillons des cartes assez véloces pour le faire, une Pi3 ou une ODROID, éventuellement une Pine64 si les besoins restent modestes. N'oubliez pas que vous êtes en architecture ARM et non x86. Et vous pouvez même créer des clusters sans aucun problème. En Pi, il existe plusieurs solutions matérielles et logicielles pour les monter et les gérer.

Plus audacieux, l'usage de la VCore. Impressionnante par sa taille ridicule, une pièce de 2 , elle offre des ressources matérielles très intéressantes. Pour créer un nanoserveur (Web, serveur de fichiers, etc.) ou monter un NAS personnel, le VoCore2 est une solution à ne pas négliger même si cela peut vous surprendre. Par défaut, la carte fonctionne avec OpenWRT, un Linux largement répandu dans les routeurs. Pour un prix agressif, l'ensemble complet, édition Ultimate, propose une solution complète. La carte seule est vendue -18 $.

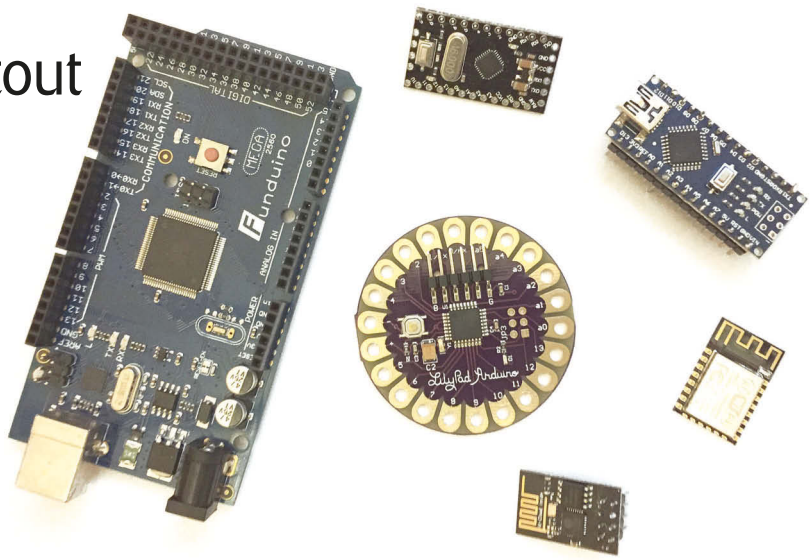
À vous de jouer maintenant !


esp8266


Raspberry Pi 3

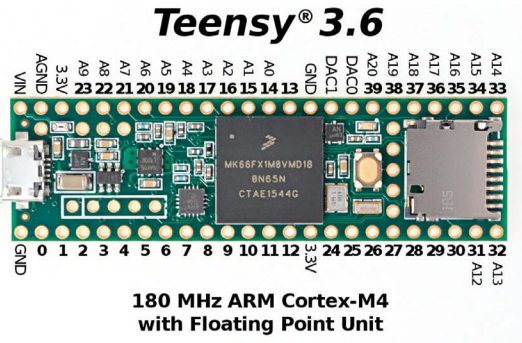

chip

Renaud Pinon : **Arduino** et **ESP** avant tout



Niveau cartes je dois avouer que je n'utilise (pour le moment) que les Arduinos et les ESP8266 (quand le Bluetooth ET le wifi pourront être utilisés simultanément sur l'ESP32, je m'en procurerai un, mais pas avant). Il m'arrive pour certains projets d'utiliser des Raspberry PI mais c'est uniquement lorsque j'ai besoin de puissance de calcul, l'inconvénient étant évidemment le temps de chargement de l'OS.

Ceci dit, même si je n'en ai jamais utilisés, les Teensy me font de l'oeil par leur puissance (toute relative certes, mais immense pour un simple micro-contrôleur), le seul défaut pour moi étant leur forme très allongée qui est difficile à placer dans un projet qu'on veut forcément le plus petit possible.



Côté Arduino, il y en a pour tous les goûts entre le UNO qui est très polyvalent, le MEGA aux possibilités immenses mais un peu encombrant (ce qui est tout de même très relatif !), le Nano très adapté pour les petits projets mais forcément limité au niveau de la mémoire (et les clones chinois sont très limités au niveau de l'ampérage qu'ils peuvent fournir aux modules qui lui sont connectés et leur port USB nécessite de télécharger un pilote), et enfin celui qui a ma préférence, le Pro MINI, qui même s'il ne possède pas de port USB permet de vraiment miniaturiser les projets (il faut simplement se procurer un module USB to UART). Et évidemment je ne parle même pas des Flora par Adafruit qui ont une taille assez réduite, peuvent pour certains être alimentés directement par une pile bouton et permettent de créer facilement des vêtements connectés.

Sinon pour en revenir aux cartes que je n'ai encore jamais utilisées mais qui me font envie, l'Omega2 d'Onion (https://onion.io/) est pour moi celle qui a l'air la plus prometteuse actuellement : un Linux en guise d'OS, WiFi intégré, une quinzaine de broches avec du série, du SPI et de l'I2C (mais malheureusement pas d'entrée analogique) le tout pour 5$! Et une taille de guêpe du tiers d'un Arduino et du quart d'un Raspberry PI !

Enfin, les conseils que je peux donner à tout maker quel que soit son niveau :

- La soudure est notre amie, il ne faut pas en avoir peur. Un simple fer à souder d'une quarantaine d'euros suffit.

- Avoir un multimètre (10 à 15 euros). C'est indispensable pour vérifier qu'un voltage est à la bonne valeur avant de brancher un module par exemple. Très utile aussi pour obtenir la valeur d'une résistance : mettre le multimètre en mode Ohm (la valeur la plus basse) et mettre les 2 connecteurs sur chaque patte de la résistance : la valeur s'affichera. Si le multimètre affiche « 1 » il suffit de le régler sur la valeur supérieure ». C'est très pratique pour qui n'a pas envie de connaître les codes couleur des résistances (sachant qu'en type 5 bandes, elles sont sur fond bleu ce qui peut être problématique pour différencier un rouge d'un violet ou d'un marron). Enfin, dernier avantage du multimètre : il peut être utilisé en mode « continuité » pour vérifier si les soudures adjacentes se touchent : si c'est le cas, il bipe !

- Avoir plusieurs résistances / condensateurs / transistors / diodes de différentes valeurs. Des lots sont en vente sur certains sites Internet pour des prix de 3 à 5 euros les 500 ou 1000 (!) unités.

- Vérifier toutes les connexions 3 fois avant d'alimenter un circuit. C'est bête, mais parfois ça sauve la vie d'un module :)

- Télécharger et utiliser Fritzing (http://fritzing.org) pour garder une trace de ses propres circuits. En effet, il arrive souvent qu'on dépouille un projet pour en faire un autre et quand on veut revenir au premier projet, un bon schéma Fritzing nous sauve la mise ! (ou du moins plusieurs heures de prise de tête !)

- Utiliser du carton pour faire des boîtiers pour nos projets, c'est bien, mais utiliser une imprimante 3D, c'est vraiment mieux ;) Il existe désormais des modèles corrects pour un prix très accessible de 300~350 euros. Le filament en lui-même s'achète pour 25 à 45 euros le kg, ce qui est aussi très accessible. Enfin, avec une bobine de 1 kg il est vraiment possible d'imprimer beaucoup de choses, les impressions étant bien souvent remplies de vide (ce qu'on appelle l'infill, qui est paramétrable dans le logiciel de « tranchage »).

Au-delà des **cartes** connues

• Nicolas Gachadoit,
développeur de robots et d'objets connectés (http://www.3sigma.fr)

J'ai envie de commencer par un cri du cœur : il n'y a pas que l'Arduino Uno et la Raspberry Pi dans la vie ! Il ne faut pas acheter aveuglément l'une de ces deux cartes sous prétexte qu'elles sont les plus populaires : il n'y a aucune garantie que l'une ou l'autre puisse s'adapter à vos besoins. Soyez curieux, il existe des matériels très intéressants, des "pépites" peu connues mais qui pourraient vous rendre de grands services dans certains cas. La bonne méthode est de lister toutes les caractéristiques et les contraintes de votre projet pour choisir ensuite la carte qui conviendra la mieux.

Cette première phase est quasiment la plus importante. Il ne faut pas hésiter à y passer beaucoup, beaucoup de temps et à faire le maximum de choses « sur le papier », avant d'acheter quoi que ce soit. Vous maximiserez ainsi vos chances de partir sur un matériel qui sera compatible avec les spécifications actuelles et futures de votre projet. Il n'existe pas dans l'absolu de plateforme meilleure que les autres, mais il en existe sans doute une (ou plusieurs) répondant mieux que d'autres à votre cahier des charges. Le choix d'un matériel adapté est une tâche valorisante et évite en général à votre montage de ressembler à un mille-feuille noyé au milieu d'un plat de spaghetti. Voyons maintenant les différents critères à prendre en compte, avec quelques exemples de cartes pouvant convenir en fonction des besoins.

Encombrement

Ce critère doit être pris en considération dès le début : inutile de vous demander quels shields vous allez ajouter à votre carte Arduino Uno si votre création doit au final être cachée dans une boîte d'allumettes. Ce dernier cas de figure est le terrain de jeu favori des versions « mini » ou « micro », qui présentent des tailles (et parfois des prix) environ 3 fois inférieures à leurs grandes sœurs, sans être en reste en termes de performances. Par exemple, la petite LinkIt Smart 7688 et son Linux OpenWRT permettent de faire du streaming vidéo avec la même fluidité qu'une « grosse » Raspberry Pi.

Alimentation

Si votre montage est hors d'atteinte d'une prise de courant pour être alimenté par un adaptateur secteur, vous devrez utiliser une batterie et mieux vaut alors s'orienter vers des cartes adaptées. En effet, une petite LiPo à une cellule (par exemple) fournit une tension comprise entre 3 et 4.2V, en fonction de son état de charge. Ceci nécessite un convertisseur de tension survolteur si vous souhaitez alimenter le montage en 5V. Dans le monde « compatible Arduino », certaines des cartes A-Star de Pololu intègrent nativement ce type de convertisseur. D'autres possèdent même un circuit de recharge de batterie (la Qduino par exemple), tout comme les cartes BeagleBone Black et C.H.I.P. dans le monde des mini-ordinateurs. Faites par ailleurs très attention à la tension de fonctionnement des broches d'E/S : 5V ? 3.3V ? 1.8V ? Ceci aura des conséquences sur les périphériques que vous brancherez sur votre système.

Connectivité

La liste des types de communications possibles avec le monde extérieur est longue : Ethernet, Wifi, Bluetooth classique, Bluetooth Low Energy, Zigbee, radio 433 ou 868 MHz, LoRa, Sigfox,... Il est parfois dommage d'associer un Arduino « de base » avec le célèbre ESP8266 alors que ce dernier (qui intègre nativement le Wifi, est plus puissant et peut se programmer à partir du même environnement de développement) peut s'utiliser seul (NodeMCU, Adafruit Huzzah). On trouve même désormais des petites cartes de prototypage incluant à la fois du Wifi et du BLE, comme la RedBear Duo ou les modules à base d'ESP32. Concernant le BLE, sachez que beaucoup de matériels ne fonctionnent qu'en esclave (afin d'être pilotés, typiquement, par un smartphone).

Applications à exécuter

Si vous souhaitez commuter un relais lors du passage d'une personne devant un capteur PIR, une carte avec un petit-micro-contrôleur suffit amplement. En revanche, le mini-ordinateur sera de façon générale le seul choix possible pour faire des traitements lourds (traitement d'image embarqué en utilisant la bibliothèque OpenCV, par exemple). Un besoin de performances en termes de traitement rapide des E/S et, en parallèle, en termes de traitement applicatif trop lourd pour un matériel possédant peu de ressources, peut conduire à associer un micro-contrôleur et un mini-ordinateur. C'était l'objectif de la carte Arduino Yun, intégrant les deux sur un PCB unique mais boudée à cause de son prix. Il existe désormais de nombreuses alternatives, comme l'ultra-compacte LinkIt Smart 7788 Duo. En mode « multi-couche », une carte Pololu A-Star qui se connecte sur une Raspberry Pi est également un bon choix.

Langage de programmation

Qu'on se le dise une bonne fois pour toutes : le « langage Arduino » n'existe pas ! Il s'agit simplement d'une bibliothèque de fonctions écrites en C++ permettant d'ajouter une couche d'abstraction dont le but est de faciliter la programmation des micro-contrôleurs. Pour programmer une carte Arduino Uno, vous devez donc avoir des notions en C/C++. Les mini-ordinateurs permettent quant à eux plus de variété puisqu'ils offrent un véritable système d'exploitation. Si vous souhaitez utiliser un micro-contrôleur avec un autre langage que le C/C++, rassurez-vous, tout n'est pas perdu. Le projet MicroPython vise à faire tourner un sous-ensemble de Python sur des micro-contrôleurs à faibles ressources (comme l'ESP8266). Vous pouvez également, entre autres, programmer vos petites bêtes préférées en Javascript avec les cartes Espruino.

Capteurs et actionneurs

Il faut prendre le plus grand soin dans le choix des capteurs et des actionneurs : lisez les datasheets pour savoir si les fonctionnalités correspondent à vos besoins et notez bien la façon dont ils s'interfacent avec un micro-contrôleur ou un mini-ordinateur car cela pourra conditionner le choix de la plateforme matérielle. Si vous avez un ou plusieurs capteurs analogiques, sachez que la Raspberry Pi ne possède pas ce type d'entrée. Au lieu de lui ajouter un module comblant cette lacune, vous pouvez la remplacer avantageusement par une BeagleBone Black, une pcDuino ou une Odroid C2, pour ne citer que les plus connues.

Vérifiez également s'il existe une bibliothèque gérant le capteur ou l'actionneur dans le langage et pour la cible que vous souhaitez utiliser. Avec l'IDE Arduino, vous pouvez aller très loin dans les tests de compatibilité car vous avez la possibilité de compiler votre code après avoir choisi le type de carte cible, sans être connecté à cette dernière : c'est idéal pour vérifier que telle ou telle librairie fonctionnera bien avec le micro-contrôleur choisi. Enfin, de nombreuses cartes intègrent désormais certains capteurs, comme l'Arduino101 avec son accéléromètre et son gyroscope 3 axes.

Fiabilité

Est-il raisonnable d'utiliser un matériel typé « Maker » dans un projet nécessitant une grande fiabilité ? Ce sujet est source de débats sans fin. Sachez cependant qu'il existe des cartes conçues dans l'optique d'une utilisation industrielle tout en permettant une programmation via l'environnement de développement Arduino. Par ailleurs, le besoin éventuel de surveiller à distance un système peut conduire à utiliser un matériel possédant une connectivité Wifi, LoRa, GSM,...

Synthèse

Idéalement, vous devriez faire un tableau listant sur chaque ligne les différents besoins de votre projet suivant les critères évoqués ci-dessus et mettre en colonne les cartes que vous avez pré-sélectionnées. Si un des critères n'est rempli par aucune, essayez de trouver un autre modèle pouvant convenir. Il existe tellement de matériels différents qu'il semble difficile de ne pas en trouver un qui conviennent… ou alors vous essayez peut-être de réaliser un montage qui défie les lois de la physique !

Voilà, vous avez maintenant trouvé le matériel idéal pour votre projet. Vient maintenant l'éternelle question : version originale ou clone (s'il existe) ? L'objectif n'est pas de trancher le débat ici, mais vous pouvez malgré tout vous demander qui « mérite » le plus de recevoir vos précieux euros : le créateur ou le copieur ? •

Visual Studio 2017

• Vivien FABING
• Alexandre NOURISSIER
Consultants Infinite Square
https://blogs.infinitesquare.com/

INFINITE SQUARE

Depuis le 7 Mars 2017 dernier, la nouvelle version de Visual Studio est enfin disponible en version finale. Côté prérequis d'installation, le support de Windows Server 2008 R2 disparaît, tandis que Windows Server 2016 fait son apparition. Concernant les prérequis matériels, les spécifications demandées sont globalement un peu plus grosses que pour la version précédente : un processeur un peu plus rapide (1.8 GHz ou plus), un peu plus de RAM (2 GO de RAM ou plus), une résolution un peu plus large (1280 par 720). C'est par contre côté espace disque nécessaire que le changement se perçoit le plus (comme décrit dans la suite de cet article, lié à la notion de Workload).

Pas de grand changement non plus dans la déclinaison des éditions disponibles : VS 2017 Community pour la version gratuite (sous conditions), VS 2017 Professional et VS 2017 Enterprise sont de la partie. À noter tout de même, l'apparition d'une nouvelle appelée "Build Tools for Visual Studio 2017" qui permet d'installer l'écosystème MSBuild pour pouvoir générer des applications sur une machine de Build, sans devoir installer l'intégralité de Visual Studio dessus.

LES NOUVEAUTÉS DE L'IDE

Installation

Les premiers changements sont visibles dès l'installation. En effet le découpage des outils et des fonctionnalités Visual Studio 2017 a été revu afin de réduire considérablement l'espace disque nécessaire à son installation. L'idée est de revenir aux scénarios réels d'utilisation de Visual Studio en proposant un outil effectivement à la carte. Le but est clair : faciliter la prise en main par les nouveaux utilisateurs venus de Visual Studio Code, et, au passage, améliorer l'expérience d'utilisation grâce à un outil plus léger et plus réactif.

Par défaut seul l'éditeur, supportant 20 langages de programmation, ainsi que les fonctionnalités de base (contrôle de code source, IntelliSense et Code Lens sauf pour l'édition Community) sont disponibles. Aucun SDK ni Template n'étant fourni, l'installation ne prend que 740Mo plutôt que la petite dizaine de Go auxquels les précédentes éditions nous avaient habitués et il en va de même avec le temps d'installation, beaucoup plus court.

C'est lors de l'installation, et par la suite en ré-exécutant l'installateur, que celui-ci nous propose d'étendre les capacités de Visual Studio 2017 en ajoutant des « workloads ». [1]

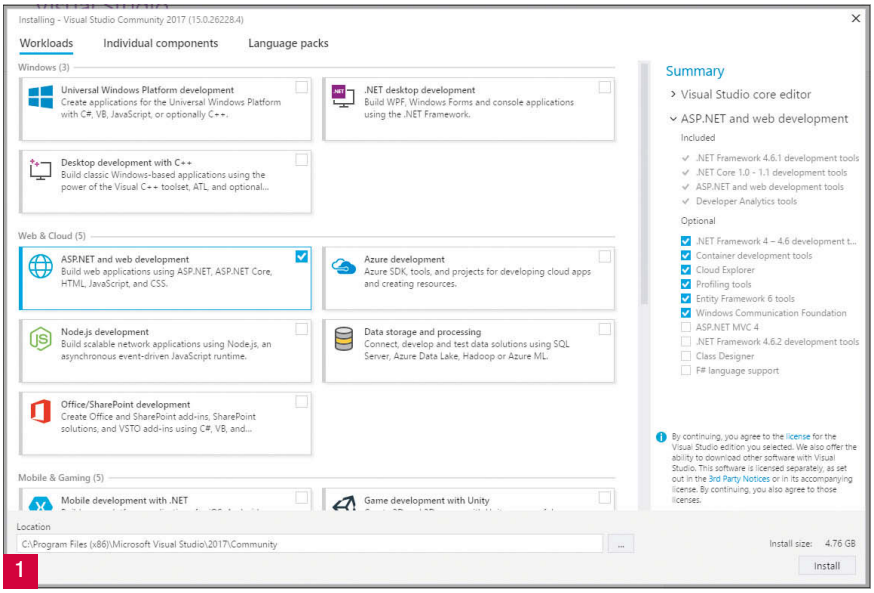
Sous cette dénomination sont regroupés des composants, des outils, des Templates et des langages de manière à pourvoir à tous les besoins des développeurs selon leurs occupations principales. Par exemple les développeurs ASP.NET disposent d'un workload "Web development", les créateurs d'applications universelles d'un workload "Universal Windows Platform development", les développeurs utilisant les fonctionnalités avancées d'Azure d'un workload "Azure development" et ainsi de suite… Notez que Visual Studio 2017 ne se cantonne pas aux technologies Microsoft mais fournit également des workloads pour des technologies telles que Node.JS, Python, C++, l'analytique, le langage R et bien d'autres. Les technologies propulsant le framework .NET Core sont inclues dans le workload « ASP.NET and web development » mais optionnelles dans le workload « .NET desktop development ».

Afin d'alléger au maximum l'installation et d'améliorer la réactivité de la plateforme, les workloads n'installent par défaut que le strict minimum et indiquent clairement depuis le même écran les outils et extensions optionnels qu'il est possible d'ajouter.

Il reste bien sûr possible de sélectionner un par un les outils dont on peut avoir besoin, les versions de framework cibles, les outils tiers (Github, Resharper) et la langue de l'éditeur.

Une fois l'installation complétée, il est possible en ré-exécutant l'installateur de retirer des workloads ou des composants, proposant ainsi une expérience plus dynamique pour permettre d'utiliser l'outil le mieux adapté au projet ou au poste du moment.

Dernier point important suite à la mise à jour de l'installateur, les extensions pourront désormais décrire les composants requis à leur bon fonctionnement afin de permettre aux utilisateurs d'installer ceux qui manquent si le cas se présente. Enfin, il est désormais possible de planifier l'installation et la désinstallation de plusieurs extensions en même temps. Plus be-

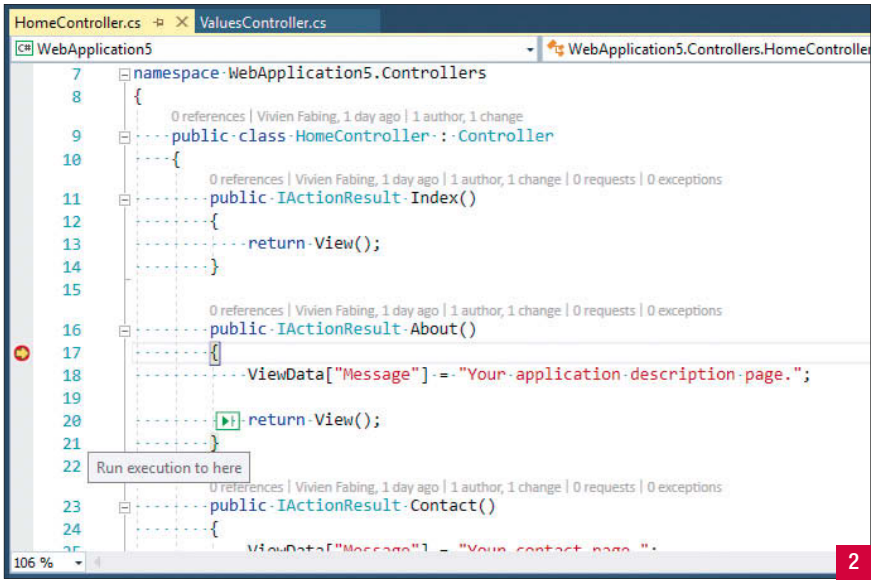


1

soin de relancer Visual Studio entre chaque opération, elles s'exécuteront dès que toutes les fenêtres seront fermées. Il est même possible de sauvegarder dans le Cloud toutes les extensions utilisées pour rendre plus naturelle l'utilisation de plusieurs environnements.
Avec cette installation revue de fond en comble, Microsoft espère faciliter les premières étapes de l'expérience des développeurs et présenter Visual Studio comme un outil accessible. Pouvoir plus efficacement capter les développeurs qui désirent tester les nouvelles technologies .NET Core et Azure depuis Visual Studio Code est stratégique pour accompagner la nouvelle image que Microsoft s'est forgée dans les communautés de développeurs, notamment open-source.

Au quotidien côté éditeur

Avant même de parler de l'éditeur en tant que tel, Visual Studio incorpore maintenant le support des fichiers de conventions de code EditorConfig. Ce format de fichier permet d'assurer à travers les outils et les projets le respect des styles définis. Parmi les éditeurs supportant ce système se trouvent IntellijIdea, WebStorm, Atom, Eclipse, Emacs, NetBeans, Sublime Text , Visual Studio 2015, 2017 et Code. Ce type de fichier était déjà supporté à travers un plugin pour la version 2015, il est maintenant complètement intégré.
Dans l'optique de réduire les frictions à l'usage de Visual Studio 2017, l'équipe de développement Microsoft annonce avoir réduit d'au moins 50% le temps de la première ouverture. De nombreuses fonctionnalités tournant autour de l'ouverture des solutions ont été revues et améliorées et il est même désormais possible d'ouvrir directement des dossiers, à l'instar de ce qu'il est possible de faire avec Visual Studio Code pour les applications ne reposant pas sur le système de projets.
Afin d'accompagner cette amélioration, le fonctionnement même du rechargement des solutions a été amélioré pour accélérer le retour au travail après un changement de branche Git. Pour ne rien gâcher, Il est aussi possible d'activer dans les options des solutions une version du chargement des solutions adaptée aux solutions traditionnelles particulièrement volumineuses, appelée **Lightweight Loading**. Enfin lors de l'utilisation en mode dossier, le support syntaxique et la recherche de symboles de beaucoup de langage ont été intégrés, et le support du débogage a été ajouté pour le C++, les projets Node.js, le C# et le VB.
L'éditeur supporte activement les nouvelles



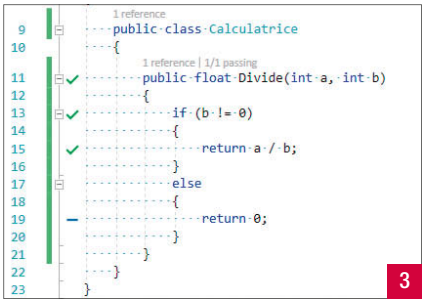
fonctionnalités de C# 7 et propose maintenant des remplacements pour des structures de codes fréquents avec les fonctionnalités apportées par cette version du langage. Par exemple la déclaration de variable « out » au moment même de son utilisation en tant que paramètre d'une méthode. Un autre exemple est de remplacer les vérifications de nullité par des opérateurs de coalescence nul.
La fonctionnalité indispensable de l'éditeur de Visual Studio, IntelliSense, a vu plusieurs améliorations et un meilleur support d'un certain nombre de langages. Il est maintenant possible avec C# et VB de filtrer les propositions de membres par types afin de faciliter l'exploration de bases de code inconnue.

Run To Click [2]

Côté débogage, une nouvelle fonctionnalité qui risque de très vite s'avérer indispensable tant elle est pratique: le **Run To Click** permet de naviguer rapidement vers une ligne de code à déboguer, sans pour autant devoir mettre des points d'arrêt partout. En mode débogage, le survol d'une ligne de code fait apparaître un petit bouton qui permet de poursuivre l'exécution du code de l'application et de s'arrêter sur cette ligne précise.

Live Unit Testing and Live Dependency Validation [3]

Côté testing, c'est la fonctionnalité de **Live Testing** qui fait son apparition : elle est facilement démarrable / stoppable depuis le menu Test de Visual Studio. Une fois activée, les tests automatiques contenus dans la solution vont être exécutés en tâche de fond ; ils vont afficher devant



chaque ligne de code un indicateur pour montrer si la ligne de code est couverte ou non par un test automatique, et si le test automatique est réussi ou en échec. De plus, en même temps que le développeur continue de modifier des lignes de code, les tests impactés vont être joués en parallèle, et vont mettre à jour les résultats en Live.

.Net Core

Nouveau format .Net core csproj

Côté .Net Core, on ne peut pas passer à côté de ce changement de paradigme : l'abandon des récents project.json au profit des .csproj. Heureusement, Microsoft a prévu une migration automatique via l'ouverture de ces projets depuis Visual Studio 2017, ou encore en exécutant la commande dotnet migrate.

.Net Core Tools 1.0

Intimement lié à ce changement des project.son, et accompagnant la sortie de Visual Studio 2017, l'outillage de .Net Core se voit doté de la sortie d'une nouvelle version **.Net Core Tools 1.0**, compatible Windows, macOS et Linux.

XAML

XAML Edit & Continue

Côté éditeur XAML, la nouvelle fonctionnalité d'**Edit & Continue** permet aux développeurs de modifier leurs fichiers XAML pendant que leur application est en train d'être déboguée, et de voir leurs modifications s'appliquer en Live dans l'application au moment de la sauvegarde de leurs modifications XAML.

(UWP) Manifest Designer Capability for creating Visual Assets

Bonne nouvelle pour les développeurs d'applications UWP, l'éditeur de Manifest a fait peau neuve, et se voit doté d'une fonctionnalité de génération d'assets visuels. Elle permet ainsi à partir d'une seule image, de générer des Tiles, Logos, Icônes et Splash Screen pour toutes les tailles et pour tous les appareils cibles de votre application.

Cross-Platform

Cordova Simulate

Côté Cross-Platform, l'apparition d'un nouvel émulateur appelé **Cordova Simulate** permet de visualiser son application dans un navigateur Web, permettant de voir ses modifications se recharger à chaud, d'émuler les plugins, etc.

Xamarin 4.3

Et côté Xamarin, Visual Studio 2017 se voit doté de sa nouvelle version 4.3, qui permet entre autres de bénéficier d'une interface graphique pour modifier les fichiers iOS de .plist et d'ajouter le support pour tvOS

Redgate DATA Tools [4]

À noter également l'annonce récente de Redgate, concernant l'intégration d'une version Core de ses produits ReadyRoll, SQL Prompt et SQL Search, incluse directement dans l'édition Enterprise de Visual Studio 2017. Cette intégration offre une alternative solide aux SSDT (SQL Server Data Tools) standards de Visual Studio, et offre également une version Pro pour les grandes entreprises souhaitant bénéficier de support professionnels et de fonctionnalités avancées.

UML et Live Dependency Validation

Pour finir sur ce tour d'horizon des nouveautés globales de Visual Studio, on regrettera la suppression soudaine des outils d'UML, même si cette disparition s'accompagne également d'une amélioration de l'outillage de validation de l'architecture des dépendances, avec notamment la fonctionnalité de Live Architecture Dependency Validation basée sur Roslyn, qui permet de signaler une violation d'utilisation d'une dépendance détectée au moment où le code est modifié.

L'INTÉGRATION AVEC AZURE ET DOCKER

Lorsque l'on parle de fluidification du processus de déploiement, difficile de ne pas entendre parler de Docker. Souvent vu comme un système de petites machines virtuelles incrémentielles, Docker est la plateforme qui permet de fiabiliser les déploiements entre les divers environnements applicatifs (Dev, Test et Prod par exemple). Jusqu'à présent, on pouvait lui reprocher de devoir obligatoirement le piloter via des lignes de commandes, qu'il ne permettait d'exécuter que des applications fonctionnant sous Unix, et enfin qu'il était difficile de l'intégrer dans l'environnement Microsoft, notamment au travers d'Azure et de VSTS (Visual Studio Team Services).

Voyons maintenant comment ces reproches ont pu être corrigés, en même temps que la sortie de Visual Studio 2017.

Les conteneurs Linux avec Docker et ASP.NET Core

Démarrage avec Visual Studio

À l'installation de Visual Studio 2017, en sélectionnant le Workload ".Net Core cross-platform development", on récupère non seulement tout l'outillage lié au développement ASP.NET Core, mais également l'intégration de l'outillage Docker dans Visual Studio.

L'intégration avec Docker des sites Web ASP.NET Core est disponible principalement au travers de deux points d'entrée :

- Lors de la création d'un nouveau site Web ASP.NET Core, une petite case à cocher permet d'ajouter automatiquement à la solution les fichiers nécessaires à l'intégration du site Web dans Docker
- Pour un site Web ASP.NET Core déjà existant, il suffit d'effectuer un clic droit sur le projet Visual Studio pour faire apparaître depuis la section "Add", l'option "Docker Support" pour ajouter également les fichiers nécessaires.

Il ne reste plus qu'à installer et configurer Docker for Windows (si ce n'était pas déjà fait), puis sélectionner la configuration de débogage sobrement intitulée "Docker" pour que Visual Studio 2017 compile puis exécute automatiquement l'application ASP.NET Core via Docker. Et voilà, le coût d'entrée pour faire fonctionner une application sous Docker en est réduit à quelques clics !

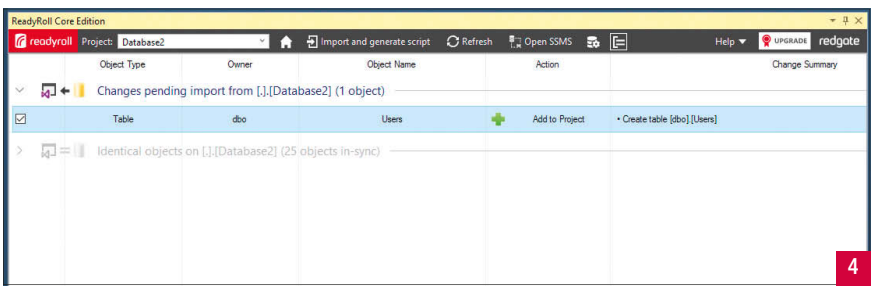
Le log de la compilation est quant à lui assez verbeux, et l'on peut voir le rapatriement des différentes images Docker sur son poste en local, notamment la fameuse image aspnetcore-build (elle-même basée sur une image Linux Debian) qui contient en seulement quelques méga-octets de quoi faire fonctionner votre application ASP.NET Core.

Bien que fonctionnant sur un conteneur Linux, il n'en reste pas moins que l'expérience en tant que développeur reste très similaire à celle à laquelle nous sommes habitués lors du développement d'applications ASP.NET Core classiques : Positionnement d'un BreakPoint pour débogage, utilisation de la nouvelle fonctionnalité de Run To Click, etc.

Publication sur Azure

Si l'intégration de Docker avec Visual Studio est indubitablement une grande réussite, le déploiement sur Azure d'une application ASP.NET Core sur un conteneur Docker n'est pas en reste. En effet, à l'instar des déploiements ASP.NET Core classiques, un simple clic droit sur le projet Web permet d'afficher l'assistant de publication du site Web. Celui-ci propose la création d'un Azure Container Registry privé (en Preview) sur lequel sera déposé le conteneur Docker de votre application, ainsi que d'un App Service Plan sous Linux (encore en Preview), sur lequel sera exécutée votre image Docker.

Si le premier déploiement peut prendre quelques minutes (le temps d'uploader le conteneur Linux, la couche ASP.NET Core, etc.), les déploiements suivants se feront plus rapide-



4

ment étant donné que seule la couche applicative (votre application Web ASP.NET Core) sera uploadée.
En bref, vous l'aurez compris, l'intégration de Docker for Windows avec ASP.NET Core est vraiment d'une utilisation la plus simple au possible avec Visual Studio 2017. Néanmoins, l'usage que nous avons vu jusqu'à maintenant reste encore réservé aux technologies ASP.NET Core multiplateformes, étant donné que les conteneurs utilisés sont des conteneurs Linux.

Les conteneurs Windows avec Docker et ASP.NET

Démarrage avec Visual Studio

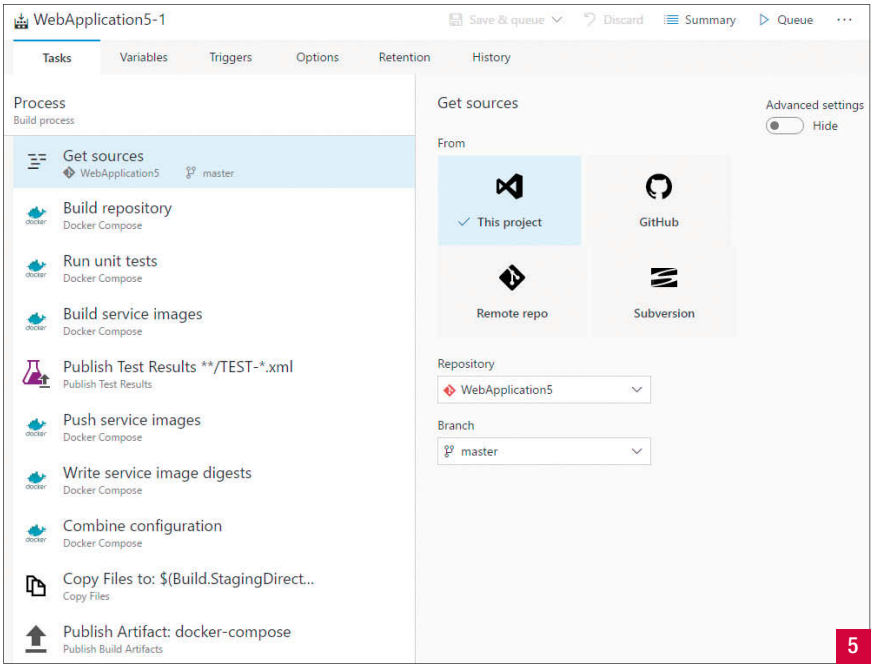
Lorsque l'on est développeur d'une application ASP.NET classique, basée sur le Framework .Net 4.6 par exemple, est-on condamné à ne pas pouvoir bénéficier du développement sous environnement Docker ? La réponse à l'heure actuelle est non, car depuis quelques temps Microsoft a sorti une image microsoft/aspnet, basée sur un conteneur sous Windows Server Core, qui permet de faire tourner sur ce conteneur Windows un site ASP.NET classique, via Docker. Grâce à ce conteneur aspnet, Visual Studio 2017 est capable d'offrir la même expérience que pour ASP.NET Core vis-à-vis de l'intégration de Docker : un clic droit sur le projet ASP.NET, et l'option "Add Docker Support" permet d'enrichir la solution des différents fichiers Docker nécessaires pour faire tourner votre application ASP.NET classique dans un conteneur.
Il faut cependant noter deux petites choses concernant l'utilisation des conteneurs Windows :

- Docker for Windows permet d'utiliser soit des conteneurs Linux, soit des conteneurs Windows, mais pas les deux en même temps. Il faudra bien veiller à ne pas oublier de configurer Docker for Windows à l'utilisation des conteneurs Windows avant d'exécuter une application ASP.NET classique.
- L'image Windows Server Core pèse à elle toute seule un peu moins de 10 Gigaoctets ! Armez-vous de patience à la première exécution le temps que cette image volumineuse soit rapatriée en local. À noter que cette problématique de taille n'est contraignante qu'au premier déploiement : les images Docker de base étant mises en cache en local, seules les couches différentielles applicatives sont redéployées à l'avenir.

Malgré tout, Microsoft a également annoncé une version plus light de Windows Serveur appelée "Nano Server". Cette version allégée de Windows Serveur contient à l'heure actuelle le nécessaire pour faire tourner un IIS, ainsi que des applications ASP.NET Core.

Publication sur Azure

Côté intégration avec Azure des conteneurs Windows depuis Visual Studio 2017, malheureusement il n'existe pas encore de solution out-of-the-box, où il suffirait juste de cliquer sur un bouton "Publier" pour pousser son conteneur ASP.NET classique sur Azure. Pour l'instant, monter soi-même son environnement de conteneurs Windows sur Windows Server 2016 reste encore la meilleure option, en partant par exemple de l'image virtuelle disponible sur le MarketPlace Azure "Windows Server 2016 Datacenter - with Containers".
À noter cependant que Microsoft a annoncé l'année dernière une Preview privée d'une version d'ACS (Azure Container Services) supportant les conteneurs Windows.



5

Le Continuous Delivery avec Docker

Démarrage avec Visual Studio et VSTS

Impossible de parler de Docker sans parler DevOps et Continuous Delivery. À ce sujet, une extension Visual Studio appelée "Continuous Delivery Tools for Visual Studio" permet de laisser Visual Studio configurer une chaîne de déploiement en entier, au travers des outils de Build et Release de VSTS (Visual Studio Team Services), pour déployer sur un ACS (Azure Container Services) orchestré par DC/OS (Datacenter Operating System).

Configuration d'ACS w/ DCOS [5]

Avant de poursuivre, il est important de bien clarifier les différentes briques mises en jeu dans cette solution :

- DC/OS (Datacenter Operating System) : sans rentrer dans les détails, c'est un système d'exploitation distribué qui permet de gérer des clusters de machines de la même manière que l'on ne gèrerait qu'une seule machine (*exemple : lorsque vous développez de nos jours, vous ne choisissez pas forcément sur lequel des cœurs du CPU va s'exécuter votre programme, eh bien DC/OS vous propose le même genre d'abstraction, mais pour un cluster de machines*) ;
- ACS (Azure Container Services) : la solution d'hébergement des conteneurs optimisée pour Azure. Elle permet notamment de configurer des agents sur lesquels seront déployés les conteneurs, à l'aide d'un orchestrateur tel que DC/OS, Docker Swarm ou encore Kubernetes ;
- VSTS (Visual Studio Team Services) : la version Cloud de TFS (Team Foundation Server), l'outil collaboratif de référence des projets utilisant des technologies Microsoft (et plus !). Avec notamment deux modules qui vont nous intéresser : la Build qui permet d'automatiser la génération de paquets applicatifs à déployer, et la Release qui permet de configurer ces paquets et de les déployer dans différents environnements.

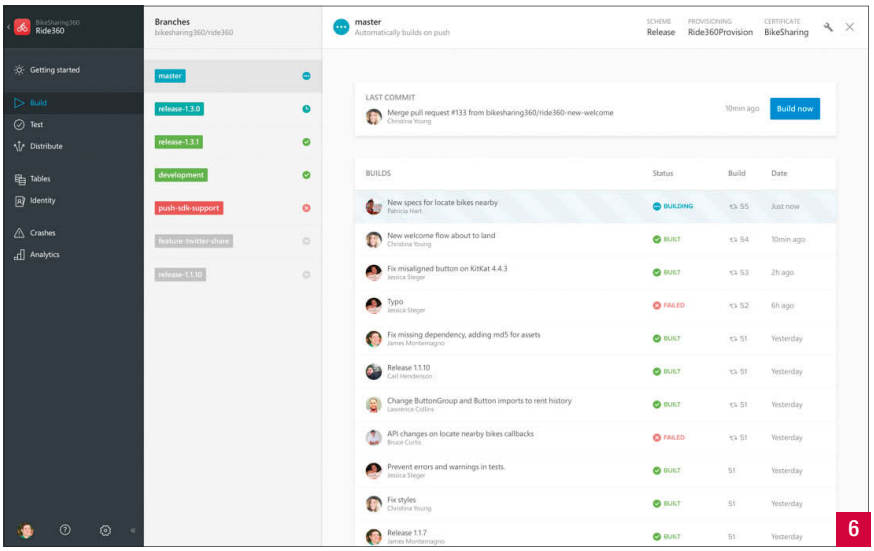
Pour commencer, il faut instancier un ACS (Azure Container Services), orchestré par un

DC/OS (Datacenter Operating System) dans votre souscription Azure. Note : à l'heure actuelle, seul ACS orchestré par DC/OS est supporté par le plugin de Visual Studio.
Durant la configuration de votre ACS, il sera nécessaire de générer des clés privées et publiques, générations qui peuvent être effectuées assez facilement via les outils en ligne de commande disponibles dans gitbash par exemple, ainsi que par des petits utilitaires que sont Putty.exe et ses dérivés (puttygen.exe et pageant.exe).
Un Azure Docker Registry sera également nécessaire, sachant que celui-ci peut être facilement créé automatiquement depuis Visual Studio en publiant un conteneur Docker ASP.NET Core sur Linux sur Azure.
Une fois tous les prérequis réunis, commencez par ajouter la solution Visual Studio contenant les conteneurs Docker dans un Repository Git de VSTS, puis effectuez un clic droit sur le projet ASP.NET Core et sélectionnez "Configure Continuous Delivery…". Vous pourrez ensuite choisir le Repository Git à utiliser, l'Azure Docker Registry sur lequel publier les images, et enfin l'Azure Container Services sur lequel les déployer et les exécuter.
Visual Studio 2017 va alors créer une définition de Build capable de générer les images Docker de votre solution, ainsi qu'une définition de Release contenant trois environnements (Dev, Test et Prod) qui vont déployer vos images Docker sur votre ACS (Azure Container Services) automatiquement.
Note : les déploiements ainsi effectués ne sont visibles que dans le réseau interne de l'ACS (Azure Container Services). Il est possible de vous connecter à ce réseau interne via un tunnel SSH créé avec l'Azure CLI, ce qui vous permettra également d'accéder à l'interface de gestion Web du DC/OS. Pour exposer publiquement et simplement le site Web déployé, vous pouvez par exemple ajouter une entrée dans la configuration de votre fichier docker-compose.yml (Exemple disponible sur : https://github.com/vfabing/programmez-vs2017-samples)

LA PLATEFORME VISUAL STUDIO 2017

Visual Studio Mobile Center

Durant l'évènement Connect qui a eu lieu le 16 Novembre 2016, une nouvelle plateforme dédiée au développement d'applications mobiles a vu le jour : Mobile Center. En effet depuis plusieurs années déjà, Microsoft s'est



doté de plusieurs services permettant de faciliter le développement d'application mobiles : Azure Mobile Services, Hockey App, Xamarin, etc. Il faisait sens de rassembler tous ces services dans une unique plateforme, dans laquelle ces différents services s'intègrent très facilement. Mais Mobile Center ne se contente pas uniquement de regrouper ces différents services ; il en propose également de nouveaux, permettant de couvrir de A à Z les besoins en développement mobile. Note : Ce service étant encore à l'heure actuelle en Preview, son utilisation est gratuite et limitée.

Démarrage

Une fois connecté au portail (https://mobile.azure.com), une première page permet, à l'instar d'HockeyApp, de visualiser les différentes applications mobiles gérées.
Il est possible d'en rajouter de nouvelles, de type iOS, Android ou Windows (à venir très prochainement).
Pour chacun de ces systèmes, différentes plateformes sont supportées, telles que React Native ou Xamarin, ainsi qu'Objective-C / Swift pour iOS et Java pour Android. Note : il est bien entendu possible d'utiliser le portail Mobile Center pour d'autres plateformes telles que Cordova, mais toutes les briques qui composent le portail ne seront pas forcément compatibles.

Distribution des Apps, récupération des Crashs et des Analytics

Tout droit issu du fonctionnement d'HockeyApp, le module Distribute permet de gérer simplement les groupes de distribution de votre application, ainsi que les différentes versions mises à disposition. Il est possible, par exemple, de composer un groupe contenant les adresses emails des Développeurs, ainsi qu'un groupe contenant les adresses emails des Testeurs. Ainsi à chaque mise à disposition d'une nouvelle version de l'application, il est possible de définir si celle-ci est accessible par ces deux groupes, ou seulement l'un d'entre eux. Les membres des groupes définis reçoivent alors un email leur permettant de télécharger et d'installer l'application ainsi partagée. Note : contrairement à HockeyApp, il n'est pas nécessaire de télécharger et d'installer une application cliente pour installer les applications partagées. Sous Android l'apk est téléchargé et facilement installable, sous iOS ce sont les API d'installation de Safari qui sont utilisées.
Étape suivante peu coûteuse et s'avérant souvent très utile : l'ajout du SDK de Mobile Center dans votre application mobile pour permettre la détection automatique des Crashs de votre application ainsi que la collecte de nombreuses statistiques Analytics. Pour chacune des technologies supportées sur Mobile Center, un tutorial présent sur la page d'accueil Mobile Center de votre application vous explique comment installer les paquets nécessaires (exemple pour Xamarin : les paquets NuGet Mobile Center, Mobile Center Analytics et Mobile Center Crashes)

Build [6]

Première brique originale de ce portail : la possibilité de connecter un compte GitHub sur lequel la brique de Build sera capable de récupérer le code source, et de le builder sur l'une des machines virtuelles mise à disposition par Microsoft. Les agents de Build étant à l'heure actuelle des machines virtuelles créées à partir d'image virtuelles OS X, seules les applications de type iOS ou Android sont supportées, mais

le support prochain des applications UWP (Universal Windows Platform) a d'ores et déjà été annoncé. Une fois la Build créée, quelques paramétrages supplémentaires, mais néanmoins importants sont disponibles, notamment :

- Déclenchement de la Build automatiquement à chaque Push ;
- Spécification des entrants nécessaires à la signature du package, afin de faciliter sa distribution sur le Store par exemple ;
- Sélection d'un groupe de distribution pour lequel un email sera envoyé lors de chaque mise à disposition d'une nouvelle version de l'application.

Test (Xamarin Test Cloud)

Basé sur Xamarin Test Cloud, le module de Test permet d'exécuter des tests d'interface graphique sur des appareils Mobiles directement depuis le Cloud. Outre les avantages apportés par l'automatisation de tests d'interface graphique sur Mobiles, c'est vraiment la grande variété des appareils Mobiles proposés qui en fait un outil incontournable (sur Android par exemple, la gamme s'étend sur plus de 200 appareils, du Google Pixel XL sur Android 7.1.1, à la tablette Motorola ZOOM sur Android 3.1, en passant par tous les Samsung Galaxy, les Nexus et autres…).

Au démarrage de ce module, un assistant de configuration s'ouvre et propose de sélectionner un groupe d'appareils Mobiles sur lequel les tests vont s'exécuter. Sur l'étape suivante, il convient de sélectionner le Framework de test utilisé ou à utiliser parmi les Framework de test actuellement supportés : Appium, Calabash, Expresso (Android uniquement), ainsi que, bien entendu, Xamarin.UITest. Pour terminer, un tutoriel s'affiche expliquant comment utiliser la CLI (Command Line Interface) de Mobile Center pour exécuter les tests automatiques, uploader le paquet de l'application, etc.

Par la suite, les différentes exécutions des suites de tests automatiques seront consultables sur cette même page, avec pour chaque exécution, la date de démarrage, la version testée, le temps d'exécution, le résultat (échoué ou réussi), le nombre de tests réussis ainsi que le nombre d'appareils testés. Il est ensuite possible de voir pour chaque résultat de test le détail, étape par étape, de l'exécution sur chacun des appareils mobiles, avec à chaque fois une capture d'écran du résultat de l'étape.

Tables, Identity (Azure Mobile Services)

Dernières briques à mentionner concernant Mobile Center : l'intégration de deux modules déjà existants via les Azure Mobile Services. La gestion du stockage de données dans des Azure Tables, ainsi que la gestion de l'identification unifiée via Azure Identity. Si ces fonctionnalités ne sont pas nouvelles, leur intégration à Mobile Center permet de faciliter encore plus leur utilisation via une configuration facilitée depuis le portail Web de Mobile Center.

Prenons tout d'abord le cas de la mise en place d'un système de stockage de données pour une application mobile via les Azure Tables. Après avoir associé une souscription Azure à Mobile Center, le portail permet de gérer en quelques clics la création et la configuration d'Azure Tables, et mettre ainsi à disposition d'APIs REST pour y stocker des données. Côté mise en place de l'accès à ces API depuis votre application mobile : elle se résumera globalement à générer un App Secret depuis le portail de Mobile Center, et d'intégrer le SDK d'Azure Mobile Services à votre application en lui précisant votre App Secret.

Concernant Azure Identity, son intégration est en tout point similaire, à savoir intégrer le SDK et lui fournir un App Secret). Et côté configuration Mobile Center, il suffira uniquement de renseigner depuis le portail Web les différents Client ID et Client Secret des différents fournisseurs (Azure Active Directory, Facebook, Google, Comptes Microsoft ou encore Twitter). En ce qui concerne le déclenchement du processus d'authentification depuis l'application mobile, deux scénarios sont possibles :

- Intégrer le SDK du fournisseur pour aller récupérer un jeton d'authentification, et s'en servir pour se connecter via le client Azure Mobile (recommandé pour bénéficier d'une interface d'authentification propre à l'OS utilisé) ;
- Faire une authentification serveur, via l'affichage d'une mire de login dans une Web View. Cette seconde méthode, bien que proposant une expérience moins intégrée, a le mérite d'être mise en place en une seule ligne de code.

RoadMap

Le portail de Mobile Center est encore tout récent, et s'il offre déjà quelques fonctionnalités très intéressantes ; il y manque encore de nombreuses fonctionnalités fondamentales. Cependant la plupart de celles-ci, ainsi que d'autres nouvelles, ont déjà été annoncées dans la RoadMap globale de ce nouveau produit, et il semble indispensable de faire un petit tour d'horizon de ce qui nous attend dans les mois à venir :

- Support des applications UWP (Universal Windows Platform) ;
- Intégration avec VSTS (Visual Studio Team Services) ;
- Module de gestion de CodePush (https://microsoft.github.io/code-push/) pour "pousser" rapidement des fichiers (HTML, CSS, JS, Images) à des applications clientes ;
- Module de gestion des Notifications Push ;
- Ainsi que de nombreuses améliorations des modules existants.

Du côté de l'ALM

Conjointement avec la sortie de cette version finale de Visual Studio 2017, l'Update 1 de Team Foundation Server 2017 sort également en version finale. Comme à l'accoutumée, les Updates de Team Foundation Server sont principalement composés des nombreuses fonctionnalités mises à disposition toutes les 3 semaines sur Visual Studio Team Services. L'Update apportant probablement plus d'une cinquantaine de nouvelles fonctionnalités, il n'est pas très intéressant de les lister une à une dans cet article (la liste exhaustive est de toute manière disponible sur la Release Note publique liée à cette Update 1). Néanmoins on peut en citer quelques-unes, telles que :

- La page d'accueil personnalisée de la collection, personnalisée avec l'usage de l'utilisateur ;
- La gestion améliorée des notifications, avec notamment la possibilité de se désabonner individuellement d'une notification de groupe ;
- L'éditeur de Process Template pour Visual Studio 2017 ;
- La dépréciation des Team Rooms au profit de Microsoft Teams, Slack ou autre ;
- Le passage en mode payant de l'extension Package Management (gratuit pour les 5 premiers utilisateurs, et les possesseurs de licences VS Enterprise).

CONCLUSION

En conclusion, cette nouvelle version de Visual Studio s'annonce prometteuse, et permet de faire un pas en avant supplémentaire vers la rapidité d'exécution au quotidien. Son système de Workload offre une flexibilité d'installation et de mise à jour non négligeable. Son intégration toujours plus poussée avec le Cloud, et notamment Docker, permet aussi aux développeurs d'accéder à des outils puissants, de manière simple et efficace.

Quelques problèmes de stabilité peuvent être parfois constatés avec cette nouvelle version, mais ces problèmes ne sont néanmoins en rien incomparables avec ceux de la version 2015. •

La **démocratisation** de l'Intelligence Artificielle pour les développeurs avec Visual Studio 2017

- François Bouteruche
  Expert Technique Visual Studio
- Aleksander Callebat
  Data Scientist
- Inès Kouraïchi
  Visual Studio Sales Lead

"With innovations like Cloud, mobile and AI, the opportunity for developers to drive real change is greater than ever. Today we are releasing Visual Studio 2017 to empower any developer design any application on any platform"

Satya Nadella, le 7 mars 2017, à l'occasion du lancement de Visual Studio 2017

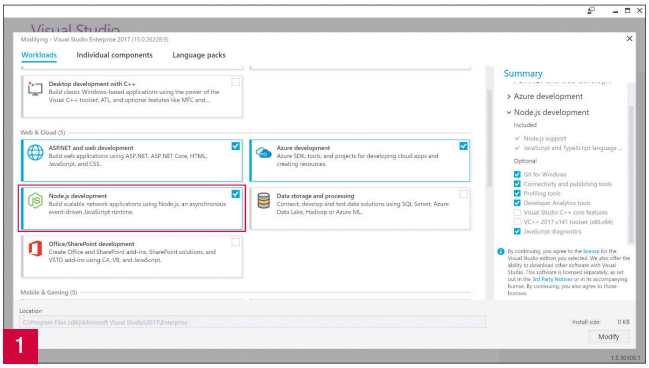
L'intelligence artificielle est pendant longtemps restée un concept qui relevait de l'imaginaire et de la science-fiction. C'est aujourd'hui une réalité, dont les domaines d'application sont nombreux : qu'il s'agisse du traitement du langage naturel, de la reconnaissance et du traitement d'images, de la robotique ou encore de l'analyse des émotions. Sans le savoir, nous utilisons au quotidien des services qui intègrent de l'intelligence artificielle : pour améliorer la sécurité des passagers, Uber a ainsi intégré dans son application un service de reconnaissance faciale pour permettre des contrôles inopinés de l'identité des conducteurs (qui doivent se prendre en photo avec leur smartphone et renvoyer l'image via leur application).
La stratégie de Microsoft est de rendre ces innovations accessibles et faciles à intégrer dans n'importe quelle application et quelle que soit la plateforme. Pour un développeur, Microsoft a à cœur de proposer des outils et des plateformes qui lui permettent de gagner en simplicité d'exécution et en productivité au quotidien. C'est l'ambition de Visual Studio 2017 : ouverture, intégration d'un maximum de technologies comme Node.js ou Python avec la suggestion de code, la détection de bugs, l'analyse du code ou le live unit testing. Visual Studio 2017 présente également des avantages pour les data scientists en offrant une plateforme de développement ouverte qui permet de manipuler des algorithmes, des datas et même l'infrastructure cCloud.

Quelques mots sur Microsoft Cognitives Services

Si vous souhaitez améliorer vos applications en y introduisant une part d'intelligence artificielle, alors Microsoft Cognitive Services est parfait. Microsoft Cognitive Services est un ensemble d'APIs, de SDKs et de services mis à disposition des développeurs pour qu'ils puissent rendre leurs applications plus intelligentes, plus engageantes et plus intuitives. Ils offrent une collection croissante de puissants algorithmes d'intelligence artificielle pour la vision, la dictée, le langage et la connaissance.
En intégrant des fonctionnalités intelligentes telles que la détection d'émotions, la reconnaissance faciale et vocale, la vision par ordinateur, la compréhension du texte et du discours, les développeurs peuvent proposer des interactions plus naturelles et contextuelles. L'expérience des utilisateurs est ainsi améliorée. L'objectif est que cette expérience soit plus personnelle afin d'améliorer la productivité.
Parmi les nombreuses nouvelles fonctionnalités qu'intègre Visual Studio 2017, la simplification de l'intégration des Microsoft Cognitives Services dans les applications Web, desktop ou mobile met à disposition de tous les développeurs l'IA leur offrant ainsi de nouvelles perspectives.
Prenons l'exemple d'un site Web de recherche de restaurant.
Aujourd'hui, les technologies de recherche intégrables à un site Web ne permettent pas de réaliser une analyse sémantique de la demande de l'utilisateur. Ainsi, la requête *je veux manger près de chez moi* ***tout sauf*** *japonais* va se résumer à une recherche des restaurants japonais qui seront présentés à l'utilisateur selon un critère de classement prédéterminé comme par exemple leur notation. Vous pouvez même tester cette phrase dans un moteur de recherche type Google ou Bing. La réponse fournie à l'utilisateur sera donc un contresens complet par rapport à ses intentions.
En utilisant le service LUIS (Language Understanding Intelligent Service), les développeurs vont pouvoir analyser la demande de l'utilisateur au préalable pour en retirer l'intention (ce que veut faire l'utilisateur) et les entités (les informations pertinentes concernant l'intention). Dans la phrase précédente, à l'issue d'un entraînement adéquat, LUIS sera capable d'identifier que l'utilisateur souhaite manger au restaurant (l'intention) et trois entités qui qualifient cette intention : le lieu, le type de cuisine et la négation. Cette dernière entité permet de savoir que les restaurants japonais doivent être exclus de la recherche.
Pour plus d'information sur la configuration d'une application LUIS : https://www.microsoft.com/cognitive-services/en-us/luis-api/documentation/home

Créer une application Web Node.js intégrant LUIS avec Visual Studio 2017

Bien sûr, Visual Studio 2017 améliore encore le support de JavaScript et TypeScript comme des langages de première classe. Mais Visual Studio 2017 va plus loin, Node.js est désormais aussi une plateforme cible standard et non plus supportée à travers une extension. Pour installer les outils nécessaires au développement Node.js, il suffit de sélectionner le *Workload* nommé *Node.js development* dans le tout nouvel outil de gestion des installations Visual Studio. [1]
Une fois les outils de développement Node.js installés, il suffit de créer un nouveau projet Node.js en utilisant par exemple le modèle de projet d'application Web Node.js Express 4. [2]
En quelques secondes, on dispose de la structure d'un site Node.js prêt à l'emploi. [3]. Pour intégrer le service LUIS dans l'application Node.js, il suffit d'installer le package npm en ligne de commande ou via l'interface de gestion des packages npm. [4]
Une fois que le package npm est installé, il n'y a plus qu'à le charger, configurer le client LUIS et appeler le service pour obtenir une analyse.

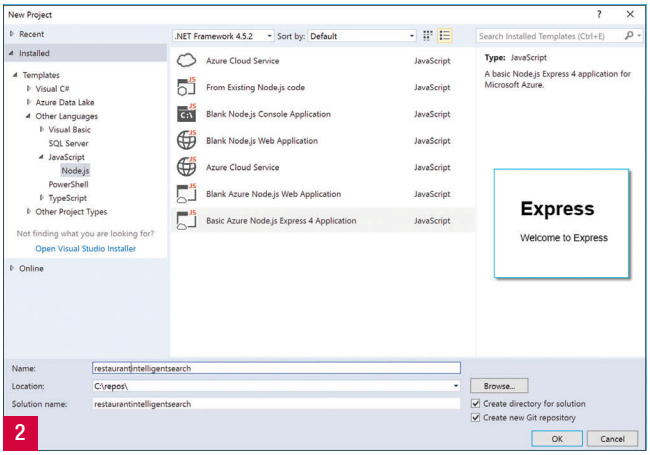


1

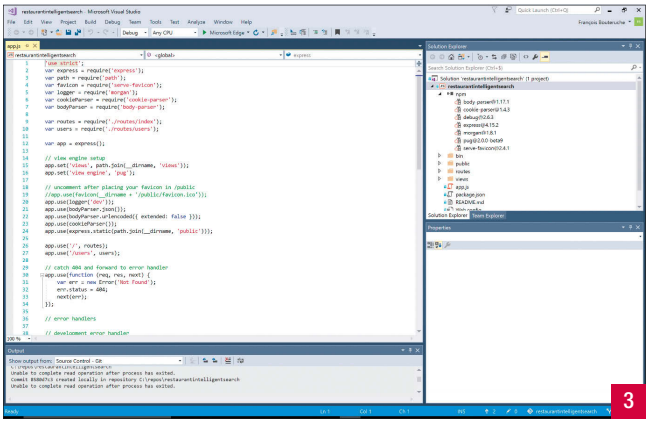
Le code suivant montre qu'en quelques lignes de code, on intègre ce puissant service de compréhension du langage dans son application Node.js. [5]. En cas de problème, on pourra facilement déboguer et analyser le contenu des réponses de LUIS avec par exemple la fenêtre permettant d'explorer le contenu des variables en mode debug, et ce, grâce à l'intégration de la plateforme Node.js à Visual Studio 2017.

Le process de data science dans Visual Studio 2017

Le développement d'un algorithme de data science se décompose en deux étapes : le nettoyage/manipulation des données, et la création du modèle à proprement parler. Microsoft va avoir une proposition de valeur à ces deux niveaux ; au niveau de l'analyse/ nettoyage des données avec une gestion simplifiée des bases de données, et le support des langages aidant la manipulation de données (SQL, R, python, F#...). Au niveau du développement du modèle à proprement parler, l'intégration de python vaut la peine que l'on jette un coup d'œil plus en détail. Regardons d'abord la manipulation de bases de données de taille conséquentes (pour l'exemple, stockées sur Azure) depuis VS. Un premier outil, si on stocke ses données en format SQL serveur est l'explorateur d'objets SQL serveur, qui permet de manipuler directement depuis l'IDE vos données. Si vos données sont stockées dans un data lake, il va également être possible d'interagir avec en un clic. Plus généralement, l'accès à son espace Cloud se fait par un explorer où l'on peut naviguer et manipuler ses fichiers. Passons maintenant à Python (pour le moment dans la preview, accessible gratuitement). L'apport qu'on remarque de prime abord est l'apparition d'options particulières au moment de créer son projet Python; tout comme il existe des template flask ou django pour créer ses applications Web, il existe maintenant trois templates liés directement au machine learning, comme par exemple le clustering, la classification, et la régression. Regardons un instant le contenu de la classe clustering; on y retrouve toutes les étapes du workload de machine learning : la fonctions permettant d'obtenir les données, celle qui convertit les données en array numériques, celle qui réduit la dimension du problème, puis le clustering à proprement parler, et, enfin, le plot pour l'interprétation des résultats. Quand on exécute l'exemple, à supposer que le kernel ait les bons packages, on tombe sur le résultat suivant : [6]. C'est un point de départ confortable pour implémenter ensuite pas à pas son processus de data science avec les tests unitaires à chaque étape, en modifiant dans l'ordre chaque fonction. Outre ce cadre agréable de développement pour sa pipeline, il faut citer la gestion des packages avec un vrai IDE, qui est une réelle force par rapport à une gestion *a mano* telle qu'on peut la connaître en développant sur des éditeurs de texte, et l'intégration du terminal dynamique et du notebook. •



2


3

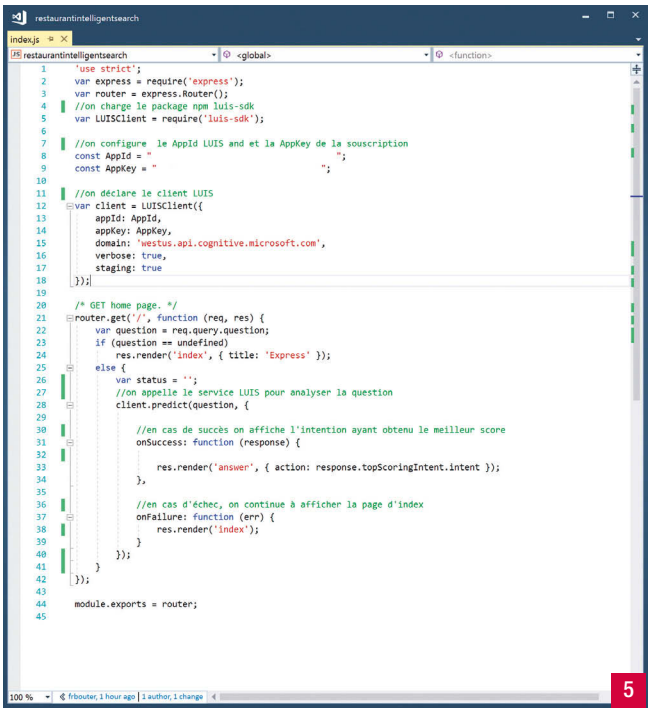
4

Pour aller plus loin :

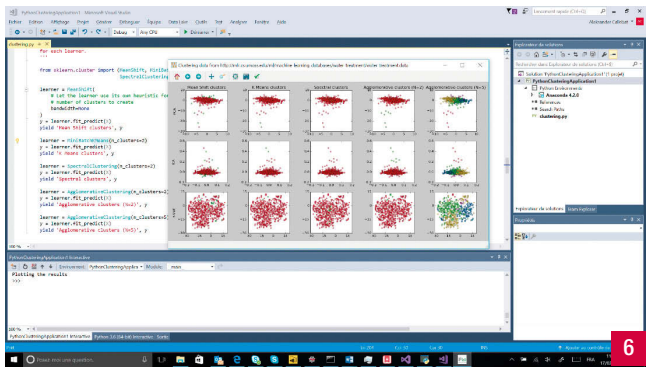
Concours d'IA collaborative dans MineCraft à destination d'étudiants en PHD : https://aka.ms/malmo-challenge

L'IA capable d'écrire du code : https://aka.ms/deepcoder

Visuel : https://blogs.msdn.microsoft.com/visualstudio/2017/03/13/visual-studio-2017-poster/

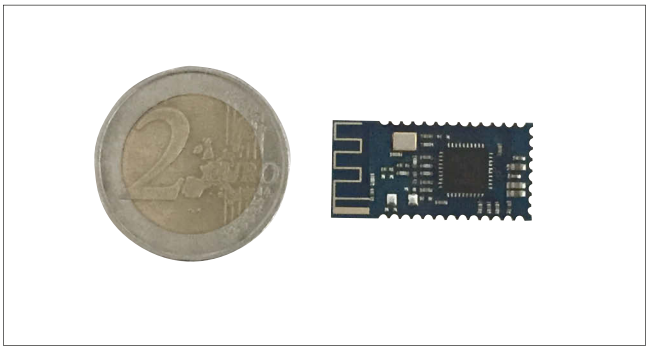


5


6

Module HM-10 : du Bluetooth LE pour votre Arduino ! Partie 1

Le module Bluetooth HC-05 est très réputé au sein de la communauté Maker : il permet de faire communiquer sans fil un montage électronique avec ordinateur par exemple. Toutefois il présente un problème de taille : il ne peut communiquer avec un iPhone, sauf en adhérant à l'onéreux programme MFi d'Apple, permettant de construire des appareil certifiés compatibles. Heureusement pour nous, le module HM-10 vient à la rescousse !

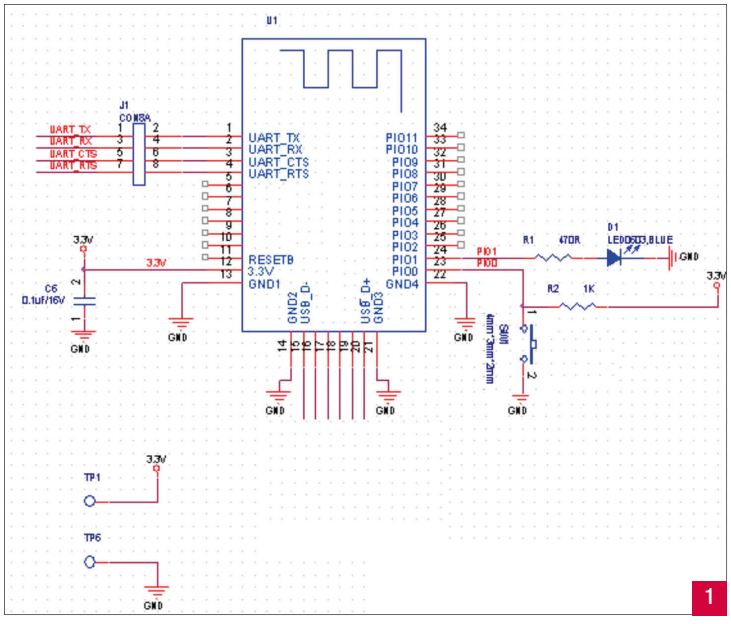


• Renaud Pinon

Le module *HM-10* est un proche cousin du module *Bluetooth HC-05*. Il suffit de les comparer l'un à côté de l'autre pour s'en rendre compte : même taille (absolument minuscule !), même disposition des connecteurs, même forme d'antenne, … seule la couleur change (vert pour le *HC-05* et bleu pour le *HM-10*). Le prix est même assez souvent similaire : entre 4 et 10 euros suivant les marchands. La réelle différence réside dans la capacité du *HM-10* à gérer le *Bluetooth Low Energy* (ou *LE*) qui est le seul protocole à fonctionner sans avoir à payer de coûteuse licence pour les appareils *iOS*. Cela ouvre de nouvelles perspectives à nos montages électroniques, avec la possibilité, par exemple, d'ouvrir une porte de garage à l'aide de son *smartphone* ou encore d'afficher sur un écran les coordonnées *GPS* actuelles du téléphone.

Configuration du module

Le moins que l'on puisse dire c'est qu'avec ses 34 connecteurs, *JNHuaMao*, le fabricant du *HM-10*, n'y est pas allé de main morte ! [1]
Rassurez-vous, la plupart sont des entrées/sorties. Nous n'utiliserons pour notre part que quatre connecteurs sur la gauche : les deux premiers (*TX* et *RX*) et les deux derniers (*VCC* - ou *3.3V* sur le schéma - et *GND*). Comme beaucoup d'autres modules du même genre, le *HM-10* fonctionne en connexion série *TTL* nécessitant simplement quatre fils : *Plus*, *Moins*, *TX* (transmission) et *RX* (réception).
Pour les besoins de cet article, nous allons connecter notre module à un *Arduino*, mais sachez que n'importe quelle carte pouvant communiquer en série via des broches *RX* et *TX* ferait l'affaire, comme un *Raspberry PI* par exemple.



1

Branchements

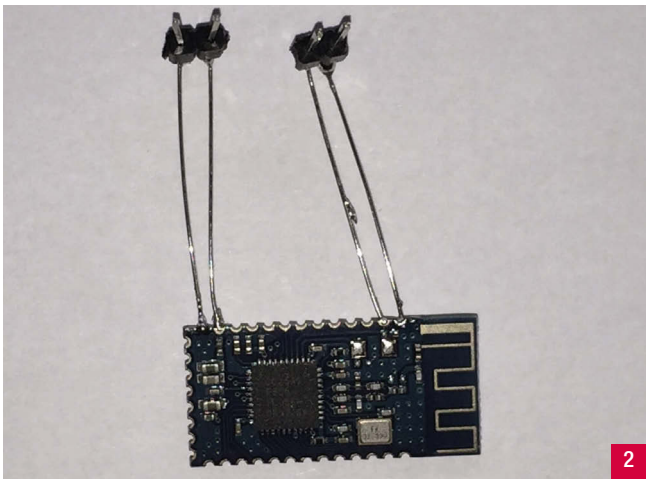
Nous allons connecter le module sur notre *Arduino* aux broches D10 et D11, puis utiliser la bibliothèque *SoftwareSerial* pour dialoguer. Nous aurions pu utiliser la bibliothèque *Serial* (tout court) en branchant *RX* et *TX* sur les broches 0 et 1 de l'*Arduino*, mais vous verrez par la suite que nous allons affecter ce port série à notre ordinateur afin d'envoyer des commandes au module par le moniteur série. Pourquoi les broches 10 et 11 ? Simplement car si (comme moi) vous possédez un *Arduino Mega*, vous ne pourrez utiliser la broche *RX* de *SoftwareSerial* que sur les entrées comprises entre 10 et 20 (les seules à gérer les interruptions matérielles), contrairement au modèle *UNO* où toutes les broches sont compatibles.

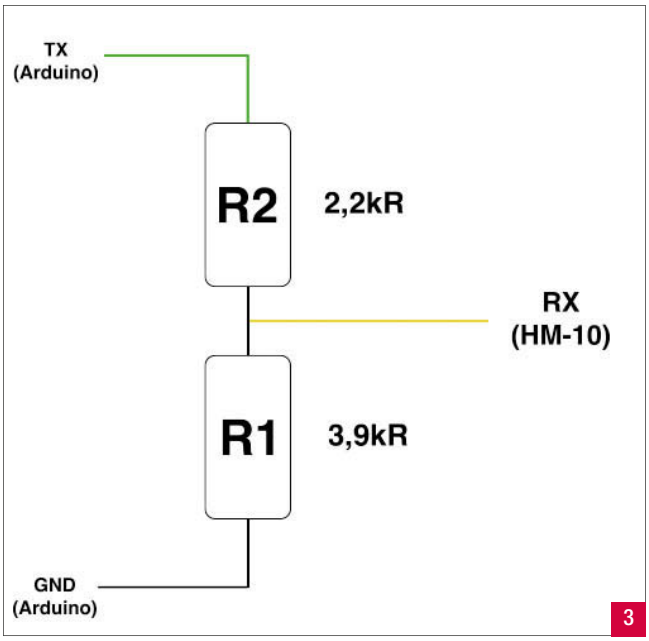
Le *HM-10* n'a pas de « broches » à proprement parler mais des pistes, la situation requiert donc de jouer du fer à souder pour en ajouter. Notez que moyennant un léger surcoût, vous pourrez acheter une version « *socket* » contenant le *HM-10* soudé sur un circuit présentant des broches classiques pour faire les branchements. Mais si comme moi vous n'avez pas la version *socket*, ou simplement si la place disponible est un facteur important dans votre projet, il va falloir chausser vos lunettes, vos yeux bioniques et vos plus beaux doigts de fée pour faire les soudures, car l'espacement entre les connecteurs est absolument minuscule. Personnellement, je coupe des pattes de résistances dont je sais que je ne me servirai jamais (5 ou 10 Mégohms par exemple), jette la résistance à la poubelle, puis soude les pattes aux connecteurs du module et enfin à des *headers* mâles. Un fois la soudure terminée, simplifiez-vous la vie : si vous avez un multimètre, réglez-le sur le mode « *continuité* » (icône avec une onde sonore), placez un de ses connecteurs sur *RX* et le second sur *TX*. Si le multimètre bipe (ou si la valeur affichée est différente de 1), c'est que vos

soudures se touchent et qu'il faut donc les refaire. Suivez la même procédure avec *VCC* et *GND* (les deux connecteurs du bas).
Pour ma part, voici le résultat de mes soudures. OK, ce n'est pas très jo-jo, mais ça fonctionne ;) [2]

Autre problème si vous vous passez de la version *socket* : le module fonctionne en 3,3V et n'est pas tolérant au 5V (3,7V maximum). Bien-sûr, nous allons l'alimenter avec la broche *3V3* de l'*Arduino*, cependant beaucoup de gens oublient que la broche *3V3* est la SEULE de toute la carte à envoyer du 3,3V : toutes les autres, lorsqu'elles sont configurées en sortie, envoient du 5V. Et le connecteur *RX* du *HM-10* n'est pas plus tolérant au 5V que son connecteur *VCC*.

Pour régler ce souci, vous pouvez utiliser un convertisseur de niveau entre le *TX* de l'*Arduino* et le *RX* du *HM-10* (pas besoin pour la broche *TX* du module : elle ne reçoit pas de courant). Mais si vous n'avez pas de convertisseur, le plus simple est encore de faire un diviseur de tension avec deux résistances. Le principe est simple : on branche le fil *TX* de l'Arduino sur une première résistance, elle-même suivie d'un câble connecté au *RX* du module, câble lui-même raccordé à une deuxième résistance, qui enfin est raccordée au connecteur *GND* de l'*Arduino*. [3]


2



3

Pour expliquer brièvement : on fait passer le courant au travers des résistances et suivant la proportion entre leurs valeurs (peu importe la valeur), le courant va être divisé. Par exemple, si on utilise deux résistances de 1 kOhm chacune, le courant sera exactement divisé par deux. Si l'une fait le double de l'autre, le courant sera divisé par 3 (ou multiplié par 2/3 suivant l'ordre). La chose importante à retenir est que le voltage perdu se transforme en chaleur et qu'il faut donc des valeurs de résistance suffisamment grandes pour éviter une surchauffe. Voici la formule exacte de la division de tension :

$$V_{sortie} = (V_{entrée} \times R1) / (R2 + R1)$$

J'ai pour ma part utilisé une résistance R1 de 3,9 kOhm et une résistance R2 de 2,2 kOhm, ce qui donne, au multimètre, 3,1V, soit assez proche de 3,3V. Notez que si vous n'avez que deux résistances identiques, cela devrait fonctionner quand même : vous obtiendrez 2,5V, ce qui devrait être suffisant pour déclencher une valeur *HIGH* chez le module.

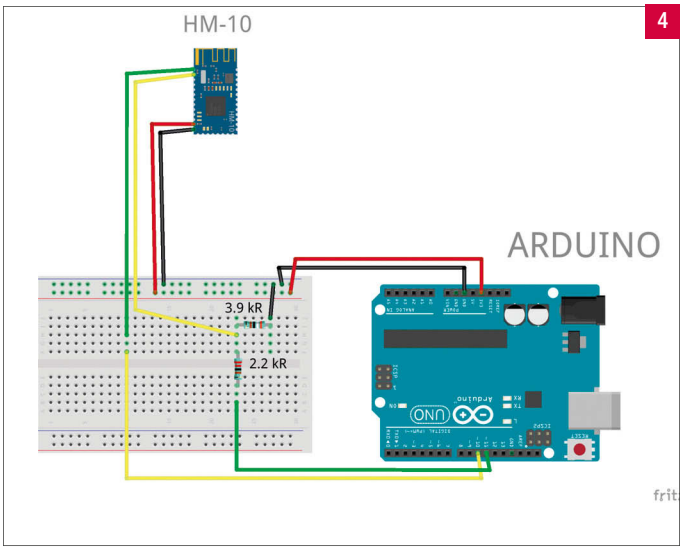
Voici à quoi ressemble le raccordement du *HM-10* avec un *Arduino* : [4]

Commandes AT

Le module est utilisable directement sans la moindre configuration. Mais nous allons le paramétrer pour découvrir les possibilités qu'il nous offre. Dans un premier temps, nous allons simplement changer le nom du module. Car oui, si vous possédez plusieurs modules l'un à côté de l'autre, autant pouvoir les différencier facilement. Dans un second temps - la partie II de l'article pour être précis - nous nous amuserons à paramétrer notre module en mode « *iBeacon* ». Ce mode, avec l'application adéquate sur votre *smartphone*, vous permet de savoir à quelle distance de votre téléphone se trouve le module *HM-10*.

Si vous connaissez l'*ESP8266* (module *WIFI*), la configuration d'un *HM-10* vous semblera familière : on envoie des commandes dites « *AT* » par la connexion série afin d'obtenir une valeur (*get*) ou de modifier une valeur (*set*). Si le module s'aperçoit que les données commencent par *AT+*, alors il interprète la suite comme étant une commande. Voici la syntaxe :

Obtenir la valeur d'un paramètre (commande *get*) :
AT+[param]?



4

Définir la valeur d'un paramètre (commande *set*) :
AT+[param]=[valeur]

Notez qu'il n'y a pas de retour chariot pour séparer les commandes. En envoyant à la suite les commandes *AT+VERSION?AT+NAME?*, vous recevrez, via la broche *TX* du module, le numéro de version du *firmware* immédiatement suivi du nom du module. Dernière chose à savoir : les commandes *AT* ne fonctionnent que si le module n'a actuellement pas de connexion *Bluetooth* établie avec un autre appareil sans fil.

Il n'y a pas de protocole *AT* officiel, donc chaque fabricant l'intègre comme il le souhaite dans son matériel. Cela soulève un problème auquel vous serez très probablement confronté : la contrefaçon. Il existe en effet une copie très présente sur le marché du module *HM-10* s'appelant le *CC41*. Si vous voulez savoir visuellement si vous possédez un *HM-10* ou *CC41*, consultez cette image : [5]

Le problème est que les sites de vente en ligne vous feront passer allègrement un *CC41* pour un *HM-10* officiel (et c'est d'ailleurs ce qu'il m'est arrivé). Même si les connecteurs sont strictement identiques, le protocole *AT* se comporte pour sa part très différemment.

Voici le résumé des différences :

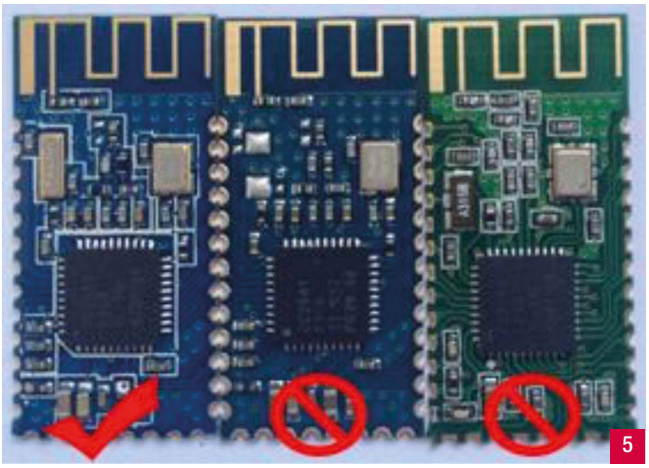
Obtenir les valeurs d'un paramètre	
HM-10	**CC41**
AT+[param]?	AT+[param][\r\n]

Définir les valeurs d'un paramètre	
HM-10	**CC41**
AT+[param]=[valeur]	AT+[param][valeur][\r\n]

Note 1 : [\r\n] = retour chariot (ASCII 13) suivi d'une nouvelle ligne (ASCII 10).
Note 2 : les commandes AT, AT+RESET et AT+RENEW sont les seules à être identiques pour les deux types de modules.

Sachez enfin que pour obtenir le listing de toutes les commandes *AT* de votre module, vous pouvez envoyer la commande :

Commande d'aide	
HM-10	**CC41**
AT+HELP?	AT+HELP[\r\n]


5

Ceci étant clarifié, il est désormais grand temps de programmer notre *Arduino* pour lui permettre d'envoyer les commandes *AT*. Voici le listing du code :

```
// Fichier HM10_config.ino

#include <SoftwareSerial.h>

// Defines:
#define kBTPinRx      10
#define kBTPinTx      11

// Variables globales:
SoftwareSerial _btSerial = SoftwareSerial(kBTPinRx, kBTPinTx);

// Initialisation:
void setup()
{
  // Démarre la communication série avec l'ordinateur :
  Serial.begin(9600);
  while (!Serial) { ; /* attente interface dispo */ }

  // Démarre la communication série avec le HM-10:
  _btSerial.begin(9600);
  while (!_btSerial) { ; /* attente interface dispo */ }
}

// Boucle principale:
void loop()
{
  // Lecture des données provenant du module Bluetooth :
  while (_btSerial.available() > 0)
  {
    Serial.print( (char)_btSerial.read() );
  }

  // Lecture des données envoyées depuis l'ordinateur:
  while (Serial.available() > 0)
  {
    _btSerial.write( Serial.read() );
  }
}
```

Vous pouvez retrouver ce code à l'adresse : https://github.com/renaudpinon/HM10. Examinons-le brièvement : nous déclarons un objet de type *SoftwareSerial* sur les broches 10 (*RX*) et 11 (*TX*). C'est là que sont branchées respectivement les broches *TX* et *RX* (avec le diviseur de tension) de notre module *Bluetooth*. Dans la fonction *setup()*, nous démarrons les communications série vers l'ordinateur et vers le module. La vitesse de communication par défaut du *HM-10* est de 9600 bauds, nous initions donc à cette vitesse l'interface série qui lui correspond (*_btSerial*). Hasard de la vie, nous utilisons la même vitesse pour la connexion série avec l'ordinateur (*Serial*).
Dans la fonction principale, nous vérifions s'il y a des données disponibles sur la connexion série du *Bluetooth* : si c'est le cas, nous transférons les données vers le port série relié à l'ordinateur. Pour terminer, nous faisons la même chose avec les données provenant de la connexion série de l'or-

dinateur afin qu'elles soient transférées au module *Bluetooth*.
Notez que pour ce programme, peu importe que vous ayez un *HM-10* ou un *CC41*. La distinction se fera au moment d'envoyer les données. Je m'explique : nous allons utiliser le moniteur série d'*Arduino IDE* afin d'envoyer manuellement chaque commande et nous paramètrerons le moniteur suivant le type de module.
Pour cela, envoyez le code précédent dans votre *Arduino* grâce à *Arduino IDE*, puis allez dans le menu *Outils / Moniteur série*. Une nouvelle fenêtre s'ouvre : réglez la première liste déroulante en bas à droite sur « *Pas de fin de ligne* » si vous avez un *HM-10* ou sur « *Les deux, NL et CR* » si vous avez un *CC41* (cela évite d'avoir à taper le *[\r\n]* à chaque fin de commande). Réglez la deuxième liste sur « *9600 bauds* ». Fermez ensuite le moniteur série et rouvrez-le, pour être sûr que vos modifications soient prises en compte. Tapez « *AT* » dans la zone de texte du haut, puis cliquez sur le bouton « *Envoyer* » : le mot « *OK* » doit alors s'afficher sur une nouvelle ligne. Si ce n'est pas le cas, plusieurs possibilités :

- Vous avez cru avoir un *HM-10* alors que vous avez un *CC41*
- Vous avez mal branché la broche *RX* ou *TX* de votre module : vérifiez que vous n'avez pas interverti les deux ! Le *RX* de l'Arduino va sur le *TX* du module et vice-versa.
- Un ou plusieurs connecteurs du module sont mal soudés. J'ai remarqué qu'avant qu'une donnée ne soit envoyée au module, un courant passe dans chacune des broches *RX* et *TX*. C'est le bon moment pour user du multimètre : réglez-le sur la valeur immédiatement supérieure à 5V continu, placez le connecteur noir sur *GND* du module et le rouge sur *VCC* : si le module est alimenté vous devez lire environ 3,3V. Placez ensuite le rouge sur *TX* : vous devez lire 3,3 V également. Placez enfin le connecteur rouge du multimètre sur RX : il doit afficher une valeur entre 2,5 et 3,7V (dépend des valeurs de résistances choisies pour le diviseur de tension). Si une de ces valeurs est 0, c'est qu'il y a peut-être un problème de soudure.
- Vous pouvez « debugger » votre circuit avec des LEDs : pour des raisons de simplicité je n'en ai pas inclus dans le montage mais rien ne vous empêche d'en mettre deux : une pour la réception de données du module et une pour l'envoi de données. Dans votre code Arduino, allumez simplement la led de réception si _btSerial.available() > 0 et allumez la LED d'envoi si Serial.available() > 0, autrement éteignez-les.

Changement de nom

Maintenant que nos commandes *AT* fonctionnent, toujours dans le moniteur série, nous allons modifier le nom de notre module. Par défaut il doit s'appeler *HM-10-quelque-chose* ou *CC41-quelque-chose*, mais nous allons le renommer en *Programmez0001*. Pour cela, nous allons simplement taper la commande :

HM-10	CC41
AT+NAME=Programmez0001	AT+NAMEProgrammez0001

Le module doit répondre le message suivant :

```
+NAME=Programmez0001
```

Rebootez ensuite le module grâce à la commande *AT+RESET*.
Pour vérifier que le paramètre est bien resté en mémoire, faites la commande :

HM-10	CC41
AT+NAME?	AT+NAME

La réponse doit être la même que précédemment. Bravo, vous avez modifié le nom du module :)
Nous allons maintenant télécharger sur notre smartphone une application gratuite permettant de communiquer avec le module afin de voir s'il est détectable. Sur *iPhone*, recherchez dans l'*App Store* l'application *HM10 Bluetooth Serial* d'*Alex Van Der Lugt*. Sur *Android*, vous pouvez télécharger l'application s'appelant *Bluetooth 4.0 BLE for Arduino* ou *Bluetooth Terminal*. Ces applications vous permettent d'afficher une liste des modules *Bluetooth* détectés à proximité et de vous y connecter pour recevoir ou envoyer des données (ce que nous ferons dans la partie II avec notre propre application sur *iOS*).
A noter que si vous avez déjà tenté de faire détecter le module par un programme similaire sur votre *smartphone* avant d'avoir changé le nom, il est très probable que ce dernier ne soit pas modifié. Pas de panique : le problème ne vient pas de votre module mais bien de votre téléphone qui a l'insupportable habitude de tout mettre en cache. Si vous rencontrez ce problème, débranchez et rebranchez votre *Arduino*, puis désactivez / réactivez le Bluetooth sur votre téléphone. Si le nom n'a toujours pas changé (ou s'il disparait), éteignez votre téléphone et attendez une ou deux minutes, puis rallumez-le. Si le nom correct n'apparaît toujours pas, j'ai remarqué que parfois, en se connectant au module dans l'application téléchargée sur le téléphone, le retour à la liste provoque un rafraîchissement du nom par le système.

CONCLUSION

Nous avons donc pu voir aujourd'hui les spécificités matérielles du module *HM-10* et de son perfide clone le *CC41*. Nous avons pu résoudre les difficultés de connexion que l'on peut rencontrer en passant du 5V au 3,3V, faire un montage *Arduino* fonctionnel, mais aussi voir quelques astuces bien utiles au moment de la soudure (je pense notamment au mode *Continuité* des multimètres, méconnue mais tellement pratique !).
Dans le prochain épisode, nous paramètrerons le module en tant qu'*iBeacon*, histoire de le retrouver au mètre près parmi tout bazar ambient qui se respecte. Et dans un registre différent, nous créerons une application *iPhone* en *Objective-C* nous permettant d'échanger des données avec notre module *Bluetooth*. •

La suite dans Programmez ! 207

arduino (clone)

Optimiser l'utilisation de la mémoire avec Doctrine

• Sophie Sound
ingénieur développement
Osaxis

Dans un projet Symfony, nous avons tous déjà rencontré la fameuse erreur « allowed memory size exhausted » lors de récupérations de données avec Doctrine. Quand cela arrive, nous essayons d'optimiser la requête, puis, à court d'idées, avons la fâcheuse tendance à vouloir augmenter le paramètre « memory_limit » de PHP. Cette erreur survient le plus souvent lors de l'hydratation des données. L'hydratation est le processus qui permet de convertir en objets les résultats provenant d'une base de données à partir d'une requête ; cela peut être très consommateur au niveau de la mémoire. Nous allons donc présenter dans cet article une façon de réduire l'utilisation de la mémoire en modifiant la façon dont Doctrine récupère les résultats.

Récupération et exportation de données

Nous allons partir d'une application backend gérant des produits. Cette application permet de lister tous les produits présents en base de données à l'aide d'une pagination et propose une fonctionnalité d'exportation au format CSV. Pour faire très simple, nous avons ici une entité « Product » contenant deux attributs, un numéro, le code du produit et un montant, le prix du produit.

```
// Entité Product
/**
 * @var int
 *
 * @ORM\Column(name="number", type="integer")
 */
private $number;

/**
 * @var string
 *
 * @ORM\Column(name="amount", type="decimal", precision=10, scale=2)
 */
private $amount;
```

En base de données, la table contient 20 000 lignes. Nous implémentons ensuite la fonctionnalité d'exportation, pour pouvoir récupérer et parcourir l'ensemble des données.

```
// Controller
public function exportAction(Request $request)
{
    // on récupère ici tous les produits
    $products = $this->getDoctrine()->getRepository('MemoryBundle:Product')->getAll();

    // on appelle ici notre fonction d'exportation
    $file = $this->export($products);

    // puis on retourne la réponse (téléchargement d'un fichier « export.csv »)
    $response = new BinaryFileResponse($file);
    $disposition = $response->headers->makeDisposition(
        ResponseHeaderBag::DISPOSITION_ATTACHMENT,
        'export.csv'
    );
    $response->headers->set('Content-Disposition', $disposition);

    return $response;
}

private function export($products)
{
    $filename = 'path/export.csv';
    $fp = fopen($filename, 'w');
    foreach ($products as $product) {
        /* ... écriture dans le fichier ... */
    }
    fclose($fp);

    return $filename;
}
```

Pour l'exportation, il faut d'abord récupérer tous les produits et c'est ici qu'a lieu l'hydratation et la saturation de la mémoire. Pour ce faire, nous présentons ici quatre façons de récupérer cette liste avec un QueryBuilder :

- « getResult »
- « getResult » avec « HYDRATE_ARRAY »
- « iterate »
- « iterate » avec « HYDRATE_ARRAY »

Pour comparer les résultats, nous utilisons la barre du développeur de Symfony3, disponible en bas de chaque page sur l'environnement de développement, dans l'onglet « Performance ».

Comparaison des méthodes

L'appel à « getResult » sans changement sur les autres clauses est un équivalent à « findAll », la méthode la plus classique pour retourner une liste. Elle s'utilise de la manière suivante :

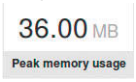
```
// Repository
public function getAll()
{
```

```
    $qb = $this->createQueryBuilder('p');
    $qb->select('p');

    return $qb->getQuery()->getResult();
}
```

Une fois la liste récupérée, nous devons la parcourir pour générer notre fichier CSV :

```
// Méthode export
private function export($products)
{
    /* ... ouverture du fichier ... */
    foreach ($products as $product) {
        fputcsv($fp, [
            $product->getNumber(),
            $product->getAmount(),
        ]);
    }
    /* ... fermeture du fichier et retour du nom de fichier ... */
}
```

Après l'appel à la fonction d'exportation, la barre du développeur nous indique ceci :



Utilisation de la mémoire avec « getResult »

Un objet étant plus coûteux à hydrater qu'un tableau, nous pouvons indiquer à Doctrine de nous retourner les résultats sous forme de tableau. C'est le paramètre « HYDRATE_ARRAY » qui va nous permettre de faire cela. On peut aussi noter que « getResult(Query::HYDRATE_ARRAY) » est équivalent à la méthode « getArrayResult() ».

```
// Repository
public function getAll()
{
    /* ... récupération du QueryBuilder ... */

    return $qb->getQuery()->getResult(Query::HYDRATE_ARRAY);
}
```

Puisque nous avons maintenant un tableau, la méthode « export » est également changée. Au lieu des appels aux méthodes « getNumber » et « getAmount », on accède aux clés « number » et « amount » :

```
// Méthode export
private function export($products)
{
    /* ... ouverture du fichier ... */
    foreach ($products as $product) {
        fputcsv($fp, [
            $product['number'],  // on récupère la donnée par la clé et non par un getter
            $product['amount'],  // idem ici
        ]);
    }
    /* ... fermeture du fichier et retour du nom de fichier ... */
}
```

La barre du développeur nous indique désormais :

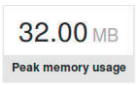
Utilisation de la mémoire avec « getResult » et « HYDRATE_ARRAY »

Nous sommes passés de 36 Mo à 14 Mo, mais que ce soit sous forme d'objets ou en passant par des tableaux, « getResult » hydrate toute la liste dès la récupération. Allons donc plus loin avec une méthode qui va nous permettre d'hydrater seulement lorsque l'on itère dessus : « iterate ». Il suffit juste de remplacer l'appel à « getResult » par « iterate » pour l'utiliser. Pour comparer les deux méthodes, effectuons dans un premier temps un appel sans le paramètre « HYDRATE_ARRAY » :

```
// Repository
public function getAll()
{
    /* ... récupération du QueryBuilder ... */

    return $qb->getQuery()->iterate();
}
```

Un itérateur étant retourné, la méthode « export » doit elle aussi être changée :

```
// Méthode export
private function export($products)
{
    /* ... ouverture du fichier ... */
    foreach ($products as $row) {
        $product = $row[0];
        fputcsv($fp, [
            $product->getNumber(),
            $product->getAmount(),
        ]);
    }
    /* ... fermeture du fichier et retour du nom de fichier ... */
}
```



Utilisation de la mémoire avec « iterate »

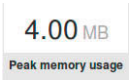
Faisons également le test en ajoutant le paramètre « HYDRATE_ARRAY » à la méthode « iterate » :

```
// Repository
public function getAll()
{
    /* ... récupération du QueryBuilder ... */

    return $qb->getQuery()->iterate([], Query::HYDRATE_ARRAY);
}
```

De la même manière qu'avec « getResult » avec « HYDRATE_ARRAY », nous récupérons un tableau :

```
// Méthode export
private function export($products)
{
    /* ... ouverture du fichier ... */
    foreach ($products as $row) {
        $product = $row[0];
        fputcsv($fp, [
            $product['number'],
            $product['amount'],
```

```
    ]);
  }
  /* ... fermeture du fichier et retour du nom de fichier ... */
}
```



Utilisation de la mémoire avec « iterate » et « HYDRATE_ARRAY »

On remarque que c'est bien l'hydratation en tableau qui permet d'optimiser grandement la mémoire. Combinée avec la méthode « iterate », la mémoire utilisée est passée de 36 Mo, avec « getResult », à 4 Mo. Dans notre cas, nous avons une réduction de 89 %. Avec des données plus conséquentes et plus réalistes, le gain en Mo peut même être encore plus important. L'explication est que « getResult » garde en mémoire toute la liste (puisque l'hydratation se fait juste après la récupération) alors que « iterate » va les hydrater un à un lorsque l'on itère dessus. Les objets pour lesquels on a déjà itéré n'étant plus utiles, PHP peut libérer la mémoire allouée au fur et à mesure. Quant au paramètre « HYDRATE_ARRAY », son atout vient du fait qu'un tableau est plus simple à hydrater qu'un objet.

Méthode	Mémoire utilisée
getResult	36 Mo
getResult avec HYDRATE_ARRAY	14 Mo
iterate	32 Mo
iterate avec HYDRATE_ARRAY	4 Mo

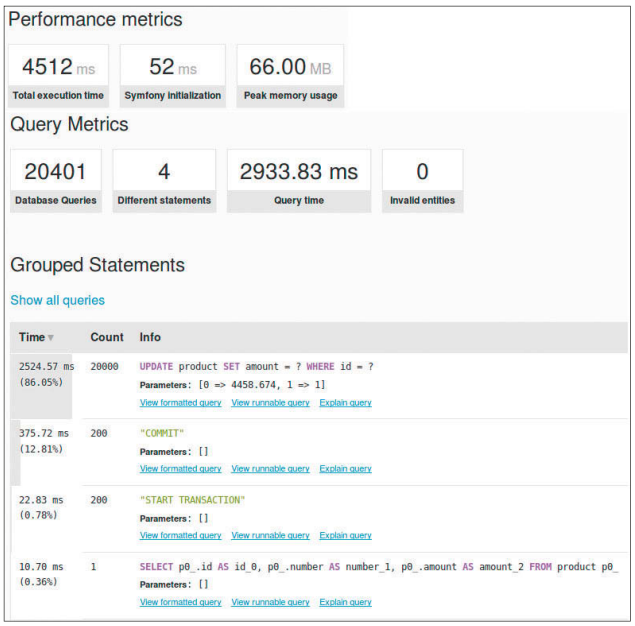
Récapitulatif de l'utilisation de la mémoire

Insertion, modification et suppression en masse

Les optimisations peuvent également se faire sur les opérations d'insertion, de modification et de suppression en masse. Prenons l'exemple d'une modification des prix des produits. La liste est récupérée avec le code du « iterate » sans paramètre :

```
$products = $this->getDoctrine()->getRepository('MemoryBundle:Product')->getAll();
$batchSize = 100;
$i = 0;
foreach ($products as $row) {
  $product = $row[0];
  $product->setAmount($product->getAmount() * 1.1);
  ++$i;
  if (($i % $batchSize) === 0) {
    $em->flush(); // exécution des mises à jour
    $em->clear(); // détache les entités de Doctrine
  }
}
$em->flush(); // exécutions des dernières mises à jour
```

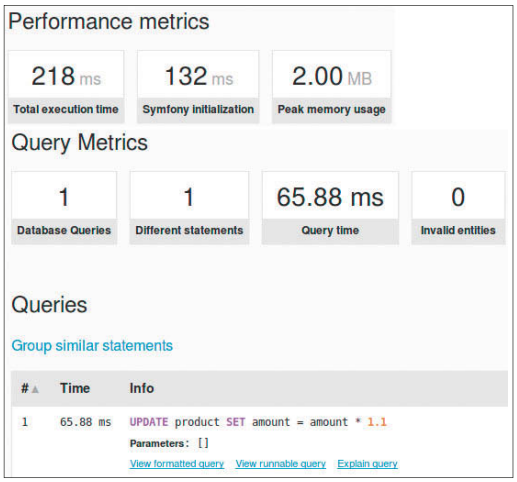
Dans cet exemple avec un QueryBuilder, on a une requête UPDATE pour chaque enregistrement de la base de données. Sachant qu'il y a 20 000 produits, nous avons autant de requêtes UPDATE, ce qui se traduit par une consommation accrue de la mémoire mais également une augmentation des temps de réponse.



Performance de mise à jour avec QueryBuilder

La méthode la plus efficace pour la mise à jour en masse est l'utilisation du DQL (Doctrine Query Language) qui va nous permettre d'exécuter une seule requête d'UPDATE en base.

```
$q = $em->createQuery('UPDATE MemoryBundle:Product p SET p.amount = p.amount * 1.1');
$q->execute();
```



Performance de mise à jour avec DQL

CONCLUSION

D'un point de vue général, retourner les résultats sous forme de tableau est donc beaucoup plus intéressant au niveau mémoire utilisée que sous forme d'objet, et le DQL est à privilégier pour les opérations de masse. Attention toutefois, « HYDRATE_ARRAY » et « iterate » ne sont pas à utiliser systématiquement. En effet, « HYDRATE_ARRAY » est plus adapté pour des fonctionnalités de lecture seule (comme ici l'exportation). Un tableau étant retourné à la place d'un objet, les fonctionnalités de l'entité (lazy loading, persistance, mise à jour, …) ne sont plus disponibles. Quant à « iterate », les collections (toMany) ne peuvent pas être récupérées et il se peut que la connexion à la base de données garde tous les résultats dans un tampon utilisant de la mémoire supplémentaire non visible par PHP. Pour les autres pistes d'optimisation plus poussées (mémoire, temps de réponse, etc.), on peut se rendre sur la documentation officielle : http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/improving-performance.html •

Un **support client optimisé** en connectant la VoIP à votre Back Office Magento Partie 1

• Jean-Marie Heitz,
Ingénieur études et développement e-commerce chez **Netapsys Grand Est**
Jean-marie.heitz@netapsys.fr

Jusqu'à assez récemment, on trouvait généralement des systèmes informatiques et des systèmes téléphoniques qui étaient cloisonnés. Pourtant, le couplage de ces deux peut fonctionnellement être assez intéressant, notamment pour des logiciels E-Commerce comme Magento : c'est le Couplage Téléphonie Informatique (CTI). Ainsi nous allons tout d'abord voir certains aspects fonctionnels du CTI, puis passer du côté de la technique en approchant les solutions d'interconnexion proprement dites, pour enfin remonter à un niveau applicatif plus élevé, en voyant que les solutions peuvent être aussi bien centralisées (par exemple avec Asterisk) que plus diffuses dans l'environnement (les possibilités des différents équipements SIP), voire même intégrées dans un environnement web.

Mais pour l'heure, commençons tout d'abord par certains aspects fonctionnels des solutions CTI. [1]

CTI : Entrevue des aspects fonctionnels

Le Couplage Téléphonie Informatique est, comme on vient de le dire, l'établissement de synergies entre deux types de systèmes qui n'étaient, jusqu'à assez récemment, peu ou pas interconnectés dans les entreprises, à savoir :

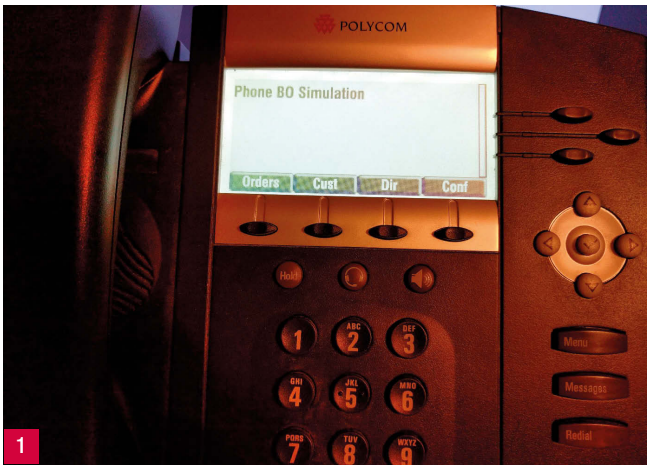
- Les systèmes informatiques, souvent vus comme le monde de la donnée, de la "data" ;
- Les systèmes téléphoniques, avec principalement les PABX, qui sont devenus des IPBX en intégrant des protocoles tels que SIP, SDP et RTP.

De cette explication vaporeuse, on comprend que ce couplage peut avoir lieu à de multiples niveaux, et peut fournir des fonctionnalités diverses. Par exemple, pour les niveaux :

- Au niveau de l'ordinateur : affichage d'informations métier sur l'appel en cours, sur le PABX, déclenchements d'appels…
- Au niveau du téléphone : affichage d'informations métier directement sur le téléphone, déclenchement de traitements métier depuis le téléphone…
- Au niveau du PABX : routage de l'appel en fonction du contexte métier, enrichissement des données métier avec les données statistiques du PABX, déclenchement d'appels sortants vers des cibles métiers.
- Au niveau d'un document HTML : intégration de liens avec des URI du schéma "tel".

En ce qui concerne l'intérêt spécifique de coupler la téléphonie avec Magento, il est assez intuitif : Magento est à la base un logiciel de commerce électronique. Cependant, il intègre à ce titre, en particulier dans sa partie Back Office, un grand nombre d'éléments qui ne sont pas exclusivement du ressort du commerce électronique :

- Une gestion des clients, avec leurs adresses, leurs groupes de clients, les commandes passées,… : on retrouve donc certains aspects CRM ;
- Une gestion des stocks, pour éventuellement éviter de vendre des articles qui pourraient ne plus être en stock : ici, on voit la disponibilité de certaines fonctionnalités d'ERP ;
- Une gestion des flux de commandes et de facturation des commandes : on a donc également des fonctionnalités liées aux logiciels de comptabilité ;
- Des rapports sur les ventes générées : on a donc des prémices d'intelligence décisionnelle.

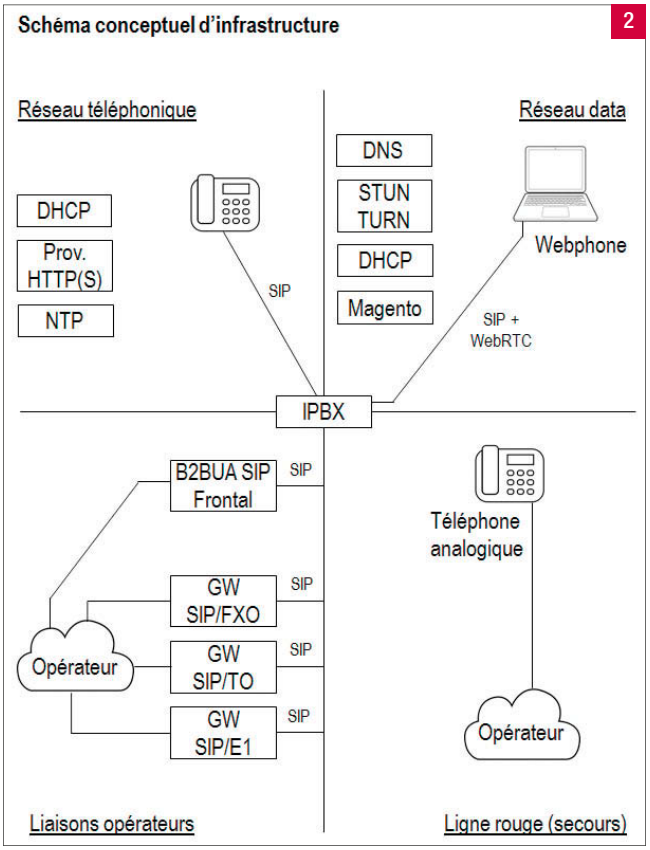


1

Exemple de simulation d'écran d'accueil de BO sur téléphone Polycom SoundPoint IP 450

On comprend donc que Magento peut devenir, pour certaines entreprises, un outil central de gestion, et qu'il peut donc être très intéressant d'y ajouter des services qui permettent aux employés de gagner en confort et en rapidité d'exécution de certaines tâches, voire fournir de nouveaux services qui peuvent aider à générer de la valeur ajoutée :

- Facilitation de communication pour un agent : affichage d'informations, déclenchement de processus métiers sur l'un des deux périphériques, annuaire métier disponible depuis plusieurs supports…
- Facilitation de communication pour un client sans pour autant mobiliser de suite une personne physique : analyse de la situation du client, et proposition d'aller rapidement au bon interlocuteur (par exemple : commercial pour une commande pas encore livrée ou un problème de paiement, technique pour un article livré récemment…), diffusion contextuelle de messages métiers préenregistrés…
- Facilitation du suivi du client : alors qu'un mail laisse des traces de lui-même étant un document papier, un appel ne peut laisser des traces que dans le PABX : d'où une utilité évidente pour le suivi du client de loguer ces interactions dans le logiciel métier
- Facilitation de la démarche commerciale : coordination des équipes de centres d'appels, aide à la préparation de campagnes téléphoniques en fonction des rapports métiers (par exemple sur les paniers abandonnés des clients logués) ; simplification de la prise de commandes par téléphone…



Toutefois, intégrer la téléphonie avec l'application Web présente des défis et des opportunités qui s'ajoutent aux problématiques usuelles de CTI :

- L'application est centralisée, ce qui constitue une facilité de maintenance puisqu'il n'y a pas besoin de mettre à jour l'ensemble des postes clients lorsque l'on effectue une amélioration ;
- Les téléphones sont généralement situés sur les postes de travail des utilisateurs : il faut donc faire la liaison entre les deux dimensions ;
- La création d'une interface Web est plus "rigide" et plus compliquée qu'un client lourd installé sur un poste : on passe toujours par le serveur Web qui fait office de serveur d'applications ;
- Magento nécessite la consultation de nombreuses pages Web, ce qui introduit une complexité sur la persistance du service de CTI au travers des pages visitées.

Afin de mieux comprendre comment l'ensemble des composants et des technologies peuvent s'architecturer, je propose un schéma conceptuel d'infrastructure tel que suit :

- La pierre angulaire de l'architecture est le PABX (ou IPBX), qui fait la liaison entre trois réseaux : le réseau de téléphonie, le réseau data, et les liaisons opérateurs ;
- Le réseau téléphonie est un réseau constitué majoritairement de téléphones SIP dédiés. De ce fait, il faut aux téléphones le nécessaire pour se configurer et se connecter au PABX ;
- Le réseau Data : c'est un réseau un peu particulier, car il est effectivement consacré au réseau informatique au sens large (machines serveurs et postes clients). Comme le cas des softphones SIP est très ressemblant à celui des hardphones SIP, je ne les considère pas sur le schéma. En revanche, je considère sur le schéma le cas d'un webphone, et l'infrastructure spécifique qui est requise. Petite précision par rapport au STUN/TURN : la plupart des systèmes exigent ces composants. En théorie, un serveur STUN devrait plutôt se situer sur un réseau où il dispose de deux IP publiques. Dans le cas présent, je suppose que le PABX a une connexion réseau dans le réseau data, ce qui explique sa position.
- Les liaisons opérateurs : par simplicité, j'ai considéré quatre types de liaison qui sont généralement utilisées, et dont je parlerai par la suite dans l'article ; on notera cependant que des sorties "plus exotiques" existent également (par exemple une passerelle permettant de sortir en GSM, ou la connexion à un téléphone portable via du bluetooth sur certains PABX tels qu'Asterisk) ;
- La ligne rouge de secours : les entreprises qui ont des employés ont des obligations légales et morales de sécurité envers ces derniers. On constate sur le schéma que les installations ne sont pas triviales, et complexes, alors que le fait de pouvoir appeler peut être par moment une réelle question de vie ou de mort. De ce fait, je pense qu'il est opportun de garder une véritable solution de secours, comme par exemple dans le schéma, une ligne analogique (cuivre), isolée du reste du système (bien entendu, il faudra chercher des solutions adaptées dans les cas qui se poseront en réalité). Et j'ajoute d'ailleurs à ce propos une petite réflexion par rapport à un cas qui m'est déjà arrivé : il ne faut pas oublier qu'un point qui est également critique avec la VOIP est l'approvisionnement en électricité pour faire marcher tous les équipements. Avec un téléphone analogique sur ligne cuivre, c'est l'opérateur qui fournit le courant nécessaire au téléphone pour fonctionner. Autrement dit, il continue de fonctionner en cas de coupure de courant, ce qui est un avantage parfois appréciable pour appeler les services de dépannage de son opérateur d'électricité... [2]

Nous allons voir d'abord les solutions d'interconnexion au réseau téléphonique, puis des possibilités de couplage téléphonie informatique fournies par un IPBX, avant de considérer celles liées aux téléphones, et certaines spécificités que l'on peut rencontrer lorsque l'on cherche à intégrer la téléphonie avec une application Web telle que Magento.

Introduction aux solutions d'interconnexion

INTERFACES [3]

Lorsque l'on parle de couplage téléphonie informatique, il faut en premier lieu effectivement réussir à interconnecter les deux types de réseaux, et obtenir des informations sur l'appelant et éventuellement l'appelé. Pour ce faire, deux types de solutions existent :


3
Photo de carte de téléphonie X101P (1 FXO)

- Utiliser un opérateur qui propose des trunks SIP et une interconnexion au réseau téléphonique : cette solution peut paraître séduisante, car elle déporte une bonne partie de la problématique vers l'opérateur. Néanmoins, elle a également des inconvénients : tout d'abord, on ouvre le réseau informatique pour faire les accès pour la VOIP. Cette ouverture de réseau ne doit pas se faire n'importe comment : il faut sécuriser l'ouverture du flux, en utilisant au minimum uniquement des connexions sécurisées (SIPS, SRTP,...) si ce n'est en utilisant des firewalls applicatifs (et dans ce cas, un chiffrement applicatif devient plus ardu) et des connexions VPN. D'autre part, il y a une problématique de QoS : en effet, on arrive facilement à contrôler la qualité de service des paquets sortants de

son infrastructure, mais on peut plus difficilement jouer sur la QoS des paquets entrants. Si l'on passe sur des ressources partagées, il peut y avoir contention sur la ressource et le service peut en pâtir. Pour être théoriquement viable, la QoS devrait s'appliquer de bout en bout, jusqu'à l'opérateur VOIP ; ce qui veut dire au minimum des contrats spécifiques. Une autre alternative est l'overprovisionning, c'est à dire avoir des surcapacités pour réussir la plupart du temps à absorber les pics de trafic. Enfin, il y a très souvent des problématiques de NAT, puisque le protocole SDP sous-jacent va utiliser les IP et ports pour indiquer les points de connexion des média. Cette problématique n'est pas uniquement du ressort de l'opérateur, car l'usager devra également avoir des clients SIP qui sont capables de mettre en œuvre les solutions techniques que l'opérateur propose (notamment STUN, TURN et ICE).

- Se connecter via des interfaces télécoms au réseau de l'opérateur. Cette solution permet de mieux dissocier le réseau téléphonique et le réseau informatique, et pourrait en théorie permettre des communications de qualité plus stable, dans la mesure où l'on cherche à passer non pas sur des réseaux à commutation de paquets, mais à commutation de circuits (établissement de circuits temporaires ou permanents entre deux points). Au niveau des types d'interfaces disponibles, on en trouve majoritairement deux/trois types :
  - Les interfaces FXO/FXS (Foreign eXchange Office / Foreign eXchange Subscriber) : ce sont des interfaces analogiques. Les interfaces FXO se branchent à l'opérateur, tandis que les interfaces FXS se branchent à des téléphones (ou équivalent). Ce sont des interfaces où la signalisation et la voix sont associées sur le même support, qui véhiculent actuellement le numéro de téléphone via des séquences de croisement de fréquences identifiant des numéros, appelées DTMF. La signalisation à l'utilisateur est assurée via des séquences sonores, tandis que l'identifiant du numéro et/ou du nom se fait initialement via l'envoi de données au début de l'appel, typiquement lors des sonneries. Lorsqu'une telle interface est utilisée pour se connecter à un opérateur, un seul numéro de téléphone est attribué.
  - Les interfaces BRI et PRI (Basic / Primary Rate Interface) : ces interfaces utilisent des codages comme Alternate Mark Inversion ou High Density Bipolar 3 qui véhiculent dans une certaine mesure un signal de synchronisation, puisque l'on fait par exemple varier le niveau du signal pour la valeur 1 (AMI) et que l'on casse une série de zéro en insérant une valeur 1 à intervalle régulier (viol de parité pour HDB3) pour ne pas perdre la synchronisation, ce qui leur permet d'atteindre des débits plus élevés que le codage V23 utilisé dans certains cas pour le callerId sur le FXO. Ce genre d'interfaces présente des trames de longueur fixe, avec des emplacements pour deux types de canaux : les canaux voix (canaux B) et le canal de signalisation (canal D). Le nombre de canaux B et D est différent en fonction de la technologie : par exemple, pour une interface BRI ISDN, on a par exemple 2 canaux B et un canal D, tandis que pour une interface PRI E1 on a 30 canaux B et un canal D. Au niveau signalisation, on utilise généralement Q921 pour la couche liaison de données et Q931 pour la couche réseau (couche d'appels). En s'intéressant à Q931, on constate qu'il y a une signalisation spécifique d'établissement d'appel, qui permet notamment de véhiculer le numéro demandé : de ce fait, il y a indépendance entre le canal utilisé et le numéro demandé, ce qui a également pour conséquence que l'on peut faire des économies d'échelles en véhiculant un nombre "approprié" de numéros au-dessus d'une même interface. La notion d'appropriation dépendant en particulier de la durée des appels, et du nombre d'appels que l'on tolère statistiquement de perdre. En ce qui concerne la présentation du nom, on constate dans la norme, qu'elle n'est prévue que dans un seul sens : du réseau vers l'usager, ce qui interdit le fait de "forger" cette information au niveau de l'utilisateur. Ceux qui s'intéressent à ce genre de technologies pourront s'y essayer grâce à l'IPBX Asterisk et les drivers DAHDI en utilisant des canaux de type TDMOE (Time Division Multiplexing Over Ethernet), qui permet d'émuler une liaison T1 (23B+1D) ; ce type de liaison a d'ailleurs une autre particularité, comme son nom l'indique : il est directement au-dessus d'Ethernet, et pas au niveau d'IP !

```
[Aug 2 09:56:12] PRI Span: 2
[Aug 2 09:56:12] PRI Span: 2 < Protocol Discriminator: Q.931 (8) len=59
[Aug 2 09:56:12] PRI Span: 2 < TEI=0 Call Ref: len= 2 (reference 1/0x1) (Sent from originator)
[Aug 2 09:56:12] PRI Span: 2 < Message Type: SETUP (5)
[Aug 2 09:56:12] PRI Span: 2 < [04 03 80 90 a2]
[Aug 2 09:56:12] PRI Span: 2 < Bearer Capability (len= 5) [ Ext: 1 Coding-Std: 0 Info transfer capability: Speech (0)
[Aug 2 09:56:12] PRI Span: 2 < Ext: 1 Trans mode/rate: 64kbps, circuit-mode (16)
[Aug 2 09:56:12] PRI Span: 2 < User information layer 1: u-Law (34)
[Aug 2 09:56:12] PRI Span: 2 < [18 04 e9 81 83 81]
[Aug 2 09:56:12] PRI Span: 2 < Channel ID (len= 6) [ Ext: 1 IntID: Explicit Other(PRI) Spare: 0 Exclusive Dchan: 0
[Aug 2 09:56:12] PRI Span: 2 < ChanSel: As indicated in following octets
[Aug 2 09:56:12] PRI Span: 2 < Ext: 1 DS1 Identifier: 1
[Aug 2 09:56:12] PRI Span: 2 < Ext: 1 Coding: 0 Number Specified Channel Type: 3
[Aug 2 09:56:12] PRI Span: 2 < Ext: 1 Channel: 1 Type: CPE]
[Aug 2 09:56:12] PRI Span: 2 < [28 14 27 4e 65 74 61 70 73 79 73 20 47 72 61 6e 64 20 45 73 74 27]
[Aug 2 09:56:12] PRI Span: 2 < Display (len=20) [ 'Netapsys Grand Est' ]
[Aug 2 09:56:12] PRI Span: 2 < [6c 0e 21 81 27 30 33 36 38 30 30 31 37 35 38 27]
[Aug 2 09:56:12] PRI Span: 2 < Calling Party Number (len=16) [ Ext: 0 TON: National Number (2) NPI: ISDN/Telephony Numbering Plan (E.164/E.163) (1)
[Aug 2 09:56:12] PRI Span: 2 < Presentation: Presentation allowed, User-provided, verified and passed (1) "0368001758" ]
[Aug 2 09:56:12] PRI Span: 2 < [70 03 80 31 30]
[Aug 2 09:56:12] PRI Span: 2 < Called Party Number (len= 5) [ Ext: 1 TON: Unknown Number Type (0) NPI: Unknown Number Plan (0) '10' ]
[Aug 2 09:56:12] PRI Span: 2 -- Making new call for cref 1
[Aug 2 09:56:12] PRI Span: 2 Received message for call 0xb6b06290 on link 0x9ef6aa4 TEI/SAPI 0/0
[Aug 2 09:56:12] PRI Span: 2 -- Processing Q.931 Call Setup
[Aug 2 09:56:12] PRI Span: 2 -- Processing IE 4 (cs0, Bearer Capability)
[Aug 2 09:56:12] PRI Span: 2 -- Processing IE 24 (cs0, Channel ID)
[Aug 2 09:56:12] PRI Span: 2 -- Processing IE 40 (cs0, Display)
[Aug 2 09:56:12] PRI Span: 2 -- Processing IE 108 (cs0, Calling Party Number)
[Aug 2 09:56:12] PRI Span: 2 -- Processing IE 112 (cs0, Called Party Number)
```

Exemple de capture d'une trame de SETUP Q931, avec les éléments numéro et nom de téléphone.

Concrètement, les interfaces se présentent généralement de trois manières différentes :

- Certains routeurs modulaires peuvent intégrer des extensions fournissant des extensions voix. C'est le cas par exemple de certains routeurs Cisco (comme la gamme des 2901) ;
- On trouve des passerelles SIP/ TDM dédiées ; des marques telles que

Patton ou Beronet proposent ce genre d'équipement ;

- On trouve également des cartes d'extension, qui peuvent par exemple être utilisées avec Asterisk : Digium propose ce genre de cartes, mais d'autres constructeurs existent aussi, comme par exemple Sangoma.

Enfin, il convient également d'aborder brièvement un autre aspect d'interconnexion : la connexion des postes de téléphonie analogique sur les équipements VOIP. En effet, il peut y avoir dans certaines entreprises un grand nombre de postes, notamment des postes analogiques, déployés, et se pose alors la question de la réutilisation ou non de ces postes. Si les besoins des utilisateurs en termes de fonctionnalités fournies directement par le poste téléphonique ne sont pas trop importantes, il peut être envisageable de continuer à utiliser ces postes téléphoniques analogiques, au travers d'équipements qui sont des équipements de couplage à part entière : des Analog Telephone Adapters (ATA), ou dans une concentration de canaux plus importante, des banques de canaux.

Les ATA permettent généralement de disposer de 1 à 4 lignes analogiques, mappées sur des comptes SIP. Je parle de lignes car on peut éventuellement relier plus qu'un seul téléphone sur une même ligne, en respectant toutefois les caractéristiques électriques de tous les équipements, dont les REN (Ringing Equivalent Number) qui est une unité de mesure pour savoir combien un téléphone a besoin de courant pour pouvoir sonner. Des fabricants d'ATA sont par exemple Cisco (SPA 2102 ou 3102) ou GrandStream (Gamme HandyTone).

Les channel banks sont plus variées : on peut trouver des channel banks qui se branchent en USB sur des systèmes PABX (certains modèles de Xorcom par exemple), tandis que d'autres peuvent s'interfacer en E1 ou T1 (chez Rhino par exemple), voire même en TDMoE (plus rare, mais cela existe chez DigiVoice). Ces différences de connexions indiquent également des différences notables de configuration logicielle et matérielle au niveau des PABX ; chaque système de branchement ayant des avantages et inconvénients qu'il faudra évaluer dans les situations en fonction du besoin.

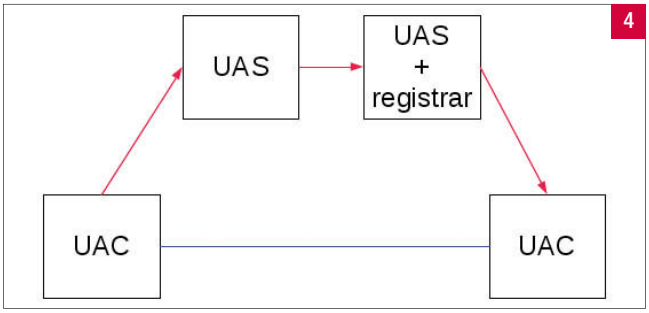
On a donc compris via ces explications comment sont récupérées les informations de l'appelant au niveau de l'appelé ; informations de numéro et éventuellement de nom qui permettront d'en savoir plus sur l'appelant et de faire du couplage téléphonie informatique sur les appels entrants.

PROTOCOLE SIP

Les protocoles de VOIP vont être importants dans le couplage téléphonie informatique. Un certain nombre de protocoles existent, mais un protocole est devenu le standard le plus utilisé : il s'agit du protocole Session Initiation Protocol : SIP.

Ce protocole est défini dans un grand nombre de RFC, dont en particulier la RFC 3261. Le protocole SIP met en œuvre plusieurs types d'entités, dont notamment :

- Des User Agent Clients, c'est à dire des clients SIP : on retrouve des clients de types assez différents, comme par exemple des téléphones SIP matériels, des softphones, des Analog Telephone Adapter (ATA), ou même des webphones (c'est à dire des systèmes applicatifs utilisant par exemple le WebRTC intégré au navigateur et une stack SIP JS). Du fait de leurs statuts, ils vont être impliqués à la fois dans la signalisation et dans le flux média d'un appel.
- Des User Agent Servers : Il s'agit de proxys SIP. Les proxys SIP font office de "routeurs avancés" pour la signalisation des appels. Non seulement ils ont pour charge de transmettre au "next-hop" les paquets SIP, mais peuvent également assurer d'autres tâches, comme la transmission d'un paquet à des cibles multiples (fork de l'appel dans le cadre de l'initiation d'une session par exemple), ou encore assurer des fonctions d'authentification. Généralement, un des proxys SIP de l'infrastructure en embarque un.
- Registrar : le registrar sert à stocker l'enregistrement de clients SIP qui ont des IP/ports dynamiques, et qui doivent donc se faire connaître pour pouvoir recevoir des appels.
- Des Back To Back User Agent (B2BUA) : ces éléments ne sont pour ainsi dire que des éléments optionnels d'une infrastructure SIP. Néanmoins, ces entités intègrent souvent des fonctionnalités avancées en leur sein, fonctionnalités qui nécessitent d'être centralisées (des salles de conférences, par exemple), ou qui pourraient ne pas être implémentées chez tous les clients. Par exemple, l'envoi de touches DTMF peut se faire de plusieurs façons, dont notamment directement dans le média audio, en fonction des codecs utilisés, ou dans le trafic RTP, sous forme d'événement, ou sous forme de message SIP INFO. Autre exemple au niveau même du protocole SIP : on peut utiliser des messages SIP reINVITE ou utiliser des messages SIP REFER dans le cadre des transferts. [4]



Exemple classique de cheminement d'infos SIP : l'UAC de gauche souhaite appeler l'UAC de droite. En bleu le chemin des médias, en rouge, le chemin d'une requête de l'UAC de gauche vers l'UAC de droite.

Avec ces quelques informations, on comprend déjà que de nombreuses typologies SIP peuvent être constituées :

- Des solutions où l'on cherchera par exemple à mettre une bonne partie de l'"intelligence" dans des entités centralisées, de manière à réduire les prérequis techniques des UA. Il est donc fort probable que l'on utilisera des B2BUA, et que ces B2BUA pourront être mis à contribution pour le couplage téléphonie informatique, d'autant plus qu'un environnement Web est un environnement client/serveur, où le serveur est donc également centralisé. Une communication serveur à serveur peut donc être mise en œuvre.
- Des solutions où l'on cherchera plutôt à mettre en avant l'utilisation du protocole SIP entre les différents UA, et où l'on aura une "intelligence requise" au niveau des UA plus forte. Dans ces cas, il faudra donc chercher plutôt à récupérer des informations auprès des UA. Dans le monde data, ceci correspond aux interfaces qui sont intégrées aux navigateurs via une logique intégrée en JS par exemple.
- Et entre les deux, on peut avoir toute une gamme de solutions intermédiaires. On comprendra qu'il est toujours possible, pour des fonctionnalités spécifiques, de mixer les deux types de solutions : router des appels pour des fonctionnalités spécifiques vers un B2BUA qui va les acheminer ensuite vers d'autres UA du réseau (par exemple : un serveur assurant la répartition des appels entrants sur un call center).

On va s'intéresser aux deux approches, à savoir aux possibilités d'information qui sont généralement intégrées aux UA (et n'utilisant pas forcément uniquement le protocole SIP), ainsi qu'à un B2BUA Open Source : Asterisk.

Dans la partie 2, nous aborderons la partie technique Asterisk et CTI. •

Retour d'expérience : MySQL vers Azure SQL Database, via Azure Data Factory

Azure Data Factory est un service Cloud de traitement de données structurées ou non structurées provenant de sources de données variées. ADF, son surnom, permet de traiter des données en entrée et de les déverser dans un stockage en sortie. Généralement en BigData, ce service se limite à effectuer une copie en « quasi-un pour un » des données. Nous verrons pourtant que cela n'est pas forcément aussi simple qu'il n'y paraît.

Loris Andaloro, *Expert Data senior chez* **VISEO**

Ihor Leontiev, *Architecte Azure, MVP Azure chez* **VISEO**

Vincent Thavonekham, *Responsable Stratégie Azure, MVP Azure chez* **VISEO**

Nous parlons de « quasi-un pour un », car il y a forcément quelques transtypages à effectuer pour passer d'une base de données à une autre. D'un point de vue théorique on est face à un ETL (Extract Transform Load), logiciel bien connu dans le monde de la **Business Intelligence**, mais pas seulement. En pratique, nous utilisons cet outil en mode ELT (Extract Load Transform) car on est ici dans le cadre du Cloud et du BigData : on ne veut pas modifier les données lors de l'import (« ingestion »), mais les enregistrer brutes dans un Data Lake. Et si on a à les modifier (« Transform ») et analyser (« Analyse »), ce sera ultérieurement, et potentiellement dans quelques années (Fig 1).

Ci-contre les principaux concepts d'Azure Data Factory (Fig 2).

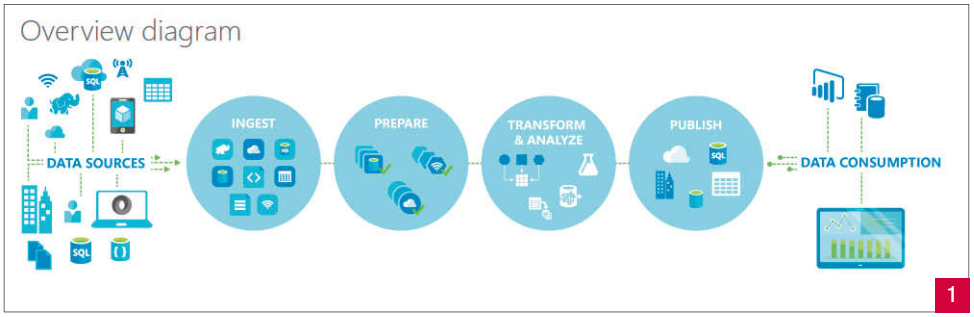
Dans une instance Data Factory, vous créez un ou plusieurs ***pipelines*** de données. Un pipeline constitue un groupe d'***activités*** qui définit les actions à effectuer. Par exemple, vous pouvez utiliser une « activité de copie » pour copier des données d'une banque de données vers une autre. De même, vous pouvez utiliser une « activité Hive » qui exécute une requête Hive sur un cluster Azure HDInsight afin de convertir ou d'analyser vos données. ADF prend en charge deux types d'activités : **déplacement des données** et **transformation des données**.

Les ***dataset*** sont des jeux de données qui peuvent être consommés ou produits par les activités. Enfin chaque exécution d'une activité produit ce que l'on appelle un ***data slice***.

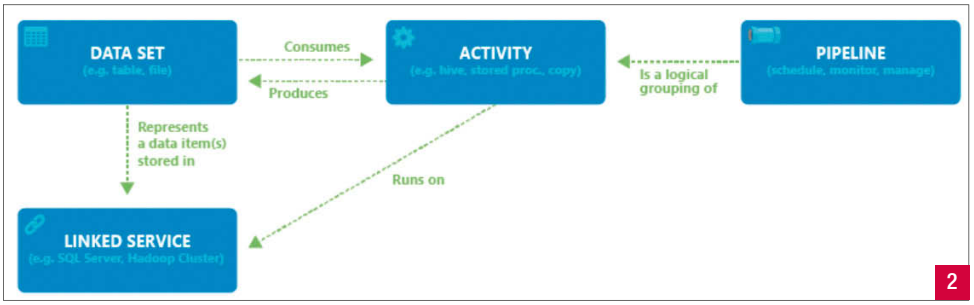
D'une part, nous allons voir comment nous avons mis en place une extraction de table depuis MySQL (On Premise) vers une base de données Azure SQL Database (Fig 3). Et d'autre part, nous allons évoquer les difficultés, qui commencent par de nombreuses incompatibilités entre les types de données, ainsi que ce qui concerne la planification des traitements.

Tout d'abord, il faut instancier ADF (Fig 4) et déployer la Gateway sur le réseau local comme précisé dans la documentation donnée en référence à la fin.

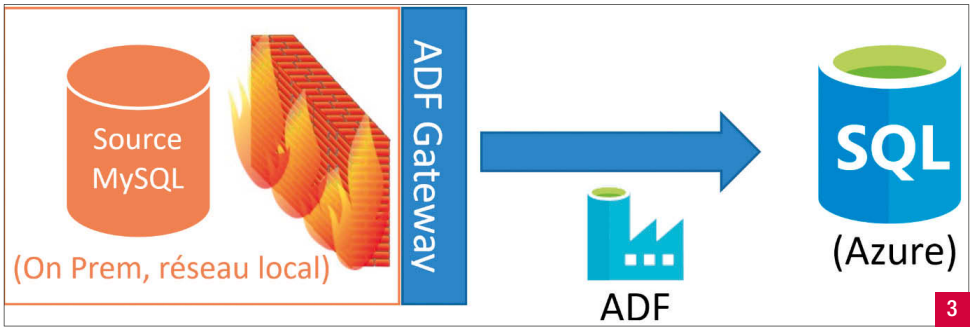
Ensuite, nous pouvons configurer le déplacement des données en cli-



1

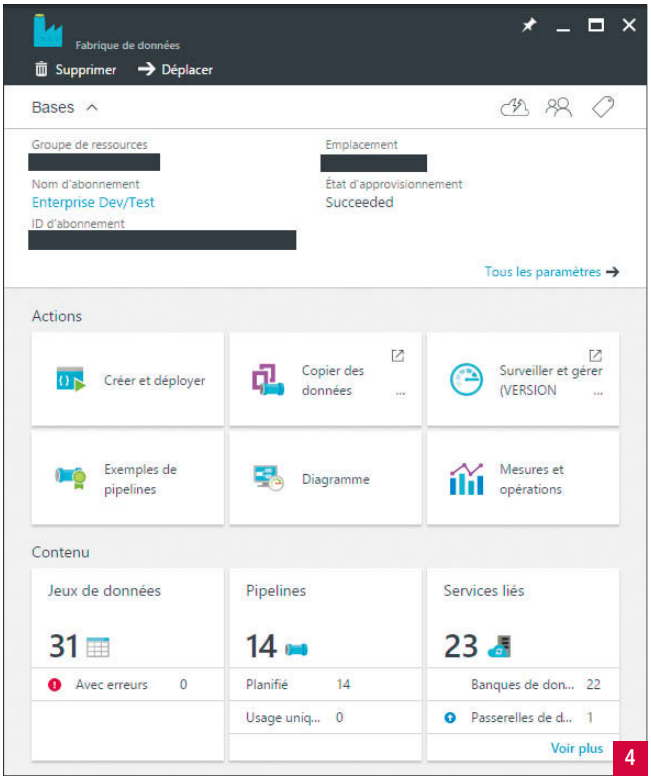


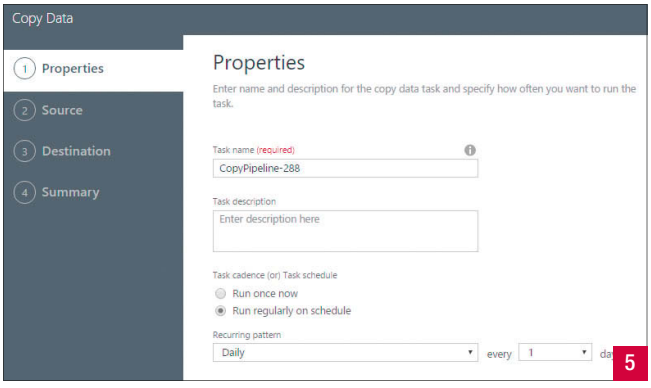
2

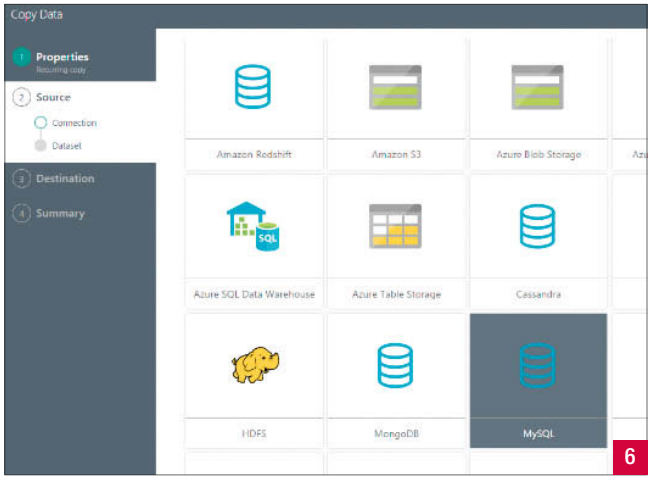


3

quant sur « *Copier des données* » afin de créer notre premier *pipeline*. Une nouvelle page s'ouvre sur une procédure pas à pas de création de pipeline (Fig 5). Attention, cet outil est encore en version beta, alors que la solution ADF est en version release. Commencez par nommer le pipeline. (Fig 5)







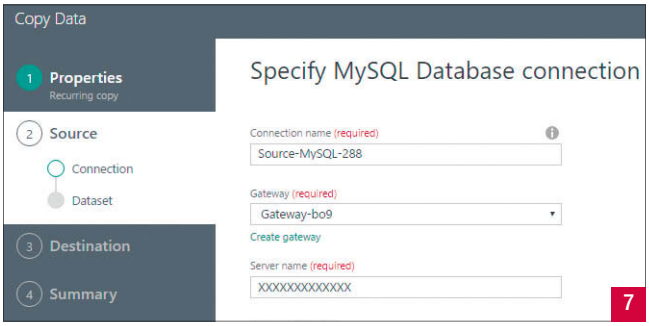
Puis renseigner la source de données MySQL, parmi de nombreuses autres options comme Amazon Redshift/S3, Azure Blob Storage/SQL Datawarehouse, Cassandra, MongoDB, … (Fig 6, Fig 7)

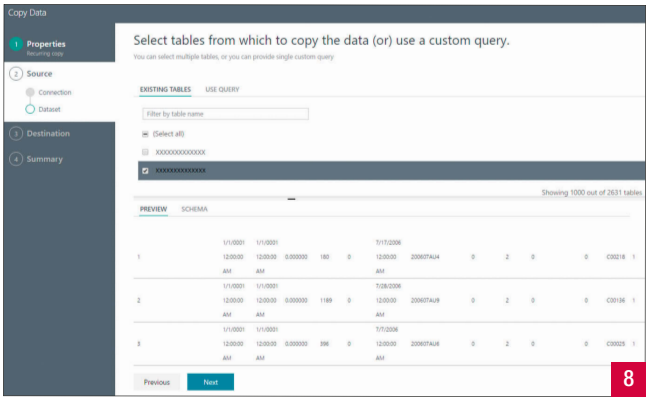
Et choisir la table source dans l'étape « Source », onglet « EXISTING TABLES » (Fig 8)

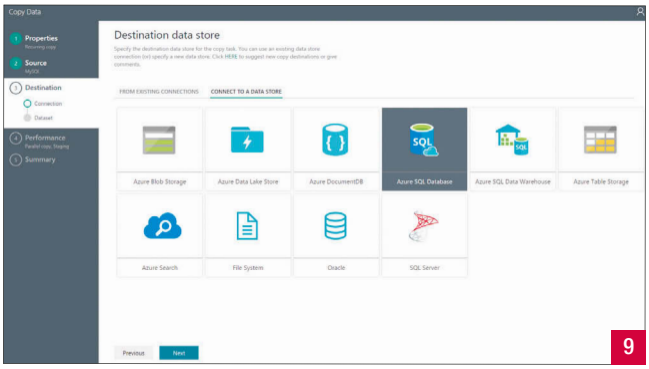
Ensuite, la base de données de destination (liste non exhaustive, Fig 9)

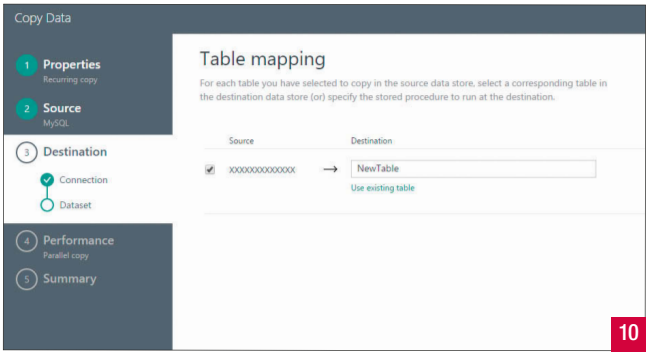
Indiquer enfin la table de destination (Fig 10)

Les premiers soucis se présentent (Fig 11) ; certains types de données semblent ne pas être supportés, empêchant donc la création du *pipeline*.





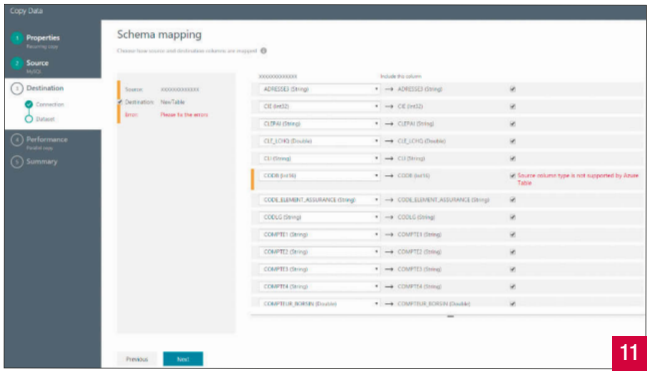




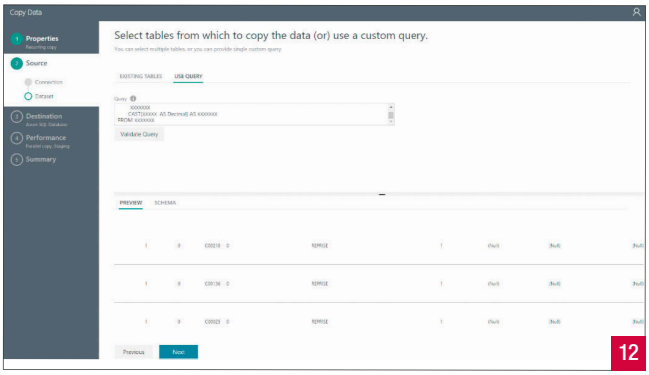
L'astuce consiste à revenir à l'étape « Source » onglet « USE QUERY » (Fig 12). Cela permet l'utilisation d'une requête avec des CAST au lieu de la table (nous verrons que cela n'est pas suffisant non plus lié à un autre point bloquant).
Voici un exemple de requête à utiliser :

```
SELECT
CAST(champ1 AS NVARCHAR) AS champ1,
      CAST(champ2 AS Signed Integer) AS champ2,
      champ3,
    IF(champ4>= '1900/01/01', champ4, NULL) AS champ4
FROM table
```

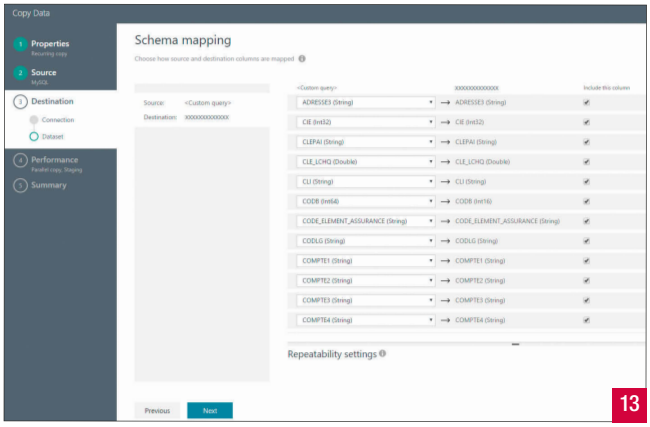
Avec des CAST convenables sur les champs problématiques, il devient possible de mapper toutes les colonnes sans erreurs. J'ai recensé les principales équivalences entre les types MySQL et SQL Server, ainsi que les opérations à effectuer au niveau de la requête SELECT.



11



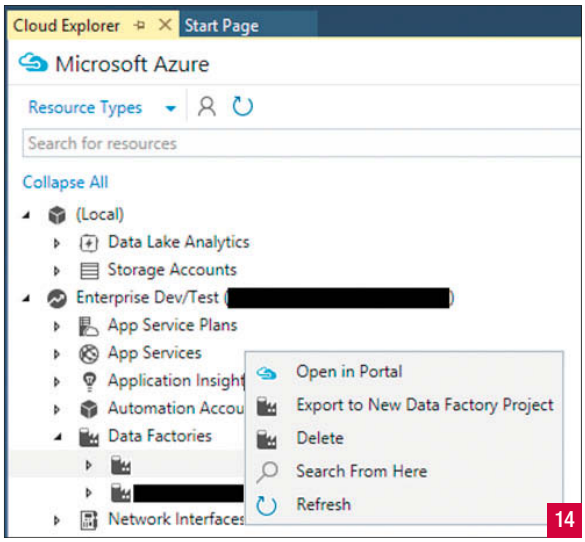
12



13

Types MySQL	Type SQL Server recommandé	Opération à effectuer
bigint	bigint	
blob	varbinary(max)	
char(X)	nchar(X)	
date	date	IF(xxxx >= '1900/01/01', xxxx, NULL) as xxxx,
datetime	datetime	IF(xxxx >= '1900/01/01', xxxx, NULL) as xxxx,
decimal	decimal	
double	float	
enum	nvarchar(max)	CAST(xxxx AS VARCHAR)
float	float	
int	int or bigint	Dépend du flag UNSIGNED MySQL : CAST (xxxx AS SIGNED BIGINT) ou CAST(xxxx AS SIGNED INT)
longblob	varbinary(max)	
longtext	nvarchar(max)	
mediumtext	nvarchar(max)	
set	nvarchar(max)	CAST(xxxx AS NVARCHAR)
smallint	smallint or int	Dépend du flag UNSIGNED MySQL : CAST (xxxx AS SIGNED INT) ou CAST(xxxx AS SIGNED SMALLINT)
text	nvarchar(max)	
time	time	
timestamp	smalldatetime	
tinyint	tinyint or smallint	Dépend du flag UNSIGNED MySQL : CAST (xxxx AS SIGNED SMALLINT) ou CAST(xxxx AS SIGNED TINYINT)
varchar(X)	nvarchar(X)	

ADF s'améliorant rapidement, ces règles évoluent également très vite également. Au fil des mois, nous constatons qu'il devient de plus en plus simple d'importer les données MySQL ; donc moins de hack de conversions et cela fonctionne sans problèmes (Fig 13).
Nous poursuivons avec l'assistant de configuration jusqu'au déploiement du pipeline, sans encombre ; cela génère un fichier JSON dans le menu *Créer et déployer*. C'est la définition du *pipeline* que nous venons de mettre en place.
Aussi, grâce au **Cloud Explorer** intégré à **Visual Studio** vous pouvez récupérer votre pipeline et l'éditer au format JSON. Pour cela il faut naviguer dans l'arborescence jusqu'à votre instance ADF puis effectuer un clic droit et choisir le menu ***Export to New Data Factory Project*** (Fig 14).



14

Il n'est en effet pas possible, pour le moment, de revenir à l'assistant Web « Copy Activity » des écrans précédents pour éditer le pipeline.
Le dernier problème bloquant de cet assistant est que nous avions plusieurs milliers de « schema mappings » à mettre en œuvre. Impossible donc de passer plusieurs jours à faire des clics dans le navigateur. D'autant que nos nombreuses tentatives désespérées mettaient systématiquement le JavaScript du navigateur sur les rotules (quel que soit le navigateur). Nous avons eu à développer un programme C# ayant la capacité de détecter les tables source et générer ces fichiers JSON à partir d'une base source.

Autres possibilités de ADF : orchestration, planification et monitoring

La planification de l'heure de lancement du traitement se fait avec le paramètre ***offset***. A ce jour, une astuce peu documentée que Microsoft Corp nous a suggérée pour choisir l'heure précise, consiste à configurer le paramètre ***start*** avec systématiquement une valeur à minuit (ci-dessous 2017-02-21T00:00:00Z). Sinon le pipeline ne fonctionnera pas.

```
"policy": {
        "timeout": "1.00:00:00",
        "concurrency": 1,
        "executionPriorityOrder": "NewestFirst",
        "style": "StartOfInterval",
        "retry": 3,
        "longRetry": 0,
        "longRetryInterval": "00:00:00"
      },
      "scheduler": {
        "frequency": "Hour",
        "interval": 24,
        "offset": "03:00:00"
      },
      "name": "Activity-0-_Custom query_->[dbo]_[xxxxxxxxx]"
],
    "start": "2017-02-21T00:00:00Z",
    "end": "9999-09-09T00:00:00Z",
    "isPaused": false,
"pipelineMode": "Scheduled"
```

La partie Monitoring d'ADF est très importante et correspond à un de ses gros points forts, et obligatoire pour faire du « Data Lineage » et ainsi auditer les données et les nombreux traitements quotidiens. Pour y accéder, cliquer l'icône « Surveiller et gérer » (Fig 4), et cela ouvre une nouvelle fenêtre. Autant dire que nous avons passé des journées entières les yeux rivés sur les différents pipelines qui s'exécutent, avec des Logs détaillés.

CONCLUSION

La documentation officielle de Microsoft, ainsi de que nombreuses recherches, ont bien aidé à avancer et nous sommes actuellement en production avec ADF. Bien que nous ayons réduit le périmètre de ce retour d'expérience à l'unique brique ADF (afin que cela soit aisément compréhensible), cette brique qui semble rapide et simple à appréhender peut se révéler bien plus compliquée qu'il n'y paraît.
Dans notre cas, au-delà de ces petites subtilités, j'avoue que sans l'aide de l'équipe produit de Microsoft Corp qui développe ADF, nous n'aurions pas réussi la mise en production ; depuis, cela fonctionne comme un charme au quotidien ! •

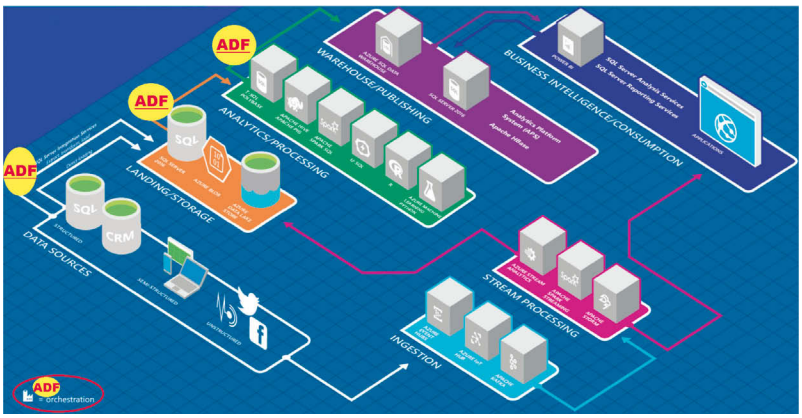
Références

Introduction ADF :
https://docs.microsoft.com/fr-fr/azure/data-factory/data-factory-introduction
Format JSON d'ADF :
https://docs.microsoft.com/en-us/azure/data-factory/data-factory-introduction
Configurer ADF Gateway sur le réseau local :
https://docs.microsoft.com/fr-fr/azure/data-factory/data-factory-move-data-between-onprem-and-cloud

POUR ALLER PLUS LOIN

Nous avons vu dans cet article un aperçu de l'utilisation d'ADF. Il est légitime de se demander pourquoi faire autant d'efforts ? alors qu'un ETL comme SSIS est déjà présent sur le marché depuis des années.

Au-delà du fait que SSIS ne soit pas « scalable », ADF va permettre d'adresser des scénarii Big Data inaccessibles jusqu'à présent.
Dans le schéma, un élément frappant est d'une part la répartition des zones de stockage, où cela diffère des traditionnels BI avec un "Staging area", un "Operational Data Store" et un "Datawarehouse" remplies et vidées quotidiennement. D'autre part, ADF gère une grande variété des sources de données : les bases de données métier de l'entreprise, le CRM, l'ERP, mais aussi les réseaux sociaux, les services Open Data, l'Internet des objets (IoT), etc… Enfin ADF travaille sur une fréquence des traitements et une puissance de calculs bien supérieur: les données sont importées, puis transformés plusieurs fois ; tantôt en (near-)temps réel (ex IoT), tantôt en batch (ex. calcul d'indicateurs récurrents chaque soir, ou analyse de sentiments Twitter/Facebook durant le lancement d'un produit).



Afin de répondre aux enjeux actuels de la Data et d'analyses orientées données, Microsoft investit de manière importante dans tous les outils Data, et en particulier sur cet outil, véritable chef d'orchestre de la Data au sein du système d'information dématérialisé de l' « entreprise 2.0 » et de l'« Industrie 4.0 » !

Embarquez des vidéos dans vos applications Android avec l'API YouTube

• Sylvain Saurel
sylvain.saurel@gmail.com
Développeur Android
https://www.ssaurel.com

Les utilisateurs d'applications mobiles sont toujours friands de contenus multimédias en tous genres. Ainsi, inclure des vidéos YouTube au sein d'une application Android peut faire la différence au moment du téléchargement de l'application sur le Google Play Store. Dans cet article, nous vous proposons de mettre en oeuvre l'API YouTube pour Android afin d'intégrer et d'interagir avec des vidéos YouTube.

Afin de permettre aux développeurs d'applications Android d'intégrer des vidéos issues de leur réseau, YouTube propose l'API YouTube Android Player. Librement téléchargeable sur le site développeurs de Google, cette API offre la possibilité d'incorporer un lecteur vidéo au sein d'une application Android autorisant le chargement et la lecture de vidéos ou de playlists YouTube. L'API permet également de contrôler finement l'expérience de lecture tout en la customisant.
Il est ainsi possible de contrôler la lecture d'une vidéo YouTube de manière programmatique que ce soit pour le démarrage de la lecture, les éventuelles pauses ou le déplacement à un instant donné dans une vidéo. Mieux encore, l'API permet d'être averti suite à certains évènements précis comme la fin du chargement d'une vidéo ou les changements d'état du lecteur YouTube embarqué. Enfin, l'API propose également des fonctionnalités facilitant la gestion des changements d'orientation ou des passages en mode fullscreen.

Clé développeur YouTube

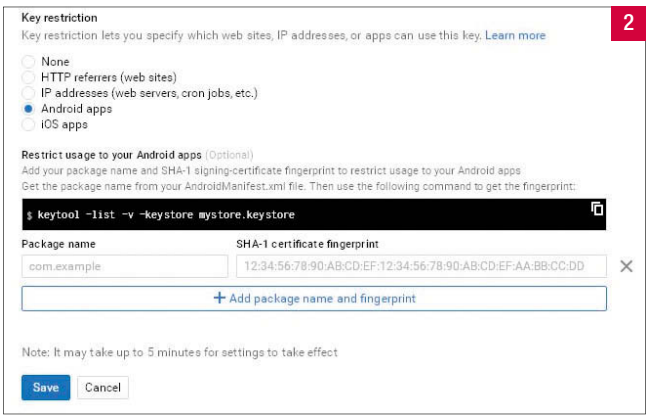
Avant de pouvoir utiliser l'API YouTube sous Android, il faut enregistrer son application dans la console développeurs de Google en suivant les étapes suivantes :
1/ Allez sur le site https://console.developers.google.com/ ;
2/ Créez un nouveau projet en le nommant MyVideoTube (figure 1) ;
3/ Dans la barre latérale de gauche, sélectionnez l'entrée "Library". Sur le panneau principal à droite, sélectionnez l'entrée YouTube Data API et cliquez sur "Enable" à la page suivante ;
4/ Dans la barre latérale de gauche, sélectionnez l'entrée "Credentials" ;
5/ L'API supporte OAuth 2.0 mais ici nous utiliserons la clé d'API pour notre application Android. Cliquez sur le bouton "Create Credentials" puis sur "API key" ;
6/ Afin de sécuriser votre clé d'API, vous devrez restreindre son utilisation au package de votre application et enregistrer l'empreinte SHA-1 du certificat utilisé pour signer l'APK de l'application (figure 2) ;

Installation de l'API

Votre projet d'application Android créé, il reste à installer l'API YouTube Player qui est téléchargeable en version 1.2.2 sur le site développeurs de Google :
https://developers.google.com/youtube/android/player/downloads/YouTubeAndroidPlayerApi-1.2.2.zip
Une fois l'archive zip ouverte, vous devez placer le fichier YouTubeAndroidPlayerApi.jar dans le répertoire libs de votre projet Android et rajouter la dépendance suivante dans le fichier de build Gradle :

```
compile files('libs/YouTubeAndroidPlayerApi.jar')
```

Création du projet dans la console développeurs



Restriction de la clé d'API à votre application

La bibliothèque ajoutée au projet, nous pouvons maintenant passer à l'intégration du lecteur vidéo YouTube au sein de l'application.

Intégration du lecteur Vidéo

L'application proposant des vidéos récupérées depuis le réseau, il est nécessaire d'ajouter la permission pour accéder à Internet au sein du manifest de l'application :

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Le lecteur vidéo proposé par l'API YouTube est représenté par l'objet YouTubePlayerView. Nous allons donc modifier le layout de l'application pour y intégrer le composant YouTubePlayerView :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
```

```
        android:orientation="vertical"
        tools:context=".MainActivity">

    <com.google.android.youtube.player.YouTubePlayerView
        android:id="@+id/youtube_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

Au niveau du code Java, il est possible d'utiliser l'objet YouTubePlayerView via le fragment YouTubePlayerFragment ou via l'activité YouTubeBaseActivity. Ici, nous allons recourir à la seconde option. Ainsi, notre activité va étendre la classe YouTubeBaseActivity tout en implémentant l'interface YouTubePlayer.OnInitializedListener qui propose différentes méthodes pour suivre la connexion de l'objet YouTubePlayerView au service en ligne YouTube.
A la création de l'activité, nous récupérons l'objet YouTubePlayerView depuis le layout XML défini précédemment avant d'appeler sa méthode initialize en passant en entrée la clé développeur générée précédemment. Une fois le lecteur initialisé avec succès, nous accédons à l'objet YouTubePlayer qui propose la méthode cueVideo. Nous appelons cette méthode avec l'ID de la video YouTube que nous souhaitons charger.
En cas d'échec à l'initialisation, il reste la possibilité de lancer une requête de récupération avec un code de requête spécifique. Le résultat de cette requête est à gérer au sein de la méthode onActivityResult au sein de laquelle nous allons appeler à nouveau la méthode initialize de l'objet YouTubePlayerView. On obtient ainsi le code suivant :

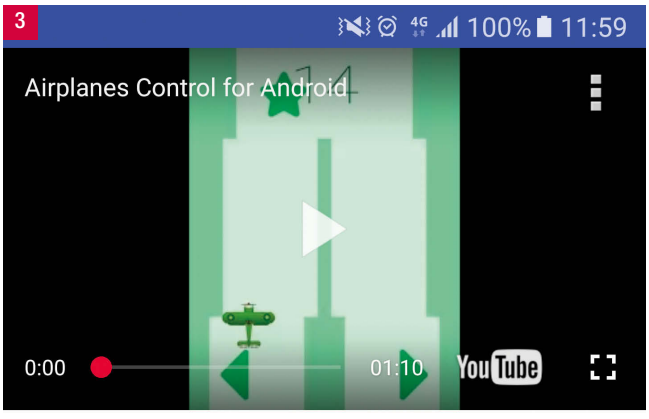
```
public class MainActivity extends YouTubeBaseActivity implements YouTubePlayer.
OnInitializedListener {

    private static final int RECOVERY_REQUEST = 1;
    private YouTubePlayerView youTubeView;
    private YouTubePlayer player;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        youTubeView = (YouTubePlayerView) findViewById(R.id.youtube_view);
        youTubeView.initialize(Config.YOUTUBE_API_KEY, this);
    }

    @Override
    public void onInitializationSuccess(Provider provider, YouTubePlayer player, boolean
wasRestored) {
        this.player = player;
        player.setPlayerStateChangeListener(playerStateChangeListener);
        player.setPlaybackEventListener(playbackEventListener);

        if (!wasRestored) {
            player.cueVideo("8igWsm6Y7y4");
        }
    }

    @Override
    public void onInitializationFailure(Provider provider, YouTubeInitializationResult
errorReason) {
        if (errorReason.isUserRecoverableError()) {
            errorReason.getErrorDialog(this, RECOVERY_REQUEST).show();
        } else {
            String error = String.format(getString(R.string.player_error), errorReason.
toString());
            Toast.makeText(this, error, Toast.LENGTH_LONG).show();
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == RECOVERY_REQUEST) {
            getYouTubePlayerProvider().initialize(Config.YOUTUBE_API_KEY, this);
        }
    }

    protected Provider getYouTubePlayerProvider() {
        return youTubeView;
    }
}
```

En exécutant l'application, on obtient alors le résultat présenté à la figure 3 avec le lecteur de vidéo YouTube parfaitement intégré à l'application.
Il est à noter que les appareils Android devront avoir l'application YouTube installée en version 4.2.16 au minimum pour que le lecteur vidéo puisse fonctionner correctement. En effet, l'API YouTube Player s'appuie sur un service spécifique distribué au travers de l'application YouTube. Néanmoins, cela n'est pas une contrainte très forte puisque la grande majorité des appareils Android ont l'application YouTube installée par défaut en version supérieure à la 4.2.16 apparue avec Android Froyo 2.2.

Suivre les évènements du lecteur Vidéo

Une fois le lecteur vidéo YouTube intégré au sein de votre application, il est fort probable que vous ayez besoin de suivre les évènements émis par le lecteur afin de pouvoir interagir avec lui. Ainsi, il est possible d'être informé lorsque le lecteur vidéo est en phase de buffering, de lecture, de pause, de recherche ou bien arrêté. On peut par exemple imaginer le démarrage d'une nouvelle vidéo une fois celle en cours de lecture terminée.
Pour gérer au mieux les évènements du lecteur vidéo, l'API YouTube Player met à disposition du développeur les interfaces suivantes :

- YouTubePlayer.PlayerStateChangeListener qui définit des méthodes callbacks invoquées lorsque l'état du lecteur vidéo change ;



Intégration du lecteur vidéo YouTube

- YouTubePlayer.PlaybackEventListener définissant des méthodes callbacks invoquées lors de la lecture d'une vidéo ;
- YouTubePlayer.OnFullscreenListener définissant des méthodes callbacks qui seront invoquées lors du passage en mode fullscreen ou du retour en mode normal ;
- YouTubePlayer.PlaylistEventListener qui définit des méthodes callbacks invoquées lorsque des évènements liés aux playlists se produisent ;

Pour les besoins du présent article, nous allons implémenter les 2 premières interfaces listées ci-dessus. Afin de suivre les différents états de la lecture de la vidéo chargée par notre application, nous implémentons l'interface PlaybackEventListener et nous afficherons un message à l'écran via un Toast à chaque changement d'état en partant de la lecture et en allant jusqu'à l'arrêt de la vidéo. La seconde interface que nous allons implémenter, l'interface PlayerStateChangeListener, va nous permettre de suivre les changements d'état du lecteur vidéo. Il va ainsi être possible de savoir lorsqu'une vidéo est en cours de chargement ou qu'une publicité est affichée dans le lecteur vidéo. La dernière étape consiste à enregistrer ces interfaces de type listener sur l'instance de l'objet YouTubePlayer obtenue suite à l'initialisation réussie de la connexion au serveur YouTube ce qui donne le code suivant : **code complet sur notre site Web.**

Définition de contrôles spécifiques

La bibliothèque YouTube Player fournit, avec le composant YouTubePlayerView, une interface déjà user-friendly pour la lecture de vidéos au sein d'une application Android. Cependant, en tant que développeur, il se peut que vous souhaitiez proposer des contrôles spécifiques à vos utilisateurs pour interagir avec une vidéo. Ici, nous allons rajouter un bouton lecture, un bouton pause ainsi que la possibilité de changer la position de lecture en laissant le soin à l'utilisateur de saisir une valeur en secondes. La mise à jour du layout prenant en compte ces modifications donne le code suivant :

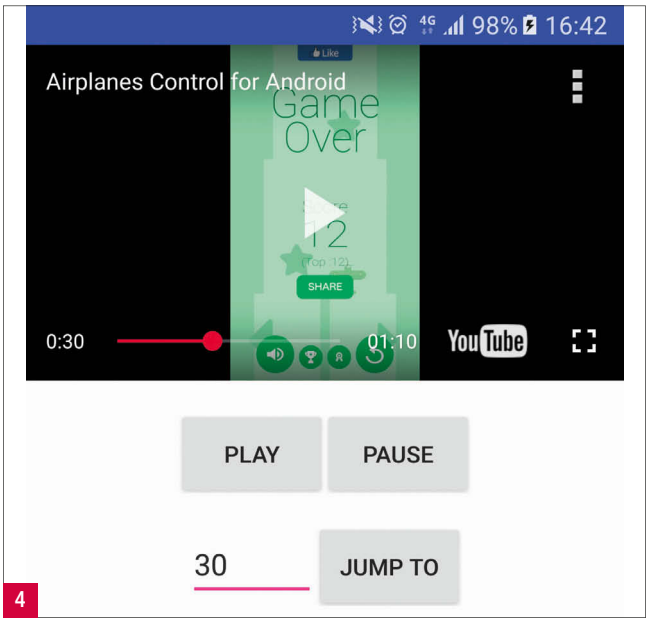
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        tools:context=".MainActivity">

  <com.google.android.youtube.player.YouTubePlayerView
    android:id="@+id/youtube_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="20dp">

    <Button
      android:id="@+id/play"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/play"/>

    <Button
      android:id="@+id/pause"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/pause"/>

  </LinearLayout>

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="20dp">

    <EditText
      android:id="@+id/seek_to_text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:inputType="number"
      android:hint="@string/seek_to_hint"/>

    <Button
      android:id="@+id/seek_to"
      android:text="@string/seek_to"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
  </LinearLayout>

</LinearLayout>
```

Côté code Java, il suffit de récupérer les instances des 3 boutons définis dans le layout ainsi que la référence du composant EditText au sein duquel l'utilisateur saisira la nouvelle position de lecture de son choix pour la vidéo en cours de lecture. Pour ce faire, l'objet YouTubePlayer, récupéré à l'initialisation de la connexion avec le serveur YouTube, propose la méthode seekToMillis que nous allons employer. Pour gérer les actions du bouton lecture, du bouton pause et du bouton de déplacement de la position de lecture, nous définissons l'objet OnClickListener suivant :

```
private View.OnClickListener btnClickListener = new View.OnClickListener() {

 @Override
 public void onClick(View view) {
  switch(view.getId()) {
   case R.id.seek_to:
    int skipToSecs = Integer.valueOf(seekToText.getText().toString());
    player.seekToMillis(skipToSecs * 1000);
    break;

   case R.id.play :
    player.play();
    break;

   case R.id.pause :
    player.pause();
    break;
  }
 }
};
```

Il ne reste ensuite plus qu'à setter cet objet OnClickListener dans la méthode onCreate de l'activité principale de notre application :

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

  youTubeView = (YouTubePlayerView) findViewById(R.id.youtube_view);
  youTubeView.initialize(Config.YOUTUBE_API_KEY, this);

  playerStateChangeListener = new MyPlayerStateChangeListener();
  playbackEventListener = new MyPlaybackEventListener();

  seekToText = (EditText) findViewById(R.id.seek_to_text);
  seekToButton = (Button) findViewById(R.id.seek_to);
  seekToButton.setOnClickListener(btnClickListener);
  playButton = (Button) findViewById(R.id.play);
  playButton.setOnClickListener(btnClickListener);
  pauseButton = (Button) findViewById(R.id.pause);
  pauseButton.setOnClickListener(btnClickListener);
}
```



Ajout des contrôles spécifiques

Une fois ces modifications effectuées, nous pouvons tester notre application et apprécier le résultat final avec la possibilité donnée à l'utilisateur de déplacer la position de lecture (figure 4).

CONCLUSION

Simple d'utilisation et puissante à la fois, l'API YouTube Player permet d'intégrer facilement des vidéos YouTube au sein d'applications Android. Proposer à ses utilisateurs des contenus multimédias de ce type peut être un facteur différenciant au moment du téléchargement (ou non) d'une application sur le Google Play Store. La bibliothèque offre également aux développeurs d'intéressantes possibilités de personnalisation qui permettent de créer une expérience utilisateur avancée. Une fois les bases de l'API détaillées dans cet article maîtrisées, il est possible d'imaginer de nombreux cas d'utilisation plus poussés en sachant que les playlists sont également supportées par le lecteur vidéo. •

Construire une application mobile connectée à une Blockchain


Guillaume Nicolas
ingénieur développeur


Damien Lecan
expert technique
SQLI Nantes

La Blockchain fait rêver, à la fois par la potentielle révolution qu'elle pourrait apporter à la société, tout en paraissant inaccessible. En effet, nombre d'études ou de POC internes conduisent à la même conclusion couperet : immature et/ou trop compliquée. S'ensuit un rejet de la technologie en attendant une plus grande maturité.
Le Bitcoin montre pourtant que les principes techniques fondamentaux des Blockchains fonctionnent, mais il paraît trop éloigné de nos usages quotidiens pour avoir l'air de s'appliquer à d'autres domaines. Chez SQLI, nous croyons à la Blockchain dès maintenant et de la manière la plus simple possible : depuis nos smartphones.

Accéder avec la Blockchain avec son smartphone, vraiment ?

Il existe de nombreuses applications qui permettent d'accéder à la Blockchain, mais l'immense majorité est dédiée aux postes de travail et autres ordinateurs de bureau ou portables. En effet, ce sont des applications dites "lourdes", car les ressources nécessaires au fonctionnement de la Blockchain sont importantes : espace disque (plusieurs giga-octets), bande passante réseau (ces giga-octets sont à télécharger...), mémoire, CPU... Le défi de l'accès à la Blockchain au travers d'un smartphone est donc conséquent. Comment concilier ces ressources nécessaires avec les "faibles" moyens apportés par les smartphones ? Pour y répondre, nous avons donc relevé le défi de construire une application mobile décentralisée Android, qui interagit *directement* avec une Blockchain, et non pas au travers d'un intermédiaire qui délocaliserait l'accès à Blockchain sur un serveur, dans le Cloud ou sur le serveur de votre choix, fusse-t-il open-source.
Nous avons décidé d'y aller par étape, de manière à rendre progressive la consommation de ressources nécessaires sur le smartphone. Par exemple, pour la première étape, nous nous restreignons à l'usage d'une Blockchain privée, ce qui limite la volumétrie de stockage et la bande passante nécessaire. De plus, nous faisons en sorte que le smartphone soit un noeud passif de la Blockchain, le plus passif possible, qui permet donc uniquement de générer des transactions sur la Blockchain et stocke une partie des blocs qui y sont générés. Par conséquent, le smartphone est un noeud qui ne mine pas. Passons maintenant à la réalisation d'une application, dont nous avons démontré la faisabilité au Devfest Nantes 2016.

Une application décentralisée Android Ethereum

Pour créer une application décentralisée (ĐApp) sur smartphone Android, nous avons besoin de plusieurs éléments : * Une application Android suivant une architecture particulière * Un client qui implémente le protocole de communication et de gestion d'un noeud blockchain. Si aujourd'hui il devient possible de mettre en place ces deux briques indépendamment les unes des autres, comment faire pour les lier au sein même d'un smartphone ? Dans un premier temps il nous faut récupérer le client blockchain compatible pour l'architecture ARM qu'offre la plupart de nos smartphones. Grâce à la communauté Ethereum, nous avons accès à différentes versions du client Geth *cross-compilé* pour des architectures différentes. Une version au format Android Archive (.aar) est donc disponible au téléchargement. Il s'agit d'une version "*wrappée*" du binaire permettant simplement de lancer un noeud geth dans une application Android. Cependant, une fois le binaire lancé, il n'existe aucun outil pour piloter ce noeud depuis une application Android. C'est pour pallier ce problème que nous avons créé notre propre librairie de communication appelée Ethereum-Java (licence MIT). Cette librairie Java a les caractéristiques suivantes :

- Elle reprend les standards de communication exposés par Web3.js (une librairie Javascript offerte par la communauté pour faire communiquer des ĐApp Web) ;
- Elle permet notamment de communiquer en IPC (inter-process communication) depuis l'application vers le noeud ;
- Elle offre une gestion de flux de données simplifiée grâce à l'utilisation de RxJava.

Son but est d'abstraire au maximum les échanges entre une application Android et un client blockchain, facilitant ainsi son utilisation pour des développeurs non expérimentés aux notions de la blockchain.
À présent, intéressons-nous aux outils que nous mettons à disposition pour orchestrer toutes ces briques qui composent notre ĐApp mobile.

Lancer un noeud Geth 1.4 dans une application Android

Note : *Geth 1.5 sorti le 15 novembre 2016 propose une API totalement revue et incompatible avec ce nous proposons ici. Nous adaptons nos travaux à cette nouvelle version, ceux-ci devraient être disponibles prochainement. Après avoir récupéré et référencé la librairie Android-Geth dans une application Android, nous avons accès aux outils de lancement du noeud Geth.*

Pour pouvoir lancer le noeud Geth au démarrage de l'application, il faut créer sa propre classe *SampleApplication* qui étend *EthereumApplication*.

```
import com.sqli.blockchain.android_geth.EthereumApplication;

public class SampleApplication extends EthereumApplication {
    ...
}
```

Cette classe va se charger de démarrer le noeud Geth dans un service Android avec les paramètres suivants : Fast ; Lightkdf ; Nodiscover ; networkid **100 ;** datadir avec la valeur getFilesDir().getAbsolutePath() + "/node" ; genesis.

```
{
"nonce": "0x0000000000000042",
"timestamp": "0x0",
"parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
"extraData": "S Q L I blockchain",
"gasLimit": "0x8000000",
```

```
"difficulty": "0x500000",
"mixhash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
"coinbase": "0x0000000000000000000000000000000000000042",
"alloc": {    }
}
```

La classe va également permettre d'être notifiée une fois le service démarré (lorsque le fichier ipc est créé). Cette notification est disponible à plusieurs niveaux : *Activity* et *Application*. Pour la classe SampleApplication, nous devons surcharger la méthode onEthereumServiceReady() pour récupérer cet évènement. Cela permet par exemple d'instancier EthereumJava (qui sera présenté dans la seconde partie de cet article). Il est en effet possible de communiquer uniquement lorsque le noeud est totalement lancé. Pour récupérer cet évènement dans une *Activity*, il faut implémenter l'interface EthereumServiceInterface et s'enregistrer auprès de votre SampleApplication.

```
import com.sqli.blockchain.android_geth.EthereumService;

public class MainActivity implements EthereumService.EthereumServiceInterface{

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    [...]
    SampleApplication application = (SampleApplication) getApplication();
    application.registerGethReady(this);
  }

  @Override
  public void onEthereumServiceReady() {
    // Do something when Ethereum node is ready

    super.onEthereumServiceReady();
  }
}
```

À ce stade, vous avez une application Android et un noeud Geth qui cohabitent dans votre smartphone et vous êtes capable de faire réagir les interfaces de votre application lorsque le noeud est opérationnel.

Accéder au noeud Geth depuis des *Activities*

Pour le moment, le pont de communication entre l'application et le noeud blockchain n'est pas établi. Pour cela, il faut démarrer la communication IPC Android d'EthereumJava lorsque le noeud est démarré. Dans la surchage de onEthereumServiceReady(), il faut utiliser le builder de *EthereumJava* en prenant soin de bien choisir le provider AndroidIpcProvider qui permet (comme son nom l'indique) de faire de l'IPC sous Android.

```
[...]
public EthereumJava ethereumjava;
[...]
public EthereumJava getEthereumJava() {
  return ethereumjava;
}

@Override
public void onEthereumServiceReady() {
  ethereumjava = new EthereumJava.Builder()
      .provider(new AndroidIpcProvider(ethereumService.getIpcFilePath()))
      .build();

  super.onEthereumServiceReady();
}
```

AndroidIpcProvider prend en paramètre le chemin absolu où se trouve le fichier ipc. Comme présenté précédemment, le fichier se situe dans le répertoire de l'application (ethereumService.getIpcFilePath()). Grâce à cette architecture, nous avons accès à l'instance d'*EthereumJava* depuis n'importe quelle *Activity* via l'appel ((SampleApplication) getApplication()).getEthereumJava().

Piloter le noeud Geth

Le client Geth expose une API de gestion du noeud, décomposée en modules (eth, personal, admin, miner, net,...). Pour que l'utilisateur de la librairie ne soit pas totalement perdu dans la liste des fonctions disponibles, nous avons fait le choix du même regroupement en module que Web3.js et d'avoir une signature relativement similaire.
Il est donc possible de récupérer les informations sur le noeud :

```
NodeInfo nodeInfo = ethereumJava.admin.nodeInfo();
nodeInfo.enode //--> enode://id@ip:port
nodeInfo.ip   //--> adresse ip d'écoute du noeud geth
nodeInfo.id   //--> identifiant du noeud geth
```

ou encore de créer et dévérouiller des comptes :

```
String password = "passwd";
String accountId = ethereumJava.personal.createAccount(password);
boolean unlocked = ethereumJava.personal.unlockAccount(accountId);
```

L'API de Geth offre également la possibilité d'appeler les fonctions de manière asynchrone. Pour gérer le multithreading associé et une écriture de code asynchrone fluide, nous avons opté pour la librairie RxJava. Elle se base sur le pattern Observer, avec une API spécifique. Lors d'un appel asynchrone, un Observable est retourné permettant de définir le comportement notre application dans les différents cas du requêtage : nouvelle donnée, terminé ou erreur.

```
Observable<NodeInfo> observable = ethereumJava.admin.getNodeInfo()
observable.subscribe(new Action1<NodeInfo>() {
  @Override
  public void call(NodeInfo nodeInfo) {
    System.out.println(nodeInfo);
  }
});
```

Vous êtes à présent prêts à développer votre première application Android décentralisée, contenant un noeud blockchain complet. À ce stade, vous pourrez vous connecter à une blockchain depuis votre smartphone, créer des portefeuilles virtuels, émettre des transactions sur le réseau ou encore analyser en profondeur la blockchain.
Important : ce projet est actuellement en cours de développement et vous est proposé en version *alpha*, non stable. Il est destiné à un usage expérimental et ne doit en aucun cas être utilisé en production.
La librairie va continuer à évoluer. Par exemple, nous souhaitons limiter les contraintes d'architecture de l'application Android en déplaçant le lancement du service Geth en dehors de la classe Application. De plus, pour le moment, toutes les méthodes de l'API exposée par Geth ne sont pas implémentées et le système de threading actuellement en place complique l'utilisation de la librairie dans une application Android. Dans la suite de cet article, nous verrons comment communiquer avec un *smart-contract* depuis une application Android au travers d'Ethereum-Java. •

Modules très utiles pour Python

Partie 2

Franck Ebel,
Expert R&D et Formations
Serval-concept / serval-formation
Responsable Licence CDAISI, UVHC
Commandant de Gendarmerie réserviste, cellule Cyberdéfense

Python est un langage riche. Les modules que nous vous proposons dans cet article s'avéreront très utiles quand vous souhaiterez vous attaquer à des bases de données ou pour gérer plusieurs clients qui vont se connecter à un serveur que vous aurez créé.
Nous allons nous intéresser dans cet article à des modules utiles pour la suite. Nous verrons donc en détail les modules re, Thread, les connexions aux bases de données, le module shelve et pickles.

PostgreSQL

Il existe différentes méthodes pour se connecter en Python à une base de données PostgreSQL. Nous allons dans cette partie utiliser psycopg.
La méthode connect() de psycopg demande comme argument une chaîne de caractères contenant toutes les informations nécessaires pour établir une connexion avec le serveur. La fonction getdsn() nous permet de récupérer toutes ces informations.

```
import psycopg2

def getdsn(db = None, user = None, passwd = None, host = None):
    if user == None:
        import os, pwd
        user = pwd.getpwuid(os.getuid())[0]
    if db == None:
        db=user
    dsn='dbname=%s user=%s'%(db,user)
    if passwd != None:
        dsn += ' password=' + passwd
    if host != None:
        dsn += ' host=' + host
    return dsn

dsn=getdsn('python_db','franck',' ','localhost')
print("Connexion a %s"%dsn)
dbh=psycopg2.connect(dsn)
print("connexion reussie.")
ddh.close()
```

Ce qui donne :

```
franck:~ franckebel$ ./script.py
Connexion a dbname=python_db user=franck password=  host=localhost
connection reussie.
franck:~ franckebel$
```

Nous savons maintenant nous connecter à une base de données postgreSQL. Nous allons maintenant voir comment envoyer des commandes. Nous devons bien sûr avoir une base de données fonctionnelle pour tester les scripts suivants. Pour lancer n'importe quelle commande nous devons en premier lieu obtenir un curseur. Le concept du curseur est d'aider à simplifier les résultats des requêtes mais pour l'instant nous allons nous focaliser sur les commandes et non les résultats. Nous allons voir maintenant comment créer une table et charger quelques données.

```
import psycopg2

def getdsn(db = None, user = None, passwd = None, host = None):
    if user == None:
        import os, pwd
        user = pwd.getpwuid(os.getuid())[0]
    if db == None:
        db=user
    dsn='dbname=%s user=%s'%(db,user)
    if passwd != None:
        dsn += ' password=' + passwd
    if host != None:
        dsn += ' host=' + host
    return dsn

dsn=getdsn('python_db','franck',' ','localhost')
print("Connexion a %s"%dsn)
dbh=psycopg2.connect(dsn)
print("connection reussie.")
cur=dbh.cursor()
cur.execute("CREATE TABLE Programmez(mon_num integer UNIQUE, ma_chaine
 varchar(30))")
cur.execute("INSERT INTO Programmez VALUES (0)")
dbh.commit()
dbh.close()
```

Ce qui donne :

```
franck:~ franckebel$ ./script.py
Connexion a dbname=python_db user=franck password=  host=localhost
connection reussie.
franck:~ franckebel$

franck:~ franckebel$ psql -d python_db
psql (9.1.2)
Type "help" for help.

python_db=# \d
        List of relations
 Schema |    Name    | Type  | Owner
--------+------------+-------+-------
 public | Programmez       | table | franck
 public | pseudo     | table | franck
```

```
(2 rows)

python_db=#
```

Le script commence par se connecter de la même manière que précédemment. Ensuite nous créons un objet curseur (cursor()) puis nous lançons execute() trois fois. Nous finissons par faire un commit() .
Un pan intéressant des systèmes de transaction (nous en avons parlé dans la partie MySQL) est que les autres utilisateurs et programmes ne voient pas les changements que nous faisons jusqu'à ce que nous les ayons terminé (commit()). Voici un exemple qui utile ce concept :

```
import psycopg2

def getdsn(db = None, user = None, passwd = None, host = None):
    if user == None:
        import os, pwd
        user = pwd.getpwuid(os.getuid())[0]
    if db == None:
        db=user
    dsn='dbname=%s user=%s'%(db,user)
    if passwd != None:
        dsn += ' password=' + passwd
    if host != None:
        dsn += ' host=' + host
    return dsn

dsn=getdsn('python_db','franck',' ','localhost')
print("Connexion a %s"%dsn)
dbh=psycopg2.connect(dsn)
print("connection reussie.")
cur=dbh.cursor()
cur.execute("DELETE FROM eni")
cur.execute("INSERT INTO ProgrammezVALUES (0)")
dbh.commit()
dbh.close()
```

Ce script remplace le contenu de la table avec une seule ligne. L'avantage est que les utilisateurs peuvent encore lire dans la table, même après le DELETE et ce jusqu'à ce que commit() soit appelé. Un problème commun dans les bases de données est que le programmeur doit souvent faire la même commande de multiples fois. Avec SQL cela veut dire par exemple faire des INSERT INTO des milliers de fois. La fonction executemany() prend comme argument une commande et une liste d'enregistrements ; chaque enregistrement peut être une autre liste ou un dictionnaire.

```
import psycopg2

rows=({'num':0, 'text' : zero},{'num':1,'text':'Un'},{'num':2,'text':'Deux'},{'num':3,'text':'trois'})

def getdsn(db = None, user = None, passwd = None, host = None):
    if user == None:
        import os, pwd
        user = pwd.getpwuid(os.getuid())[0]
    if db == None:
        db=user
    dsn='dbname=%s user=%s'%(db,user)
    if passwd != None:
        dsn += ' password=' + passwd
    if host != None:
        dsn += ' host=' + host
    return dsn

dsn=getdsn('python_db','franck',' ','localhost')
print("Connexion a %s"%dsn)
dbh=psycopg2.connect(dsn)
print("connection reussie.")
cur=dbh.cursor()
cur.execute("DELETE FROM eni")
cur.executemany("INSERT INTO ProgrammezVALUES (%(num)d,%(text)s)",rows)
dbh.commit()
dbh.close()
```

Ce script va insérer les quatre lignes contenues dans rows. Quand la base de données est créée et fonctionnelle, il faut bien sûr pouvoir récupérer nos données pour l'application. Les requêtes sont envoyées via la méthode execute() et les résultats sont obtenus via une méthode fetch(). Il existe différentes méthodes fetch. La méthode fetchall va récupérer tous les champs et va retourner une liste. Chaque élément de la liste correspond à un enregistrement. La totalité de la requête est chargée en mémoire.

```
import psycopg2

def getdsn(db = None, user = None, passwd = None, host = None):
    if user == None:
        import os, pwd
        user = pwd.getpwuid(os.getuid())[0]
    if db == None:
        db=user
    dsn='dbname=%s user=%s'%(db,user)
    if passwd != None:
        dsn += ' password=' + passwd
    if host != None:
        dsn += ' host=' + host
    return dsn

dsn=getdsn('python_db','franck',' ','localhost')
print("Connexion a %s"%dsn)
dbh=psycopg2.connect(dsn)
print("connection reussie.")
cur=dbh.cursor()
cur.execute("SELECT * FROM eni")
rows=cur.fetchall()
print rows
for row in rows:
    print row
dbh.commit()
dbh.close()
```

Ce qui donne :

```
franck:~ franckebel$ ./script6.py
Connexion a dbname=python_db user=franck password=  host=localhost
connection reussie.
(0,'ZERO')
(1,'Un')
(2,'Deux')
(3,'trois')
```

Nous pourrons aussi utiliser fetchmany() pour récupérer ligne par ligne ;

```
#!/usr/local/bin/python3
#--*--coding:utf8--*--

import psycopg2


def getdsn(db = None, user = None, passwd = None, host = None):
    if user == None:
        import os, pwd
        user = pwd.getpwuid(os.getuid())[0]
    if db == None:
        db=user
    dsn='dbname=%s user=%s'%(db,user)
    if passwd != None:
        dsn += ' password=' + passwd
    if host != None:
        dsn += ' host=' + host
    return dsn

dsn=getdsn('python_db','franck',' ','localhost')
print("Connexion a %s"%dsn)
dbh=psycopg2.connect(dsn)
print("connection reussie.")
cur=dbh.cursor()
cur.execute("SELECT * FROM eni")
cur.arraysize=2
while 1:
  rows=cur.fetchmany()
  print("Obtenu %d resultats de fetchmany()."%len(rows))
  if not len(rows):
     break
  for row in rows:
     print(row)
dbh.close()
```

Nous pourrons utiliser aussi fetchone() pour récupérer les résultats un à un. D'autres méthodes sont disponibles pour récupérer certaines données ou les traiter, mais la documentation bien fournie sur le site de python.org ou sur différents sites Web dédiés à Python vous permettra de trouver ce dont vous aurez besoin dans vos futurs développements

Module thread

Le thread est le moyen pour un programme d'effectuer une tâche plusieurs fois en parallèle. Nous pourrons donc dans un script lancer un certain nombre de fois la même tâche sur un seul processeur. Le module thread va nous permettre de travailler avec de multiples threads.

```
import _thread

def bonjour(nbr):
  print('Bonjour du  thread %s\n'%nbr)
_thread.start_new_thread(bonjour,(0,))
_thread.start_new_thread(bonjour,(1,))
_thread.start_new_thread(bonjour,(2,))
```

Nous créons ici trois thread, chacun écrivant à l'écran « bonjour du thread » avec son numéro correspondant qui est donné en argument lors du démarrage de chaque nouveau thread. Le résultat à l'écran devrait être :

- Bonjour du thread 0
- Bonjour du thread 1
- Bonjour du thread 2

Threading

Ce module threading est un autre moyen de travailler avec les threads. Nous retrouverons dans threading une classe thread() et une méthode run(). Nous devrons instancier un objet thread et appeler la méthode start(). La méthode run() sera exécutée dans un nouveau thread. Cette méthode contiendra le code que nous voulons exécuter en parallèle.

```
from threading import Thread

class MyThread(Thread):
  def run(self):
     print('Bonjour du thread %s' % self.name)

for i in range(3):
 my_thread = MyThread()
 my_thread.name = i
 my_thread.start()
```

Ce script va créer trois thread et nous aurons à l'écran :

- Bonjour du thread 0
- Bonjour du thread 1
- onjour du thread 2

class Lock()

Crée une nouvelle primitive de synchronisation. Deux méthodes sont ensuite accessibles : acquire() et release().
acquire([blocking=1]) : acquiert le verrou et renvoie True en cas de succès. Si blocking est à 1 ou n'est pas spécifié, l'appel de cette méthode bloque le thread si le verrou est déjà locké par un autre thread. Si blocking est à 0, acquire() se contente de renvoyer False pour signaler que le verrou est déjà pris.
Release() : libère le verrou, autorisant d'autres threads à le reprendre. Si plusieurs threads sont en attente de ce verrou, un seul thread est autorisé à l'acquérir. Appeler cette méthode sur un verrou qui n'est pas fermé lève une exception.
La classe Rlock est identique mais permet au thread qui a le verrou de rappeler la méthode acquire() sans provoquer de deadlock. Cette variation simplifie grandement la conception du code, surtout lorsque des fonctions récursives entrent en jeu. Rlock est un lock réentrant.

```
import time
From_threading import Thread

THREADS = 3

def main():
  manager = ThreadManager()
  manager.start(THREADS)


class ThreadManager:
  def __init__(self):
    pass
```

```
    def start(self, threads):
        thread_refs = []
        for i in range(threads):
            t = MyThread(i)
            t.daemon = True
            print('démarrage du thread %i' % i)
            t.start()
        for t in thread_refs:
            t.join()


class MyThread(Thread):
    def __init__(self, i):
        Thread.__init__(self)
        self.i = i

    def run(self):
        while True:
            print('Bonjour du thread # %i' % self.i)
            time.sleep(.25)


if __name__ == '__main__':
    main()
```

Nous créons ici 3 threads, ceci est déclaré dans la variable THREADS.
Une première classe MyThread() est créée avec une méthode run() qui va afficher à l'écran « Bonjour du thread » puis son numéro.
Une attente de 0.25 seconde sera alors générée.
Une deuxième classe ThreadManager contient la méthode start().
Celle-ci prend comme paramètre le nombre de threads et va donc créer autant de threads que nécessaire grâce à la boucle for.
Elle nous donnera comme indication que le thread démarre.

Démarrage du thread 0
Bonjour du thread # 0
Démarrage du thread 1
Bonjour du thread # 1
Démarrage du thread 2
Bonjour du thread # 2

CONCLUSION

Grâce aux modules que nous venons d'étudier, nous allons pouvoir commencer à créer des scripts assez conséquents qui vont pouvoir automatiser beaucoup de tâches. Dans un prochain article nous appliquerons ce que nous avons vu afin de créer des scripts "utiles" et réels puisque utilisé dans le cadre de mes audits en sécurité et en forensic.
Vous pouvez retrouver mes vidéos sur python, scapy, le forensic en python sur ma chaîne youtube : http://www.youtube.com/c/franckebel •

Ce qui compte le plus dans un projet

Une publication Nefer-IT, 7 avenue Roger Chambonnet, 91220 Brétigny sur Orge - redaction@programmez.com
Tél. : 01 60 85 39 96 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : S. Saurel
Nos experts techniques : P. Toth,O. Denoo, C. Villeneuve, M. Philippon, M. Perrin, M. Grief, J. Paquet, A. De Amaral, M. Dulac, I. Kempf, F. Gutierrez, C. Dufour, J. Raffre, R. Pinon, N. Gachadoit, V. Fading, A. Nourissier, F. Bouteruche, A. Callebat, I. Kouraïchi, S. Sound, J-M Heitz, L. Andaloro, I. Leontief, V. Thavonekham, G. Nicolas, D. Lecan, F. Ebel, Commitsrip.
Couverture : © DragonImages - **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 -
Publicité : pub@programmez.com - Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, avril 2017

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

Ou encore sur votre tablette