OPENCV 4.0 | OCR EN PYTHON | JDBI | GO

programmez.com

Programmez!

Le magazine des développeurs

227 MARS 2019

ASSISTANTS VOCAUX

Créer / Développer / Les usages

© AlexLMX

Les tests sont-ils les meilleurs amis du développeur ?

© monsitj

M 04319 - 227 - F: 6,50 € - RD

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

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

100% RÉSISTANT À L'EAU

EXTRÊMEMENT STABLE

POSE FACILE

COMPATIBLE TOUTES SURFACES

RÉSISTANT AUX RAYURES

SILENCIEUX

RÉSISTANT AUX TACHES

US PATENT APPROVED # 9,156,233

ÉMISSIONS DANS L'AIR INTÉRIEUR* A+

*Information sur le niveau d'émission de substances volatiles dans l'air intérieur, présentant un risque de toxicité par inhalation, sur une échelle de classe allant de A+ (très faibles émissions) à C (fortes émissions).

EN 14041

DOP-ELV

USFLOORS Int IIc
Industriezone 10
9770 Kruishoutem
Belgium

resilient floor covering

GALERIE RENOVATION

COREtec® the Original

EDITO

Bug ou feature ? Si on ne pouvait plus troller, la vie de dév serait triste

Débat sans fin : bug ou feature ? Chacun aura son avis. Le bug est une fonction non prévue à l'origine. C'est vrai que si cette nouvelle fonction fait planter son app toutes les 30 secondes, cette feature n'est pas la meilleure.

Eh oui, c'est la saison du dossier test de Programmez! Chaque année, mon passé de testeur me rattrape. Revenons aux fondamentaux : qu'est-ce que le test, les différents types de tests (non ? pas possible ! // mode sarcasme), pratique du test notamment avec des services cloud. Et n'oubliez pas la journée française du test logiciel en avril prochain !

Quelques exemples qui démontrent l'importance des tests :

1 : l'alunissage de la mission Apollo 11 a failli échouer suite à une alarme et une série de messages d'erreurs. L'ordinateur de bord recevait trop d'informations et avait fini par saturer, bref un overflow ! Reboot ! Mais cela n'influença pas la suite de la mission.

2 : la Nasa perd la mission Mars Polar Lander suite à une erreur logicielle qui avait mal interprété les données : le système crut que la sonde était en phase d'atterrissage sur Mars. Et hop, on coupe les moteurs bien trop tôt et la sonde s'écrase ! Et cet échec mit aussi en évidence un autre problème : l'utilisation du système métrique et du mile. Une erreur de conversion entre les deux systèmes de mesure avait contribué à la perte de la sonde…

3 : Décembre 2017 : blocage du nouveau système informatique de la gare Montparnasse (Paris) gérant notamment l'aiguillage.

4 : En 1996, le premier vol d'Ariane 5 fut un échec. Le dépassement d'une valeur a provoqué l'autodestruction. La limite logicielle était en réalité celle utilisée dans les logiciels d'Ariane 4. Or l'accélération d'Ariane 5 était bien plus forte…

5 : Nike déploie un logiciel de logistique trop rapidement plus ou moins stabilisé, sur un outil de commande ancien. Résultat ? Plusieurs millions de chaussures sont commandées par erreur… Comme quoi, parfois, mieux vaut être prudent quand on remplace des briques logicielles.

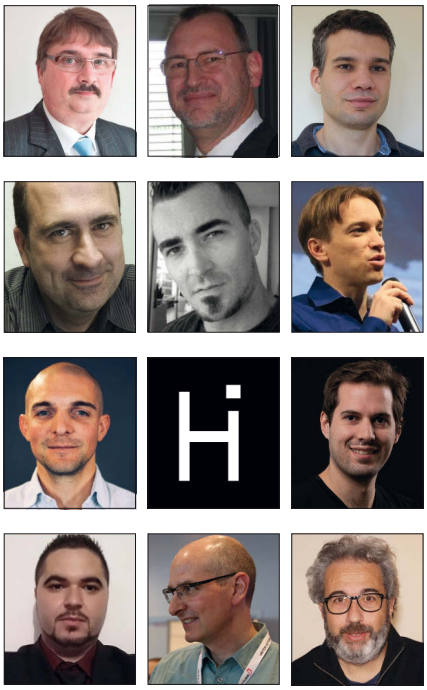
6 : Printemps 2018, plus de 450 000 certificats d'immatriculation ne pouvaient pas être traités et générés dans les préfectures « virtuelles ». La cause était un afflux de demandes et une cascade de bugs du logiciel déployé.

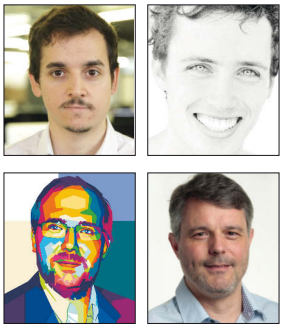
7 : avril 2018, 8 500 bureaux de poste sont paralysés durant une demi-journée. L'informatique est totalement plantée !

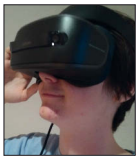
Notre dépendance à la technologie, à l'informatique, est tellement énorme que le moindre plantage, ou une mauvaise implémentation, a une conséquence en cascade sur nos vies, surtout quand cela touche aux services publics, aux transports, etc.
Ressortez les bouliers et les bougies.

François Tonic
ftonic@programmez.com

SOMMAIRE







Dans le prochain numéro !
Programmez! #228, dès le 29 mars 2019

Java 12 :
toutes les nouveautés du nouveau Java

Plongée au coeur de PostgreSQL

Dernière ligne droite pour la directive copyright

La directive copyright actuellement débattue par les autorités européennes fait trembler les géants du net américain. Les articles 11, qui instituent un droit voisin pour les articles de presse, et 13, qui impose une détection automatique des contenus par les éditeurs de services, sont au centre des polémiques. Le texte est récemment sorti de la phase de « trilogue », ce qui signifie que le parlement, la commission et le conseil de l'UE sont tombés d'accord sur une version commune du texte. Les articles 11 et 13 sont maintenus, mais des exceptions ont été prévues afin de limiter l'application de ces articles aux acteurs les plus importants. Le texte doit maintenant être approuvé par le parlement en séance plénière et par le conseil de l'UE.

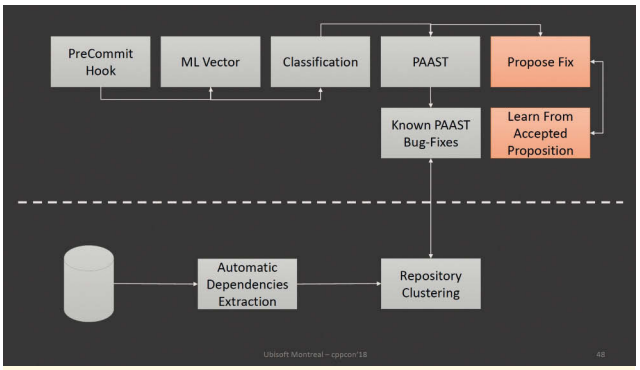
Internet : la Russie veut pouvoir se couper du monde

La Russie a annoncé vouloir procéder à un test visant à vérifier si le réseau internet russe pouvait fonctionner en vase clos, coupé du reste de l'internet. Cette mesure vise à garantir la résilience du réseau russe et des principaux services en cas de conflit informatique avec d'autres grandes puissances, selon une proposition de loi discutée devant le parlement russe. Ce test vise à récolter les retours des acteurs du secteur afin de nourrir les amendements de ce projet de loi et vérifier que tout fonctionnera bien comme prévu le jour où le cyberespace russe décidera de larguer les amarres.

bitcoin

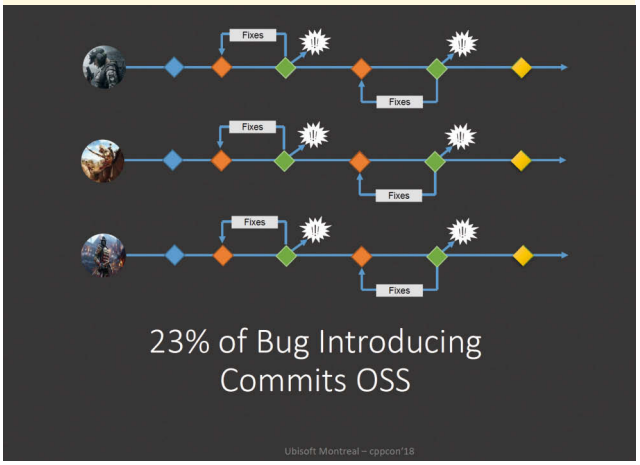
145 millions $ en cryptomonnaie gelés après la mort du propriétaire

QuadrigaCX est une bourse d'échange de cryptomonnaie comme il y en a aujourd'hui beaucoup. Mais celle-ci a annoncé au mois de février la mort de son fondateur. Problème : celui-ci aurait apparemment été l'unique possesseur du mot de passe permettant d'accéder aux cryptomonnaies stockées sur le porte-monnaie hors ligne de la place de marché. Soit l'équivalent de 145 millions de dollars. Bien évidemment, le trou dans la trésorerie fait craindre le pire pour QuadrigaCX, et certains envisagent la possibilité que toute l'affaire ne soit qu'un coup monté du fondateur pour se tirer avec la caisse. Encore une belle journée dans le fabuleux monde des cryptomonnaies.

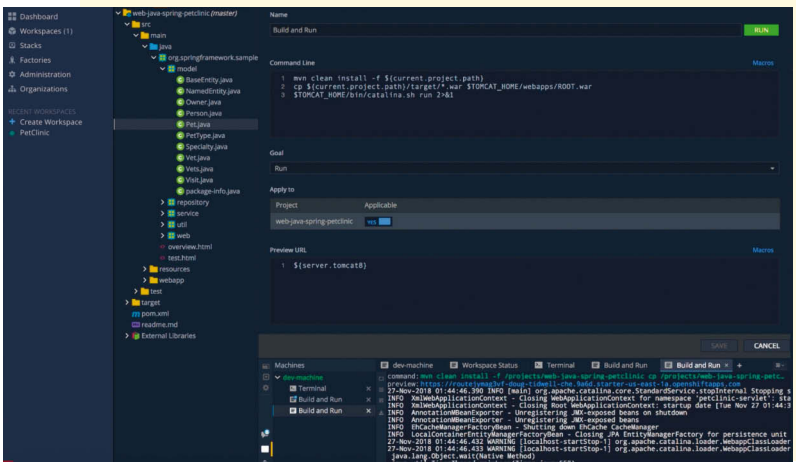


UBISOFT ET MOZILLA : un partenariat pour inspecter les travaux finis

Les développeurs de Firefox ont annoncé un partenariat avec Ubisoft. Mozilla entend en effet utiliser un outil développé en interne par Ubisoft, Clever Commit. Celui-ci se présente comme un outil d'assistance au développement et propose d'analyser les commits poussés sur un gros projet afin de détecter les bugs avant que ceux-ci ne soient poussés sur la branche principale du projet. Clever Commit a recours à des outils de machine learning pour analyser et reconnaître le code susceptible de contenir des bugs.



Red Hat lance CodeReady Workspace



Kubernetes se dotera lui aussi de son environnement de développement. Red Hat a ainsi annoncé la mise à disposition d'un environnement de développement intégré baptisé CodeReady Workspace. Cet environnement se distingue par une intégration complète avec Kubernetes : l'environnement fonctionne ainsi sur un cluster Kubernetes. Celui-ci s'exécute sur le Paas Open Shift proposé par Red Hat. Et fonctionne entièrement dans le navigateur du développeur.

Vote électronique : la Suisse met son système à l'épreuve

La Suisse entend étendre son système de vote électronique à l'échelle fédérale. La poste Suisse a proposé une implémentation du système, mais ouvre un bug bounty afin de vérifier la fiabilité et la sécurité du système. Les récompenses prévues s'élèvent jusqu'à 30 000 francs suisses, soit 26 000 euros et le bug bounty sera ouvert du 25 février au 24 mars 2019. Pendant cette période, les chercheurs en sécurité sont donc invités à tenter de trouver les failles du système et à les signaler auprès des organisateurs.

PARIS 16 & 17 MAI

+ JOURNÉE WORKSHOPS LE 15 MAI

{ newcrafts }

April Wensel
fondatrice de Compassionate Coding
@aprilwensel

Alberto Brandolini
créateur de l'Event Storming
@ziobrando

Emily Bache
auteur de The Coding
Dojo Handbook @emilybache

Brian Marick
signataire du manifeste Agile
et auteur @marick

et bien
d'autres...

Newcrafts 6e édition
c'est le rendez-vous
indépendant
& international
des développeur.se.s
pro & passionnés !

#craftingSoftware #ddd #agile
#devOps #design #bigData
#functionalProgramming
#architecture #opensource

{ TECHNOLOGIES }
{ PRATIQUES }
{ ARCHITECTURES }

PASS 2 OU 3 JOURS
toutes les infos sur ncrafts.io
NOTRE CFP EST OUVERT ! cfp.ncrafts.io

Vous avez envie de nous soutenir ? Devenez sponsor ! sponsoring@ncrafts.io

Baromètre Hired de la recherche d'emploi
Android et iOS se rebiffent

Habituellement cantonnées aux dernières places du baromètre mensuel Hired de la recherche d'emploi des développeurs, les technologies Android et iOS figurent respectivement à la deuxième et la quatrième place de ce même baromètre au mois de janvier, aux côtés des valeurs sûres que sont React, DevOps et Vue.

En janvier, les technologies Android et iOS ont ainsi généré pour les profils Tech qui les avaient mises en avant 6,2 et 5,9 demandes d'entretien en moyenne. Au mois de décembre, le nombre de propositions reçues était de 4,7 pour Android et de 3,9 pour iOS. « Nous constatons que les entreprises ont de plus en plus besoin de développeurs mobiles, notamment du côté des grands comptes », explique Antoine Garnier-Castellane, directeur du bureau français de Hired. Intercalés entre Android et iOS, on retrouve ensuite à la première et la troisième place, React, avec une moyenne de 6,8 propositions d'entretien et DevOps, qui en a généré 6. Cette technologie, qui reste une des plus difficiles à trouver (moins de 1 % des inscrits sur Hired en janvier) remonte ainsi très haut dans le classement après un mois de décembre lors duquel elle n'avait pas déchaîné les recruteurs avec seulement 3,5 demandes d'entretiens. Vue, Angular, Node et Python suivent ensuite dans le classement s'affirmant comme des valeurs sûres. De l'autre côté du baromètre, on retrouve PHP et Go avec 4 et 4,2 demandes d'entretiens générées respectivement. C'est d'ailleurs une belle dégringolade pour le Go qui était en décembre crédité de 6,2 propositions d'entretiens et trônait sur la deuxième place du baromètre. Au cours des 6 derniers mois, Go est d'ailleurs détrôné par DevOps qui lui prend la deuxième place du classement, derrière React. Node, Ruby, Vue et Python suivent ensuite.

Du point de vue des profils, Java, Node, PHP, React et Python restent les technologies les plus représentées sur Hired et totalisent à elles cinq plus de 70 % des candidatures sur Hired en janvier, comme sur les six derniers mois. DevOps, Ruby et iOS ferment la marche en représentant respectivement 0,93 %, 2,34 % et 2,8 % des profils en janvier. Pour ces derniers, le constat est également similaire sur les 6 derniers mois et les chiffres n'évoluent que très peu.

Tous les mois, Programmez ! publie en exclusivité le baromètre Hired des technologies les plus recherchées par les entreprises. Elles peuvent permettre aux développeurs de connaître les nouvelles tendances du recrutement pour se former ou se démarquer des autres candidats.

Janvier 2019

Technologies demandées	Pourcentage de candidatures développeurs	Technologies demandées	Nombre moyen de demandes d'entretiens
Java	17,06 %	React	6,8
Node	16,36 %	Android	6,2
PHP	13,79 %	DevOps	6
React	12,85 %	iOS	5,9
Python	12,85 %	Vue	5,6
Angular	7,24 %	Angular	5,6
Vue	4,67 %	Node	5,3
Android	4,67 %	Python	4,7
,NET	4,21 %	,NET	4,6
Go	3,04 %	Ruby	4,6
iOS	2,80 %	Java	4,3
Ruby	2,34 %	Go	4,2
DevOps	0,93%	PHP	4

De août 2018 à janvier 2019

Technologies demandées	Pourcentage de candidatures développeurs	Technologies demandées	Nombre moyen de demandes d'entretiens
Java	19,09 %	React	7,5
Node	15,34 %	DevOps	6,9
Python	12,83 %	Go	6,3
PHP	11,74 %	Node	6,2
React	11,27 %	Ruby	6,2
Angular	9,15 %	Vue	5,8
Android	5,24 %	Python	5,6
,NET	4,69 %	Angular	5,3
Go	3,60 %	Java	4,7
Vue	3,05 %	PHP	4,5
Ruby	2,97 %	Android	4,4
iOS	2,74 %	,NET	4,2
DevOps	1,02%	iOS	4,2

Congrès & Expo

∞e édition

BIGDATA PARIS BY Corp

Big Data Paris revient
les 11 & 12 mars 2019
au Palais des Congrès

ACCELERATE THE FUTURE !

Venez vivre une expérience riche au cœur du Big Data !

- **Vous informer grâce aux conférences et ateliers**
- **Découvrir les nouveautés technologiques**
- **Networker avec 250 exposants et 17 000 visiteurs**

Inscrivez-vous des maintenant sur www.bigdataparis.com

François Tonic

Boitier eGPU : faut-il réellement l'utiliser ?

Votre GPU est trop ancien ou trop poussif pour vos jeux, la VR/VA, la 3D, les affichages 4k ou 5k ? Au lieu de changer de machine ou de carte, les boîtiers externes eGPU sont souvent cités comme LA solution. Oui et non. Déjà le premier impératif est de disposer d'un port Thunderbolt 3 sur sa machine.

Rappelons que l'eGPU permet d'utiliser une carte graphique en externe, à la place ou parallèlement à une GPU interne. Généralement, cette solution sert à des opérations très lourdes comme le montage vidéo temps réel, les jeux, la 3D, etc. Quand on commence à installer des écrans 4K ou même un immense écran 42'', mieux vaut un GPU qui dépote. Et dans ce cas, seul le Thunderbolt 3 offre des débits suffisants.

Toutes les GPU du marché ne sont pas supportés et ce, pour plusieurs raisons :

- Certains boîtiers sont trop courts pour installer certaines cartes ;
- La consommation en watts de certaines cartes est si importante qu'il n'est pas possible de les utiliser ;
- Les constructeurs de boîtiers indiquent les GPU (officiellement) supportés.

Windows supporte plutôt bien les eGPU et les GPU supportés sont nombreux. Sur Mac, les dernières versions de macOS étendent enfin le support de ces boîtiers mais les GPU supportés sont très limités. On a le choix entre AMD et AMD. Certaines NVidia peuvent être utilisées, tout dépend du boîtier et des pilotes disponibles.

Sur Linux, le eGPU peut être utilisé mais le support est rudimentaire sur de nombreuses distributions. Et les constructeurs de boîtiers ne listent pas Linux dans les plateformes supportées.

Prix du boîtier nu (sans GPU) :
à partir de 399 €

Soyons clairs : si votre objectif est d'accélérer l'affichage de votre IDE, le eGPU n'aura aucun intérêt. Ni en temps de compilation.

ANDRÉ : UNE MACHINE ASSEMBLÉE EST UN BON COMPROMIS.

Amateur de C++, André recherchait une machine à la fois modulaire et puissante. Il s'est naturellement intéressé aux machines sur mesure. Il utilisera sa machine principalement pour Visual C++. La compilation se fait en parallèle sur un Mac Mini 2018 (via LLVM) pour la compatibilité macOS. Pour la compilation Linux, un Mac Mini 2014 a été passé sous Linux et la compilation passe sur GCC.

Son choix s'est porté sur :

- Boîtier Fractal R6 Black
- Intel Core i9 9900K
- Carte mère Asus Prime Z390A
- Stockage : Samsung Serie 970 EVO NVMe M.2 1 To
- Alimentation 750W d'Antec EA750G Pro
- GPU : MSI GeForce FTX 1060 Gaming X 6 Go
- Mémoire : Corsair Vengeance LPX Black DDR4 2666 Mhz CAS 2x16 Go
- Graveur DVD Asus
- Windows 10 Home 64 bits
- Ventilateur : Noctua NH-U12S
- Carte WiFi Asus

Tarif : 2 600 € (montage inclus)

Notre avis : une belle configuration avec des composants de qualité. La carte mère choisie propose une connectique complète. Le seul défaut : l'absence de Thunderbolt 3. La partie GPU offre un bon ratio prix / performance. Un GPU de type 1080 Ti aurait été intéressant mais à un prix largement au-dessus du 1060 et le budget de la machine aurait dépassé les 3 000 € ! L'usage de la machine et l'écran de 29'' Ultra Wide LG ne nécessitent pas une carte supérieure.

STÉPHANE MISE SUR UN POSTE LÉGER MAIS COMPLET

Stéphane développe depuis +20 ans. Aujourd'hui, il mise sur un poste de travail à la fois minimaliste mais très complet. Fini les gros PC ou les portables, désormais il utilise une Surface Pro 4 de Microsoft équipée de 8 Go de mémoire et de 286 Go de stockage. Comme le confort visuel est important, surtout quand on développe sur Visual Studio, sous Windows 10, il connecte 3 écrans, dont 1 écran de 27'' et 1 de 22'' via le Surface Dock. La résolution est de 1920 x 1080. Cela peut paraître faible mais au quotidien, elle est bien adaptée.

En complément, Stéphane utilise deux petits écrans : 1 de 12'' et de 8''. Ils servent avant tout aux plannings et aux schémas des fonctions à développer.

NOUVEAUX BOÎTIERS CHEZ MSI

Le construction MSI propose deux nouveaux boîtiers orientées gamers :

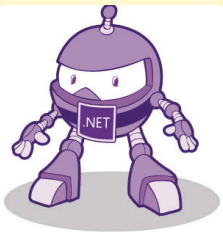
- MPG Gungnir 100 : design plutôt agressif et pas désagréable. Il propose 7 slots, 2 baies 3,5, rétroéclairage ;
- Mag Vampiric 010 : design plus sage mais tout aussi imposant que le premier. Flux d'air optimisé.

Il faudra patienter encore quelques semaines. Prix non communiqués.

MEETUP PROGRAMMEZ! #3

SPÉCIAL

.NET CORE 3.0
VISUAL STUDIO 2019



14 mars à partir de 19h

Les intervenants de la soirée :

Andrés Talavera
(développeur, Ideastud.io)

Pierrick Blons
Team Leader Azure (SQLI)

Nicolas Gautier
Chef de Projet Microsoft (SQLI)

Où

Arolla
21, rue du bouloi
75001 Paris
Batiment D au fond à gauche

Métros / RER :
Métro 1 : station Louvre-Rivoli
Métro 4 / RER A, B & D : Châtelet - Les Halles

Inscriptions & infos pratiques : https://www.programmez.com/content/programmez-meetup-3-inscriptions

Nos prochains meetups :

25 avril : *C++ et les dernières évolutions*
28 mai : *Java 12, le nouveau Java*
25 juin : *quantique, vers l'infini et au-delà*

Meetups organisés par [Programmez!] Le magazine des développeurs

Mars

A partir du 5 mars : .NEXT On Tour / France

Dans toute la France, Nutanix mettra cette année l'accent sur les déploiements cloud hybride et multicloud, notamment en présentant les dernières solutions esquissées lors de son évènement .Next Europe qui s'est tenu à Londres en novembre dernier.
Les évènements .NEXT on Tour de Nutanix sont l'opportunité pour les équipes IT de découvrir les dernières technologies cloud hybrides et leurs effets sur la transformation d'entreprise et de fixer le cap de leurs migrations vers le cloud.

- Lille, le 5 mars
- Lyon, le 12 mars
- Marseille, le 14 mars
- Toulouse, le 21 mars
- Nantes, le 26 mars
- Paris, le 9 avril

Agenda et informations :
https://www.nutanix.com/next/on-tour/

11 & 12 mars : BigData Paris

L'évènement big data revient à Paris. Cette année, les secteurs banque/assurance seront à l'honneur dans les conférences. Ce salon est l'occasion pour rencontrer et échanger avec les principaux acteurs du monde big data et de la donnée en général.
https://www.bigdataparis.com/2019/

20 au 22 Breizh Camp / Rennes

BreizhCamp est le rendez-vous Rennais qui regroupe les développeurs de tous poils pour un moment de convivialité et de partage. Pour sa 9ème édition, et conformément au rituel bien établi, BreizhCamp a choisi une thématique inspirante. Vous serez donc invités à rejoindre les 'CodeBusters' pour chasser le code mort dans le hall de l'Université Rennes 1, du 20 au 22 mars. Si, comme les années précédentes, exposants et speakers sont inspirés par ce thème, ça promet d'être épique. https://www.breizhcamp.org

Avril

9 : Journée française des tests logiciels

Cet évènement a su s'imposer au fil des années comme le rendez-vous fédérateur de la communauté française des tests logiciels. **En effet, avec plus de 1 000 participants l'année dernière, 43 sponsors et 16 conférences thématiques** mettant à l'honneur les témoignages de grands comptes et organisations, la JFTL est aujourd'hui un lieu d'échanges privilégié de l'ensemble des acteurs de ce marché, qui peuvent à cette occasion, rencontrer les experts les plus reconnus du secteur et échanger leurs expériences entre professionnels.
http://www.cftl.fr/JFTL/accueil/

Du 17 au 19 : Devoxx France

https://www.devoxx.fr

Mai

Du 15 au 17 : Riviera Dev / Sophia Antipolis

http://rivieradev.fr

16 & 17 : Newcrafts / Paris

La conférence sur la qualité du code a ne pas rater ! Cette 6e édition continuera à explorer la qualité du code, les bonnes pratiques.
https://ncrafts.io

23 & 24 : Android Maker / Montrouge

La grande conférence Android française revient pour une nouvelle édition ! Comme durant les précédentes éditions, l'objectif est de parler Android, écosystème, développement, roadmap, retour d'expérience. L'agenda sera connu dans quelques semaines.
Site : https://androidmakers.fr

23 & 24 : Mixit / Lyon

https://mixitconf.org/fr/

Juin

4 : Paris Container Day / Paris

Le 4 juin prochain aura lieu la quatrième édition du Paris Container Day. Le Paris Container Day est la conférence pionnière en France dédiée à l'écosystème des conteneurs et de ses bonnes pratiques. Pour cette nouvelle édition, les participants pourront assister à une vingtaine de conférences techniques, retours d'expériences et des fast tracks. Le thème de cette année est "Standards & Craftsmanship". Organisée par Zenika.
https://paris-container-day.fr

6 & 7 : Best of Web / Paris

http://bestofweb.paris

11 au 14 : INFORSIF / Paris

Depuis 1982 le congrès INFORSID (INFormatique des ORganisations et Systèmes d'Information et de Décision) rassemble chaque année des chercheurs et des professionnels afin d'échanger sur les problématiques d'ingénierie et de gouvernance des systèmes d'information.
Site : http://inforsid.fr/Paris2019/

14 : DevFest Lille

Le DevFest Lille Saison 3, est la 3ème édition d'un évènement complet de conférences sur le web, le mobile, le cloud et les objets connectés. Plus de 700 participants attendus cette année au Kinépolis de Lomme, le plus grand cinéma d'Europe. Une nouvelle saison pleine de surprises pour ce Devfest Lille sous le thème des Séries TV qui se déroulera le 14 juin prochain
https://devfest.gdglille.org

24 au 28 :
COMPAS 2019 / Bayonne
Compas est la Conférence francophone en informatique autour des thématiques du parallélisme, de l'architecture et des systèmes. L'édition 2019 aura lieu du 24 au 28 juin à l'IUT de Bayonne.
Site : https://2019.compas-conference.fr/

27 & 28 :
Sunny Tech / Paris
https://sunny-tech.io

Concours Prologin

Le concours national d'informatique Prologin est ouvert à tous les jeunes de 20 ans et moins. Ce concours entièrement gratuit est l'occasion pour les jeunes passionnés d'informatique de démontrer leurs talents et de rencontrer d'autres passionnés d'informatique. Prologin est organisé par des étudiants de l'EPITA, de l'École Normale Supérieure, ainsi que de l'École Polytechnique. La grande finale se déroulera en mai prochain à l'EPITA Paris : 36 heures de code pour les 100 meilleurs candidats !
Site : https://prologin.org

Girls Can Code !

Les stages Girls Can Code ! c'est une semaine d'initiation à la programmation pour les collégiennes et lycéennes. On y apprend les bases de la programmation avec le langage Python, et de quoi se lancer dans des petits projets de son choix ! Les stages sont gratuits, entièrement organisés par des bénévoles de l'association Prologin.
Au programme : apprentissage du code, ateliers pratiques, réseaux. Une occasion de découvrir le métier de développeur et le monde de l'informatique. Une excellente initiative que nous ne pouvons que soutenir !
Pour en savoir plus : https://gcc.prologin.org/

Le coin maker

Tech Inn'Vitré :
du 1er au 3 mars à Vitré.

TFEEA :
le 5 mars à Paris (ministère de l'Économie) Makers et travailleurs en situation de handicap.

Metromix :
28 & 29 mars (hackathon sur la mobilité urbaine).

Makeme Fest Nantes :
du 12 au 14 avril sur la Foire de Nantes.

Makeme Fest Angers :
du 25 au 29 avril sur Foire d'Angers.

[Programmez!] Le magazine des développeurs

Conférences Programmez!

Retrouvez tous les meetups Programmez! :

14 mars à partir de 19h / Paris :
.Net Core 3 & Visual Studio 2019

25 avril :
C++ 17 et les prochaines évolutions

28 mai :
Java 12, le nouveau Java

25 juin :
Quantique, vers l'infini et au-delà

Agenda complet et inscription :
https://tinyurl.com/y5svgs2j

Retrouvez les vidéos des dernières conférences DevCon sur :
https://tinyurl.com/yyesy9yb

CONFÉRENCES ORGANISÉES PAR ZENIKA

27 juin : 2ème édition du DataXDay

https://dataxday.fr/

DataXDay est l'unique conférence Data qui réunit DataScientists, Data Engineers et Data Architects autour d'une quinzaine de conférences techniques. Venez échanger autour de la Data Science du PoC à la mise en production, des architectures Microservices et Serverless, de Craftsmanship, de sécurité, de Cloud, etc. Rendez-vous le 27 juin prochain.

7-8 Octobre : 4ème édition de la FrenchKit

La FrenchKit sera de retour pour une quatrième édition les 7 et 8 octobre prochains. La FrenchKit est la première conférence iOS et macOS en France. Durant ces 2 journées, certains des développeurs les plus reconnus de la communauté internationale traiteront un large éventail de sujets, des API Cocoa à Swift.

28 novembre : XebiCon, Build The Future

https://xebicon.fr/

A travers une soixantaine de conférences, la Xebicon vous donnera les clés pour tirer le meilleur des dernières technologies. Conférence techniques, retours d'expériences clients, hands-on, venez partager et échanger sur les nouveautés technologiques autour du Cloud, de la Data, des nouveaux standards d'Architecture, des nouveaux langages Front-End, de l'IoT, de la culture DevOps, des transformations agiles à l'échelle ou encore de la mobilité.

Merci à Aurélie Vache pour la liste 2019, consultable sur son GitHub :
https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

Olivier Denoo
Président du Comité Français des Tests Logiciels

Le test logiciel doit se réinventer !

CFTL
Comité Français des Tests Logiciels

Notre monde change. Il devient agile et s'affranchit du passé, la tête de plus en plus dans les nuages, comme si le ciel n'était même plus une limite. Il prend peu à peu la mesure des nouveaux phénomènes de communication et s'improvise social, collaboratif et participatif. Il est impatient et s'alimente de l'immédiateté, conséquence directe d'une société des écrans. Ambigu et versatile, il peine à distinguer le vrai du faux. Il oublie vite, ne pardonne rien, et réécrit sans cesse l'histoire à la lumière du présent. Il veut tout, et tout de suite, et refuse de payer l'addition si d'aventure elle se révèle salée. Parfois naïf et péchant par excès de confiance, il veut tout et son contraire : la liberté absolue mais derrière des murs ; la publicité, mais en privé ; la qualité, mais pas chère ; les choix, mais pas leurs conséquences…

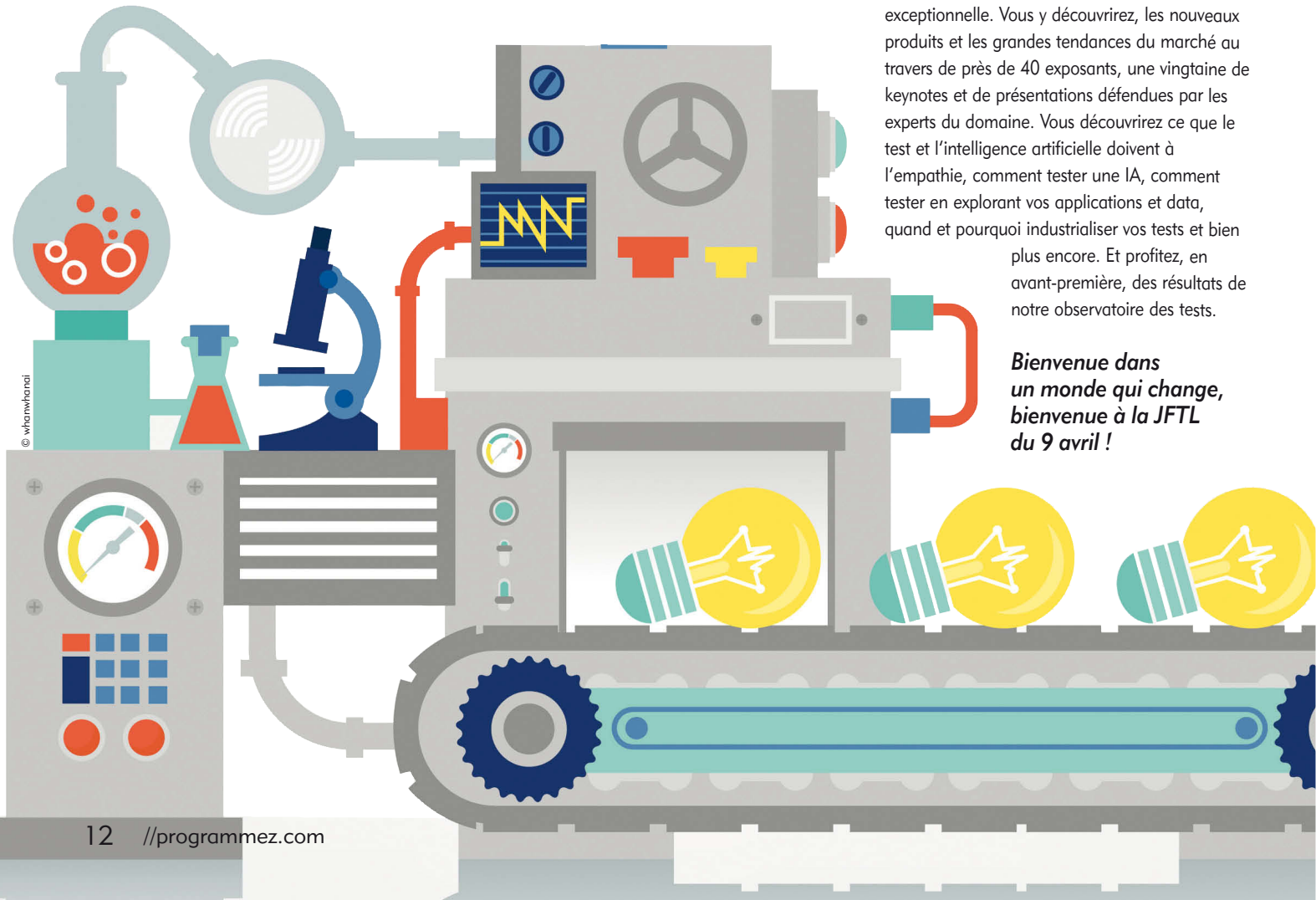
Notre monde change et l'informatique change avec lui. Toujours plus présente, elle s'invite partout, grignote chaque espace de nos vies pour gagner toutes les couches de la population, toutes les tranches d'âge ; et alors qu'elle peine encore à intégrer l'indispensable expérience utilisateur (Ux), qu'elle fait encore trop souvent l'impasse sur les « soft skills » et l'Humain ; elle lorgne déjà sur une intelligence de plus en plus artificielle prétendue complémenter notre bêtise humaine, ou au pire la corriger.

Face à ces mutations, le monde du test n'est pas non plus épargné. Alors que son positionnement traditionnel, durement acquis, s'étiole peu à peu au profit de nouvelles pratiques, aux barrières plus souples et aux rôles plus flous ; il doit à la fois se réinventer et faire face aux nombreuses innovations technologiques et à leurs contraintes. Une véritable révolution pour les testeurs, fraîchement entrés dans les métiers des TIC !

Non seulement ils ne doivent plus tester la même chose, mais encore doivent-ils tester autrement, développer de nouvelles techniques, de nouvelles compétences.

Ne manquez pas la fusée du progrès. Embarquez avec nous pour un voyage au cœur de la nouvelle qualité informatique sur cette journée exceptionnelle. Vous y découvrirez, les nouveaux produits et les grandes tendances du marché au travers de près de 40 exposants, une vingtaine de keynotes et de présentations défendues par les experts du domaine. Vous découvrirez ce que le test et l'intelligence artificielle doivent à l'empathie, comment tester une IA, comment tester en explorant vos applications et data, quand et pourquoi industrialiser vos tests et bien plus encore. Et profitez, en avant-première, des résultats de notre observatoire des tests.

Bienvenue dans un monde qui change, bienvenue à la JFTL du 9 avril !



© whanwhanai

Marc Hage Chahine
Key Member centre d'expertise de test d'Altran (ITQ). Créateur et animateur du blog La taverne du testeur

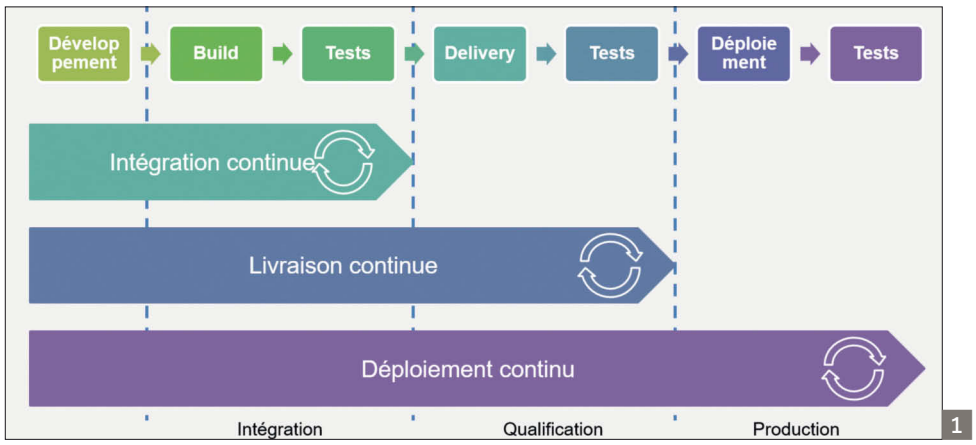
Collaboration entre testeur et développeur dans une équipe agile

Les équipes agiles – et plus généralement les équipes pluridisciplinaires – ont comme atout principal de regrouper un grand nombre de compétences en leur sein. Les tests étant une spécialité du logiciel à part entière, il est fortement recommandé d'avoir au moins 1 testeur de métier dans chacune de ses équipes.

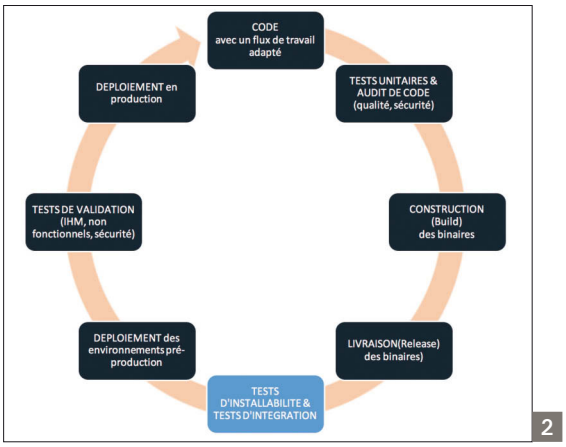
Le testeur agile, membre à part entière de l'équipe, est là pour plusieurs raisons, la principale est d'apporter une vision qualité à l'ensemble de l'équipe en mettant en place des processus, une stratégie et une prise de conscience de l'équipe sur le fait d'atteindre la juste qualité, c'est-à-dire une qualité d'un niveau satisfaisant pour les utilisateurs sans faire de sur-qualité.

Pour être vraiment efficace le testeur agile ne peut agir seul, la qualité est l'affaire de toute l'équipe ! Le testeur agile collabore avec l'ensemble des membres de l'équipe de développement sur l'ensemble des points liés à la qualité. Il doit alors convaincre de l'importance et l'apport de valeurs de ces propositions. Cela commence avec un travail fait conjointement avec l'équipe pour savoir comment on valide les nouvelles fonctionnalités (par exemple au moyen de Definition of Done et de plans de tests ou de revues). Il faut aussi savoir comment on valide que les nouvelles fonctionnalités n'ont pas engendré de régressions trop importantes (quels tests automatiser et comment ?). Connaître le niveau de qualité pour les différents livrables est également important tout comme l'optimisation du coût de l'ensemble de ces tests avec l'ajout de processus pour, par exemple, tester tôt (ex : revues). Tout cela n'est possible qu'avec une bonne connaissance du produit, du contexte, des risques engendrés avec les nouveaux changements. Afin d'avoir accès de la manière la plus efficiente à l'ensemble de ces informations, le testeur doit échanger, discuter et surtout travailler de concert avec les développeurs (qui connaissent les risques techniques) et les analystes métiers (qui connaissent les risques fonctionnels). Les approches BDD (Hiptest, Cucumber…) ou ATDD visuel (Yest, CA ARD, MaTeLo) ont d'ailleurs pour but (en plus de faire des tests une documentation vivante) de faciliter les échanges entre les personnes de ces différents métiers en proposant un langage compréhensible par tous et en limitant les barrières d'incompréhensions. Le BDD et l'ATDD permettent aussi de synthétiser le besoin en amont et donc d'éviter de développer des fonctionnalités non désirées.

Enfin, il est bon de rappeler que le testeur agile doit également collaborer avec l'équipe en faisant autre chose que du test. Il peut collaborer en travaillant sur des spécifications ou des tâches de développement à travers des revues ou même de l'écriture. Un testeur agile est avant tout un membre de l'équipe agile et il doit faire ce pour quoi il est le plus utile à l'équipe à l'instant t. Cela se traduit dans la majorité des cas par du test mais pas lorsqu'il y a une baisse de charge inhérente sur ce point.



1



2

Collaboration entre testeur et développeur sur une chaîne d'intégration continue

Dans une chaîne d'intégration le travail de collaboration entre les testeurs et les développeurs (et idéalement les opérationnels) est encore plus important. Tout d'abord afin de définir quels seront les tests ajoutés à la chaîne d'intégration continue tout en s'assurant que ces tests ne rendent pas les merge trop longs. Les différents niveaux d'intégration continue sont l'intégration continue, la livraison continue et le déploiement continu et peuvent être résumés avec ce schéma : 1

A chaque niveau d'intégration continue certains tests sont nécessaires (plus d'informations à ce sujet dans l'article dédié du livre édité par le CFTL : Les tests en Agile).

Voici un processus « standard » de déploiement : 2

Intégration continue

Avoir de l'intégration continue revient à impacter toute l'équipe de développement.

Il faut donc, au minimum, automatiser l'exécution des tests unitaires ainsi que les tests de qualité de code. Ici le testeur n'a pas beaucoup à apporter à part rappeler cette nécessité et potentiellement suggérer l'ajout de tests « vitaux ». Un autre apport potentiel important du testeur à ce niveau-là est la mise en place des rapports automatiques suite à l'exécution des tests. Pour faire de l'intégration continue il est nécessaire d'avoir des outils de développement adaptés. On peut penser à des outils comme Git (gestion de versions), GitlabCI ou Jenkins pour l'ordonnancement d'exécution des scripts. C'est au niveau de l''ordonnancement que se décide l'ajout des tests avec par exemple l'exécution des tests unitaires ou le déclenchement d'outils comme Sonarqube. De même, la phase de Construction (Build) fait également partie de l'intégration continue et doit être automatisée. On peut alors avoir différents binaires compilés et conditionnés en paquet debian, .jar, image docker ou n'importe quel autre format.

Livraison continue

Avoir une chaîne d'intégration continue allant jusqu'à la livraison continue, c'est aller jusqu'à proposer le packaging automatique. Le binaire créé précédemment est envoyé dans un dépôt comme Artifactiry ou Nexus afin qu'il devienne installable. La livraison impacte également les opérationnels (s'ils ne font pas partie de l'équipe). De plus l'avancement étant plus important, les tests à exécuter sont plus nombreux. Dans le cas de la livraison continue, il faut également implémenter des tests d'installabilité (il n'y a pas d'intérêt d'avoir un .exe, .jar ou .apk… s'il n'est pas installable) et des tests d'intégration (les tests unitaires, isolés, n'étant plus suffisants). Ici le testeur est là pour rappeler l'intérêt d'avoir ces tests et de les implémenter et bien sûr d'aider l'équipe à les implémenter. Dans cette étape, le testeur est beaucoup plus important que pour une intégration continue classique car, avec la livraison continue, arrivent de nombreuses problématiques comme le temps d'exécution des tests (expliqué par leurs nombres) ou encore la nécessité d'avoir des environnements de test ainsi que des données de test adaptées (les tests d'installabilité demandent un environnement de test). C'est généralement à partir de ce moment-là que l'on commence à parler de technologies comme Docker et de parallélisation des tests.

Déploiement continu

Enfin, il y a le déploiement continu qui va jusqu'au déploiement en production. Il y a donc des impacts clients. Il est alors nécessaire d'avoir des tests fonctionnels avec notamment des tests IHM mais aussi des tests non-fonctionnels comme les tests de performances, les tests de sécurité, d'adaptabilité… Pour les tests IHM il existe de nombreux outils.

L'outil le plus en vogue pour les tests web étant Selenium. Il existe néanmoins d'autres solutions comme des solutions d'éditeurs permettant de faire des tests sur clients lourds (UFT de HP, TestComplete de Smartbear…), mais aussi des solutions de KDT (Keyword Driven Testing) permettant de simplifier l'automatisation pour les testeurs fonctionnels. On peut citer par exemple RobotFramework HP BPT ou encore Agilitest comme solution de KFT.

Pour les tests de performances, il existe de nombreux outils matures comme Jmeter ou Neolaod, néanmoins il existe également d'autres outils sur le marché comme WebPageTest ou encore Greenspector qui s'est spécialisé dans les tests de performances (temps de réponse mais aussi consommation de batterie) pour mobiles.

Les tests de sécurité ont comme principal représentant Fortify qui, à l'image d'un Sonar, va inspecter le code pour détecter des failles de sécurité. Je recommande d'ailleurs de l'ajouter dès la phase d'intégration continue.

Enfin, il n'y a pas d'outils spécifiques pour des tests d'adaptabilité. Je ne peux cependant que conseiller d'avoir des tests utilisant des variables (par exemple le navigateur ou la version de l'OS), ce qui permet de limiter le nombre de tests à maintenir, mais aussi utiliser des outils comme Docker afin d'avoir des environnements de test adaptés.

C'est dans cette étape que le testeur prend le plus d'importance. Tout comme pour la livraison continue, il y a des problématiques de temps d'exécution des tests qui sont dans ce cas encore plus importants. Il y a la nécessité d'avoir des environnements et données de tests fiables ainsi que des bilans facilement lisibles et compréhensibles. Pour répondre à toutes ces problématiques, le testeur apporte son savoir au niveau des outils de test, sur la parallélisation des tests, sur la sélection des tests à exécuter (réussir à prendre le minimum de test tout en assurant la qualité souhaitée). A ce moment, le testeur ne peut plus faire l'économie de l'étude des mesures en production. Quelles sont les anomalies les plus fréquemment remontées ? Quelles sont les faiblesses du produit ? Comment adapter les tests afin de les rendre plus efficaces tout en ne prenant pas plus de temps. Tout cela ne peut se faire qu'en étroite collaboration avec l'ensemble des personnes travaillant sur le produit.

Conclusion

Le testeur agile est amené à collaborer étroitement avec l'ensemble des membres de son équipe de par la mise en place de processus comme les revues et l'utilisation de plans de tests. De même, le testeur agile se doit également de faire autre chose que du test pour le bien de l'équipe, cette pratique lui permet d'ailleurs d'avoir une meilleure compréhension des problématiques des membres de son équipe et des faiblesses du produit.

Néanmoins, la plupart des processus peuvent être mis en place de manière « unilatérale ». De même un développeur peut écrire son code sans connaître à l'avance les tests qui seront exécutés. Dans ce cas on ne peut pas dire que le testeur et le développeur collaborent; cela se ressent malheureusement sur la qualité du produit et de l'environnement de travail, ce qui fait émerger les problématiques des méthodes de développement classiques réapparaissent. Les nouvelles approches comme le BDD et l'ATDD sont de très bons outils pour limiter ces écueils.

Cette nuance est inexistante avec une chaîne d'intégration continue pour la simple et bonne raison qu'un développeur seul ne peut pas mettre en place une chaîne efficace sans l'expertise d'un testeur, et que l'inverse est vrai également. Si l'on veut faire de la livraison, ou même du déploiement continu, il est nécessaire d'avoir les connaissances des exploitants, des développeurs, des analystes métier et des testeurs, car ces chaînes nécessitent de fortes compétences techniques, la connaissance de nombreux outils mais aussi énormément de savoir sur les tests et leur sélection. •

Bernard Homès
Président TESSCO sas
Fondateur et ex-président du Comité Français des Tests Logiciels (bhomes@tesscogroup.com)
Avec plus de 35 ans d'expérience en Tests de logiciels dans des postes clefs à dimension internationale , Bernard exerce en qualité de Consultant Senior au sein d'entreprises renommées dans la banque, l'aéronautique, le spatial et les télécommunications.

Types de tests : utilité et impacts

Il existe de nombreux types de tests et il est tout à fait normal que vous soyez perdu dans l'explosion de termes plus ou moins complexes. Je vous propose de synthétiser les divers types de tests, chacun avec une utilité et un impact différent.

Le « test » devrait être vu comme de la vérification et de la validation (« V&V »). Vérification et Validation impliquent tous les deux de fournir des preuves que des exigences ont été respectées. Cela implique d'avoir des exigences mesurables et atteignables. On parle de IV&V quand les actions de V&V sont effectuées par une équipe indépendante. La validation se focalise sur un « usage voulu », ce qui n'est pas le cas pour la vérification. Les tests peuvent être dynamiques (càd. nécessiter l'exécution du code de l'application à tester) ou statiques (sans exécuter le code à tester).

Comme toute autre activité, les tests doivent être efficaces et efficients. Efficaces en termes de détection de défauts, efficients en limitant l'effort nécessaire pour atteindre ces objectifs.

Le cycle de développement importe peu sur les tests : tous les types de tests mentionnés ci-dessous peuvent être effectués quel que soit le cycle de développement. Dans un cycle (itération, sprint, …), on commence par faire des tests statiques et des revues, puis on effectuera du test unitaire, ensuite du test d'intégration, puis du test système, avant de faire du test d'acceptation. Chaque sprint après le premier devra s'assurer que les évolutions n'amènent pas de régressions dans ce qui a été développé précédemment, ceci impliquera du test de (non)régression et parfois du test automatisé. Lors de la mise en place de solutions de livraison continue (p.ex. DevOps), il faudra s'assurer, par des tests d'intégration de systèmes et des tests de systèmes-de-systèmes, que les fonctionnalités de bout-en-bout ne sont pas impactées. Des tests de performances, de sécurité et d'utilisabilité seront aussi nécessaires.

Tests statiques et revues sont des activités de vérification de composants ne nécessitant pas l'exécution de ce composant. Ces activités peuvent utiliser des outils (on parlera d'analyse statique) ou des êtres humains (on parlera de revues). Le formalisme et le niveau de détail peut varier, entraînant une variation de l'efficacité de ces activités.

- Utilité : l'efficacité des revues a été démontrée depuis les années 70 et varie de 50 jusqu'à 85% de détection des défauts. Ces activités peuvent être effectuées avant même que le logiciel ne soit exécutable, donc très en amont de la livraison, ce qui laisse plus de temps pour les corrections.
- Impact : la détection précoce de défauts, y compris de défauts de conception, permet un gain de temps et d'efficacité très important.

Tests de composants (ou tests unitaires) : ces activités sont principalement exécutées par les équipes de développement pour vérifier si les composants unitaires sont corrects sur le plan fonctionnel et non-fonctionnel. Les tests unitaires sont les premiers tests que subissent les composants développés. Les tests de type TDD, BDD et ATDD ne répondant pas à la définition du test, ils devraient être considérés comme des exigences automatisées, et, en tant que tels, sont très utiles. Dans le test de composants, le niveau de granularité des composants unitaires peut varier de tout petit (fonction, méthode ou classe) à très gros (application ou système).

- Utilité : des tests de composants bien conçus permettent de trouver de nombreuses anomalies qui peuvent être corrigées immédiatement. Les diverses techniques de conception de test étant rarement enseignées aux développeurs, le niveau d'efficacité des tests unitaires est souvent faible. Les tests de composants sont très importants en ce qu'ils permettent d'avoir confiance dans la qualité des composants et donc de permettre une meilleure focalisation des autres types de tests.
- Impact : les tests de composants sont souvent effectués par les développeurs dès la fin du codage et permettent de trouver – et de corriger – rapidement les défauts. Quand ils ne sont pas (ou pas bien) exécutés, les défauts ne pourront être découverts que dans des phases ultérieures de test, ce qui impactera négativement l'efficience. Il est important d'avoir un environnement de test de composant séparé de l'environnement de développement.

Tests d'intégration : ces tests ont pour objet de vérifier l'intégration de composants les uns avec les autres. On se focalisera donc sur la recherche de défauts dans les flux et messages échangés entre des composants testés unitairement. Ici aussi on mesure à la fois les aspects fonctionnels et non fonctionnels. Divers modèles d'intégration existent (« big bang », de bas en haut, de haut en bas, par transaction ou par fonctionnalité) chacun avec ses avantages et ses inconvénients propres. Les tests d'intégration nécessitent à la fois des connaissances techniques (développement et réseau) et des connaissances fonctionnelles (métier). Quand la granularité des composants est très importante (p.ex. des systèmes au sein d'un système d'information ou d'un système-de-systèmes) on parlera de « Tests d'intégration système ». Ces tests d'intégration système sont principalement fonctionnels et vérifient la bonne connexion des systèmes les uns aux autres ainsi que la qualité des flux (mapping, performances, etc.).

- Utilité : s'assurer que des composants sont en mesure d'échanger correctement des informations permet de limiter les soucis lors des tests systèmes et des tests d'acceptation. Cependant ce type de tests est souvent oublié ou bâclé. Il en résulte des soucis lors de l'intégration des composants, surtout quand un système logiciel est introduit dans un système d'information ou un système-de-systèmes.
- Impact : si les tests d'intégration sont insuf-

fisants, les tests systèmes deviennent une séquence d'exécutions suivies d'arrêts et de reprise après correction. Pour mettre en œuvre des tests d'intégration correctement, tous les composants (ou systèmes) intégrés doivent être présents et préalablement testés. Ceci implique un investissement significatif en termes d'environnements : il en faudra un par type de test.

Tests système : ces tests – aussi appelés tests de bout en bout – ont pour objectif de trouver des défauts d'aspects fonctionnels et non fonctionnels dans le système développé, c'est-à-dire quand tous les composants sont intégrés. Si le système est composé d'autres systèmes, on parlera de tests de systèmes-de-systèmes. Vu la complexité croissante des systèmes et l'explosion combinatoire des tests, il n'est pas matériellement possible de garantir une couverture complète par des tests systèmes. Un environnement de test système spécifique, différent des environnements de test d'intégration, de test de composant et de développement est nécessaire, ainsi qu'un processus maitrisé de gestion des versions et des anomalies.

- Utilité : permet de vérifier le fonctionnement de bout-en-bout du système dans un environnement similaire à celui de production. C'est à ce niveau que seront identifiées toutes les interactions des composants entre eux. Le nombre de combinaisons de chemins possibles et de types de données rend une couverture complète des tests système difficile à atteindre. Les tests système s'effectuant au bout de la chaîne de développement, le temps disponible pour corriger les anomalies est restreint.
- Impact : les tests systèmes sont les derniers tests effectués avant la fourniture du logiciel pour acceptation par le client. Il restera donc peu de temps pour corriger les anomalies trouvées et pour s'assurer que les corrections n'introduisent pas de régressions (voir plus loin tests de (non)régression).

Tests d'acceptation : ces activités servent à démontrer aux utilisateurs (aux clients) que l'application développée peut être acceptée en production. Ce ne sont pas à proprement parler des tests car ils n'ont pas pour objet de trouver des défauts. Divers types de tests d'acceptation existent (beta tests, phases pilotes, etc.).

- Utilité : ces activités permettent de vérifier les exigences contractuelles et d'acquérir des preuves de leur bon fonctionnement, et montrent aux utilisateurs que l'application peut être acceptée en production. Tout défaut identifié à ce stade aura un impact négatif sur la satisfaction des clients.
- Impact : ces tests sont souvent effectués par des utilisateurs finaux qui se focalisent sur leur utilisation dans le contexte de leur métier. Les tests d'acceptation sont rarement exhaustifs (ils ne couvrent pas tous les chemins), formels (ils ne démontrent pas une couverture de toutes les exigences contractuelles) et l'identification de cas complexes ou d'exception est peu fréquent.

Tests de (non) régression : toute modification peut avoir un impact (un effet de bord) sur d'autres parties de l'application ou du système. Il faut s'assurer, à chaque changement et pour chaque niveau de test, qu'aucun effet de bord indésirable (aucune régression) n'est introduit. Selon le niveau de risque, cela pourrait nécessiter la réexécution de tous les tests existants sur l'application. Vu la fréquence des changements (évolutions et corrections), seul un sous-ensemble des tests est réexécuté. Les tests de (non)régression se prêtent bien à l'utilisation d'outils d'automatisation des tests, pour rejouer rapidement un grand nombre de tests dans un temps réduit.

- Utilité : le nombre de défauts trouvés par des tests de (non)régression est généralement très faible, mais l'effort nécessaire pour exécuter ces tests est relativement élevé. Ces exécutions limitent les risques liés aux modifications avant la mise en production de versions corrigées.
- Impact : vu l'investissement nécessaire pour assurer une absence de régressions et les délais très courts avant l'implémentation (la mise en production) des modifications ou évolutions, les tests de (non)régression sont souvent assez limités. Il en résulte une augmentation faible mais constante des défauts dans les applications au cours de leurs existences. Dans les cycles de développement Agiles et DevOps, des tests de (non)régression devraient s'assurer que les évolutions ou changements n'ont pas introduits d'effet de bord sur les composants développés dans les itérations (sprint, cycle, etc.) précédentes. Il sera donc intéressant d'automatiser ces tests afin de les exécuter régulièrement.

Tests automatisés : toute tâche de test peut – plus ou moins facilement – bénéficier de l'utilisation d'outils. Nous avons

Quelques outils de tests selon DZone

Product	Selenium	Katalon Studio	Unified Functional Testing	TestComplete	watir
Available since	2004	2015	1998	1999	2008
Application Under Test	Web apps	Web (UI & API), Mobile apps	Web (UI &API), Mobile, Desktop, Packaged apps	Web (UI &API), Mobile, Desktop apps	Web apps
Pricing	Free	Free	$$$$	$$	Free
Supported Platforms	Windows Linux OS X	Windows Linux OS X	Windows	Windows	Windows Linux OS X
Scripting languages	Java, C#, Perl, Python, JavaScript, Ruby, PHP	Java/Groovy	VBScript	JavaScript, Python, VBScript, JScript, Delphi, C++ and C#	Ruby
Programming skills	Advanced skills needed to integrate various tools	Not required. Recommended for advanced test scripts	Not required. Recommended for advanced test scripts	Not required. Recommended for advanced test scripts	Advanced skills needed to integrate various tools
Ease of Installation and Use	Require advanced skills to install and use	Easy to setup and use	Complex in installation. Need training to properly use the tool	Easy to setup. Need training to properly use the tool	Advanced skills needed to integrate various tools

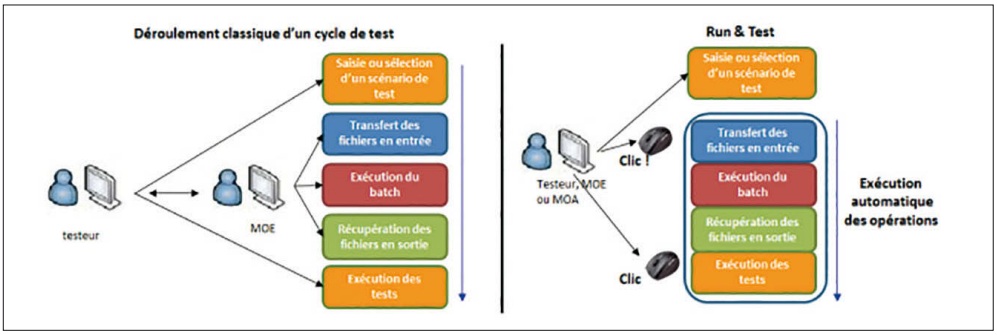
principalement les tâches de gestion (suivi, reporting, gestion des anomalies, etc.) et les tâches d'exécution des scénarios et cas de test. De très nombreux outils existent, chacun se focalisant sur un – ou plusieurs – aspects des activités de test. Nous considérons principalement les activités d'enregistrement et d'exécution des scripts de test, ainsi que les activités de reporting du test.

- Utilité des outils d'exécution des tests : l'exécution périodique de tests de (non)régression justifie d'utiliser des outils qui peuvent réexécuter des scénarios de test, simuler les actions d'utilisateurs et s'assurer que les résultats obtenus correspondent aux résultats attendus. L'intérêt de tels outils est de limiter le temps d'exécution, voire de permettre d'exécuter un grand nombre de tests avec un nombre réduit de testeurs. La mise en œuvre d'un outil d'exécution des tests doit être considérée comme un projet de développement à part entière et ne pas négliger les aspects de reporting de ces exécutions.
- Utilité des outils de reporting des tests : l'exécution d'un test n'apporte de valeur au projet que quand elle permet de prendre des décisions sur le projet. Il est donc primordial pour le chef de projet de test de fournir des preuves que les exigences contractuelles spécifiées ont été couvertes par des cas de test passés avec succès. Donc la traçabilité depuis les exigences jusqu'à l'exécution des tests sur une version connue des composants logiciels est primordiale.
- Impact des outils d'exécution des tests : si on se limite à des activités de capture et de rejeu de scripts de test, ces outils d'automatisation n'ont qu'une utilité limitée : l'investissement nécessaire pour la conception, la mise au point et la maintenance des scripts de test est très élevé et le nombre de défauts identifiés limité. Par contre, utiliser ces outils dans le cadre de tests dirigés par les données (DDT) ou de test dirigés par des mots clé (KDT) permet de réutiliser les mêmes scripts de façon très efficace et efficiente en donnant un certain niveau d'intelligence aux scripts de test.
- Impact des outils de reporting des tests : des statistiques d'avancement, des courbes de tendances permettent au chef de projet de test de suivre son projet et d'anticiper les actions à mettre en place pour limiter les risques. Beaucoup d'outils ne fournissent pas ce type d'informations, mais permettent d'exporter des données qui seront plus facilement analysées et mise en page par des tableurs.

Tests de sécurité, d'utilisabilité, de performances, etc. : ces tests peuvent exister à chaque type de test (chaque niveau de granularité de test) et se focalisent sur des caractéristiques – non fonctionnelles – décrites dans des standards comme ISO25010.

- Utilité : toutes les caractéristiques qualité ont un intérêt plus ou moins important à court, moyen ou long terme pour les parties prenantes. Le niveau d'importance et donc l'investissement que l'on devrait consentir variera selon l'environnement.
- Impact : comme pour les tests fonctionnels, ces activités devraient être effectuées le plus tôt possible dans le cycle de développement.

Une analyse comparative de l'utilité, de l'impact, de l'efficacité et de l'efficience de chacun des types de test mériterait un chapitre complet plutôt qu'un simple article. Si nous voulons simplifier, nous pourrions comparer les types de tests à des filtres aux mailles de plus en plus fines. Toute suppression d'un filtre (toute suppression d'un type de test) augmentera le nombre de défauts à identifier par les filtres suivants, sinon ils seront identifiés par les utilisateurs finaux. On voit donc que plusieurs types de tests, plusieurs niveaux de test sont nécessaires pour assurer la qualité des logiciels livrés. •



Principe de l'automatisation des tests, d'après eccellium.fr

Exemples d'outils

Des centaines d'outils de test sont disponibles sur le marché, que ce soient des outils commerciaux ou des outils open source.

Le choix d'un ou de plusieurs outils dépendra de votre contexte (organisationnel, technique, humain), des délais disponibles et des objectifs à atteindre. Quel que soit l'outil sélectionné, vous devrez prévoir une stratégie de mise en œuvre à long terme ainsi qu'une mesure du retour sur investissement.

Le coût initial d'un outil n'est qu'un des paramètres à prendre en compte. Vous trouverez ci-contre une liste non exhaustive d'outils pouvant être utilisés pour chacun des types de test.

Tests statiques et revues :
Parasoft, Polyspace, Coverity, Cast, CppCheck, CppDemand, …

Tests de composants :
SilkTest, Selenium, Katalon, QTP, SOAPSonar, W3C CSS Validator, JUnit, CppUnit, VisualStudio Test Professional, …

Tests d'intégration :
Panaya, VectorCAST/C++, LDRA, FitNesse, Rational Integration Tester, …

Tests système :
Squash TA, Panaya, SilkTest, Tricentis Tosca, TestLink, QTP, Rational Robot, …

Tests d'acceptation :
ALM-QC, Tricentis Tosca, Ranorex, Watir, SoapUI, …

Reporting et administration des tests :
ALM, MS-Test Manager, Squash TM, TestComplete, Mantis, Bugzilla, RedMine, …

Tests de (non) régression :
Selenium, Katalon, Panaya, QTP, SilkTest, eValid, …

Tests de sécurité :
OWASP Code Crawler, …

Tests de performances :
Load Runner, Neoload, SilkPerformer, WebLoad, JMeter, …

Polarion ALM : un chef d'orchestre pour gérer les exigences et les tests logiciels

La prise de conscience de l'importance des tests logiciels a émergé vers la fin des années 1960 suite à ce qui a été nommé « la crise du logiciel », qui a mis en évidence les difficultés rencontrées pour satisfaire au fameux triptyque « coût, délai, qualité » des projets. (Notons que derrière Qualité se cachait aussi la Conformité, à savoir l'adéquation aux exigences exprimées ou réglementaires). Près de 50 ans après, même si la typologie de tests n'a pas profondément changée (unitaire, intégration, système, acceptation, etc.), l'avènement des Frameworks de tests unitaires, de l'intégration continue, de nouvelles approches (BDD Behavior Driven Development), les contraintes de l'automatisation des tests au niveau des IHMs, et enfin des pratiques Agile ont permis d'inverser « la pyramide des tests », son socle n'étant plus aujourd'hui les tests dits fonctionnels, mais les tests unitaires, avec une automatisation poussée des couches basses.

Avant de rentrer dans le vif du sujet des « tests logiciels », nous définissons et exécutons des tests pour délivrer des solutions de qualité, répondant aux besoins des futurs utilisateurs et clients. Cette recherche de qualité passe en premier lieu par une gestion efficace des exigences, ne dit-on pas que parmi leurs critères d'acceptation, qu'elles doivent être vérifiables, mesurable ou encore traçable ?

Une couverture globale du cycle de conception du logiciel

Ce caractère indissociable entre les artefacts exigences et tests rend nécessaire de s'appuyer sur un chef d'orchestre et c'est exactement le rôle que joue **Polarion ALM** dans une chaîne de développement logiciel.

Polarion ALM est une solution unifiée de gestion du cycle de vie des applications et systèmes, s'appuyant sur un référentiel centralisé, prônant la collaboration entre toutes les parties prenantes (comment, un testeur peut échanger avec un développeur ou un analyste métier ?), le pilotage des processus (ou comment guider un acteur dans son workflow), avec cette flexibilité permettant de gérer de manière efficiente et cohérente tous types d'artefacts, nommés **Work Items** : Que ce soit des Work Items de type exigence, user story, cas de test, plans de test, ou défaut, les mêmes règles et principes s'appliquent pour notamment :

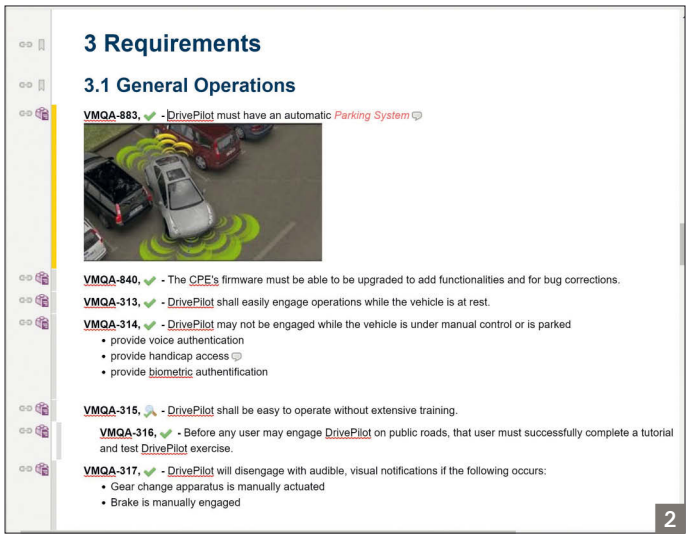
- Spécifier les exigences et artefacts de tests dans des documents LiveDoc collaboratifs, versionnés, historisés, facilitant les revues et pouvant être signés électroniquement,
- Analyser l'impact des changements et le taux de couverture par la traçabilité entre les exigences et les tests,
- Délivrer des métriques sur l'ensemble des artefacts du projet.

Polarion ALM est une solution reconnue et largement déployée pour spécifier, documenter et approuver les exigences dans les secteurs du développement logiciel, des systèmes embarqués, ou dans des domaines fortement réglementés comme les dispositifs médicaux ou l'avionique. 1 2

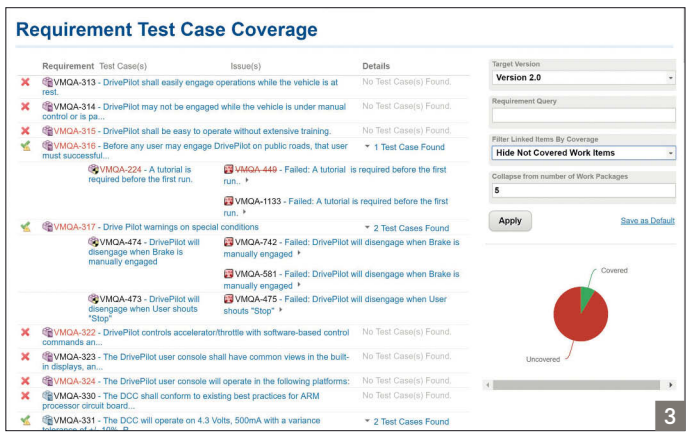
L'utilisation de Polarion ALM pour aligner les tests et les mesurer en regard des exigences, couvrir les activités propres aux tests logiciels comme définir la stratégie de test, organiser et définir les artefacts de tests, piloter leur exécution, consolider les informations et apporter de la visibilité sur l'ensemble – avancement, résultats, conformité –, permet de faciliter la continuité entre les différentes phases des projets : Il n'y a pas de rupture, les utilisateurs communiquent autour d'un langage commun, une même interface utilisateur, avec une navigation aisée entre les différents Work Items par la traçabilité, une visibilité « sur mesure » en fonction des profils (ex : un analyste métier pourrait lire les plans de tests, sans pouvoir les modifier, et inversement, un Testeur peut consulter voire commenter les documents de spécification, ce qui est fort utile pour son métier, mais sans pouvoir les mettre à jour). 3

Comment les choses se déroulent-

1



2



3

elles quand les tests sont spécifiés dans un environnement différent que celui des exigences ? Nous voyons alors les exigences être copiées dans l'environnement de test. Ces exigences dupliquées sont alors associées aux cas de tests qui seront alors exécutés avec des résultats stockés dans l'environnement de test. Alors certes des intégrations sont possibles et existent entre Polarion ALM et d'autres gestionnaires de tests, mais par exemple, comment et où se mesure la couverture des exigences initiales par les tests, que ce soit en termes de définition ou d'exécution, dans l'outil de gestion des exigences ou l'outil de tests ? Comment communiquer et gérer aisément le workflow d'une user story qui passe en état final « *verified done* » une fois les tests validés par les acteurs QA ? Des difficultés parmi d'autres qui sont facilement levées quand les barrières entre activités du génie logiciel sont ouvertes.

Dans la pratique, le testeur va pouvoir rédiger sa spécification de tests dans un document LiveDoc, valoriser les propriétés du test (sévérité, type de test, etc.), relier ces tests aux autres Work Items (exigences, tâches, fonctions, etc.) par lien de traçabilité adéquat. 4

Cette approche pour rédiger les plans et cas de tests est novatrice en permettant donc de s'appuyer sur ce format documentaire LiveDoc, alors que généralement, les tests sont décrits uniquement dans une structure tabulaire hiérarchisée. Avec Polarion ALM, les utilisateurs bénéficient des deux vues systématiquement, il n'y a pas de double-saisie dans une représentation document, puis dans une représentation table. Un utilisateur peut basculer en un clic d'une vue à l'autre. Procédant ainsi, il est possible d'appliquer aux spécifications des tests les mêmes bonnes pratiques que pour les spécifications des exigences, à savoir élucidation, documentation, revue par les pairs, approbation ou refus de la spécification, et signature électronique si requise par la réglementation. 5

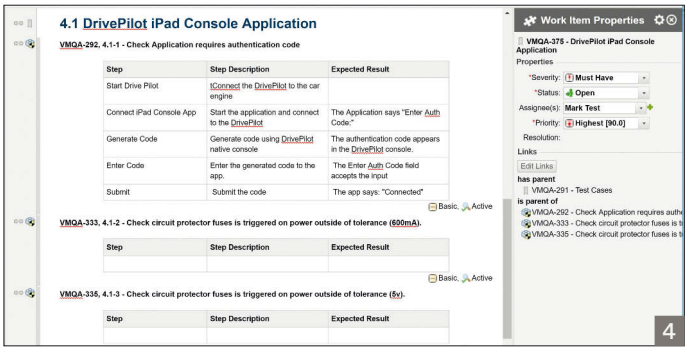
Donner de l'importance au processus de test

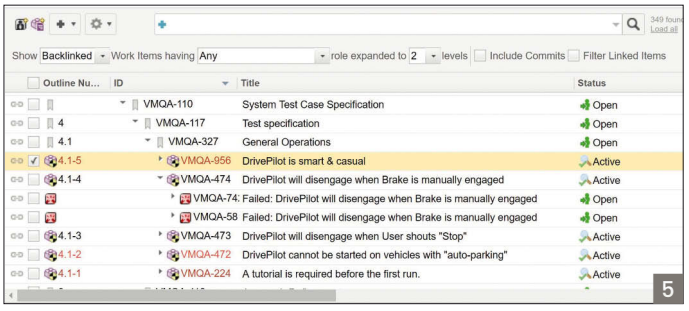
Pour les utilisateurs habitués aux gestionnaires de test « traditionnels », qui peuvent être dirigistes dans la manière de décrire les tests, d'exécuter les campagnes, et mesurer l'avancement ou la couverture, Polarion ALM peut dérouter au premier abord car il donne de la latitude aux administrateurs de la plate-forme pour mettre en place un processus de test ad-hoc, sur mesure : Il n'y a pas qu'un seul type de cas de test, pas un seul workflow, ni une seule approche pour analyser les résultats des campagnes. Polarion ALM offre la souplesse pour concevoir le modèle de données et le processus adapté à l'organisation et au cycle de vie des projets. Ce workflow sur mesure peut être couplé à la signature électronique des résultats de tests pour répondre notamment à la norme médicale 21 CFR Part 11. L'*Audit Trail* sous Polarion permettra de retracer et consulter l'ensemble de l'historique propre aux tests, aux exigences et user stories vérifiées, aux anomalies générées ou même au code source impacté. 6

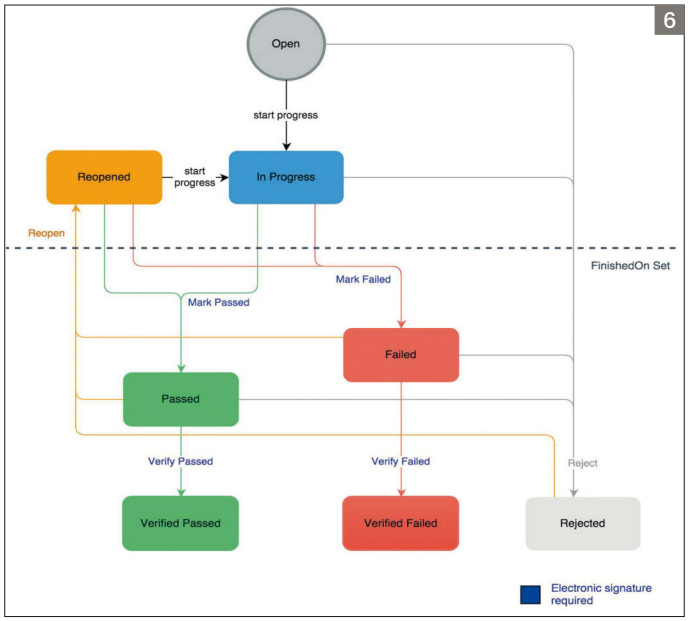
Piloter les campagnes de tests manuels

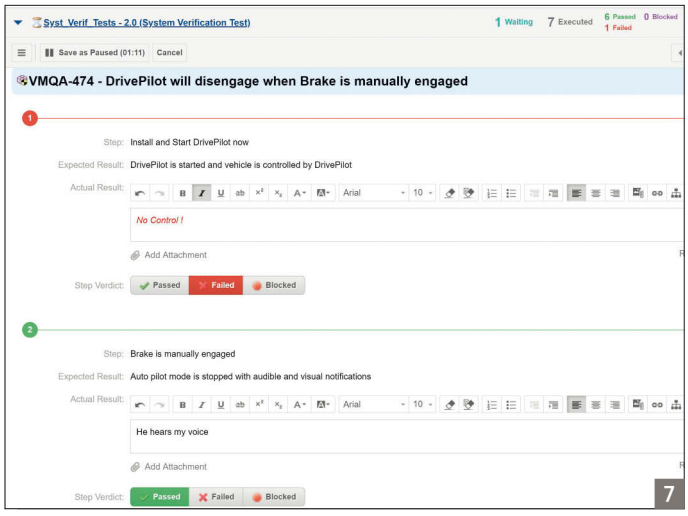
Même si une automatisation plus poussée des tests est visée pour les tests unitaires, les tests d'acceptation ou la couverture de l'*Happy Path*, l'exécution des tests de recette est encore souvent manuelle. Ici encore le focus est mis sur l'adaptabilité. Pour sélectionner les cas de tests à exécuter dans un *Test Run* pour un nouveau *build* du logiciel, l'approche la plus intéressante est de définir un modèle qui va permettre d'alimenter automatiquement ce *Test Run* à partir du contenu d'un document de spécification de tests, ou à partir d'une requête. Cette sélection automatisée peut se faire au fil de l'eau : un nouveau test est créé dans le document pendant que les campagnes sont en cours, le ou les *Test Runs* liés à ce document sont dynamiquement mis à jour. Les étapes de tests peuvent également être variabilisés afin de pouvoir couvrir les mêmes cas de tests tout en exploitant des jeux de données différents. 7

La page rapportant de la progression et des résultats des tests donne une vision d'ensemble de l'état de la campagne et est aisément personna-









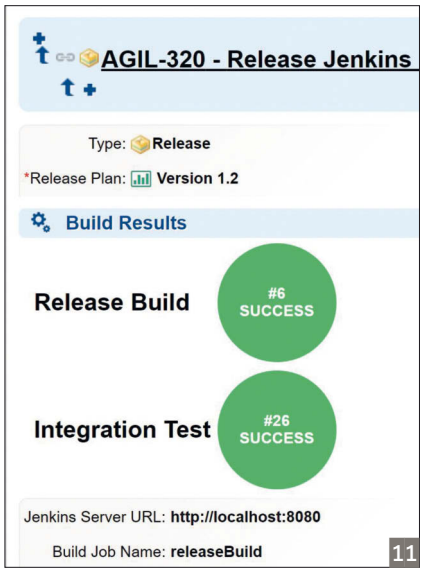
lisable. Dans notre exemple, une fiche d'anomalie a été produite automatiquement et liée au test (dans sa révision d'exécution) et au *Test Run* en échec. Le cycle d'analyse et de résolution de ce défaut constaté sera suivi au niveau de ce Work Item *Defect*, avant de pouvoir reprendre l'exécution. Pour cette gestion des anomalies, vous avez le choix de les gérer sous Polarion ALM, ou dans un autre environnement dédié (notamment Jira pour lequel un connecteur de synchronisation existe). 8

Automatiser les tests dans un pipeline de livraison continue

Quand on automatise les tests, en particulier ceux des niveaux inférieurs de la pyramide (tests unitaires et tests d'intégration), on emploie au-delà des automates eux-mêmes, différentes solutions qui vont nous permettre de mettre en place une chaîne d'intégration continue (CI) et de déploiement continu (CD), par exemple avec des outils comme Jenkins ou GitLab. Polarion ALM se veut par ailleurs agnostique dans le choix des automates retenus. En effet, du fait de la diversité et portée des tests, les entreprises utilisent en moyenne plus de trois outils d'automatisation et d'exécution de ces tests. Les intégrations natives de Polarion et son ouverture via ses Java APIs/Web Services publiés dans le *SDK (Software Development Kit)* permettent de mettre l'outil ALM au cœur de ce framework complet de *Build*, *Test*, *Deploy*, tout en gardant le lien fort avec les exigences ou user stories initiales. 9

Prenons par exemple l'utilisation du serveur d'intégration continue Jenkins : Les résultats des tests automatisés seront remontés sous Polarion sous la forme d'un Test Run, avec une présentation et une consolidation identique comme pour les tests manuels, et ainsi bénéficier d'un environnement unifié pour le suivi des tests. Ce transfert des données vers le gestionnaire de test s'effectue via un fichier résultat produit par l'automate au format xml (format xUnit), ou par une intégration sur mesure via les Web Services. 10

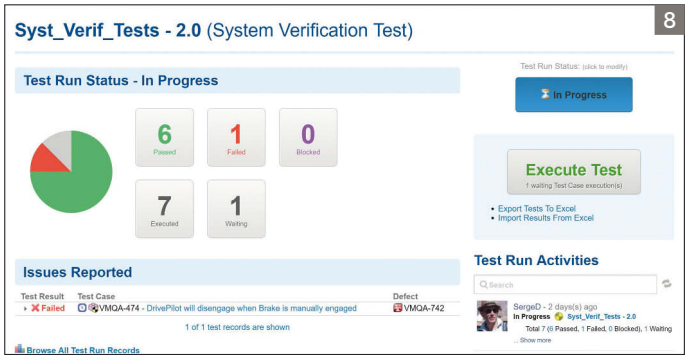
Jenkins est un chef d'orchestre, il peut se suffire à lui-même pour soumettre les jobs. Cependant pour s'affranchir de devoir basculer d'un environnement à l'autre, Polarion ALM montre à nouveau sa flexibilité et son ouverture en permettant de déclencher toute une chaîne de traitement, qui va successivement soumettre les jobs de Builds sous Jenkins, remonter leur statut et générer les Test Runs avec les résultats de tests sous Polarion. 11

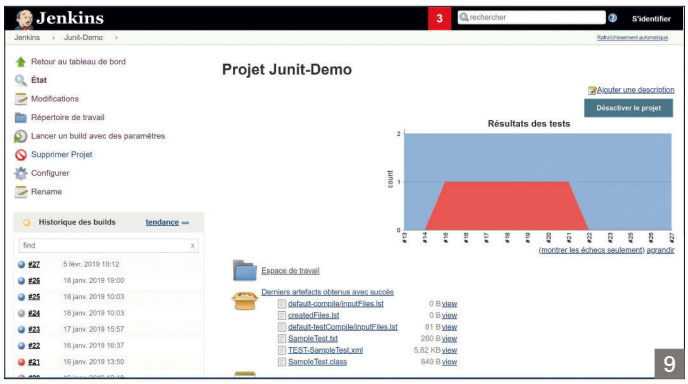


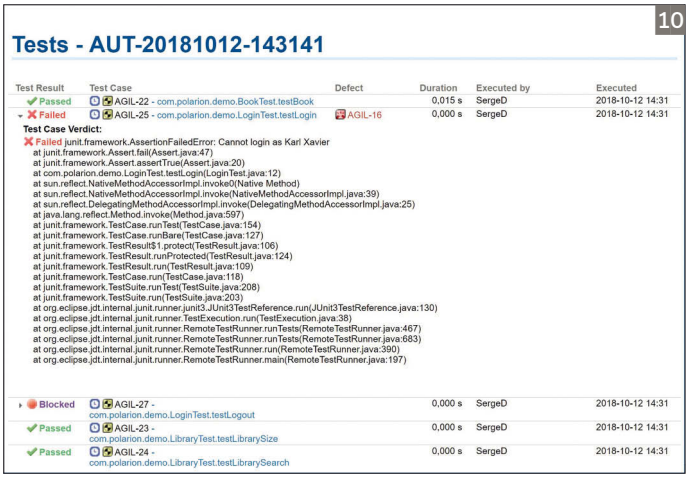
Un référentiel centralisé mais ouvert pour accroitre la qualité des logiciels

Tout comme les pratiques Agile qui tendent à rapprocher les différents acteurs impliqués dans la conception logicielle, qu'ils soient Product Owner, développeurs ou testeurs, Polarion ALM se veut être l'environnement central qui va permettre d'unifier et rendre aisément disponibles l'ensemble des données relatives aux activités de tests, et au-delà aux activités de spécifications, de tâches de développement, de gestion des anomalies, etc. Ce lien étroit entre toutes ces données va permettre de faciliter la communication et la collaboration entre les acteurs, de donner plus de visibilité pour une meilleure prédictibilité des releases, de s'affranchir des duplications d'artefacts de tests entre environnements disparates, de disposer des éléments de traçabilité permettant une analyse d'impact plus rapide et plus simple. Dans des frameworks de tests modernes, Polarion ALM ne couvre pas tout cela seul, le logiciel s'ouvre vers l'existant, s'intègre avec les logiciels tiers des pipelines CI/CD et s'adapte aux besoins et visions des architectes logiciels.

•







Olivier Bouzereau
avec l'équipe du projet STAMP

STAMP : les tests automatisés soutiennent l'équipe DevOps

Les tests automatisés s'imposent vite en développement et en intégration continus ; ils améliorent leur célérité pour déployer des services débarrassés de nombreux bugs. Illustration avec l'amplification des tests Java proposés par la suite d'outils open source STAMP.

L'automatisation des tests logiciels fait l'objet de multiples expérimentations en ce moment. Les concepteurs d'applications distribuées et de jeux en ligne sont friands de coopérations avec les universitaires. Plusieurs pistes sont explorées pour accélérer la mise à disposition de nouveaux services plus fiables, plus sûrs, plus stables : algorithmes d'IA, tests par mutation et techniques d'amplification de tests tentent de couvrir davantage de lignes de codes et de suggérer des améliorations plus précises.

Ubisoft développe ainsi avec l'Université Concordia à Montréal un renifleur de bugs capable de détecter les soumissions à risques avec une précision de 79% en moyenne. Google et Facebook s'appuient respectivement sur le Machine Learning et sur les tests par mutation pour leurs déploiements à grande échelle.

Néanmoins, avec les tests automatisés, deux écueils principaux restent à éviter : des délais d'exécution trop longs et des résultats de tests différents à chaque itération - ce dernier cas, dit flacky test, n'apporte pas d'information fiable sur le code testé.

Les contributeurs du projet STAMP (Software Testing Amplification), soutenus par la commission européenne dans le cadre du programme Horizon 2020, développent quatre outils open source de tests automatisés ciblant les applications Java. Actuellement en bêta (https://l.ow2.org/stp-beta), ces outils s'inscrivent dans l'approche DevOps, un cycle agile où les nouvelles versions de programmes peuvent être mises en production très fréquemment. La méthodologie, les outils et cas d'usage STAMP reflètent l'enjeu actuel consistant à livrer régulièrement de nouvelles versions d'applications exemptes de défauts prévisibles, en optimisant leurs tests.

Amplifier les tests en mode DevOps

Suivant l'approche DevOps, les développeurs et les exploitants informatiques communiquent et coopèrent plus étroitement. Ils rapprochent leurs objectifs et s'entendent autour de nouveaux automatismes, la qualité du logiciel devenant l'affaire de chacun.

Paradoxalement, c'est en perturbant des tests créés manuellement que l'on cherche à rendre le logiciel plus stable en production.

L'activité quotidienne du testeur de logiciel ressemble aux efforts de Sisyphe, héros malheureux de la mythologie grecque puni à rouler éternellement son rocher en haut d'une colline par Zeus et son frère Hadès, en réprimande de ses provocations. Naturellement, à l'approche du sommet, le rocher retombe, forçant Sisyphe à réitérer sa tâche. S'il pouvait renverser la colline comme on retourne un bol, son défi serait résolu : le rocher, même bousculé, serait enfin stabilisé. C'est ce que les outils STAMP proposent de faire tout au long du cycle DevOps (figure 1). L'énigme moderne revient donc à provoquer une perturbation adéquate aux tests logiciels, aux différents stades de la conception, de la transformation et de l'exécution de l'application. 1

Design, tests et intégration continue

Quels sont les prérequis pour faire tourner les outils STAMP ? Il suffit d'utiliser Java en version 8 ou plus récente et de savoir s'orienter dans un dépôt Git. Mais surtout, il faut avoir une pratique des tests unitaires déjà en place, faute de quoi, rien ne pourra être amplifié.

Les outils STAMP s'exécutent en ligne de commande ou via un plug-in Maven ou Gradle pour gérer les dépendances et les modules externes. Ils sont également intégrés aux environnements Eclipse et IntelliJ,

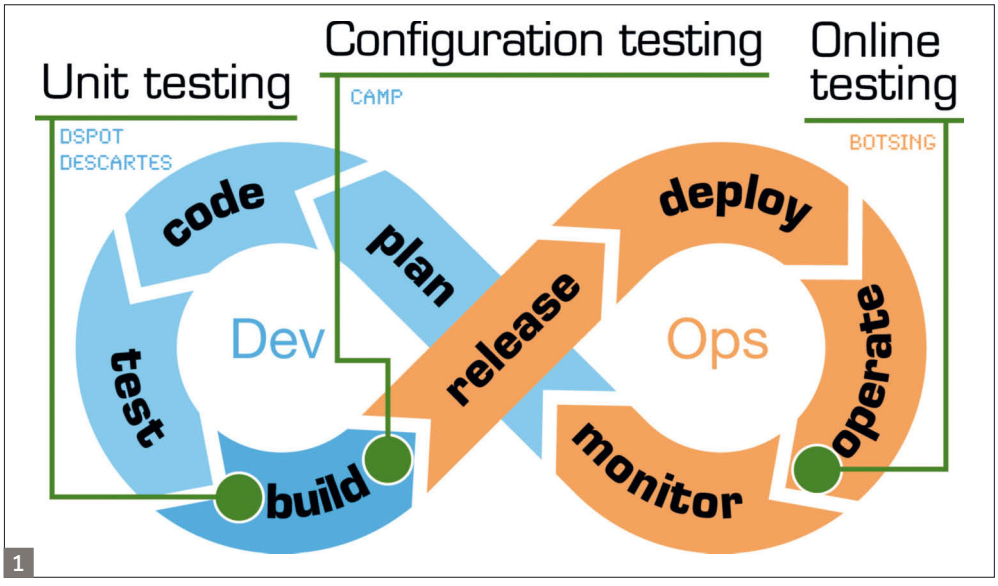


Figure 1 : Le cycle DevOps et les outils automatisés STAMP

```
106     @Override
107     public boolean equals(Object object)
108     {
109         boolean result;
110
111         // See http://www.technofundo.com/tech/java/equalhash.html for the detail of this algorithm.
112 2   1. equals : All method body replaced by: return true → SURVIVED
113     2. equals : All method body replaced by: return false → KILLED
114         } else {
115             if ((object == null) || (object.getClass() != this.getClass())) {
116                 result = false;
117             } else {
118                 MacroId macroId = (MacroId) object;
119                 result =
120                     (getId() == macroId.getId() || (getId() != null && getId().equals(macroId.getId())))
121                         && (getSyntax() == macroId.getSyntax() || (getSyntax() != null && getSyntax().equals(
122                             macroId.getSyntax())));
123             }
124         }
125         return result;
126     }
127 }
```

2

Figure 2 : L'outil open source Descartes détecte une faiblesse dans le test qui couvre la méthode equals()

deux IDE bien connus des développeurs.

Les tests automatisés s'immiscent désormais dans un pipeline d'intégration et de déploiement continus. Les programmes DSpot et Descartes de la suite STAMP agissent dès les tests unitaires, lors de la conception du programme. DSpot génère automatiquement des variations autour de tests unitaires pré-existants. L'objectif de Dspot est d'améliorer la suite de test pour limiter le risque de régression. Le fonctionnement de Descartes est détaillé dans le paragraphe suivant.

Les outils Camp et Botsing interviennent ensuite, respectivement au niveau des configurations puis de l'exécution. Camp amplifie les tests de configuration en environnement Docker et Botsing génère des nouveaux tests qui permettent de reproduire des crashs produits à l'exécution. Camp vise à vérifier les bugs susceptibles de se produire dans plusieurs combinaisons d'environnements, côté serveur et côté navigateur, en déployant et testant plusieurs configurations et versions de logiciels successivement. En effet, certains dysfonctionnements n'apparaissent que dans une combinaison spécifique de logiciels et pas dans les autres. Au stade de l'exécution, si un cash se produit, le contexte est sauvegardé sous la forme d'une stack trace. Botsing vise à générer un test qui produit la même trace. Ce test est ensuite ajouté à la suite de tests existante pour éviter que le bug qui a produit le crash ne soit réintroduit dans les prochaines versions de l'application.

Améliorer efficacement la suite de tests

STAMP contribue à améliorer la suite de tests et à réduire les crashs applicatifs. Pour déterminer si les changements dans la suite de tests sont efficaces, l'équipe du projet s'appuie sur les tests par mutation. L'intuition de cette technique est simple : un cas de test est bon s'il est capable de détecter un bug. Si un outil introduit des bugs dans l'application, il devrait exister au moins un test qui le détecte ; si ce n'est pas le cas, la suite de tests est trop faible et doit être améliorée. Tout le processus (injection de bugs, exécution des tests, évaluation de la capacité des tests à détecter les bugs) est automatique et produit en sortie une liste des bugs non détectés ainsi qu'une liste de tests à améliorer.

Un outil de mutation comme Descartes transforme automatiquement certaines instructions de l'application, créant ainsi des mutants. Un mutant est une version du programme dans lequel on injecte une faute (par exemple une négation sur une condition). Le test unitaire, créé préalablement par le développeur, devrait détecter cette mutation, mais ce n'est pas toujours le cas. Il existe plusieurs raisons pouvant l'expliquer : les données passées en entrée du programme par le test ne sont pas pertinentes pour déclencher l'exécution du bug ; les assertions du cas de test ne sont pas suffisantes pour observer la défaillance produite par le bug ; ou bien l'effet du bug est difficilement observable par manque de méthodes qui permettent de récupérer l'état du programme. Au fil des itérations, on cherche à réduire le nombre de mutants non détectés pour améliorer la fiabilité de l'application elle-même, à travers l'amélioration des tests et ses indices, le taux de couverture du code et le score de mutation.

2

Un exemple de mutation de test

Prenons un exemple avec la fonction 'Remplir le réservoir' et son test associé 'le réservoir est-il plein (Oui/Non)?'. Une première mutation possible devient : 'le réservoir est rempli à moitié'. Le test de cette fonction répondra 'Oui' ou 'Non', selon la façon dont il a été codé. S'il répond 'Oui', le réservoir est censé être plein ce qui est faux ; le changement n'a pas été détecté. Dans ce cas, le mutant survit. Il est bien visible pour que le développeur puisse améliorer le test. Si le test répond 'Non', on suppose qu'il a bien détecté le changement ; le test est valide et le mutant est alors tué, ce qui est le but recherché.

Avec l'outil Descartes, la mutation est dite extrême ; la méthode est vidée de son contenu ou retourne 0 au lieu d'une valeur prévue ; dans notre exemple, cela donnerait 'Aucune goutte n'a été ajoutée dans le réservoir'. Si un tel mutant n'est pas détecté par le test, une amélioration du test s'impose. La mutation extrême réduit de

POURQUOI AUTOMATISER LES TESTS ?

Dans les applications d'entreprise, de nombreuses fonctions imbriquées compliquent la maintenance et l'évolution du code. Faute de tests bien écrits, des bugs surviennent provoquant un comportement inadapté, des transactions erronées ou des failles de sécurité. Tout l'enjeu du test automatisé consiste à fournir un nouvel examen de vérité. Le test par mutation trouve quasiment toujours quelque chose à corriger sur une grande suite de tests. Il y parvient même dans un délai raisonnable avec la mutation extrême, l'altération automatique du code d'origine pouvant être très gourmande en ressources.

façon drastique le nombre de mutants générés, permettant de gagner du temps à l'exécution des tests. On peut ainsi insérer la mutation dans une chaîne de construction continue de l'application.

Les tests par mutation mis à l'épreuve

L'outil Descartes a été évalué sur cinq applications Java professionnelles et leurs tests respectifs fournis par les participants industriels du projet STAMP. Les développeurs ont apprécié les verdicts fournis par l'outil qui ont permis d'améliorer les assertions de nombreux tests et d'ajouter des cas de test pertinents qui manquaient dans la suite de tests.

Ainsi Descartes a-t-il permis de détecter qu'une fonctionnalité majeure du programme Sat4j développé par Daniel Le Berre, à l'Université d'Artois et membre du Centre de Recherche en Informatique de Lens (CRIL - CNRS/Université de l'Artois) n'était pas testée. Intégré à la plateforme de développement ouverte Eclipse, SAT4J fournit un ensemble d'outils de raisonnement pour le langage Java très utilisé à travers le monde. Une de ses innovations majeures a été d'offrir un tri spécifique dans la recherche de la solution du problème SAT selon une nouvelle heuristique. Or ce tri n'était pas testé, ce que la mutation extrême fournie par Descartes a révélé immédiatement. Le test a été optimisé depuis. Pour mémoire, rappelons qu'un problème SAT de « satisfaisabilité booléenne » est un problème de décision qui s'exprime sous la forme d'une formule de logique propositionnelle. Résoudre un tel problème consiste à déterminer s'il existe une assignation des variables propositionnelles rendant la formule vraie.

Corriger des bugs automatiquement

Parvient-on à corriger automatiquement certains bugs ? C'est tout l'objectif de certains outils tel Repairnator, développé par l'Inria (France) et KTH (Suède). En quelques semaines, ce robot open source, branché sur la plateforme d'intégration continue Travis CI, a analysé des centaines de builds échoués et a produit automatiquement cinq correctifs, prétendument écrits par Luc Esape - un développeur fictif. Ces solutions ont été prises en compte et acceptées par des développeurs humains qui ont fusionné les correctifs suggérés à leur propre code sans même se douter de la nature du correcteur : "Il s'agit d'un jalon important pour la compétitivité humaine dans la recherche en génie logiciel sur la réparation automatique de programmes", estime Martin Monperrus, professeur d'informatique à Stockholm (KTH) impliqué dans les projets Repairnator et STAMP. A court terme, cela pose de nouvelles questions, notamment juridiques : une fois le logiciel corrigé automatiquement, qui devient responsable des défauts persistants ?

•

L'ÉDITEUR XWIKI DÉTECTE DES BUGS PERNICIEUX

Parmi les innovations les plus utiles de la suite STAMP, l'éditeur XWiki note la faculté d'exécuter des tests fonctionnels de l'interface utilisateur de sa plateforme collaborative, dont les modules sont écrits en Java.

« Nous sommes en mesure de lancer des tests sur plusieurs configurations et environnements de production, notamment lorsque XWiki fonctionne avec Tomcat, Jetty, MySQL, PostgreSQL, avec les navigateurs Chrome, Firefox et la suite LibreOffice, dans toutes les combinaisons de ces environnements et avec différentes versions. Cela nous a déjà permis de détecter plusieurs bugs qui survenaient uniquement sur certains environnements », souligne Vincent Massol, le directeur technique d'XWiki.

QUELLE RENTABILITÉ POUR UNE SUITE DE TESTS AUTOMATISÉS ?

Il faut coder ses propres tests, les rédiger à la main correctement, avant de pouvoir les amplifier. Pour déterminer si une suite de tests automatisés sera rentable, on peut évaluer le coût de l'application en cas de défaillance. Si elle n'est plus disponible ou préconise de mauvais choix, combien l'entreprise perdra-t-elle ? D'autres risques surviennent avec la baisse de qualité, comme des brèches de sécurité applicative. L'organisation est souvent prête à investir pour contourner ces risques, en commençant par les plus critiques. Des coûts de formation, de licences et d'infrastructure supplémentaires s'avèrent cependant nécessaires pour dérouler les tests et interpréter correctement les résultats. Le retour sur investissement s'avère d'autant plus rapide que les défaillances traitées sont critiques, les mises à jour fréquentes et les corrections pertinentes.

STAMP : UNE COOPÉRATION À DIX POUR AMPLIFIER LES TESTS

Porté par un consortium de dix partenaires européens, le projet STAMP fédère quatre partenaires académiques (INRIA, Sintef, TUDelft, KTH), cinq industriels (ActiveEon, Atos, Engineering,TellU, XWiki) et la communauté open source OW2. L'équipe composée d'une trentaine de chercheurs et d'ingénieurs a conçu une méthode d'amplification des tests et un jeu d'outils de tests automatisés gratuits et au code ouvert, ciblant les applications Java et les équipes DevOps. A leur tour, celles-ci peuvent développer, configurer et mettre en production des services distribués stables, plus fréquemment. Les quatre principaux programmes sont DSpot, Descartes, Camp et Botsing, disponibles en bêta (https://l.ow2.org/stpbeta). Les deux premiers interviennent au niveau des tests unitaires - DSpot génère de nombreux tests à partir des tests unitaires déjà écrits. Descartes complète l'outil de tests par mutation Pitest (http://pitest.org) en y apportant une nouvelle stratégie dite extrême, tandis que Camp amplifie les tests de configuration en environnement Docker et Botsing facilite la détection de crash à l'exécution.

Automatisation des tests logiciels : les enjeux de l'industrialisation

Partie 1

L'agilité, l'instantanéité et la mise à jour permanente sont les maîtres mots de cette dernière décennie, et l'industrie de l'IT est au cœur de cette transformation. Cela peut être considéré comme antinomique avec la volonté d'amélioration constante de la qualité… sauf à être capable de qualifier des systèmes de plus en plus complexes, de plus en plus rapidement et fréquemment.

Jean-Baptiste Marcé
Co-founder, CEO/CTO CloudNetCare

D'après IEEE, la qualité logicielle est le degré avec lequel un système, composant ou processus satisfait aux exigences spécifiées et besoins ou attentes de ses clients / usagers.

Un des premiers besoins critiques est bien évidement d'éviter les défaillances…

Les dysfonctionnements logiciels peuvent apparaître comme conséquence de défauts à différents niveaux (Spécification, Code ou Système). Pour réduire les risques d'occurrences de défaillances en production, les tests logiciels sont indispensables.

La complexité croissante des architectures entraîne la multiplication des actions de tests à effectuer et l'augmentation de la fréquence des mises à jour entraîne la nécessité de tester de plus en plus vite.

La seule réponse à cette problématique est l'automatisation, et plus particulièrement l'industrialisation de l'automatisation, car sans processus industriel, pas d'amélioration possible !

L'objectif de cet article est de synthétiser les enjeux actuels de l'automatisation grâce à nos retours d'expériences concrets et réussis d'industrialisation de tests automatisés. Nous évoquerons les avantages, les risques et surtout les éléments à prendre en compte pour réussir un processus d'automatisation.

DISTINCTION ENTRE LES TESTS ET LES RESPONSABILITÉS DES ÉQUIPES

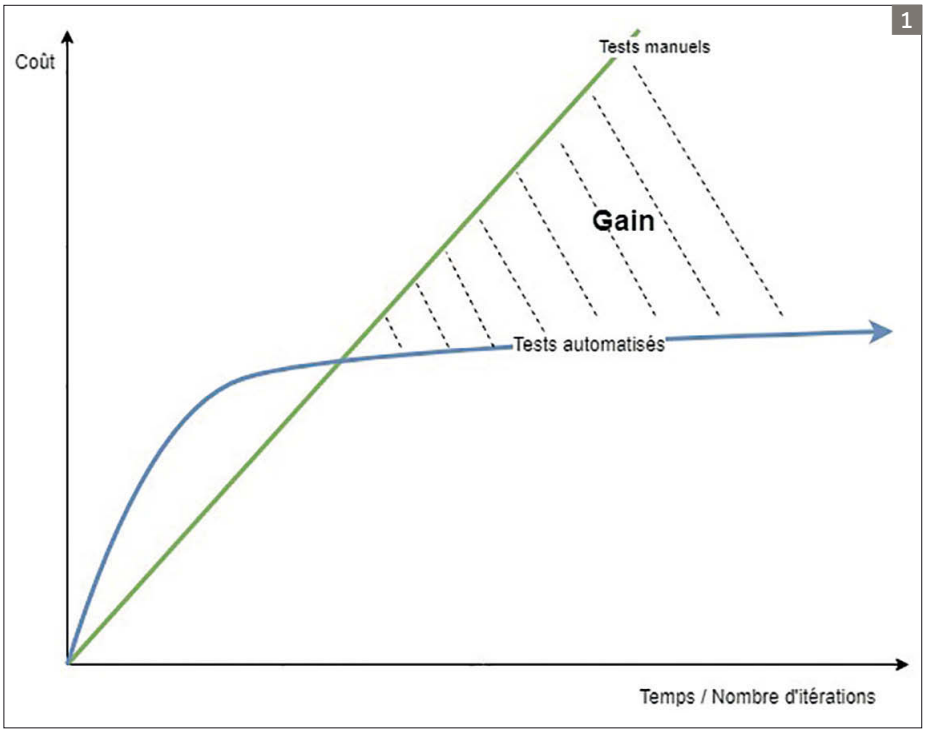
Dans le cycle de développement logiciel, nous pouvons communément distinguer deux familles de tests automatisables, les tests sous la responsabilité des équipes de développement (principalement les tests de composants, tests unitaires) et les tests sous la responsabilité des équipes « Quality Assurance ou QA » (plus généralement les tests fonctionnels / non fonctionnels que l'on retrouve aux niveaux des tests d'intégration, tests systèmes et tests d'acceptation).

Les tests unitaires sont écrits par les développeurs dans leur environnement de développement (Visual Studio, IntelliJ, Eclipse, …) avec des frameworks dédiés. Ils sont bouchonnés (pas d'interaction avec des données externes – services ou Bases de données) et ont pour objectif de tester les algorithmes de manière unitaires et reproductibles. Ils sont lancés idéalement à chaque « Build » et doivent être très rapides. Dans une démarche TDD (Test Driven Development), ils sont même créés avant d'écrire les algorithmes.

Les tests fonctionnels / non fonctionnels sont plus du ressort des équipes QA (concepteurs de tests, responsables de test, testeurs) dont le travail consiste à gérer la couverture applicative dans son ensemble. Le test fonctionnel a pour objectif de tester les fonctions applicatives d'un système incluant les jeux de données, les interactions entre les systèmes et le contexte d'exécution de production. Le test non fonctionnel teste des caractéristiques non fonctionnelles d'un applicatif comme par exemple la tenue en charge, la performance, l'usabilité, la maintenabilité, ….

Dans le cadre de l'automatisation des tests, apparaît un nouvel acteur qui devient indispensable dans le cadre de l'industrialisation : le testeur automaticien. Il est en charge de la gestion des tests automatisés, de la création des scénarios, de leur maintenance. Il est responsable de l'exécution de la couverture de test et de l'interprétation technique des résultats.

Mur de téléphones pour tests automatisés



Acteurs :

- Développeurs : en charge des tests unitaires dans l'environnement de développement
- Equipe QA :
  - Concepteur : conçoit et analyse en continu la couverture de test ;
  - Responsable des tests / testeur : planification et contrôle des tests, compétence fonctionnelles et interface avec les développeurs ;
  - Automaticien : en charge de l'automatisation des tests.

Les outils pour automatiser des tests :

L'automatisation des tests nécessite l'utilisation d'outils dédiés. Ces outils ont pour fonction de gérer le processus d'automatisation sur des environnements cibles (ex : navigateurs pour des applications Web, Smartphone pour des apps, …). Ils sont capables de d'orchestrer le lancement à grande échelle de parcours de tests dans des contextes reproductibles et hétérogènes. Ils enregistrent toutes les actions et synthétisent les résultats rapidement pour analyse.
Les bénéfices des outils d'automatisation :

Tester plus vite :

La productivité des tests est augmentée, ce qui réduit la charge des tests manuels résiduels et le travail répétitif (risque d'erreur).

Tester plus :

En élargissant considérablement la couverture de tests et le périmètre de qualification. En libérant du temps aux équipes de recette pour tester de nouvelles fonctionnalités.
La non-régression fonctionnelle peut faire partie intégrante de la couverture automatisée.

Tester mieux :

En diminuant les erreurs de manipulation, car les robots sont reproductibles.
La reproductibilité permet une évaluation objective et fiable des résultats.
En capitalisant d'une campagne à l'autre avec un accès aux résultats normalisés et historisés automatiquement.

Bien appréhender les risques de l'automatisation

Même si les bénéfices sont flagrants, il faut prendre garde aux risques sous-jacents :

Une sous-estimation du temps, du coût et de l'effort :

- Pour réaliser la première phase de mise en place d'une couverture de tests automatisés ;
- Pour obtenir des bénéfices significatifs et continus, car cela nécessite probablement un besoin d'adaptation du processus historique de tests.

Une confiance excessive dans l'outil :

- Considérer l'outil comme substitut à la conception des tests : la conception restera toujours du ressort du testeur ainsi que la remise en cause régulière de la couverture ;
- Négliger des problèmes de relation et d'interopérabilité entre les outils connexes (outils gestion des exigences, outils de gestion des incidents, outils de suivi des défauts…) ;
- Négliger le risque de déshérence d'un logiciel ou projet open source ou libre : la prise en compte continuelle de l'évolution des technologies est primordiale ;
- Négliger d'évaluer la réactivité du vendeur ou de la communauté open source pour le support, les mises à jour et la correction des défauts ;
- Négliger d'anticiper les imprévus, comme l'incapacité de support d'une nouvelle plateforme (exemple : nouvelle version d'iOS) ;
- Si l'outil d'automatisation est déployé sur site, il est indispensable d'anticiper les temps de configuration et de mises à jour récurrents. Si par exemple les tests sont effectués dans des navigateurs internet, la mise à jour doit être anticipée tous les 15 jours (cycle de release de Firefox/Chrome/Edge) ;
- Attente irréaliste placée dans la capacité des outils d'automatisation : tout n'est pas automatisable !

Points de vigilance :

Pour optimiser le taux de réussite de l'automatisation, il faut s'assurer que les tests à automatiser soient basés sur des processus de tests manuels matures et documentés.
Il faut également ne pas sous-estimer l'effort et l'implication nécessaire de l'équipe QA (concepteur, responsable des tests, testeur) dans les phases d'élaboration des tests automatisés et pendant leur exécution.
La gestion des données (élaboration des jeux d'essai, jeux de données) est un élément fondamental qui implique souvent une charge de travail importante à ne pas sous-estimer. 1

Conclusion Partie 1

Dans cette première partie, nous avons vu un aperçu des différents types de tests automatisables, de la responsabilité des différentes équipes (développeurs, testeurs, automaticiens, …), des bénéfices de l'automatisation mais aussi des risques à appréhender.
Dans une seconde partie, nous ferons un focus sur les tests fonctionnels UI où nous aborderons les différents types de solutions disponibles aujourd'hui et les enjeux du choix des outils d'automatisation.

Lionel Duport
Ingénieur Concepteur Développeur
SQLI

SQLI DIGITAL EXPERIENCE

sonarqube

Comment améliorer la qualité de code d'un projet Android en utilisant Sonar ?

« Every minute spent on not-quite-right code counts as interest on that debt »
Ward Cunningham – 1992

Ward Cunningham (connu pour être le créateur du concept de wiki) utilise l'analogie de la dette financière pour mettre en évidence que, comme les intérêts s'accumulent lorsqu'on contracte une dette, le coût de développement augmente si le logiciel comporte des imperfections techniques. Le remboursement de la dette technique liée au développement d'un logiciel prend du temps. Cela demande d'apprendre les bonnes pratiques pour le futur, mais aussi d'épurer le passif. Avant d'entreprendre des actions qui vont vous permettre de réduire votre dette technique, il faut d'abord la mesurer !

niveau 100

En théorie ça se passe bien

Il m'arrive souvent de prendre en charge la maintenance évolutive d'applications qui ont succédé à plusieurs équipes différentes et qu'il faut faire évoluer dans le temps. En général, plusieurs développeurs ont travaillé sur ces applications et n'ont pas toujours eu à suivre les mêmes conventions de codage. **L'architecture se trouve dégradée car l'enchaînement de fonctionnalités ne respecte pas toujours l'architecture initiale.** Il en résulte donc dans le temps une **dette technique importante liée à cet historique.**

SonarQube était déjà utilisé dans l'agence SQLI Lyon pour certains projets iOS. J'ai donc mis en place cet outil sur l'ensemble de mes projets Android. **SonarQube est un outil très puissant permettant de mesurer la qualité de votre projet** de plusieurs façons : respect des règles de code, documentation du code, analyse des tests unitaires mis en place, et duplication du code.

SonarQube est une plateforme open source développée par SonarSource permettant de mesurer la qualité du code source en continu. SonarQube propose des rapports sur le code dupliqué, les normes de codage, les tests unitaires, la complexité du code, les commentaires, les bugs et les vulnérabilités de sécurité.

L'inspection de la dette technique est catégorisée en différentes mesures : Fiabilité, Sécurité et Maintenabilité avec pour chacune, une note attribuée en fonction de la qualité de cette mesure : A, B, C, D, E.

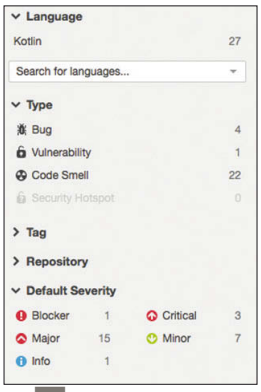
Quant aux métriques de dette technique, chaque règle rajoute une durée d'effort allant de 5 minutes à 8 heures.

Lors de l'analyse d'un projet sous SonarQube, un profil de qualité est appliqué en fonction du langage détecté. Ce profil contient l'ensemble des règles qui vont servir à identifier les points à corriger. Le profil peut être modifié afin de répondre aux besoins spécifiques d'un projet. **Il existe des profils pour Java et Kotlin, mais aussi pour C, C++, C#, PHP, Objective C et bien d'autres.** À l'heure actuelle, SonarQube supporte plus de 25 langages et fonctionne pour tous les projets Android. 1

Ces règles vous permettront de voir les problèmes et erreurs dans votre code. Ils sont triés en 3 catégories :

- **Bug** : Ce sont les bugs bloquants de votre code. Ils peuvent empêcher la compilation ou planter le programme.
- **Vulnérabilité** : Ce sont des erreurs pouvant nuire à la sécurité de votre code.
- **Code smell** : il s'agit des parties inutiles de votre code, par exemple des variables initialisées mais non utilisées.

Ces différentes règles sont catégorisées ensuite suivant leur sévérité : bloquant, critique, majeure, mineure et informative.



1 *Les règles du plugin Kotlin*

Et maintenant passons à la pratique !

Prêt à passer à la pratique ? **Nous allons maintenant regarder comment mettre en place SonarQube dans votre projet Android !** Pour cela rien de très compliqué :

1 - Installer le serveur

Pour commencer, il faut installer le serveur SonarQube qui se trouve sur la page de téléchargement du site web : https://www.sonarqube.org/downloads Ensuite, dézippez le fichier dans le répertoire de votre choix, puis lancez le serveur avec les commandes suivante sur Windows (SONARQUE_HOME étant le dossier de votre fichier dézippé) :

```
<SONARQUBE_HOME>\bin\windows-x86-xx\StartSonar.bat
```

Sur les autres systèmes d'exploitation :

```
<SONARQUBE_HOME>/bin/[OS]/sonar.sh console
```

2 - Installer le plugin Kotlin

Dans le cas d'un projet Kotlin, il faudra ajouter le plugin. Téléchargez le plugin à l'adresse https://docs.sonarqube.org/display/PLUG/SonarKotlin puis placez le fichier dans le dossier de plugins :

```
<Dossier>/extensions/plugins
```

Dans ce dossier se trouvent déjà d'autres plugins existants (Java, JavaScript, Python, PHP ...).

3 - Configurer Gradle

Votre serveur n'est pas encore lancé mais il est prêt à analyser un

projet Android, que cela soit sur Kotlin ou Java. Il reste à ajouter le plugin Sonar et configurer ses propriétés dans votre projet Android. Pour cela, il suffit d'ajouter la configuration suivante dans le fichier Gradle de l'application. Pour Gradle 2.1 et les versions supérieures :

```
plugins {
  id "org.sonarqube" version "2.6.2"
}
```

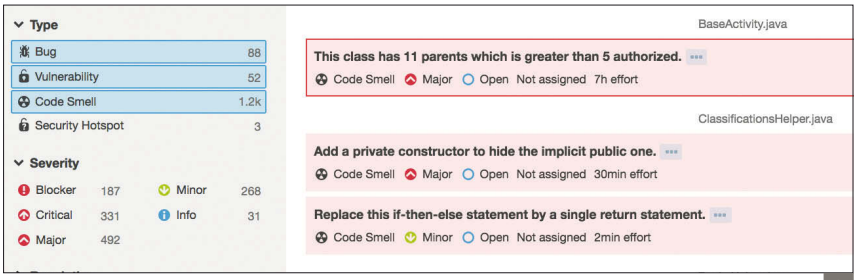
Pour les versions Gradle inférieure à 2.1 :

```
apply plugin: 'org.sonarqube'

buildscript{
  repositories {
    maven { url 'https://plugins.gradle.org/m2/' }
  }
  dependencies{
    classpath "org.sonarsource.scanner.gradle:sonarqube-gradle-plugin:2.6.2"
  }
}
```

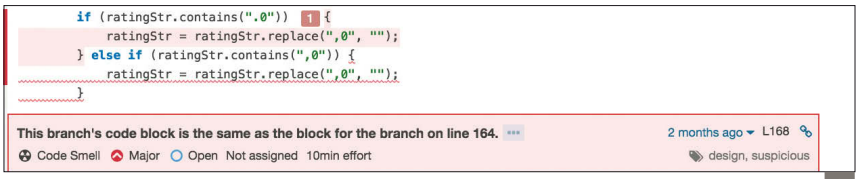
Une fois le plugin Gradle Sonar importé, une tâche sonarqube doit être créée avec différentes propriétés :

```
sonarqube {
  properties{
    property "sonar.host.url", "http://localhost:9000"
    property "sonar.jdbc.url", "jdbc:mysql://localhost:3306/sonar"
    property "sonar.jdbc.driverClassName", "com.mysql.jdbc.Driver"
    property "sonar.jdbc.username", "*****"
    property "sonar.jdbc.password", "*****"

    property "sonar.projectName", "Project Name"
    property "sonar.projectKey", "project_key"
    property "sonar.projectVersion", android.defaultConfig.versionName
    property "sonar.language", "kotlin"
    property "sonar.sources", "src"
    property "sonar.java.binaries", "build"
   }
}
```

Ces propriétés propres à Sonar permettent de configurer l'analyse de votre projet :
sonar.host.url : URL de votre serveur Sonar ;
sonar.jdbc.url : URL de la base de données ;
sonar.jdbc.driverClassName : type de la base de données ;
sonar.jdbc.username / **sonar.jdbc.password** : login et mot de passe de la base de données ;
sonar.projectName : le nom d'affichage de votre projet sur l'interface web ;
sonar.projectKey : une clef unique pour chaque projet ;
sonar.projectVersion : la version de votre projet ;
sonar.language : définissez la langue du code source à analyser : « java » ou « kotlin ». Si rien n'est défini, une analyse multilingue sera déclenchée ;
sonar.sources : le chemin vers les fichiers sources de votre projet ;
sonar.java.binaries : chemins vers les répertoires contenant les fichiers de bytecode compilés correspondant à vos fichiers sources.



2



3

4 - Lancer l'analyse de votre projet

Votre projet Android est configuré, il vous reste à lancer une analyse de votre projet en exécutant la tâche Gradle Sonarqube. Cette tâche va exécuter la commande Sonar qui va effectuer l'analyse puis envoyer les résultats au serveur. Si le scan s'est bien déroulé, vous pourrez retourner sur l'interface web de SonarQube et y voir votre application dans la liste des projets.
Une fois le scan effectué, les informations sont automatiquement mises à jour et stockées dans la base de données SonarQube.

5 – Utiliser le serveur pour améliorer votre code

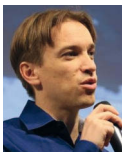
La page « Overview » vous donne une vue d'ensemble de votre projet. Vous y trouverez le nombre de bugs potentiellement détectés sur votre application, mais aussi la métrique de dette technique en nombre de jours. Deux pourcentages sont aussi présents sur cette page qui vous donneront une vision globale du taux de couverture des tests unitaires, ainsi que le pourcentage de duplication du code. Afin d'appliquer les corrections nécessaires à votre projet, vous pouvez vous rendre sur la page « Issues » afin de filtrer les erreurs par types ou sévérités. Sur le volet de droite s'affichent les règles à appliquer pour chaque classe Java. 2
En cliquant sur une règle (ici encadrée de rouge), la classe s'ouvre et affiche les corrections à appliquer. 3
Ici deux branches dans une structure conditionnelle ont la même implémentation. La correction à appliquer sera donc de fusionner les deux branches en une seule :

```
if (ratingStr.contains(".0") || ratingStr.contains(",0")) {
  ratingStr = ratingStr.replace(",0", "");
}
```

En appliquant chaque correctif sur les différentes règles, vous réduisez petit à petit votre dette technique !
Les analyses peuvent être bien sûr lancées de façon automatique sur un projet géré par des outils de versioning GIT, couplés à des outils d'intégration continue comme Jenkins ou Gitlab CI.
Voilà de quoi remettre de l'ordre dans un projet de développement et mettre en application de bonnes pratiques de développement sur une application Android au sein d'une équipe de plusieurs personnes. •

Les tests : une évolution, pas une révolution

Dans cette interview, Virgile Delécolle et Guillaume Alex, respectivement DevOps Presales Manager France & BeLux et DevSecOps Evangelist au sein de la société Micro Focus, échangent sur leurs visions et la place des tests dans une approche DevOps et présentent comment la société Micro Focus accompagne ses clients sur l'intégralité du cycle de test.



Virgile Delécolle
DevOps Presales Manager France & BeLux

Agiliste de la première heure en France, Virgile a, après une première expérience de développeur dans l'industrie Ferroviaire, rejoint l'édition logicielle chez Peregrine en tant que testeur et coach. Grace son approche pragmatique de ce disrupteur méthodologique, il a su faire évoluer la R&D française tout en continuant de suivre les évolutions de ce courant. Aujourd'hui, il est certifié SAFe DevOps et met ses compétences au service des clients Micro Focus dans un rôle de Manager Avant-Vente tout en restant en contact permanent avec la R&D pour échanger et travailler sur les roadmaps produits.

Guillaume Alex
DevSecOps Evangelist

Guillaume compte plus de 20 ans d'expérience dans l'IT principalement en tant que consultant chez des éditeurs anglo-saxons comme Mercury Interactive, HP et Hewlett Packard Enterprise. Passionné par la technologie, il a eu l'opportunité d'exercer différentes fonctions telles que développeur, chef de projet et consultant technique pour des entreprises de toutes tailles. Aujourd'hui avant-vente chez Micro Focus, il s'attache à aider ses clients à optimiser leurs processus DevSecOps notamment grâce à l'automatisation des tests et la virtualisation des services.

Pourquoi êtes-vous sensibles aux tests?

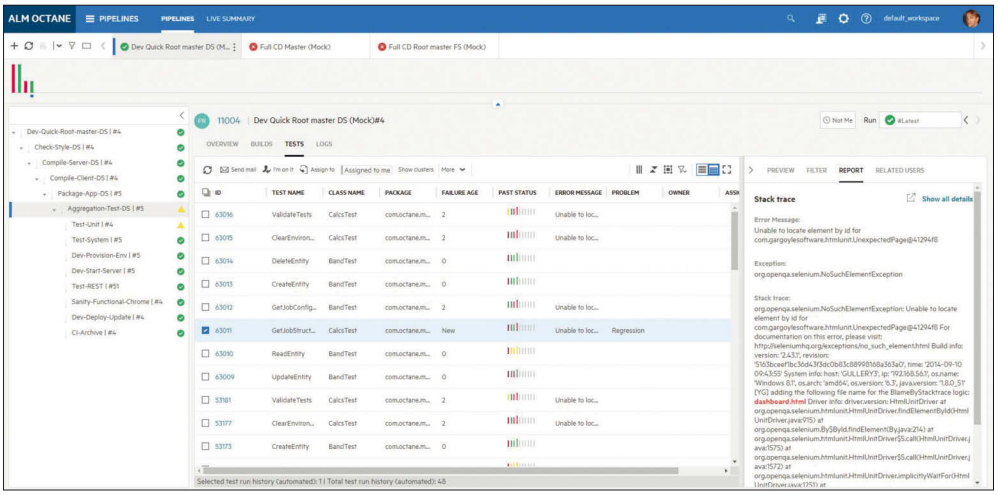
Virgile Delécolle : *J'ai démarré ma carrière dans une équipe d'Extrême développeurs : pas une ligne de code n'était écrite sans un test qui la couvre, pas une fonctionnalité n'était validée sans un test de recette automatisé et pas une livraison n'était réalisée sans vérification de la non-régression. Extrême, surtout en 2001, mais ça m'a formaté pour le reste de ma carrière.*

Guillaume Alex : *Comme Virgile, j'ai commencé à travailler en tant que développeur. En revanche, dans le cadre de cette première expérience, le test n'était pas du tout considéré comme essentiel. Ce n'est qu'un peu plus tard, lorsque j'ai participé au lancement des premiers sites de réservation en ligne du groupe Accor HOTEL que j'ai compris que tester était indispensable. Depuis, j'ai consacré toute ma carrière à prêcher la bonne parole : la non-qualité a un tel impact négatif que le test doit être au cœur du cycle de développement de toute application.*

Aujourd'hui, quelle est la place des tests pour vous dans une approche DevOps ?

Virgile Delécolle : *C'est le garde-fou de toute la démarche. On veut être plus réactif, plus souple, aller plus vite jusqu'en production mais il est impensable de le faire au détriment de la Qualité.*

Guillaume Alex : *Les entreprises que nous rencontrons qui ont déjà mis en pratique une approche DevOps ont adapté leurs méthodes de travail, l'architecture des applications et leurs modes de déploiement pour rendre possible la fourniture rapide de fonctionnalités mais aussi pour vérifier la Qualité à chaque étape.*



Les modes de déploiement ? Quel lien avec les tests ?

Guillaume Alex : *Des « Quality gates » sont mises en place entre les différentes plateformes et conditionnent le passage à l'étape suivante. Les processus de déploiement automatisé doivent en tenir compte, c'est-à-dire exécuter les tests et prendre en considération leurs résultats.*
Virgile Delécolle : *Des pratiques telles que les « dark launches » ou les « features toggles » permettent aussi de tester très tard ou de recueillir le feedback d'un échantillon d'utilisateurs avant la mise à disposition générale.*

Que les tests aient autant de place, c'est une révolution?

Virgile Delécolle : *En fait non. Il y en a toujours eu autant…. En théorie. Quand on regarde la pyramide des tests, il y en a à chaque étape mais pendant longtemps, seuls les testeurs s'en occupaient, donc*

faute de temps et de moyens, tous n'étaient pas effectués. Depuis l'adoption plus large des méthodes agiles, la Qualité est devenue l'affaire de tous, donc il y a plus de tests, de tous les types et réalisés par plus de personnes. C'est la raison pour laquelle je parle d'évolution et pas de révolution.

Guillaume Alex : *La révolution, c'est que la performance et la sécurité applicatives ont enfin voix au chapitre. Plus sérieusement, les tests de performance et de sécurité ne sont pas nouveaux mais ce n'est que depuis peu qu'ils sont également intégrés aux chaînes CI/CD et que leurs résultats ont autant de poids dès le début du cycle de développement. On parle de DevSecOps mais les tests de sécurité sont implicites dans une approche DevOps, au même titre que les tests fonctionnels ou de performance.*

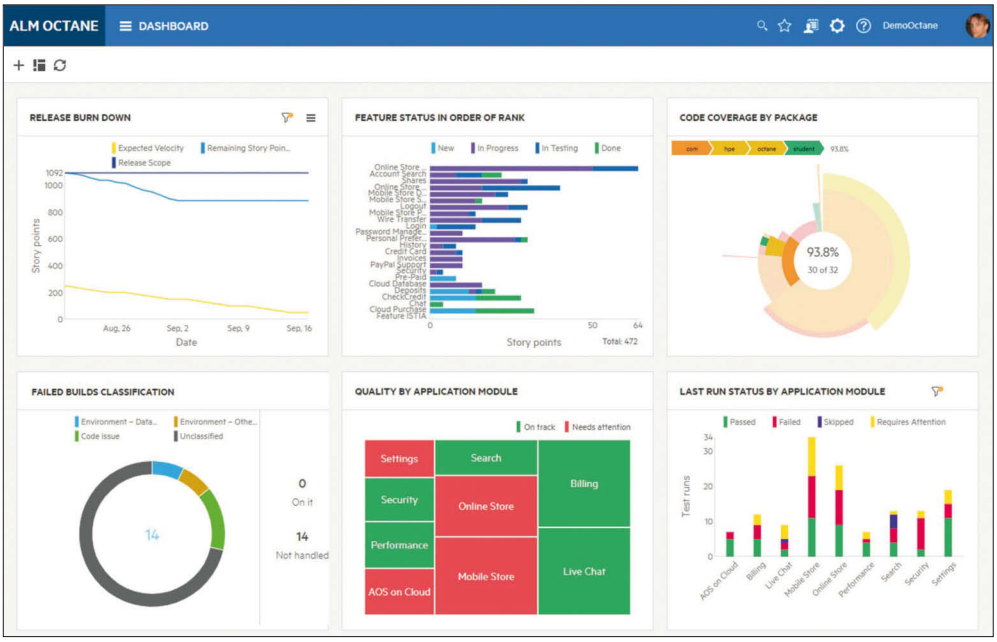
Donc plus de types de tests de différentes natures…

Guillaume Alex : *Au risque d'enfoncer des portes ouvertes, il y a aussi beaucoup plus de tests en intégration ou en recette. Dans le cas d'un logiciel par exemple, on testait déjà tous les OS, serveurs Web, middlewares et bases de données qu'on devait supporter. Aujourd'hui, toutes les applications sont développées en responsive web design et de ce fait doivent être testées sur tous les principaux navigateurs du marché (Chrome, Safari, Firefox, IE, Edge…), sur différents OS (Linux, Mac et Windows) ainsi que les OS mobiles (Android et iOS), mais aussi en différentes versions y compris les versions Betas.*

Virgile Delécolle : *N'oublions pas non plus que les applications doivent être disponibles en plusieurs langues, nos clients travaillant souvent à l'International. Heureusement que l'on peut automatiser la majorité des tests sur les dernières technologies.*

Donc tout est réglé ?

Virgile Delécolle : *Heureusement que non, sinon on n'aurait plus besoin de nous :-). Le défi n'est plus d'avoir les moyens de tester ou de pouvoir automatiser. Le défi c'est quoi faire de tous ces résultats de tests. Il faut trier, analyser, relier, assigner…*



Si tout est fait à la main, on revient au temps où les tests ralentissaient la Value Stream et où il arrivait fréquemment qu'on ne les attende pas.

Guillaume Alex : *Si tout est « vert », pas de problème. Pareil si tout est rouge. Si par contre, plusieurs centaines de tests sont en échec sur plusieurs milliers, par où commence-t-on ? Il n'est pas nécessairement question d'avoir tous les tests au vert mais plutôt de gérer le risque. On peut promouvoir un package sur une plateforme même si tout ne va pas mais on veut savoir quoi. Plus on se rapproche de la production, plus les critères sont stricts et souvent encore, la dernière « Quality gate » requiert une approbation manuelle.*

Comment Micro Focus peut aider ?

Guillaume Alex : *Avoir un seul référentiel pour le fonctionnel, la performance et la sécurité, avoir une traçabilité automatique entre les exigences/stories et les exécutions de test sont un minimum pour être efficace. C'est le rôle d'ALM Octane d'être le backbone de la Qualité. L'utilisation de l'IA va aussi beaucoup apporter. On peut par exemple analyser les logs et regrouper les tests ayant potentiellement la même origine (cf : Pipeline tests clustering dans ALM Octane)*

Virgile Delécolle : *Le machine learning permet aussi de catégoriser automatiquement les échecs d'un build et d'identifier les personnes susceptibles d'aider à la résolution (cf : Build failure autocatégorisation et Suggested users dans ALM Octane). Si le build ne sort pas : pas de test et donc pas de visibilité sur la Qualité. L'amélioration continue doit s'appliquer aux spécifications, au code, à la chaîne de fabrication et aux tests.*

Tout ceci n'est valable que pour le Web ?

Virgile Delécolle : *C'était un exemple, on ne s'appellerait pas Micro Focus si on n'adressait pas aussi le Cobol. Il y a par exemple un framework de tests unitaires (xUnit like) dans Enterprise ou Visual Cobol, et on peut sans problème intégrer des tests automatiés frontend (3270 ou web) au pipeline de build.*

Guillaume Alex : *Le DevOps n'est pas réservé au Web. En effet, rien n'empêche d'avoir un train de releases SAFe incluant du mainframe ou de faire un projet de migration d'ERP avec des chaînes CI/CD. Par ailleurs, quelle que soit la technologie, la virtualisation de services peut être un outil précieux pour s'assurer que tous les composants ou systèmes impliqués soient disponibles et représentatifs.*

Donc pour résumer, des tests, des tests et encore des tests ?

Guillaume Alex : *Des tests fonctionnels, de performance et de sécurité !*

Virgile Delécolle : *Sans oublier d'analyser tous les résultats. Nous sommes tellement confiants dans la Qualité de nos solutions que nous vous invitons à les télécharger et à les essayer gratuitement sur notre site web* www.microfocus.com •

Hamza Iqbal
Consultant web Front-End
https://blogs.infinitesquare.com/

Maîtrise de la qualité en projet web : le point de vue du développeur

Partie 1

Aujourd'hui, presque n'importe qui a la possibilité de créer son site web et même de lancer sa boutique en ligne à moindre frais. Mais qu'en est-il de la possibilité de faire des sites et des services en ligne de qualité ? Est-ce une notion applicable, mesurable et durable dans le temps ? D'ailleurs, qu'est-ce qui définit la qualité web et qui concerne-t-elle ?

C'est à ce type d'interrogation que répond l'organisme Opquast, dont le but est d'accompagner les professionnels du web à améliorer leurs services.

Nous allons découvrir les principes et les fondements de la Qualité Web, la démarche proposée par **Opquast** pour en maîtriser les coûts et les risques, ainsi que la méthode de certification des professionnels aux bonnes pratiques.

Opquast (https://www.opquast.com/) est un organisme français de formation et de certification autour de la Qualité Web. Derrière ce nom se cache un acronyme : *Open Quality Standards*.

Depuis sa création en 2004, l'objectif d'Opquast est de contribuer à l'amélioration de la qualité des services en ligne en proposant, sous licence ouverte (Creative Commons BY-SA), des référentiels et des outils à destination des professionnels du Web. Opquast est le pionnier mondial de la qualité Web, dont il est un acteur connu et reconnu.

L'activité d'Opquast tire son essence de la publication et du maintien d'une *check-list* de **bonnes pratiques** (https://checklists.opquast.com/oqs-v3/) qui se présentent sous forme d'affirmations universelles, consensuelles, réalistes et vérifiables.

En plus de proposer un référentiel de bonnes pratiques sur la qualité web, Opquast édite également des référentiels sur le web mobile, le SEO, la performance et l'éco-conception web .

Qu'est-ce que la qualité Web ?

Aujourd'hui, la majorité des secteurs d'activité est soumise à des **exigences** et des **contrôles de qualité** par le biais de règles et de critères stricts. Les professionnels de l'alimentaire, de l'automobile, du bâtiment ou de la distribution répondent à des **référentiels** précis pour nous garantir la qualité de leurs offres respectives et nous fournir le meilleur service possible. Souvent, un simple label qualité peut nous faire pencher pour un produit / service ou un autre.

Alors pourquoi ne pourrait-on pas avoir d'exigence en matière de qualité web ?

Pourquoi ne pas disposer de critères objectifs visant à améliorer et renforcer la qualité web ?

Le problème est que la qualité web n'est pas tangible et reste **subjective**, dépendante de l'utilisation et de l'expérience de chacun. Parfois, le manque de qualité n'est même pas perçu tant les utilisateurs sont habitués aux mauvaises expériences sur Internet.

Opquast tente de rationaliser les choses en définissant la Qualité Web de la manière suivante :

> *"La qualité web représente l'aptitude d'un service en ligne à satisfaire des exigences explicites ou implicites."*

De cette définition découlent 3 objectifs :

- Déterminer les attentes et exigences telles qu'elles sont formulées par les utilisateurs ;
- Trouver les réponses à des problématiques que les utilisateurs ne savent pas forcément formuler mais qui s'avèrent contraignantes dans leur usage ;
- Définir une méthode de management, d'évaluation, d'amélioration et de pérennisation de la qualité web.

Quel est le périmètre de la qualité web ?

Élie Sloïm (Président et fondateur d'Opquast) et Éric Gateau (Temesis) ont conçu un **modèle générique et simple** synthétisant les attentes des utilisateurs, mettant en avant les qualités principales espérées sur un site / un service en ligne et fournissant une vision transversale des corps de métiers contribuant à la qualité web.

Il s'agit du modèle **VPTCS** (pour **V**isibilité, **P**erception, **T**echnique, **C**ontenus et **S**ervices). 1

- La **Visibilité** désigne l'aptitude d'un site à être rencontré par ses utilisateurs potentiels (référencement, positionnement, communication, réseaux sociaux, publicité, ...) ;
- La **Perception** représente son aptitude à être utilisable et correctement perçu par ses utilisateurs (ergonomie, graphisme, navigation, design d'interaction et d'information, ...) ;
- La **Technique** concerne son aptitude à fonctionner correctement (accessibilité, intégration, performance, sécurité, ...) ;
- Les **Contenus** recouvrent l'aptitude à délivrer de l'information de qualité (rédaction, orthographe, vocabulaire, traduction, architecture de l'information, structuration, ...) ;
- Les **Services** déterminent son aptitude à proposer, accompagner, et/ou générer la réalisation de services de qualité (e-commerce, logistique, suivi, service client, réactivité, ...) ;

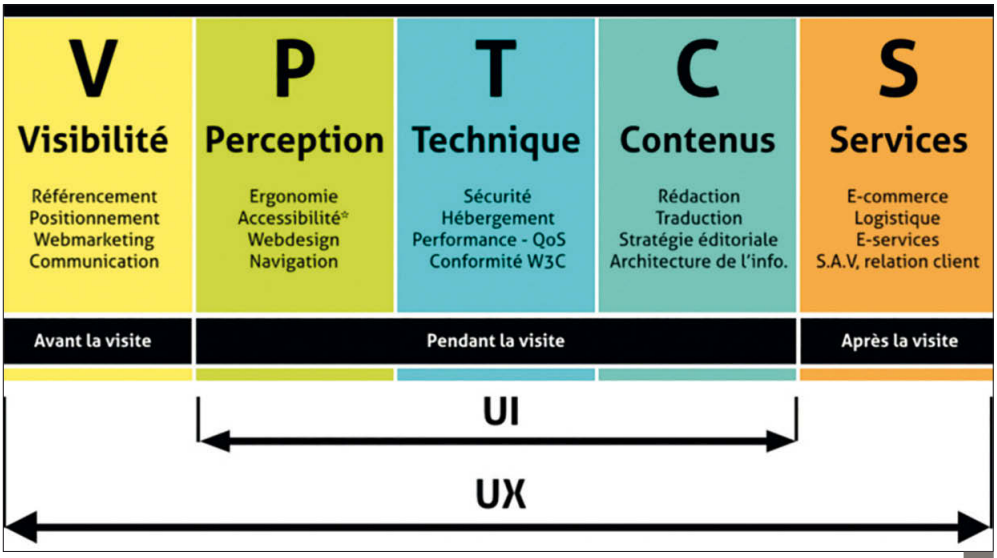
La qualité web couvre donc l'Expérience Utilisateur web au sens large : avant, pendant et après la visite du site.

Pourquoi un périmètre aussi **large** ? Tout simplement car la qualité, la réputation et la confiance accordées à un site vont

également dépendre de ce qu'il se passe en **amont** et en **aval** de celui-ci. Un site facile à trouver et ayant plusieurs liens référents est **favorisé** par les moteurs de recherche et réduit la **méfiance** des utilisateurs. De l'autre côté, un site répondant et accompagnant correctement les utilisateurs dans leurs démarches **fidélisera** sa cible sur le long terme et améliorera sa réputation. C'est une façon plus poussée d'**optimiser** les contenus et les services en plus de la mise en place de principes ergonomiques et de solutions techniques, choses qu'on ne trouve que sur la partie visible du site en question.

Pour en revenir au modèle, comme dit précédemment, il s'agit d'une approche **transversale** et **générique**. Sa particularité réside dans le fait qu'il met au même plan tous les aspects d'un site de façon **équilibrée** sans privilégier d'approche spécifique. Il est suffisamment segmenté pour définir une chaîne de production web ainsi que la catégorisation des différents intervenants.

Pour comprendre et appliquer en détail les spécifications de qualité, il est nécessaire d'aller **au-delà** du modèle VPTCS. C'est pourquoi la démarche qualité web se décompose en un ensemble de **bonnes pratiques**.



Modèle VPTCS (simplifié) pour la qualité web 1

Qu'est-ce qu'une bonne pratique ?

De la formulation d'une exigence subjective…

Quand un client demande un site avec une navigation ergonomique et intuitive, il s'agit de propos libres d'interprétation et passibles de débats sans fin (avec le client et entre les professionnels eux-mêmes) et donc potentiellement sources d'erreurs. Comment faire consensus et répondre efficacement à cette demande ? C'est ce qu'Opquast a fait en commençant à proposer et publier un ensemble de **bonnes pratiques** génériques à destination des professionnels du Web.

… à la définition d'un critère objectif…

Pour exister, chaque bonne pratique doit être :

- **Utile** : a-t-elle une valeur ajoutée démontrable pour les utilisateurs ?
- **Réaliste** : est-elle réaliste compte tenu des contraintes inhérentes aux projets ?
- **Consensuelle** : a-t-elle reçu l'approbation de suffisamment de professionnels du secteur ?
- **Universelle** : est-est valable au niveau international ?
- **Testable** : est-elle testable et vérifiable en ligne de manière automatisée ou par un intervenant tiers et non spécialiste ?

Une bonne pratique doit également respecter un critère d'**unicité** : on ne teste qu'une seule chose à la fois. Elle ne doit également pas faire de référence à une solution technique (risque d'obsolescence), ni à une valeur numérique (pas de consensus), ni à une norme, un standard ou une législation (pas d'universalité ni de validité dans le temps).

Là où la Qualité Web est quelque chose de difficilement **mesurable** à cause de son caractère **subjectif**, les bonnes pratiques sont des critères **objectifs**, **vérifiables** et utilisables comme axes d'amélioration **planifiables** et **gérables** à l'échelle d'un projet.

… et générique

Une bonne pratique n'a **pas vocation à brider** les intervenants dans leurs domaines d'expertises mais plutôt de les **guider** et de leur permettre d'anticiper des erreurs évidentes de non-qualité.

Par exemple : les bonnes pratiques vont demander aux graphistes de fournir un contraste suffisant pour la lisibilité du texte mais en aucun cas leur imposer une palette de couleurs ou une limitation du choix des polices de caractères. Pareil pour les développeurs à qui l'on demandera d'effectuer certaines configurations au niveau du serveur (minification de fichiers, mise en cache, sécurité, …) mais pas de restreindre leurs choix technologiques. De même, les bonnes pratiques n'empièteront pas au niveau des législations en vigueur (ex : déclaration à la CNIL) mais axeront plutôt leurs objectifs autour de la fidélisation et de la relation de confiance entre les utilisateurs et le site. Quelques exemples de bonnes pratiques à destination des profils techniques

Bonne pratique n°12 :

Chaque identifiant HTML n'est utilisé qu'une seule fois par page

Objectifs :

- Éviter les erreurs de restitution du contenu ou d'interaction via les scripts.
- Limiter les risques d'interprétation hasardeuse du Document Object Model (DOM) par des agents utilisateurs différents.

Bonne pratique n°17 :

Le codage de caractères utilisé est UTF-8

Objectifs :

- Recourir à un jeu de caractères international ;
- Prévenir les défauts d'affichage ;
- Faciliter la manipulation des contenus par les utilisateurs et les développeurs.

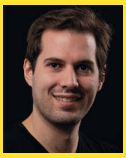
Bonne pratique n°206 :

Le serveur envoie les informations permettant la mise en cache des contenus.

Objectifs :

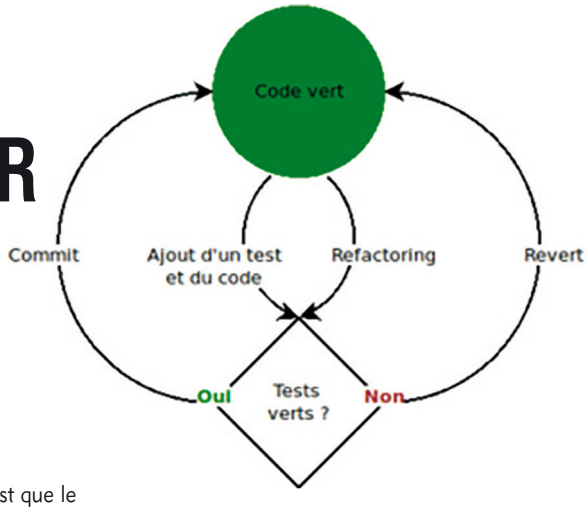
- Accélérer l'affichage des contenus et permettre une navigation plus fluide ;
- Réduire les coûts de bande passante.

La suite de cet article dans le prochain numéro



Xavier Detant
Consultant et formateur à Zenika Paris

TDD est mort, longue vie à TCR



Cela fait quelques mois maintenant que Kent Beck a publié son article sur TCR :
https://medium.com/@kentbeck_7670/test-commit-revert-870bbd756864

Pour résumer l'article, TCR est un acronyme de `test && commit || revert`. L'idée est d'avancer avec des petits pas maîtrisés et de revenir en arrière automatiquement au moindre dérapage. Cette pratique est contradictoire avec TDD qui impose une étape où un nouveau test doit ne pas passer. Ce qui est surprenant venant de la personne qui a inventé TDD !

La grande contrainte de TCR : le code doit toujours être vert

C'est la première chose qui saute aux yeux : si la commande `test && commit || revert` est lancée alors que le code ne compile pas ou qu'il ne passe pas les tests (code rouge), alors tous les changements sont immédiatement supprimés afin de retomber dans un état stable (code vert).
Cela semble contre-productif au départ et c'est exactement pour cette raison qu'il faut essayer (TDD aussi semble contre-productif à première vue). Après quelques heures d'essai (seul et à plusieurs), il apparaît que ce n'est pas contre-productif du tout et que c'est très fun.
Ce n'est pas contre-productif car TCR nous impose d'avancer à notre rythme. Tout le monde veut aller le plus vite possible, mais TCR vient nous mettre une limitation de vitesse en fonction de nos capacités du moment. Au moindre faux pas, TCR met le pied sur le frein un peu comme un régulateur de vitesse dans une voiture. Si le développeur est en forme et arrive à faire de grandes avancées dans son code, TCR ne l'empêchera pas, mais s'il est fatigué et/ou trop confiant, alors les changements seront annulés jusqu'à ce que le développeur fasse preuve d'humilité. Au final, on se retrouve à avancer à la vitesse maximale qui reste sûre pour la stabilité du code.
TCR est fun (et un peu frustrant) car on a l'impression de jouer au poker contre le code. Quand on lance la commande on peut s'entendre dire «je parie que ce code est vert ! Tapis !». Si on a raison, on gagne un commit, si on se trompe, on perd son code.

La grande force de TCR : le code est toujours vert

Le grand avantage de cette démarche est que le code est toujours dans un état stable et qu'il est commité (très) souvent. La conséquence est que le code est toujours intégrable. On peut donc pousser l'intégration continue à son paroxysme.
Il est par exemple imaginable de faire tourner un script du genre :

```
while(true){
  git pull --rebase && git push
}
```

Ainsi, au moindre changement commité, il est intégré et prêt à être partagé avec les autres membres de l'équipe.
Cela peut sembler extrême (comme l'Extreme Programming) mais imaginez une autre personne qui travaille sur le même morceau de code que vous sans que vous le sachiez. Si vous travaillez de manière «classique» vous vous en rendrez compte quand vous et elle aurez fini. Avec cette méthode, vous serez au courant au plus tôt : dès que les deux auront fait passer leur premier test. Ce sera alors l'occasion de discuter avec elle voir de pairer !
Encore une fois, on gagne en productivité !

La grande faiblesse de TCR : le code est toujours vert

Du fait que le code soit toujours vert, il est impossible d'observer un test passer au rouge pour le faire passer au vert ensuite. Or il s'agit là d'une étape importante de TDD.
En effet, cette étape permet de valider :
- Que le test vérifie effectivement quelque chose ;
- Que le code ne fait pas déjà passer le test ;
- Que le message d'erreur est suffisamment explicite pour comprendre ce qui s'est mal passé.

Le point 1 peut être mis de côté si nous partons du principe que cela est trivial, il suffit d'avoir un `assert` dans le test (spoiler alert, il est possible d'avoir un `assert` et de ne tester aucun code de production). Cependant, les points 2 et 3 ne peuvent être validés qu'en lançant le test ! Or, si nous faisons ça avec TCR, le test disparaîtra suite au revert. TCR semble donc incompatible avec TDD

Oui, mais…

Rien ne nous interdit de lancer la commande `test` sans le reste. Par exemple, quand on pense que le test doit être rouge, alors on ne lance que `test` sinon on lance `test && commit || revert`. Et voilà ! TDD et TCR sont parfaitement compatibles. Il suffit de ne pas être dogmatique sur la commande de TCR (qui est, pour rappel, juste une idée lancée il y a deux mois).
À partir de là, il est possible d'aller plus loin. Par exemple, à l'étape rouge, pourquoi ne pas lancer `test && revert`. Cela aura pour effet de forcer le test à être rouge et apporte les mêmes bénéfices que TCR pour la phase vert (maîtrise de la vitesse et fun).
Allons encore plus loin ! Pourquoi ne pas lancer `test && revert || commit` ? Pour une raison simple : on se retrouve à commiter du test rouge. Or, pour rappel, nous avons un script qui fait de l'intégration en boucle. Nous allons donc partager du code rouge avec tous nos collègues !
Dans le cadre de TCR, cela aura pour effet de reverter tout changement qu'ils feraient. C'est idéal pour se faire des ennemis !
Cependant, si nous changeons notre script d'intégration continue en :

```
while(true){
  git pull --rebase && test && git push
}
```

Alors aucun risque de pousser du code rouge !
Cette piste me semble très prometteuse et c'est ce que j'explore à l'heure actuelle. Je partagerai mon expérience dans un prochain article. En attendant, je me fais écho de Kent Beck et vous encourage fortement à essayer TCR pour voir ce que ça pourrait changer pour vous.

•

Anthony Giretti
MVP, MCSD
Developer, Blogger, Speaker
anthony.giretti@gmail.com
http://anthonygiretti.com

Tests unitaires et tests d'intégration sur ASP.NET Core 2.2.

Partie 1

Souvent négligés, les tests unitaires permettent de vérifier qu'une implémentation reste conforme à la règle de fonctionnement attendue, évitant les régressions en cas d'évolution du programme. Les tests d'intégration, eux, servent à vérifier ce que les tests unitaires ne peuvent pas faire : tester le fonctionnement de chaque couche du logiciel entre elles. Exemples avec FluentAssertions, Xunit, NSubstitute et TestServer.

niveau 200

Dans cet article, je vais vous présenter les outils nécessaires à la réalisation de ces tests et vous montrer l'utilisation de ces derniers avec un exemple de test unitaire et de test d'intégration dans ASP.NET Core 2.2. Voici les outils que nous allons utiliser :

FluentAssertions

FluentAssertions est une librairie d'assertion. C'est un ensemble de méthodes d'extension .NET qui vous permet de spécifier plus naturellement le résultat attendu d'un test de type TDD ou BDD. Cela permet d'entreprendre des tests avec une syntaxe intuitive et simple. Pour en savoir davantage sur FluentAssertions, la documentation se trouve ici : https://fluentassertions.com/documentation/

XUnit

xUnit.net est un outil de test unitaire gratuit, open source et axé sur la communauté pour .NET. Écrit par l'inventeur original de NUnit v2, xUnit.net est la dernière technologie en matière de tests unitaires de C #, F #, VB.NET et d'autres langages .NET. xUnit.net fonctionne avec ReSharper, CodeRush, TestDriven.NET et Xamarin. Il fait partie de la fondation .NET et est régi par son code de conduite. Il est sous licence Apache 2 (une licence approuvée par OSI). Pour en savoir davantage, vous pouvez consulter sa documentation ici : https://xunit.github.io/#documentation

NSubstitute

NSubstitute est un framework de mocking .NET. Ce dernier est une alternative conviviale aux autre frameworks de mocking .NET comme Moq, Rhino.Mocks. Pour rappel, le mocking sert à créer de faux objets, notamment instancier des objets à partir des interfaces injectées dans nos constructeurs de classe via l'injection de dépendance. La documentation est accessible à cette adresse : http://nsubstitute.github.io/

TestServer

ASP.NET Core inclut un hôte de test qui peut être ajouté aux projets de test d'intégration et utilisé pour héberger des applications ASP.NET Core, servant des demandes de test sans avoir besoin d'un hôte Web réel. En somme, il va permettre d'émuler un serveur pour exécuter de véritable appels sur notre Web API. Voici le lien de la documentation Microsoft : https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.testhost.testserver?view=aspnetcore-2.2

INSTALLATION

Pour utiliser ces outils, nous allons commencer pour télécharger leur package respectif :

```
PM> Install-Package FluentAssertions
PM> Install-Package xunit
PM> Install-Package xunit.runner.console
PM> Install-Package NSubstitute
PM> Install-Package Microsoft.AspNetCore.TestHost
```

PRÉPARATION ET CRÉATION D'UN TEST UNITAIRE

Nous allons prendre comme exemple un middleware permettant de gérer les erreurs dans ASP.NET Core 2.2.
Pour rappel un middleware ASP.NET Core est un logiciel assemblé dans un pipeline d'applications pour gérer les demandes et les réponses. Chaque composant :

- Choisit si la requête doit être transmise au composant suivant du pipeline ;
- Peut effectuer des travaux avant et après le prochain composant du pipeline.

Voici un exemple de middleware qui gère des erreurs globalement dans notre application ASP.NET Core, logue les erreurs et contrôle la réponse au client en lui renvoyant un message d'erreur formaté qui n'est autre qu'un objet JSON contenant un code d'erreur, un message d'erreur et une liste d'erreurs si requis.

```
namespace WebApiDemo.Models
{
  public class Error
  {
    public string Code { get; set; }
    public string Message { get; set; }
    public IEnumerable<string> Errors { get; set; }
  }
```

```
}

using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Net;
using System.Threading.Tasks;
using WebApiDemo.Models;

namespace WebApiDemo.Middlewares
{
  public class CustomExceptionMiddleware
  {
    private readonly RequestDelegate _next;
    private readonly ILogger<CustomExceptionMiddleware> _logger;

    public CustomExceptionMiddleware(RequestDelegate next, ILogger<CustomException
Middleware> logger)
    {
      _next = next;
      _logger = logger;
    }

    public async Task Invoke(HttpContext context)
    {
      try
      {
        await _next.Invoke(context);
      }
      catch (Exception ex)
      {
        await HandleExceptionAsync(context, ex);
      }
    }

    private async Task HandleExceptionAsync(HttpContext context, Exception exception)
    {
      var response = context.Response;
      var message = "Unhandled error";
      var code = "00009";
      var errors = new List<string>();

      response.ContentType = "application/json";
      response.StatusCode = (int)HttpStatusCode.InternalServerError;

      // log de l'erreur
      _logger.LogError(exception, exception.Message);

      // Réponse
      await response.WriteAsync(JsonConvert.SerializeObject(new Error
      {
        Code = code,
        Message = message,
        Errors = errors
      }));
    }
  }
}
```

Comme vous pouvez le constater nous avons un constructeur qui prend en paramètre un *RequestDelegate* et un *ILogger*.
Un *RequestDelegate* n'est autre qu'un délégué prenant en paramètre un objet *HttpContext*.
Voici la signature du délégué :

```
//
// Summary:
//    A function that can process an HTTP request.
//
// Parameters:
//   context:
//    The Microsoft.AspNetCore.Http.HttpContext for the request.
//
// Returns:
//    A task that represents the completion of request processing.
public delegate Task RequestDelegate(HttpContext context);
```

Préparation du test

Vous l'aurez surement compris nous venons d'identifier les objets à « mocker » ou « simuler » pour tester notre middleware qui en dépend.
Il ne nous reste maintenant plus qu'à identifier ce que nous allons tester. Comme je l'ai dit plus haut, notre middleware logue les erreurs et contrôle la réponse au client en lui renvoyant un message d'erreur formaté qui n'est autre qu'un objet JSON contenant un code d'erreur, un message d'erreur et une liste d'erreurs si requis.
Nous allons donc tester :

- Le Content-Type, nous attendons la valeur "application/json" ;
- Le StatusCode, nous attendons la valeur 500 ;
- Nous nous attendons à ce que logger entre en action une fois ;
- Et nous nous attendons à ce que la réponse renvoie un code avec la valeur "00009", le message "Unhandled error" et une liste d'erreurs vide.

Maintenant que nous avons identifié comment initialiser notre test et quoi tester, nous sommes prêts à écrire ce dernier.

Création du test

Etape 1 : Création du squelette du test

```
using FluentAssertions;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;
using NSubstitute;
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Threading.Tasks;
using WebApiDemo.Middlewares;
```

```
using WebApiDemo.Models;
using Xunit;

namespace UnitTestsDemo
{
  public class CustomExceptionMiddlewareTests
  {
    [Fact]
    public async Task WhenAGenericExceptionIsRaised_CustomExceptionMiddlewareShould
HandleItToProperErrorResponseAndLoggerCalled()
    {
      // Test ici
    }
  }
}
```

C'est une étape importante pour bien commencer un test. Nous avons besoin du runner de test, et nous avons choisit *Xunit*. Pour que le test puisse être découvert et déclenché, il est nécessaire d'ajouter l'attribut suivant sur chaque méthode de test : *[Fact]*.

Il est également nécessaire lors de l'écriture d'un test unitaire de le nommer correctement, pourquoi ? car un test unitaire doit être facile à lire et à comprendre, en un coup d'œil il doit être possible de le comprendre et de le réparer rapidement si besoin est. C'est pourquoi ne vous sentez pas gêné de le nommer avec un nom à rallonge. D'ailleurs le nom devrait avoir deux parties, la première à gauche de « _ » devrait énoncer le scenario du test, la partie de droite comment le test devrait réagir.

Interpretation du test nommé ainsi :

```
« WhenAGenericExceptionIsRaised_CustomExceptionMiddlewareShouldHandleItToProper
ErrorResponseAndLoggerCalled"
== Quand une exception est levée, le custom middleware devrait la gérer en renvoyant l'erreur
 adéquate dans la réponse et appeler le logger.
```

C'est clair non ?

Etape 2 : on arrange le test en préparant les données nécessaires au test.

```
[Fact]
public async Task WhenAGenericExceptionIsRaised_CustomExceptionMiddlewareShouldHand
leItToProperErrorResponseAndLoggerCalled()
{
// Arrange
    var loggerMock = Substitute.For<ILogger<CustomExceptionMiddleware>>();
    RequestDelegate requestDelegate = ((innerHttpContext) =>
    {
        throw new Exception("Oooops error!");
    };
    var middleware = new CustomExceptionMiddleware(requestDelegate, loggerMock);

    var context = new DefaultHttpContext();

    context.Response.Body = new MemoryStream();
}
```

Nous avons besoin d'une instance de *ILogger<CustomExceptionMiddleware>* et c'est ici que *NSubstitute* entre en jeu. Il va nous permettre de créer un « Mock » de ce dernier et c'est la méthode statique *Substitute.For<T>()* qui s'en charge.

Nous avons notre premier paramètre constructeur pour le middleware à tester, il nous reste à obtenir une instance de *RequestDelegate* comme décrit ci-dessus. Pas besoin de mocker quoi que ce soit, nous pouvons obtenir facilement une instance d'un *RequestDelegate* puisqu'il s'agit d'une classe concrète et non pas d'une interface.

Il nous reste plus qu'à obtenir une instance de *HttpContext* qui sera passée en paramètre de notre méthode *Invoke* par la suite. Pour ce faire nous allons instancier la classe *DefaultHttpContext*. C'est une nouveauté apportée par ASP.NET Core et .NET standard, avant l'arrivée de ces derniers (ASP.NET classique) il fallait mocker par soi-même *HttpContext* et cela était relativement lourd.

Nous finissons notre arrangement en initialisant la propriété *Body* de la réponse.

Etape 3 : On agit en exécutant le test :

```
//Act
await middleware.Invoke(context);
```

C'est tout!

Etape 4 : On interprète les résultats en faisant nos assertions

Code complet sur programmez.com & github

Après avoir repositionné le pointeur au début du Stream dans le body et désérialisé la réponse, nous allons pouvoir tester si la réponse obtenue correspond bien à celle attendue.

Pour ce faire allons enfin utiliser *FluentAssertions*.

Est-ce que l'objet retourné correspond bien aux attentes ?

```
objResponse
 .Should()
 .BeEquivalentTo(new { Message = "Unhandled error", Errors = new List<string>() , Code = "00009" });
```

Est-ce que le statut Http correspond bien aux attentes ?

```
context.Response.StatusCode
 .Should()
 .Be((int)HttpStatusCode.InternalServerError);
```

Est-ce que le content-Type correspond bien aux attentes ?

```
context.Response.ContentType
.Should()
.Be("application/json");
```

Est-ce que le logger a été invoqué une fois conformément aux attentes ?

```
loggerMock.Received(1);
```

Nous venons de réaliser un test unitaire dans ASP.NET Core 2.2 avec des outils modernes et performants.

Pour parfaire notre pratique de tests, nous allons maintenant voir comment créer des tests d'intégration.

La suite de cet article dans le prochain numéro

Laurent YOVANOVITCH
Arolla

niveau 200

Tester un repository Azure Cosmos DB

Les repositories sont un composant important de toute application mettant en jeu la persistance de données. Ils embarquent souvent des requêtes complexes, et pourtant ils font rarement l'objet de tests. Un test de repository doit s'adresser à une vraie base de données et devient donc un test d'intégration. Il faut placer la base dans un état connu, et postérieurement faire le ménage. Je montre ici comment le faire avec Cosmos DB. Nous verrons ensuite comment exécuter ces tests en intégration continue dans un pipeline Azure Devops.

Présentation d'Azure Cosmos DB

Cosmos DB est une base de données NoSql disponible sur le cloud Azure(1). Il s'agit d'une base PaaS (Platform as a Service) : vous n'avez pas à gérer de serveurs. Si on dispose d'une souscription Azure, il suffit de quelques clics pour créer un compte Cosmos DB prêt à l'emploi. Il est aussi possible de créer une souscription Azure d'essai qui donne un accès limité à Cosmos DB. Cosmos DB est multi-API. Chaque API définit une organisation des données. Les API disponibles sont : Core (SQL), MongoDB, Gremlin (Graphe), Cassandra, et Table. Nous utiliserons l'API Core, aussi appelée SQL API, qui permet d'accéder aux documents avec un langage inspiré de SQL. Un compte Cosmos DB contient une ou plusieurs bases de données dans lesquelles on peut créer des collections. Les données sont enregistrées sous forme de documents Json dans les collections.

Préparation de l'environnement

SDK Dotnet Core

Le code est écrit en C# et s'exécute sous .NET Core 2.2, il fonctionnera donc sur Windows, MacOS et de nombreuses distributions Linux. Installez le SDK .NET Core si vous ne l'avez pas déjà fait(2). Le code est transposable à d'autres plateformes pour lesquelles le client SQL API est disponible, notamment Java, Node et Python.

Cosmos DB

Pour la base Cosmos DB, le plus simple est d'utiliser l'émulateur Cosmos DB(3). Celui-ci peut être installé localement ou dans un conteneur Docker Windows. Les identifiants de connexion inclus dans le code sont ceux de l'émulateur, il n'y a donc pas à les modifier si vous utilisez l'émulateur.
Malheureusement, l'émulateur ne fonctionne que sous Windows. Pour tester avec d'autres OS, créez un compte gratuit sous Azure qui vous donnera accès à Cosmos DB. Pour utiliser une base Cosmos DB dans Azure, remplacez les identifiants dans le fichier appSettings.json par ceux de votre base.

Récupérer le code

Si vous ne souhaitez pas saisir le code en recopiant l'article, vous pouvez le récupérer sur Github : https://github.com/ervilho/testCosmosDb

(1) https://docs.microsoft.com/fr-fr/azure/cosmos-db/introduction

(2) Installer dotnet core SDK : https://dotnet.microsoft.com/download

(3) Installer cosmos DB emulator : https://docs.microsoft.com/fr-fr/azure/cosmos-db/local-emulator

Initialiser le projet

Si vous démarrez *from scratch*, vous pouvez initialiser le projet en lançant le script init.ps1 ou en tapant les commandes suivantes :

```
dotnet new xunit --name cosmosDbtests # Crée le projet
cd cosmosDbtests # va dans le dossier du projet

dotnet add package Microsoft.Azure.DocumentDb.Core # Ajoute le SDK CosmosDb
dotnet add package NFluent # Ajoute la librairie d'assertions

# Ajoute les librairies de Configuration dotnet
dotnet add package Microsoft.Extensions.Configuration
dotnet add package Microsoft.Extensions.Configuration.Json
dotnet add package Microsoft.Extensions.Configuration.EnvironmentVariables
```

Une fois le projet initialisé, on peut compiler et lancer les tests par la commande `dotnet test`

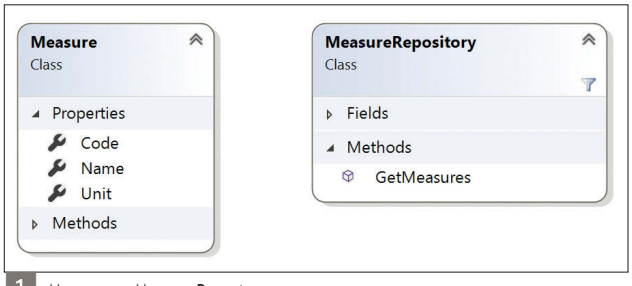
Le code à tester

Nous souhaitons tester une classe **MeasureRepository** qui comporte une méthode **GetMeasures.** Cette méthode envoie tous les objets **Measure** présents dans une collection CosmosDb.
Measure représente un type de mesure physique avec un code, un nom et une unité, par exemple { "h", "Hauteur", "m" } pour une hauteur en mètres. 1

Créer la classe Measure

Ajoutons le fichier Measure.cs avec :

```
public class Measure
{
  public string Code { get; set; }
  public string Name { get; set; }
```



1 *Measure et MeasureRepository*

```
    public string Unit { get; set; }
}
```

Créer le repository

Pour accéder aux données de la collection le repository utilise :

- Une instance de DocumentClient pour effectuer les requêtes ;
- L'Uri de la collection à utiliser.

GetMeasures utilise une requête SQL pour récupérer l'ensemble des mesures présentes dans la collection.

```
public class MeasureRepository {
    private readonly DocumentClient _documentClient;
    private readonly Uri _documentCollectionUri;

    public MeasureRepository(
        DocumentClient documentClient,
        Uri documentCollectionUri)
    {
        _documentClient = documentClient;
        _documentCollectionUri = documentCollectionUri;
    }

    public async Task<IEnumerable<Measure>> GetMeasures()
    {
        var query = _documentClient.CreateDocumentQuery(
            _documentCollectionUri,
            "SELECT * FROM c WHERE c.pk = 'measure' ORDER BY c.Code");
        var measures = await query.AsDocumentQuery().ExecuteNextAsync<Measure>();
        return measures;
    }
}
```

Écriture du test

La classe de test

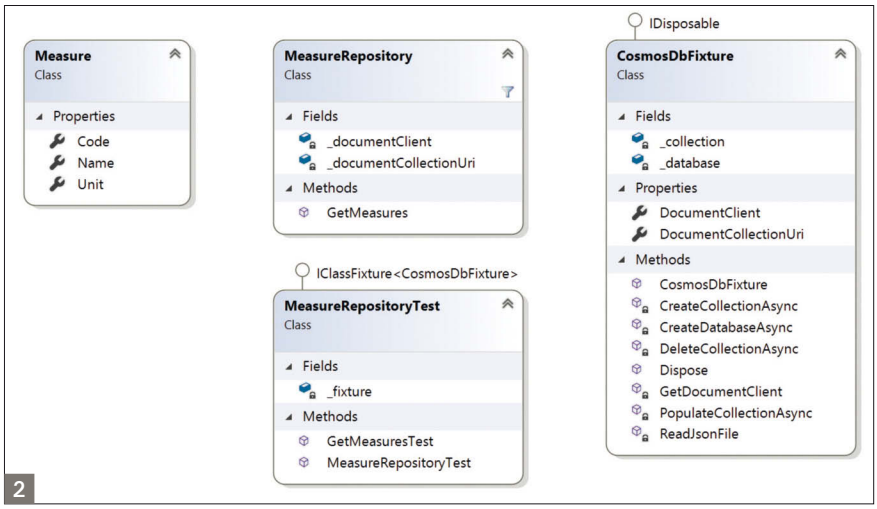
Nous allons écrire une classe de test **MeasureRepositoryTest** pour tester notre méthode.

```
public class MeasureRepositoryTest : IClassFixture<CosmosDbFixture>
{
    private readonly CosmosDbFixture _fixture;

    public MeasureRepositoryTest(CosmosDbFixture fixture)
    {
        _fixture = fixture;
    }
}
```

Pour initialiser la collection avant l'exécution nous allons créer une classe **CosmosDbFixture** et déclarer que la classe de test implémente **IClassFixture<CosmosDbFixture>**. L'interface **IClassFixture** ne comporte aucune méthode, mais indique à xUnit d'effectuer les opérations suivantes(4) :

- Avant l'exécution des tests, créer une instance de **CosmosDbFixture** et l'injecter dans le constructeur de **MeasureRepositoryTest** ;

(4) https://xunit.github.io/docs/shared-context#class-fixture



2

- Exécuter les tests ;
- Après l'exécution des tests, appeler la méthode **Dispose** de **CosmosDbFixture** si elle implémente **IDisposable**.

Dans le constructeur de CosmosDbFixture nous appellerons le code d'initialisation et dans Dispose, le code de nettoyage.

Nous conservons **fixture** dans un champ privé car elle expose le client Cosmos DB nécessaire pour le repository.

Nous nous retrouvons donc avec quatre classes :

- **Measure** représente une mesure issue de la base ;
- **MeasureRepository** la classe à tester ;
- **MeasureRepositoryTest** la classe contenant le test ;
- **CosmosDbFixture** qui gère l'initialisation et le nettoyage de la base de données. 2

La classe CosmosDbFixture

CosmosDbFixture va créer et initialiser la collection, et exposer l'Uri de la collection et l'instance de DocumentClient nécessaires au repository.

```
public class CosmosDbFixture : IDisposable
{
    private readonly string _database;
    private readonly string _collection;

    public CosmosDbFixture()
    {
        _database = "test";
        _collection = $"test_{Guid.NewGuid()}";
        DocumentCollectionUri = UriFactory.CreateDocumentCollectionUri(_database, _collection);

        DocumentClient = GetDocumentClient();
        CreateDatabaseAsync().Wait();
        CreateCollectionAsync().Wait();
        PopulateCollectionAsync().Wait();
    }

    public Uri DocumentCollectionUri { get; }

    public void Dispose()
    {
```

```
        DeleteCollectionAsync().Wait();
        DocumentClient.Dispose();
    }
}
```

Le rôle des méthodes est le suivant :

- GetDocumentClient : lit la configuration dans appSettings.json et instancie DocumentClient ;
- CreateDatabaseAsync : crée la base de données _database si elle n'existe pas encore ;
- CreateCollectionAsync : crée la collection _collection ;
- PopulateCollectionAsync : lit le fichier measures.json et écrit les documents dans la collection ;
- DeleteCollectionAsync : supprime la collection _collection ;
- Dispose : appelle DeleteCollectionAsync et dispose DocumentClient.

GetDocumentClient

Cette méthode instancie l'objet **DocumentClient** qui est notre point d'accès à Cosmos DB. Il sera exposé en tant que propriété par **CosmosDbFixture** et injecté dans le repository. Il permet aussi de créer la base et les collections dans les méthodes suivantes.

```
private DocumentClient GetDocumentClient()
{
    var builder = new ConfigurationBuilder()
        .AddJsonFile("appSettings.json")
        .AddEnvironmentVariables();
    var configuration = builder.Build();

    return new DocumentClient(new Uri(
        configuration["cosmosDbEndpoint"]),
        configuration["cosmosDbAuthKey"]);
}
```

Dans cette méthode nous créons d'abord un objet **builder** qui permet de récupérer la configuration à partir du fichier appSettings.json et des variables d'environnement. Si une variable d'environnement est présente sa valeur écrasera celle du fichier de configuration. Ceci va nous servir dans le cadre de l'intégration continue. Une fois l'objet **configuration** obtenu, nous créons un **DocumentClient** à partir des propriétés cosmosDbEndpoint et cosmosDbAuthKey de la configuration.

CreateDatabaseAsync

```
private async Task CreateDatabaseAsync()
{
    var database = new Database { Id = _database };
    var response = await DocumentClient.CreateDatabaseIfNotExistsAsync(database);
    Check.That(new[] { response.StatusCode }).IsOnlyMadeOf(HttpStatusCode.OK, HttpStatus
Code.Created);
}
```

Cette méthode crée la base Cosmos DB si elle n'existe pas. Nous appelons simplement la méthode **CreateDatabaseIfNotExistsAsync** du **DocumentClient**. La ligne « Check » est une assertion de la librairie NFluent qui vérifie que le résultat de la méthode est un code de succès (**OK** si la base existait déjà, **Created** si elle a été créée).

CreateCollectionAsync

Cette méthode crée la collection. Elle est un peu plus complexe que les précédentes car la collection nécessite de nombreux paramètres.

```
private async Task CreateCollectionAsync()
{
    var databaseUri = UriFactory.CreateDatabaseUri(_database);

    var pks = new Collection<string>(new[] { "/pk" });
    var partitionKeyDefinition = new PartitionKeyDefinition() { Paths = pks };
    var indexingPolicy = new IndexingPolicy(new Index[]
    {
        new RangeIndex(DataType.Number, -1),
        new RangeIndex(DataType.String, -1),
    });
    var collection = new DocumentCollection() {
        Id = _collection,
        PartitionKey = partitionKeyDefinition,
        IndexingPolicy = indexingPolicy };
    var options = new RequestOptions() { OfferThroughput = 400 };
    var response = await DocumentClient.CreateDocumentCollectionAsync(
                        databaseUri, collection, options);
    var code = response.StatusCode;
    Check.That(code).IsEqualTo(HttpStatusCode.Created);
}
```

partitionKeyDefinition définit la clé de partition pour la collection. Les collections Cosmos DB n'ont pas de limite de taille, mais le moteur les partitionne en partitions de 10 Go maximum. Ici nous déclarons que le partitionnement doit se faire sur la propriété « pk » des documents. Tous les documents qui ont la même valeur pour la propriété « pk » seront donc placés dans la même partition.
indexingPolicy définit la stratégie d'indexation des données pour la collection. Ici nous déclarons que les propriétés de type Number et String doivent être indexées en Range et sans limite de précision (-1) afin de permettre des recherches par des clauses WHERE avec des comparaisons, comme par exemple SELECT * FROM c WHERE c.Name > 'J'. Cette requête échouerait si String n'était pas un RangeIndex.
Nous créons ensuite un objet **DocumentCollection** qui déclare le nom de la collection (**Id**), la clé de partitionnement et la stratégie d'indexation.
Options déclare la capacité de la base : OfferThroughput est le nombre d'unités de requête (RUs) qui peuvent être traitées simultanément. Une unité de requête correspond à 1 ko en lecture et 0,2 ko en écriture. 400 est le nombre minimum de RUs possibles pour une collection, et on peut monter jusqu'à plusieurs centaines de milliers... mais les RUs servent aussi d'unité de facturation !
Finalement nous appelons **CreateDocumentCollectionAsync** avec tous les objets créés précédemment pour réaliser la création de la collection. La dernière ligne vérifie que la création a réussi.

ReadJsonFile

Cette méthode lit le fichier measures.json qui contient les données à insérer dans la collection.

```
private JArray ReadJsonFile()
{
  var serializer = new JsonSerializer();

  using (var sr = new StreamReader("./measures.json"))
  {
    using (var jsonTextReader = new JsonTextReader(sr))
    {
      return (JArray)serializer.Deserialize(jsonTextReader);
    }
  }
}
```

PopulateCollectionAsync

PopulateCollectionAsync appelle **ReadJsonFile** pour récupérer les données et les insère dans la collection.

```
private async Task PopulateCollectionAsync()
{
  var documents = ReadJsonFile();

  foreach (JObject document in documents)
  {
    var options = new RequestOptions() {
    PartitionKey = new PartitionKey((string)document.GetValue("pk")) };
    await DocumentClient.CreateDocumentAsync(
          DocumentCollectionUri, document, options);
  }
}
```

D'abord nous appelons **ReadJsonFile** pour récupérer la liste des documents sous la forme d'un tableau Json.
Nous itérons ensuite sur les documents pour les créer un par un. Pour chaque document, donc, nous créons un objet **PartitionKey** qui indique dans quelle partition le document doit être inséré. Cet objet est créé à partir de la propriété « pk » de notre document. Ensuite il n'y a plus qu'à appeler CreateDocumentAsync avec l'Uri de la collection, le document lui-même et les options.

Dispose et DeleteCollectionAsync

Ces méthodes servent au nettoyage après l'exécution des tests. **Dispose** est appelé par xUnit lorsque tous les tests ont été exécutés.

```
public void Dispose()
{
    DeleteCollectionAsync().Wait();
    DocumentClient.Dispose();
}

private async Task DeleteCollectionAsync()
{
    var response = await  DocumentClient.DeleteDocumentCollectionAsync(Document
CollectionUri);
    Check.That(response.StatusCode).Equals(HttpStatusCode.NoContent);
}
```

Dispose appelle **DeleteCollectionAsync**() qui supprime la collection que nous avons créée, puis **DocumentClient.Dispose** pour libérer les ressources retenues par **DocumentClient**.
DeleteCollectionAsync appelle simplement **DeleteDocumentCollectionAsync** pour supprimer la collection, puis vérifie le résultat.

Le corps du test

Finalement, nous pouvons écrire le test lui-même. Nous instancions le repository, appelons la méthode et vérifions le résultat.

```
[Fact]
public async Task GetMeasuresTest()
{
  // Arrange
  var expectedMeasures = new[] {
    new Measure{ Code = "I", Name= "Current", Unit="A" },
    new Measure{ Code = "U", Name= "Voltage", Unit="V" }
  };

  var subject = new MeasureRepository(
    _fixture.DocumentClient,
    _fixture.Database,
    _fixture.Collection);

  // Act
  var actual = (await subject.GetMeasures()).ToArray();

  // Assert
  Check.That(actual).HasSize(expectedMeasures.Length);
  for (int i = 0; i < expectedMeasures.Length; i++)
  {
    Check.That(actual[i]).HasFieldsWithSameValues(expectedMeasures[i]);
  }
}
```

Finalement, nous pouvons exécuter le test par dotnet test et il passe ! 3

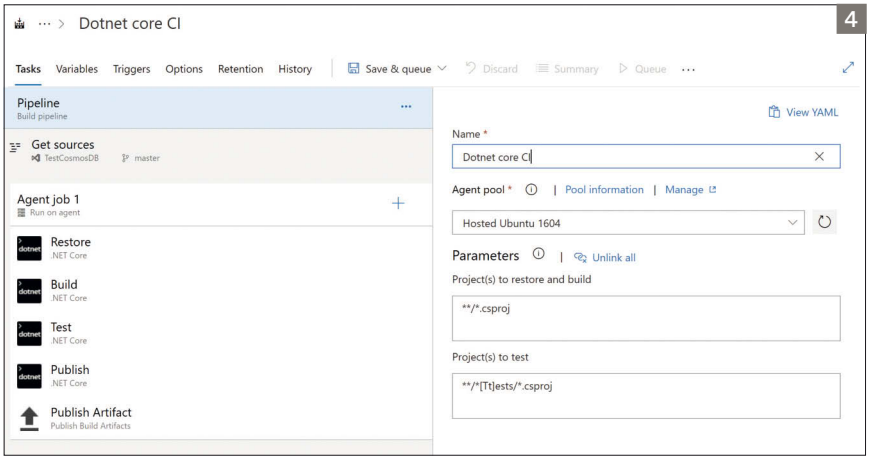
3

Intégration continue avec Azure Devops

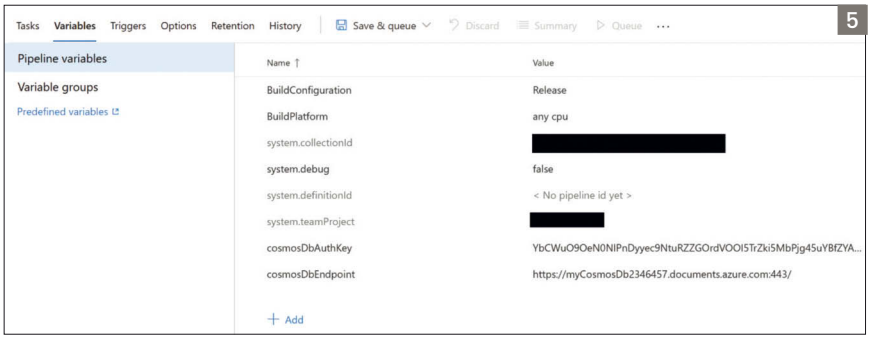
Maintenant que nous avons un test qui passe en local, il s'agit de le faire passer en intégration continue. Azure Pipelines, le composant « Build & Release » d'Azure DevOps, nous permet de créer facilement un pipeline d'intégration continue pour un projet dotnet core, qui va compiler le code et exécuter les tests.

Pour la base de données nous avons deux choix :

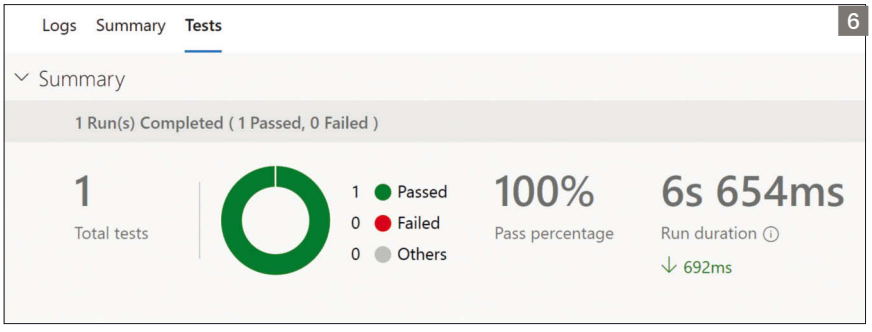
- Utiliser l'émulateur dans le cadre du build d'intégration continue ;
- Utiliser une véritable base Cosmos DB.



4



5



6

Utiliser l'émulateur dans le build d'intégration continue

Il existe une extension qui permet d'utiliser l'émulateur dans un build(5). L'extension permet d'ajouter la tâche « Run Azure Cosmos DB Emulator container » dans votre build. Comme l'émulateur a toujours les mêmes endpoint et authkey, il n'est même pas nécessaire de modifier la configuration.

Cependant :

- Comme l'émulateur ne fonctionne que sous Windows, le build doit forcément s'exécuter sur un agent Windows, ou un agent « Windows Container » ;
- Sur un agent « hosted », l'initialisation de l'émulateur est très longue, et même tombe en timeout. Il vaudrait donc mieux réserver ce cas à l'utilisation d'un agent privé.
- La tâche est en preview et n'est pas définitive.

Utiliser une vraie base Cosmos DB

Au lieu d'utiliser l'émulateur, on peut créer un compte Cosmos DB dédié à l'intégration continue. Comme en local, le test créera une base dans le compte si elle n'existe pas, puis des collections dans la base.

En fin de test, les collections créées sont supprimées par la méthode Dispose de **CosmosDbFixture**.

Pour mettre en œuvre cette solution :

- Créez un compte Azure Cosmos DB, notez-le endpoint et la clé.
- Dans Azure Devops, créez un pipeline de build avec le template « ASP.NET Core ». On obtient ceci : 4

Cliquez sur Variables et ajoutez les variables **cosmosDbEndpoint** et **cosmosDbAuthKey** avec pour valeur l'endpoint et la clé de votre compte Cosmos DB. 5

Exécutez le build. Durant l'exécution, les variables du pipeline sont accessibles en tant que variables d'environnement. CosmosDbFixture va utiliser ces variables d'environnement en priorité sur les valeurs définies dans le fichier appSettings.json, et le test passe. 6

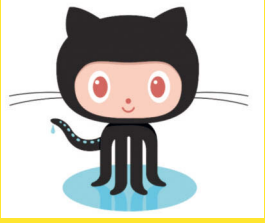
Conclusion

Cette méthode nous permet d'écrire et d'exécuter des tests automatisés en isolation sur les repositories Cosmos DB. Nous pouvons ainsi valider que nos requêtes sont correctes et assurer la non-régression.

La méthode est transposable à d'autres bases de données, à partir du moment où on est capable d'initialiser les tables assez rapidement pour pouvoir exécuter les tests.

N'hésitez plus et testez vos repositories !

(5) https://docs.microsoft.com/fr-fr/azure/cosmos-db/tutorial-setup-ci-cd



Les codes sources des articles sont disponibles sur programmez.com et GitHub.com/francoistonic

Tests de benchmark et automatisation : une véritable alternative aux développements pilotés par les tests (TDD) sur mainframe

Véronique Dufour-Thery
Vice-Présidente Europe du Sud & Moyen Orient
Compuware

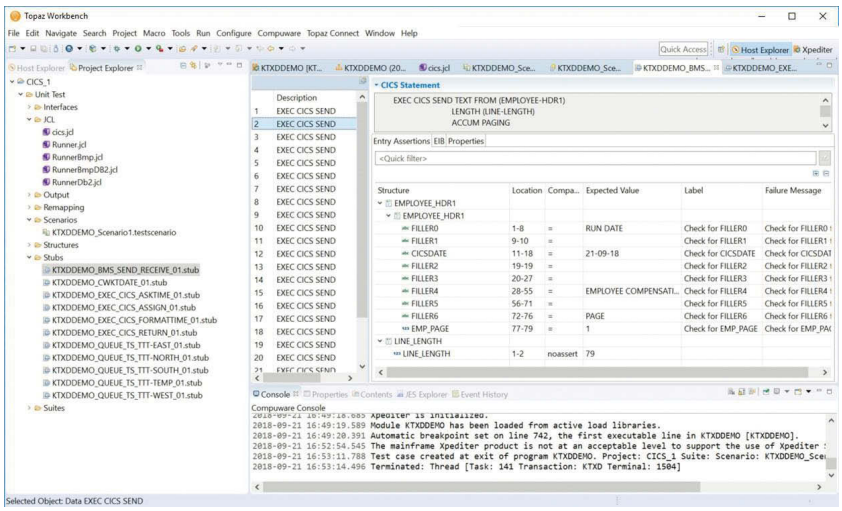
Sous la pression constante du marché et des entreprises, les développeurs doivent sans cesse accélérer la livraison d'applications sans pour autant faire l'impasse sur la qualité. La pratique de développement consistant à coder d'abord et tester ensuite n'est aujourd'hui plus une option. En réalité, plus vous tardez à tester un nouveau code généré, plus vous augmentez le risque de dysfonctionnements, limitez l'agilité globale de l'entreprise et impactez la satisfaction de vos clients.

Aujourd'hui, de plus en plus d'équipes DevOps utilisent une méthode de développement piloté par les tests, *Test-driven development* ou TDD, qui consiste à écrire des tests qui échouent avant de se lancer dans l'écriture d'un nouveau code applicatif. En plaçant la phase de tests en amont du développement et en mettant l'accent sur l'automatisation, les cycles de développement logiciels sont largement réduits et les goulets d'étranglement supprimés. Le temps de latence est également considérablement réduit, optimisant in fine le pipeline d'innovations digitales délivrées aux clients.

En revanche, le développement piloté par les tests ne fonctionne que si le nouveau code n'a peu ou pas de dépendances. Dans la plupart des grandes entreprises, les technologies stimulant l'engagement des clients, nouvellement codées, dépendent souvent d'applications matures et de données hébergées dans le mainframe dont la maintenance impacte la quasi-totalité du nouveau code. Dans ce contexte, l'approche de développement piloté par les tests ne peut s'appliquer.

L'alternative : les tests de benchmark

Pour que les équipes mainframe puissent tirer parti de la valeur d'une approche de développement piloté par les tests, une alternative est de créer un ensemble de tests de benchmark sur le code existant, avant de le faire évoluer, pour éviter des bugs inattendus. Une fois ces tests effectués, ils pourront ensuite être copiés et leurs assertions et variables modifiées, pour garantir que l'évolution du programme s'est correctement effectuée.



L'objectif ici est de changer les assertions du test pour qu'elles correspondent aux valeurs attendues une fois le code modifié. Le développeur pourra ensuite exécuter le code mis à jour et le vérifier en considérant les nouvelles assertions et en utilisant efficacement les scénarios de tests existants (intégrant les modifications d'assertions).

Cette approche s'apparente au développement piloté par les tests utilisés dans la maintenance du code. Elle fournit une base de référence de comportements existants et permet aux équipes de maintenance mainframe de vérifier le nouveau comportement du code plus rapidement, améliorant ainsi l'agilité de l'entreprise.

Automatiser les processus avec un framework de tests

Choisir un outil de tests automatisés idoine permet de créer rapidement un ensemble de tests unitaires et fonctionnels pour les tests de non-régression et de vérifier que le code fonctionne correctement après une modification. Lorsque des modifications de code entraînent l'échec des tests de non-régression, le développeur doit déterminer si les modifications apportées fonctionnent correctement, si les scénarios de tests sont corrects ou doivent être mis à jour, enfin, si des modifications supplémentaires du code sont nécessaires. Bien souvent, l'échec des tests de non-régression impliquent la nécessité d'une mise à jour.

En complément, un outil de couverture de code moderne aide à comprendre la quantité de code utilisée par les tests de non-régression. Plus le nombre de codes exécutés par les scénarios de tests est élevé, plus le taux d'échecs des modifications est important. Par ailleurs, la sécurité sera d'autant plus forte si le pourcentage de couverture de code est élevé.

Accroître la confiance des développeurs

A l'instar des bénéfices générés par le développement piloté par les tests, le filet de sécurité offert par les tests de non-régression avant toute modification de code sur le mainframe augmente considérablement la confiance des développeurs. Ces derniers pourront ainsi améliorer la qualité du nouveau code et augmenter la rapidité et l'efficacité avec laquelle il est livré, produisant de facto plus de valeur à l'entreprise et à ses clients. •

Abonnez-vous à Programmez! Abonnez-vous à Programmez! Abonne

Programmez! Le magazine des développeurs

Offres 2019

1 an ..... **59€**
11 numéros
+ 1 vidéo ENI au choix :
- Symfony 3*
  Développer des applications web robustes
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

2 ans ..... **89€**
22 numéros
+ 1 vidéo ENI au choix :
- Symfony 3*
  Développer des applications web robustes
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

** Offre limitée à la France métropolitaine*

Nos classiques

1 an 11 numéros	49€*
2 ans 22 numéros	79€*
Etudiant 1 an - 11 numéros	39€*

** Tarifs France métropolitaine*

Abonnement numérique

PDF ..... 35€
1 an - 11 numéros

Option : accès aux archives 10€

Souscription uniquement sur
www.programmez.com

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 : Symfony 3
☐ Vidéo : Raspberry Pi

PROG 227
Offre limitée, valable jusqu'au 29 mars 2019

☐ 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**

A DÉCOUVRIR D'URGENCE

NOUVEAU !

LE CADEAU GEEK IDÉAL

Découvrez l'âge d'or des micro-ordinateurs de 1973 à 2007

9,99 €
(+ 3 € de frais postaux)

116 pages - Format magazine A4

☐ **Découvrez l'âge d'or des micro-ordinateurs de 1973 à 2007** :
9,99 € (+3 € de frais postaux) = 12,99 €

Commande à envoyer à :
Programmez!
57 rue de Gisors - 95300 Pontoise

PROG 227

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

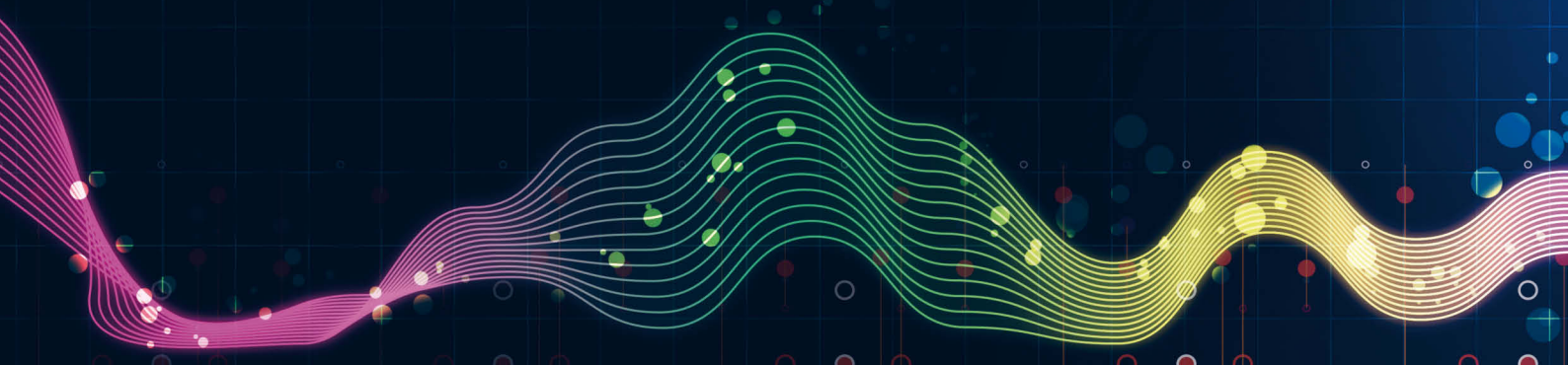
Code postal : Ville :

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !

commandez sur www.programmez.com

Assistant vocal : du DIY, du maker et les références du marché !



On promet un grand avenir aux assistants vocaux tels qu'Amazon Echo, Google Home et dans une moindre mesure Apple HomePod. On peut distinguer plusieurs couches : le matériel, la couche purement logicielle pour assurer la connexion aux services et traiter les interactions avec l'utilisateur, et, enfin, la couche de services pour pouvoir élargir les fonctionnalités de l'assistant.

François Tonic

A la rédaction, dans notre atelier, modestement appelé LabProgrammez!, nous avons monté et testé des shields, des hats, des capteurs pour construire notre propre assistant. Notre premier objectif était de connecter le matériel à un « moteur » d'assistant vocal et d'interagir, même simplement, avec lui. Naïvement, nous pensions ces prototypes rapides à réaliser. Mais au fur et à mesure de nos tests, nous avons rapidement compris que la réalité n'est pas aussi simple que les fournisseurs de matériels veulent nous le faire croire.

Finalement, c'est quoi un assistant ?

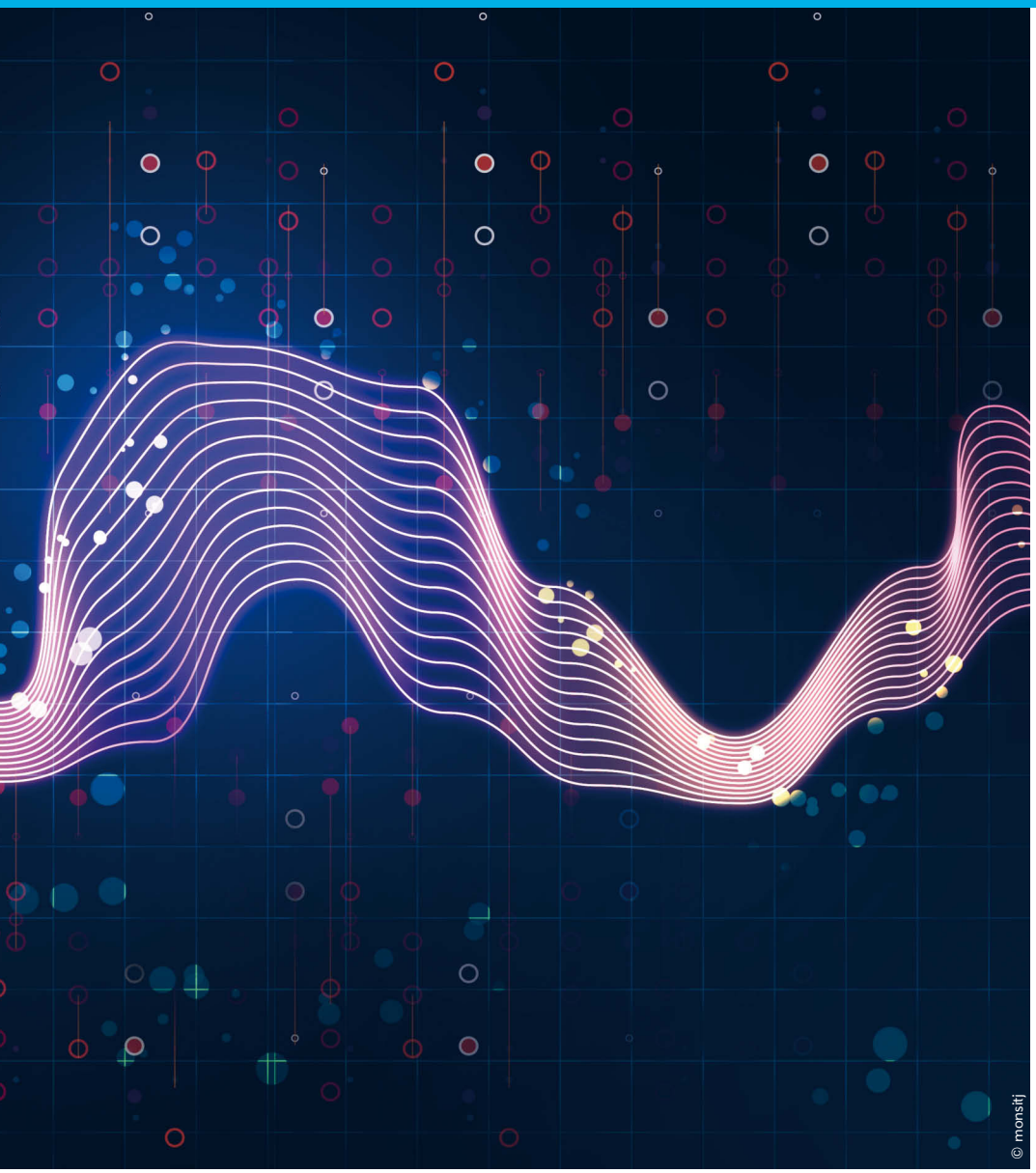
Si nous regardons les assistants vocaux sur le marché, ils ressemblent à des enceintes grandes, petits, moyennes, le plus souvent en forme de cylindre. A l'intérieur, de l'électronique ; tout particulièrement la connectivité réseau, un peu de logiciels avec un bon firmware !

Ce sont avant tout des assistants connectés mais pas à n'importe quoi. Typiquement, ce sera un service dédié : Amazon Alexa, Google Assistant, Microsoft Cortana, Apple Siri, etc. Tous ces moteurs sont disponibles via des assistants physiques. Ces moteurs, pas tous, sont utilisables avec des plateformes matérielles tierces, notamment Raspberry Pi et équivalent. Ce qui offre des perspectives intéressantes pour tous hackeurs / makers.

Ces assistants fonctionnent en mode connecté. Ils ont besoin du réseau et de l'accès aux services cloud pour pouvoir fonctionner : l'assistant envoie les requêtes, le moteur et les services reçoivent les requêtes et les traite, puis, le résultat est envoyé à l'assistant qui répond à l'utilisateur. Selon le service, les interactions seront plus ou moins nombreuses. C'est un des défauts de ces assistants car ils sont encore limités, c'est d'ailleurs une critique récurrente du HomePod d'Apple.

La qualité du réseau est un facteur important pour l'exécution rapide des requêtes. Si vous avez un assistant, vous en avez déjà fait l'expérience : latence importante entre la question et la réponse, échec de la demande ou perte de la requête, quelque part dans le réseau ou un serveur…

Une solution est d'utiliser un moteur totalement offline, c'est ce que propose par exemple le Français Snips, Open JARVIS.



L'un des intérêts de ces assistants est de pouvoir déployer des apps dédiées, par exemple sur Alexa, les skills. Ces « apps » sont disponibles sur la gamme du constructeurs (Echo, Echo Dot, Dot Spot). Pour les fournisseurs de ces assistants, le but est de proposer des modèles de développement les plus intégrés et les plus simples possibles. Si nous prenons l'exemple d'Alexa, l'objectif est que tout le monde puisse concevoir un skill sans code.

Aujourd'hui, quel est le marché réel des assistants ? Très difficile de le savoir car le succès de ces matériels reste limité.

Et pourquoi ne pas construire son assistant ?

Il est possible de créer son propre assistant vocal. Le principal reste identique :

- Une partie matérielle ;
- Les couches logicielles ;
- Un accès réseau pour les moteurs connectés ;
- Des applications / usages.

Naturellement, cette approche n'est pas aussi intégrée que d'acheter une solution prête à l'emploi. Mais on perd le côté amusant de le faire soi-même. D'autre part, nous n'avons pas accès aux mêmes niveaux de services mais c'est suffisant dans 99 % des usages que l'on veut en faire.

Avouons toutefois que ce n'est pas toujours facile. Le manque d'intégration se fait parfois cruellement ressentir quand l'assistant persiste à ne pas fonctionner pour d'obscures raisons. L'installation ressemble souvent à un parcours du combattant, même quand les docs sont disponibles. La documentation est l'un des gros problèmes des solutions matérielles disponibles sur le marché. Durant nos tests, nous devions parfois trouver des tutos et des docs tierces pour régler un problème d'installation ou de configuration. Régulièrement, on tombe sur une commande CLI avec tel outil qui n'est pas mentionné auparavant… Et on s'aperçoit que l'on a loupé une partie de l'installation… Sur la console développeur Google Assistant, on peut avoir un conflit ID entre deux projets. Pour régler le problème, on supprime un des deux projets pour lever l'erreur au démarrage du moteur sur notre Raspberry Pi + ReSpeaker…

Quelques solutions matérielles

Nous allons rapidement aborder quelques plateformes matérielles que nous avons testées :

- ReSpeaker de Seeed Studio avec Google Assistant et Snips ;
- Grasp.io ;
- Voice Hat.

Grasp.io n'est pas un véritable assistant vocal mais nous pouvons piloter à la voix des capteurs.

Après plusieurs jours de tests négatifs, nous n'avons pas retenu :

- Les shields Arduino, excepté le module Movi ;
- Matrix Voice / Creator.

Pour ces cartes, nous avons rencontré de multiples problèmes : instabilités des piles techniques, non reconnaissance des shields, impossibilité d'installation les dépendances nécessaires.

Cependant, vous n'avez pas besoin de ce matériel pour créer un Google Home. Une Raspberry Pi avec un micro et un haut-parleur suffit pour supporter et utiliser Google Assistant…

Un conseil : quand la plateforme fonctionne, n'installez pas des librairies et SDK non supportés officiellement par les constructeurs. Au mieux, vous déstabilisez la configuration, au pire, vous le cassez. Et vous serez obligé de remonter la configuration. Mais bon, on a beau le savoir, on continue à installer des libs particulièrement bancales. •

Sébastien Stormacq
AWS Developer Evangelist

Comment programmer ces apps ?

niveau 100

Les assistants vocaux sont de plus en plus populaires. Des millions d'Amazon Echo ont été vendus en 2018. C'est donc une opportunité pour les développeurs d'explorer un nouveau terrain de jeux et pour votre business de communiquer avec vos clients.

1

Amazon Alexa permet aux développeurs d'ajouter des fonctionnalités à celles fournies par Amazon. Par exemple, quand vous sortez un appareil Amazon Echo de sa boîte, une fois la configuration initiale effectuée, vous pouvez lui demander plein de choses, comme "Quel temps fait-il aujourd'hui à Toulouse ?" [1]. Grâce à un ensemble d'APIs en libre accès, tout le monde peut ajouter de nouveaux contenus à Alexa, en développant une skill. Une skill est une application vocale, servant à personnaliser votre appareil Echo. Elles sont disponibles dans la section Skill Store de l'application mobile Alexa [2] et sur le site web d'Amazon. Les clients peuvent utiliser une skill en le demandant explicitement à Alexa : "Alexa, ouvre inspire-moi", où "inspire-moi" est le nom d'invocation de la skill [3]. Alexa fera aussi de son mieux pour passer une requête à une skill quand elle ne connaît pas la réponse à une question. Par exemple : "Alexa, donne-moi une recette de poulet curry" sera envoyé à la skill Marmiton [4]. Cet article vous explique comment vous pouvez créer votre skill et la rendre accessible aux utilisateurs d'Alexa.

Il existe plusieurs sortes de skills Alexa, des plus simples qui vous donnent les nouvelles du jour (Flash Briefing), aux Custom Skills, qui permettent de gérer presque n'importe quel dialogue, en passant par les skills de domotique, qui permettent de contrôler des objets connectés comme des ampoules ou des thermostats [5] [6]. Cet article décrit comment créer une "Custom Skill", le type de skill le plus polyvalent.

Préparation : ce dont vous avez besoin

Pour développer et publier une skill, vous avez besoin d'un compte (gratuit) sur https://developer.amazon.com/alexa. Votre compte habituel amazon.fr sera reconnu. Ce compte vous permet d'accéder à la console de développement Alexa et à ask, la ligne de commande Alexa (CLI). La CLI a l'avantage de pouvoir être scriptée, les scripts peuvent faire l'objet de versions, etc. Comme il est aussi plus facile de communiquer des commandes par écrit plutôt que des copies d'écran, j'ai choisi de construire ce tutoriel autour de la ligne de commande. Si vous préférez suivre un tutoriel basé sur la console graphique, vous pouvez suivre les exemples, en anglais, donnés sur https://developer.amazon.com/alexa-skills-kit/tutorials.

Vous aurez aussi besoin d'un compte AWS (aws.amazon.com). Notez qu'il s'agit de comptes différents avec des noms d'utilisateurs et mots de passe potentiellement différents, il n'y a pas de Single Sign-On (SSO) entre les deux systèmes. Vous aurez besoin de la clé d'accès et la clé secrète d'un utilisateur IAM de votre compte AWS. Si vous utilisez déjà l'interface de ligne de commande AWS CLI (si vous avez une configuration dans ~/.aws/config), vous ne devez rien faire de spécial. Dans le cas contraire, les instructions, en anglais, sont disponibles dans la documentation de ASK [8].

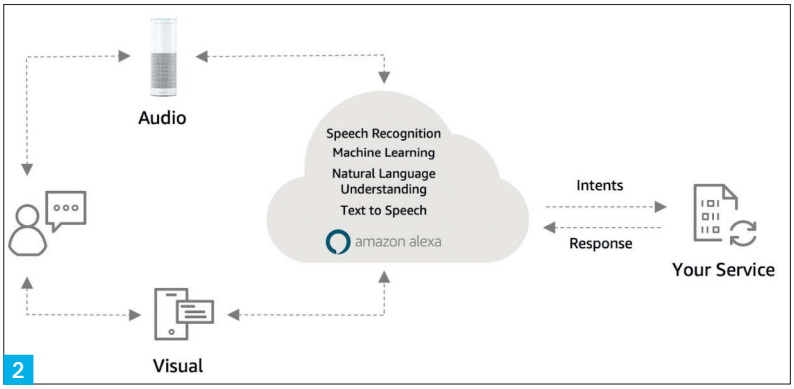
La ligne de commande est écrite en Node.js [9], vous devez donc installer node (LTS, 64 Bits pour votre OS) selon les instructions en ligne [11]. Sous Windows, vous aurez aussi besoin de Node.js Windows Building Tools [10] qui peuvent s'installer après node avec la commande : npm install -g --production windows-build-tools.

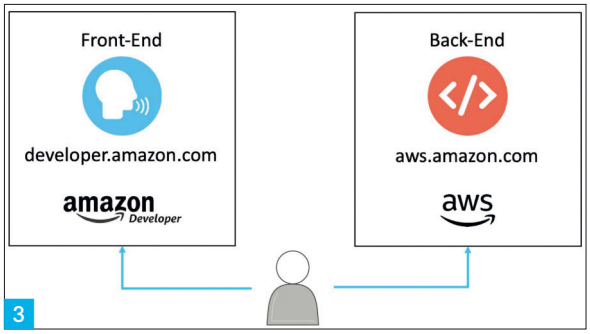
Enfin, git est nécessaire pour suivre les exemples de cet article. Vous pouvez suivre les instructions sur le site de Git [12].

La première étape de ce tutoriel sera donc d'installer et de configurer la ligne de commande. La documentation en ligne contient tous les détails, je ne donne que les 5 principales étapes, dans l'esprit TL;DR (Too Long; Didn't Read) :

Mais avant de commencer...

Il y deux principes fondamentaux à comprendre. Le premier est qu'Alexa vit dans le cloud. Les appareils Amazon Echo se déclenchent quand ils reconnaissent le mot "Alexa" (ou "Amazon" ou "Echo") et, à partir de ce moment, envoient les sons capturés vers les serveurs d'Amazon. Ces serveurs vont appliquer deux traitements sur ces sons. Le premier est appelé "Reconnaissance Automatique de la Parole" (Automatic Speech Recognition - ASR), il consiste à transformer des sons en texte. Par exemple, quand je dis "Alexa, lance Michael Jackson avec Spotify", l'ASR produit la suite de caractères "l-a-n-c-e..." etc. Le second s'appelle "Compréhension du Langage Naturel" (Natural Language Understanding - NLU), il va transformer la suite de caractères ci-dessus en intentions et entités. Dans l'exemple précédent, mon intention est de jouer de la musique ("PlayMusic" par exemple) et les entités sont "Michael Jackson" (Artiste) et "Spotify" (Source). Le résultat du NLU est un document JSON, qui ne contient plus de référence directe à la phrase capturée par l'appareil. 1





Une fois que le NLU a terminé son analyse, un routeur va envoyer une requête à un service qui sera capable de répondre à la question ou la commande, Spotify dans mon exemple ci-dessus, un service de météo quand vous demandez les prévisions météo, etc. S'il s'agit d'une commande destinée à une skill ("Alexa, ouvre inspire-moi"), la requête JSON sera envoyée à un web service écrit et hébergé par le développeur de la skill. Tout ce que ce web service doit faire est analyser la requête, faire ce qui est nécessaire pour collecter les données de la réponse (appeler une base de données, appeler un algorithme de prédiction, appeler un autre service web, etc.) et répondre avec un document JSON qui contient le texte que Alexa devra "lire" à l'utilisateur d'une part et, d'autre part, des éléments graphiques à afficher dans l'application Alexa ou sur l'écran de l'appareil (pour les Echo Show ou Echo Spot).

L'ensemble du processus ne prend que quelques centaines de millisecondes et est illustré ci-dessous. 2

Le second principe à comprendre est que les algorithmes d'intelligence artificielle qui permettent la reconnaissance et la compréhension de la parole (ASR et NLU) sont de type d'apprentissage supervisé [7]. C'est-à-dire qu'ils ont besoin de données pour pouvoir extrapoler le résultat d'une analyse. En tant que développeur de skills, vous devez donc donner à Alexa la liste des intentions comprises par votre skill, des exemples de phrases qui peuvent déclencher ces intentions, ainsi que la liste des entités incluses dans vos intentions, leur type et leurs valeurs. Ces données s'appellent le Modèle de Conversation (Language Model). Il fait partie intégrante des livrables que vous devez créer lorsque vous créez une skill pour Amazon Alexa.

En tant que développeur, vous devrez donc écrire trois groupes de livrables :

- Un web service pour recevoir et répondre aux requêtes envoyées depuis Alexa ;
- Le modèle de conversation (liste des intentions, des entités, et les exemples de phrases associés) ;
- Les méta données requises pour lister votre skill sur le Skill Store d'Amazon (un nom, une icône, une description etc.).

Pour le besoin de ce tutoriel, nous allons utiliser AWS pour notre code mais ce n'est pas une obligation. Alexa enverra ses requêtes à n'importe quel web service, du moment que vous supportez HTTPS avec un certificat valide. Vous pouvez donc choisir d'héberger votre skill où vous le souhaitez.

Le modèle de conversation et les métadonnées de la skill seront quant à eux entrés dans la console de développement Alexa (developer.amazon.com) ou avec la ligne de commande (ASK CLI), comme illustré ci-dessous. 3

Pour ce tutoriel, nous allons utiliser AWS Lambda pour héberger le code de la skill. AWS Lambda vous offre un environnement d'exécution de votre code sans que vous ayez à installer ou maintenir de machines virtuelles. Vous déposez votre code et AWS Lambda se charge du reste. AWS Lambda est facturé $0,2 par million de requêtes, le premier million chaque mois est offert. Si vous n'avez pas d'autres utilisation de Lambda, vous ne payerez donc pas les requêtes faites dans le cadre de ce tutoriel.

Maintenant que vous maîtrisez les principes, mettons les mains dans le code.

Etape 0 : installation et configuration de ASK CLI

Une fois `node` installé, ouvrez un terminal et tapez `npm install -g ask-cli` pour installer la ligne de commande `ask`.

La première fois que vous installez `ask`, vous devez initialiser votre environnement avec les détails de votre compte AWS et Développeur Amazon : `ask init`. La commande va vous demander votre profile AWS et ouvrir votre navigateur par défaut pour vous connecter sur votre compte Amazon. Si vous êtes sur une machine sans navigateur ou sans interface graphique, une instance EC2 par exemple, vous pouvez utiliser l'option ci-après: `ask init --no-browser`

Etape 1 : Hello World

La command `ask new` vous permet de créer votre première skill, en copiant un exemple depuis GitHub. La commande vous demande de choisir l'environnement d'exécution de votre skill (NodeJS ou Python), puis vous propose une liste d'exemples à partir desquels vous pouvez démarrer. Pour ce tutoriel, choisissez NodeJS et "Hello World".

```
$ ask new
? Please select the runtime Node.js V8
? List of templates you can choose Hello World
? Please type in your skill name: skill-sample-nodejs-hello-world
Skill "skill-sample-nodejs-hello-world" has been created based on the chosen template
$ cd skill-sample-nodejs-hello-world/
$ ls
$ LICENSE.txt README.md hooks instructions lambda models skill.json
```

La commande copie tous les fichiers requis et crée un dossier pour votre projet, avec la structure suivante :

```
.
├── .ask
├── hooks
├── lambda
├── models
└── skill.json
```

Vous pouvez explorer ces fichiers avec votre éditeur de code favori, la structure du dossier est la suivante :

- .ask – un dossier caché qui contient le fichier de configuration de ASK CLI : config . Ne touchez pas à ce fichier :-)
- hooks – un dossier qui contient les scripts (optionnels) à exécuter avant et après un déploiement. La commande installe deux scripts par défaut : post_new_hook et pre_deploy_hook.
- lambda – un dossier qui contient le code source de la fonction.
- models – un dossier qui contient les modèles de conversation de votre skill, un par langue. La commande propose un modèle pour l'anglais (US) (en-US.json). Nous ajouterons le français plus tard dans ce tutoriel.
- skill.json – un fichier qui contient les métadonnées de votre skill, requises pour publication sur le Skill Store, entre autres.

Etape 2 : localisation en Français

A ce stade, nous pourrions déjà déployer la skill en anglais, mais ajoutons d'abord le support pour le français. Vous devez changer trois fichiers pour traduire l'exemple en français :

- Le modèle de conversation, pour donner à Alexa les exemples de phrases et les valeurs des entités en français ;
- Les métadonnées de la skill telles que son nom, la description, les phrases d'exemple qui seront proposées à vos clients ;
- Le code de la fonction Lambda, pour les réponses envoyées, qui seront lues avec la voix d'Alexa.

en-US.json

Renommez (ou copiez) le fichier qui contient le modèle anglais US (en-US.json) et créez un fichier pour le Français de France : fr-FR.json. Éditez le nom d'invocation de la skill (les accents sont importants) et les phrases d'exemples pour les intentions non-standard (HelloWorldIntent). Le fichier complet doit ressembler à ceci (en gras les parties modifiées).

```
{
 "interactionModel": {
  "languageModel": {
   "invocationName": "***démo programmer***",
   "intents": [
    {
     "name": "AMAZON.CancelIntent",
     "samples": []
    },
    {
     "name": "AMAZON.HelpIntent",
     "samples": []
    },
    {
     "name": "AMAZON.StopIntent",
     "samples": []
    },
    {
     "name": "HelloWorldIntent",
     "samples": [
      "***bonjour***"
     ]
    }
   ]
  }
 }
}
```

skills.json

Sous “locale”, vous trouverez un bloc d'information en anglais, remplacez-le par un bloc pour le français (fr-FR) :

```
"fr-FR": {
      "summary": "***Description courte***",
      "examplePhrases": [
       "***Alexa ouvre démo programmez***",
       "***bonjour***",
       "***aide***"
      ],
      "name": "skill-sample-nodejs-hello-world",
      "description": "***Description longue***"
    }
```

index.js

Il reste ensuite à éditer le code source de la fonction Lambda qui va envoyer les réponses à Alexa. Chaque phrase en anglais doit être traduite (lignes 11, 16, 27, 31, 42, 47, 59 & 63). Je ne donne qu'un exemple ici, le code complet est disponible sur GitHub (https://github.com/sebsto/programmez.com-alexa)

```
const speechText = 'Bienvenue sur Alexa, vous pouvez dire bonjour.';
```

Etape 3 : déploiement

Vous pouvez maintenant déployer votre skill pour la tester. La commande ask deploy va déployer à la fois votre skill sur developer.amazon.com et votre fonction Lambda sur AWS.

```
$ ask deploy
Profile for the deployment: [default]
------------------- Create Skill Project -------------------
Skill Id: amzn1.ask.skill.aaaaaaaa-bbbb-cccc-dddd-000000000000
Skill deployment finished.
Model deployment finished.
[Warn]: No runtime and handler settings found for alexaUsage "custom/default" when creating
 Lambda function. CLI will use "nodejs8.10" and "index.handler" as the Runtime and Handler
 to create Lambda. You can update the runtime and handler for the target Lambda in the project
 config and deploy again if you want to set differently.
Lambda deployment finished.
Lambda function(s) created:
[Lambda ARN] arn:aws:lambda:eu-west-1:9999999999:function:ask-custom-skill-sample-nodejs
-hello-world-default
[Info]: No in-skill product to be deployed.
Your skill is now deployed and enabled in the development stage. Try simulate your Alexa skill
 skill using "ask dialog" command.
```

Vous pouvez ignorer l'avertissement ([Warn]). Notez l'ARN de la fonction Lambda qui a été créée pour vous. Vérifions le déploiement de la skill.

```
$ ask api list-skills
{
  "skills": [{
    "apis": ["custom"],
    "lastUpdated": "2019-01-29T23:13:55.758Z",
    "nameByLocale": {
      "fr-FR": "skill-sample-nodejs-hello-world"
    },
    "publicationStatus": "DEVELOPMENT",
    "skillId": "amzn1.ask.skill.aaaaaaaa-bbbb-cccc-dddd-000000000000",
    "stage": "development"
  }]
}
```

Vérifions le déploiement de la fonction lambda.

```
$ aws lambda get-function --function-name ask-custom-skill-sample-nodejs-hello-world-default
{
 "Configuration": {
  "FunctionName": "ask-custom-skill-sample-nodejs-hello-world-default",
  "FunctionArn": "arn:aws:lambda:eu-west-1:9999999999:function:ask-custom-skill-sample
-nodejs-hello-world-default",
  "Runtime": "nodejs8.10",
  "Role": "arn:aws:iam::9999999999:role/ask-lambda-skill-sample-nodejs-hello-world",
  "Handler": "index.handler",
  "CodeSize": 165031,
  "Description": "",
  "Timeout": 8,
  "MemorySize": 128,
  "LastModified": "2019-01-29T23:13:54.850+0000",
  "CodeSha256": "XOZk0LdCMOWmwn7grVkIN1NxYXR0LbrxRFUF9lJwDEs=",
  "Version": "$LATEST",
  "TracingConfig": { "Mode": "PassThrough" },
  "RevisionId": "b5528d44-1905-476f-9296-0ad43e7e7872"
 },
 "Code": {
 "RepositoryType": "S3",
 "Location": "https://awslambda-eu-west-1-tasks.s3.eu-west-1.amazonaws.com/snapshots/
[redacted]"
 }
}
```

Tout semble correct. Il est maintenant temps de tester la skill. Nous allons montrer deux techniques de tests en ligne de commande. Vous pouvez aussi utiliser le simulateur dans la console de développement Alexa (https://developer.amazon.com/alexa) ou un appareil Amazon Echo enregistré sur le même compte Amazon.
La commande ask dialog permet de simuler un dialogue à haut niveau. La commande ask simulate permet d'inspecter le détail des requêtes et des réponses reçues et envoyées par votre fonction Lambda. Les deux commandes attendent le paramètre --locale pour explicitement indiquer la langue dans laquelle vous souhaitez réaliser le test.

```
$ ask dialog --locale fr-FR
User > lance démo programmer
Alexa > Bienvenue sur Alexa, vous pouvez dire bonjour.
User > bonjour
Alexa > Bonjour !
User >
```

Tapez CTRL-C pour sortir.

```
$ ask simulate --locale fr-FR --text "lance démo programmer"
⊠ Simulation created for simulation id: 29a8ccfd-f952-4f88-8c2a-a0ed61db10a6
```

La commande ask simulate vous donne un document JSON en réponse qui contient le détail de la simulation. Vérifiez la présence des champs suivants :

```
"status": "SUCCESSFUL",
result –> skillExecutionInfo –> invocations –> invocationRequest –> body
result –> skillExecutionInfo –> invocations –> invocationResponse –> body
```

Enfin, il est conseillé de tester sur un appareil Amazon Echo pour être le plus proche possible de l'expérience que vous proposerez à vos clients. Aucune action additionnelle n'est nécessaire pour tester sur votre appareil. Si l'appareil est enregistré sur votre compte de développement, vous pouvez des á présent dire : "Alexa, ouvre démo programmez".
Félicitation, vous avez écrit votre première skill pour Alexa. Les étapes suivantes vont vous permettre d'aller un peu plus en détails.

Etape 4: slots et résolution d'entités

Il est courant dans un dialogue de poser une question et d'agir en fonction de la réponse fournie. Pour capturer des réponses dans votre skill, il est nécessaire d'abord de poser une question et de laisser le temps à l'utilisateur de répondre.
Modifions donc la réponse ci-dessus, pour inclure une question. En ajoutant un appel à reprompt(), le SDK dit à Alexa de laisser la session ouverte pour capturer une réponse. Dès la réponse entendue, Alexa enverra une seconde requête à votre fonction Lambda. Cette requête contiendra l'élément de réponse capturé. En cas de non réponse, Alexa attendra 8 secondes et reposera la question une seconde fois. Après le second délai de 8 secondes, Alexa fermera automatiquement la session si l'utilisateur ne fournit pas de réponse. Voici le nouveau code de HelloWorldIntentHandler, les parties en gras indiquent les modifications:

```
handle(handlerInput) {
  const speechText = 'Bonjour, comment vous appelez-vous ?';
  const reprompt =  'Quel est votre nom ?';

  return handlerInput.responseBuilder
    .speak(speechText)
    .reprompt(reprompt)
    .withSimpleCard(CARD_TITLE, speechText)
    .getResponse();
}
```

A cette question, il est probable que l'utilisateur réponde "je m'appelle Sébastien" ou juste "Sébastien" ou d'autres variations autour

de ce thème. Pour capturer cette réponse, nous allons ajouter une intention dans notre modèle de conversation. Cette intention, à la différence de HelloWorldIntent, contient une variable, aussi appelée "entité" ou "slot". Comme toute variable, elle a un nom (first_name) et un type AMAZON.Firstname. Vous pouvez choisir un des nombreux type prédéfinis par Amazon (date, time, artiste, ville, prénom etc) [13] ou créer votre propre type avec une liste de valeurs spécifiques. Dans fr-FR.json, ajoutez :

```
{
  "name": "MyNameIsIntent",
  "slots": [{
    "name": "first_name",
    "type": "AMAZON.FirstName"
  }],
  "samples": [
    "c'est {first_name}",
    "{first_name}",
    "je m'appelle {first_name}"
  ]}
}
```

Enfin, nous allons ajouter un handler pour cette intention dans notre fonction Lambda. Dans index.js, ajoutez :

```
const MyNameIsIntentHandler = {
 canHandle(handlerInput) {
  return handlerInput.requestEnvelope.request.type === 'IntentRequest'
   && handlerInput.requestEnvelope.request.intent.name === 'MyNameIsIntent';
 },
 handle(handlerInput) {
  // dans la vraie vie, attention aux 'undefined' ! Vérifiez que first_name existe dans l'objet.
  const firstname = handlerInput.requestEnvelope.request.intent.slots.first_name.value;
  const speechText = `Bienvenue ${firstname}, ravie de vous rencontrer`;

  return handlerInput.responseBuilder
   .speak(speechText)
   .withSimpleCard(CARD_TITLE, speechText)
   .getResponse();
 },
};
```

Ce handler lit la requête reçue pour trouver le prénom donné par l'utilisateur (handlerInput.requestEnvelope.request.intent.slots.first_name.value) et construit une réponse personnalisée. Pour votre information, jetez un coup d'oeil à la requête que Alexa enverra à votre fonction Lambda quand l'utilisateur répondra "je m'appelle Sébastien" :

```
"request": {
  "type": "IntentRequest",
  "requestId": "amzn1.echo-api.request.123",
  "timestamp": "2019-01-30T10:47:32Z",
  "locale": "fr-FR",
  "intent": {
    "name": "MyNameIsIntent",
    "confirmationStatus": "NONE",
    "slots": {
      "first_name": {
        "name": "first_name",
        "value": "sébastien",
        "confirmationStatus": "NONE",
        "source": "USER"
      }
    }
  }
}
```

Nous sommes prêts pour tester nos modifications, pour ce faire, il faut redéployer la skill et la fonction lambda :

```
$ ask deploy
Profile for the deployment: [default]
------------------- Update Skill Project -------------------
Skill Id: amzn1.ask.skill.aaaaaaaa-bbbb-cccc-dddd-000000000000
Skill deployment finished.
Model deployment finished.
Lambda deployment finished.
Lambda function(s) updated:
[Lambda ARN] arn:aws:lambda:eu-west-1:9999999999:function:ask-custom-skill-sample-nodejs-hello-world-default
[Info]: No in-skill product to be deployed.
Your skill is now deployed and enabled in the development stage. Try simulate your Alexa skill
 skill using "ask dialog" command.
```

Puis tester avec le simulateur :

```
$ ask dialog --locale fr-FR
User > lance démo programmer
Alexa > Bienvenue sur Alexa, vous pouvez dire bonjour.
User > bonjour
Alexa > Bonjour, comment vous appelez-vous ?
User > Sébastien
Alexa > Bienvenue Sébastien, ravie de vous rencontrer.
```

Etape 5 : tests

La différence entre un bon développeur et un expert : l'expert écrit des tests pour tout. Avoir un banc de tests unitaires et de tests d'intégration complets vous permet de gagner en confiance quand vous faites des changements, en tournant vos tests à chaque changement ou à chaque build, vous pouvez être confiant que vos changements n'ont pas introduit d'effets de bord.

Les tests unitaires vont tester vos fonctions NodeJS en local, sans besoin de déployer sur Lambda. Vous pouvez utiliser n'importe quel framework de test, selon votre langage. Pour ce tutoriel, j'ai choisi Mocha [14]. Mocha va exécuter une suite de tests présents dans un répertoire dédié. L'idée est la suivante :

- Capturer les requêtes envoyées à votre fonction Lambda lors de tests avec le simulateur. Vous pouvez voir les requêtes en utilisant la commande ask simulate, ou en utilisant le simulateur web dans la console de développement ou, enfin, dans les logs de votre fonction Lambda (disponibles dans le service CloudWatch Logs), si vous imprimez les requêtes depuis votre code. Cette étape ne doit être réalisée qu'une seule fois.
- Dans votre code de test, vous importez votre skill, et le fichier JSON qui contient la requête.

- Dans la fonction before() de votre test, vous appelez votre fonction lambda en lui passant la requête JSON et vous capturez la réponse.
- Vérifiez vos assertions sur la réponse (est-ce que la session reste ouverte ? est-ce que certains mots clés sont présents ? Etc.)

Le résumé du code ci-dessous, vous trouverez le code complet sur GitHub [15]

```
// import de la requête à tester
import r from './request/launch_request.json';
const request: RequestEnvelope = <RequestEnvelope>r;

// import de la skill
import { handler as skill } from '../src/index';

// pour stocker la réponse
let skill_response: ResponseEnvelope;

describe('Hello World test : LaunchRequest', function () {

  // before() est exécuté une seule fois, avant tous les it()
  before(() => {
    return new Promise((resolve, reject) => {
      // j'appelle le handler de la skill
      skill(request, null, (error, responseEnvelope) => {
        // je capture la réponse
        skill_response = responseEnvelope;
        resolve();
      });
    });
  });

  // dans chaque test, je vérifie un ensemble d'assertion sur la réponse
  it('it responds with a valid response structure', () => {
    expect(skill_response).to.have.property("version");
    expect(skill_response.version).to.be.equal("1.0");
    expect(skill_response).to.have.property("response");
  })

});
```

Finalement pour lancer le test depuis un terminal : "mocha test/*.js"
Il existe aussi des outils tiers, spécifiquement pensés pour tester des skills Alexa, Bespoken par exemple [16]. Bespoken permet d'écrire des tests unitaires sans écrire de code, juste en écrivant un scénario de tests et vos assertions en YAML, comme montré dans l'exemple ci-dessous :

```
- test: "Sequence 01. Test scenario: launch request, no further interaction."
- LaunchRequest: # LaunchRequest is not an utterance but a request type
 - response.outputSpeech.ssml: "Here's your fact"
 - response.card.type: "Simple"
 - response.card.title: "Space Facts"
 - response.card.content: "/.*/" # Regular expression indicating any text will match
```

Vous pourrez en apprendre plus sur Bespoken Unit Testing en lisant leur documentation [17]. Bespoken offre également un mode proxy [18] qui permet de tester votre code sur votre laptop depuis un appareil Alexa, sans passer par Lambda.

Aller plus loin

Vous venez d'écrire, de déployer et de tester une skill en français sur Alexa. Vous souhaitez la publier ? Pour rendre votre skill disponible dans l'application Alexa, vous devez la soumettre auprès de l'équipe de certification. Prenez le temps de lire les conseils avant la soumission [19]. Un premier conseil : veillez à ne pas laisser de fautes d'orthographe dans la description de la skill et de vous assurer que les phrases d'exemples fournies dans skills.json sont présentes dans votre modèle de conversation (fr-FR.json).
Vous pouvez soumettre votre skill avec la commande :

```
ask api submit --skill-id <votre skill_id>
```

Où trouver de l'aide ?

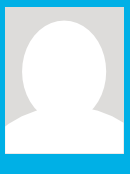
Si vous êtes coincés pendant le développement, pas de panique, des experts d'Amazon et la communauté de développeurs vous accompagnent tout au long de votre développement.

- Le forum des développeurs Alexa : https://forums.developer.amazon.com/spaces/165/index.html (utilisez le menu "Space" en haut à droite pour accéder à différentes categories)
- Le canal slack : https://amazonalexa.slack.com/
- StackOverflow, avec les tags "alexa", "alexa-skill" ou "alexa-skills-kit" : https://stackoverflow.com/

Quoi que vous codiez, codez-le bien.

Liens cite dans ce tuto

[1] https://www.amazon.fr/gp/help/customer/display.html?nodeId=201602230
[2] https://www.amazon.fr/b?node=13944548031
[3] https://www.amazon.fr/Sebastien-Stormacq-Inspire-Moi/dp/B07C76DZV9
[4] https://www.amazon.fr/Marmiton/dp/B07B9H4YVQ/
[5] https://www.amazon.fr/Philips-Hue/dp/B078XCX8NG
[6] https://www.amazon.fr/IKEA-of-Sweden-AB-TRÅDFRI/dp/B076SYBVKR/
[7] https://fr.wikipedia.org/wiki/Apprentissage_supervisé
[8] https://developer.amazon.com/docs/smapi/manage-credentials-with-ask-cli.html#create-aws-credentials
[9] https://nodejs.org/fr/
[10] https://nodejs.org/fr/download/
[11] https://www.npmjs.com/package/windows-build-tools
[12] https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Installation-de-Git
[13] https://developer.amazon.com/docs/custom-skills/slot-type-reference.html
[14] https://mochajs.org/
[15] https://github.com/sebsto/programmez.com-alexa/tree/master/lambda/custom/test
[16] https://github.com/bespoken/bst
[17] https://read.bespoken.io/unit-testing/getting-started/
[18] https://read.bespoken.io/cli/commands/#proxy
[19] https://developer.amazon.com/docs/devconsole/test-and-submit-your-skill.html
[28] https://developer.amazon.com/docs/smapi/proactive-events-api.html



Laurent Vincent
IBM Cloud & Watson Platform Solution Architect, Europe
Retail Solution Architect
Business Development and ISVs
Business Partner Solution Hub

Prise en main de Watson Assistant

niveau 100

Le sujet de l'assistant vocal génère beaucoup de fantasmes ou de déceptions.

De nombreux films d'anticipation ou de science-fiction font intervenir des assistants vocaux sous des formes variées :

- En 1968, HAL 9000, l'IA en charge des fonctions et de la maintenance du vaisseau spatial https://fr.wikipedia.org/wiki/HAL_9000 dans « 2001 l'odyssée de l'espace » . Il sera capable de participer avec fluidité à une conversation. Certains y voient une référence à un grand constructeur dans l'informatique, si on remplace les lettres de son nom par la lettre qui la suit dans l'alphabet (HAL -> IBM).
- Dans une version plus glamour, nous avons le film « Her » de 2013 où le personnage tombe amoureux de son Assistante Vocale, https://fr.wikipedia.org/wiki/Her .
- TARS dans Interstellar en 2014 https://fr.wikipedia.org/wiki/Interstellar .

Nous pourrions donner de nombreux autres exemples, mais aujourd'hui, prenons un peu de temps pour comprendre ce qu'est un Assistant, ses cas d'usages et hasardons-nous dans la création de notre premier assistant vocal.

Qu'est-ce qu'un assistant vocal ?

Simplifions le propos. Un assistant vocal est toute technologie portée par un artéfact physique (principalement des objets connectés) ou virtuel (un programme) pilotée via la voix dont l'objectif est de vous simplifier la vie en effectuant des tâches ou des services simples. Sa force est de comprendre le langage naturel pour vous aider à trouver une réponse à une question, ajouter une réunion à votre agenda, vous donner la météo ou allumer la lumière…

« Assistant » est un terme générique qui a une variété de dénominations :

- **Assistant personnel :** embarqué dans des progiciels, il vous assiste dans des tâches simples. Il utilise le langage naturel et peut faire des recherches en ligne ou dans une base de connaissances. Nous pouvons interagir avec lui par la voix ou en tapant du texte.
- **Smart Assistant :** ce terme référence un objet physique qui comme précédemment offre des services. Pour le grand public, c'est ce que nous appelons smart speaker ou enceinte connectée. Nous trouvons de nombreux acteurs dans ce domaine : Amazon Echo, Google Home, Apple Homepod.
- **Chatbot :** un chatbot utilise principalement des échanges textuels. Il peut simuler une conversation humaine dans un contexte limité. Les réponses de ces chabots peuvent être enrichies par des contenus media tels que des photos, des vidéos ou l'affichage d'une carte pour localiser un lieu. Beaucoup d'entreprises utilisent ce type d'agent au sein des réseaux sociaux pour interagir avec leurs clients pour promouvoir un produit ou un service.
- **Assistant vocal :** le sujet du jour, ou le mot clé est la voix. L'assistant vocal est un assistant digital qui utilise la reconnaissance vocale, le « Natural Language processing » pour comprendre la demande de l'utilisateur, et un synthétiseur vocal pour restituer la réponse.

Les modes de communication :

L'idée ici est de revoir quel support peut employer un utilisateur pour communiquer avec un assistant.

- **Site web :** il utilise principalement des chatbots dont l'objectif est de répondre aux questions fréquentes et d'aider à la navigation au sein du site ou de proposer à l'utilisateur une nouvelle façon d'obtenir des conseils. Par exemple Orangebank a implémenté Djingo, Ingenico a un chatbot de paiement. Malheureusement peu d'entre eux utilisent la voix.
- **Réseaux sociaux :** mis en œuvre sur Twitter, Facebook Messenger ou d'autres réseaux, les Assistants sont essentiellement des outils marketing et promotionnels.
- **Application mobile :** il existe de nombreux mobiles intégrant un assistant vocal dont Siri, Google Assistant et Bixby pour Samsung. L'assistant peut piloter de nombreuses actions telles qu'appeler un contact, envoyer un message, ajouter un rendez-vous dans votre agenda, et cela sans les mains. Les applications, quant à elles, fournissent la possibilité de piloter à distance des objets connectés : allumer la lumière, ajuster le thermostat, déverrouiller la voiture ou la porte et bien plus.
- **Objet connecté :** ces objets envahissent notre quotidien et tout objet du quotidien peut devenir connecté. Nous trouvons des réfrigérateurs, des fours avec lesquels nous pouvons converser. A quand le grille-pain intelligent ? Des produits émergent :
  - la montre connectée, compagnon du smartphone, peine à faire ses preuves ;
  - la voiture connectée, avec la possibilité de lancer des commandes vocales telles que « allume les feux », « donne-moi la direction de la maison », « où est le parking le plus proche ? » ;
  - l'enceinte connectée : c'est l'objet qui a une progression dans son adoption plus rapide que celle du smartphone. 23% des adultes américains possèdent une enceinte connectée ou Smart Speaker. Il existe deux raisons principales à une telle explosion : tout d'abord, après la révolution internet et du mobile, la population est avide de nouvelles technologies ; ensuite, le coût de ces enceintes. Les prix débutent à moins de 100 euros, l'objectif étant d'accrocher le consommateur et par la suite de lui vendre des services complémentaires. Dans ce marché nous retrouvons, entre autres, les principaux acteurs que sont :
    - Amazon avec Alexa dans ses enceintes Echo et Dot,
    - Google avec Google Assistant dans son enceinte Google Home,
    - Apple avec Siri dans Homepod.
- **Robot :** le robot est la façon ultime d'imiter le comportement humain et sa gestuelle. Son exploitation est actuellement plutôt

récréative ou évènementielle. Les Hôtels Hilton ont utilisé les robots Nao de Softbank comme concierge.

- Jeux vidéo : pendant des années, l'interaction homme machine pour les jeux vidéo s'est limitée à l'utilisation de claviers ou Joysticks. Même lorsque l'on parle de VR (Réalité virtuelle) on utilise des manettes. Dans les différentes séries ou films Star-Trek, les équipages parlent à leur vaisseau spatial pour lui donner des ordres ou obtenir un compte-rendu de situation. Aujourd'hui nous passons de la fiction à la réalité avec les jeux dont le premier du genre qui permet aux joueurs d'utiliser la voix : « Start Trek : Bridge Crew ». Le joueur peut désormais utiliser le langage naturel ! Plutôt sympa ! C'est une utilisation qui devrait être croissante.

Un peu d'histoire

En 1962 à la Seattle World's Fair, IBM présente sa solution Shoebox (boîte à chaussures). La solution a la taille d'une boîte à chaussures et identifie 16 mots, les chiffres de 0 à 9 et exécute des fonctions mathématiques.

Dans les années 70, des scientifiques de Carnegie Mellon (DARPA) créent Harpy qui reconnaît un millier de mots.

De gros progrès ont été réalisés lorsque les applications ont été capables de comprendre une phrase. Dans les années 90, IBM, Apple et d'autres créent des solutions de reconnaissance vocale. Apple démarre avec Plaintalk en 1993 pour son Macintosh. Dragon en 1997 est le premier dictaphone capable de retranscrire la voix en texte.

Les cas d'usages

Nous devons dissocier les cas d'usages grand public et professionnels, les objectifs étant totalement différents.

Le commun des mortels commande son assistant vocal plus qu'il ne mène une conversation avec lui. Les cas d'usages les plus fréquents sont, dans cet ordre :

- Poser une question,
- Écouter de la musique en Streaming,
- Demander la météo,
- Demander l'heure,
- Écouter la radio.

Ce sont des échanges simples.

Dans le milieu professionnel, nous imaginons mal l'ensemble des employés, au sein d'un open-space, parler à son assistant vocal. Cela deviendrait une joyeuse cacophonie.

Par contre, l'assistant vocal évite la nécessité d'utiliser les mains dans certains cas. Par exemple lorsqu'il y a une contrainte liée à l'environnement comme dans des laboratoires où l'on doit être équipé de gants pour éviter tout risque de contamination. Il se peut aussi que les mains soient utilisées pour la manipulation ou la maintenance d'engins.

Mais, dans le cadre professionnel la première implémentation est la substitution d'agent humain par des assistants vocaux dans les centres d'appels. Pourquoi un tel succès ?

- Les assistants vocaux font du 24/24, 7/7 sans fatigue ;
- Ils permettent de répondre immédiatement aux premières demandes des utilisateurs, sans attendre un agent libre ;
- Les échanges entre l'opérateur et l'utilisateur sont scénarisés, l'opérateur suivant une trame de questions suivant les réponses de l'utilisateur. Il y a donc peu d'effort à retranscrire cela dans un Assistant. De plus, les centres d'appels enregistrent les échanges ce qui peut être utilisé comme base de vérité (ground truth) pour l'entraînement de l'Assistant.
- Il est facile d'évaluer le retour sur investissement.

Votre premier assistant vocal :

Pour la création de votre premier assistant, le cas d'usage sera simple, il aura le rôle d'un concierge d'hôtel à qui vous pourrez demander des informations sur les services de l'hôtel tels que : où est la piscine? Quand est-elle ouverte ? Etc.

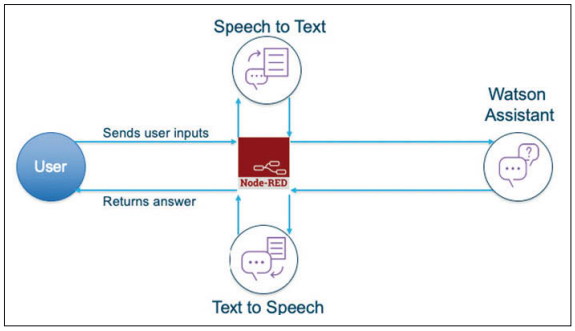
La construction de l'Assistant via les services Watson représente moins de 20% de l'effort total. Il est nécessaire en premier lieu de créer ce que l'on nomme la vérité de terrain (Ground Truth). Ceci constituera la base d'entraînement de votre assistant afin de lui permettre de comprendre la demande de l'utilisateur. C'est sa partie « Intelligent ». Cela consiste à classifier des exemples de phrases d'utilisateur en intention (intent). Par la suite, on assignera à ces intentions (intent) un processus de réponse. Tout ce travail a été fait pour vous et cette base de terrain vous est fournie.

Vous trouverez l'ensemble des sources nécessaires à la création : https://github.com/laurentvinc/VoiceAssistant

1- Téléchargez sur votre PC l'ensemble des fichiers.

Pour notre assistant vocal, nous allons avoir besoin de 4 services :

- Watson Speech to Text : qui transcrira la voix en texte.
- Watson Assistant : qui portera toute la logique de la conversation et permettra d'identifier l'intention de l'utilisateur.
- Watson Text to Speech : qui restituera la réponse de façon vocale.
- NodeRed : qui permettra l 'intégration de ces trois services.



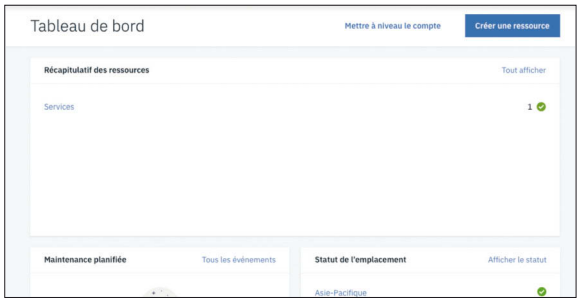
Pour l'exercice, vous n'avez besoin que d'un ordinateur avec un micro et un haut-parleur.

Commencer avec IBM Cloud :

2 - Enregistrez-vous sur la plateforme IBM Cloud : https://cloud.ibm.com

Une adresse e-mail vous sera utile pour la confirmation de la création du compte.

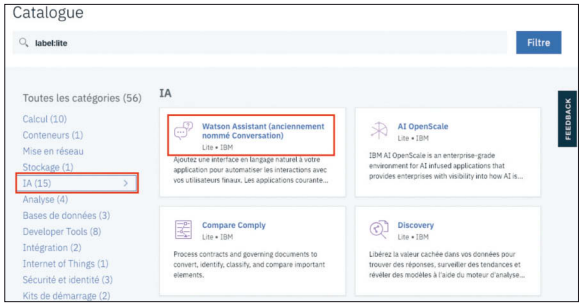
Vous voici maintenant dans votre tableau de bord. Celui-ci peut être vide si vous n'avez pas créé de service précédemment.



Les services Watson : Watson Assistant

Nous allons successivement créer l'ensemble de nos services.

3 - Cliquez sur Créer une ressource **pour lancer le catalogue, puis sur** IA (15) **menu.**

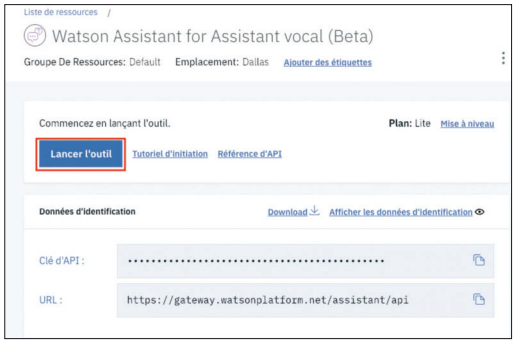


4 - Cliquez sur la tuile Watson Assistant.

Je vous invite à définir un nom de service et à conserver les paramètres par défaut. Bien choisir le Plan de tarification simplifié qui est gratuit.

5 - Cliquez sur Créer **pour finaliser la création.**

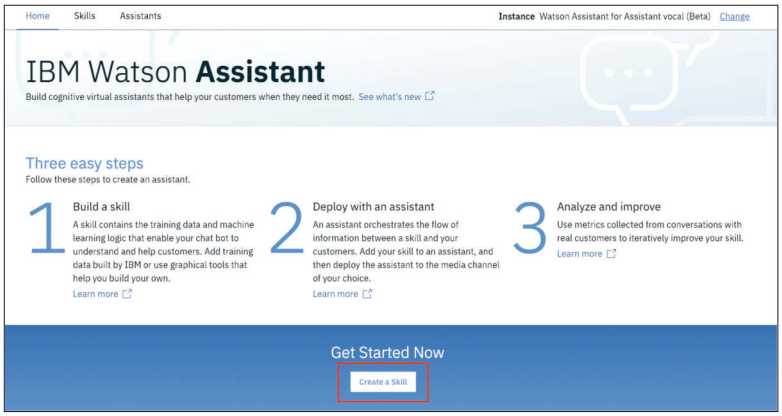
Votre service est instancié, il faut maintenant l'entraîner !



6 - Cliquez sur Lancer l'outil **pour finaliser la création.**

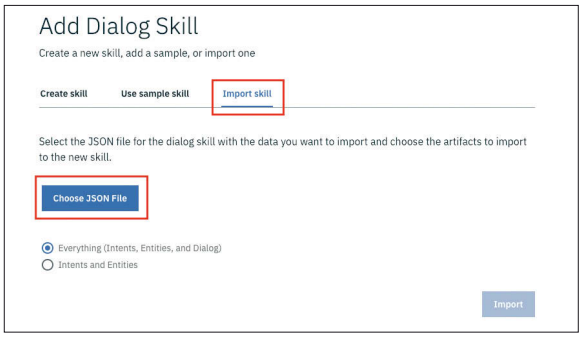
Nous allons créer les artéfacts nécessaires à la gestion de votre conversation. Ces artefacts sont : intents, entities, Dialog, Skill et Assistant.

7 - Dans la page d'accueil, cliquez sur 'Create skill' (milieu bas de la page).



8 - Cliquez sur 'Create new' puis 'Create' pour la création d'un 'Dialog Skill' (si nécessaire).

9 - Choisissez l'onglet 'Import Skill ', puis 'Choose JSON File'.



10 - Sélectionnez le fichier *Skill.json* précédemment téléchargé.

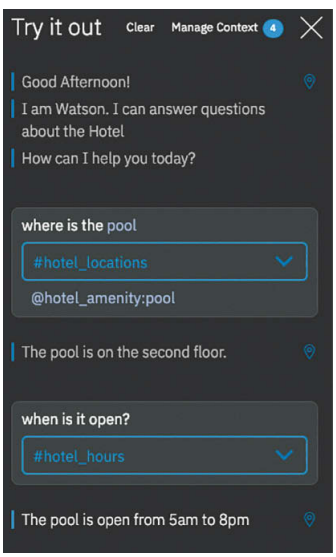
11 - Cliquez sur 'Import'.

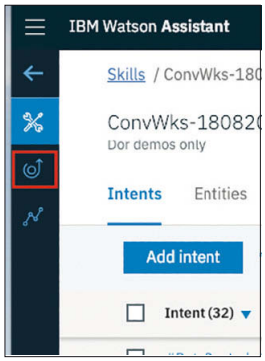
Votre 'Skill' pour l'assistant est en cours d'entraînement et devrait être disponible dans moins d'une minute. Vous pouvez le tester en cliquant sur '**Try it**' (en haut à droite).

Vous obtiendrez le message d'accueil de votre Assistant et vous pouvez commencer à lui poser des questions comme 'Where is the pool ?' et 'When is it open ?'. Si vous obtenez les réponses suivantes, votre 'skill' est prêt.

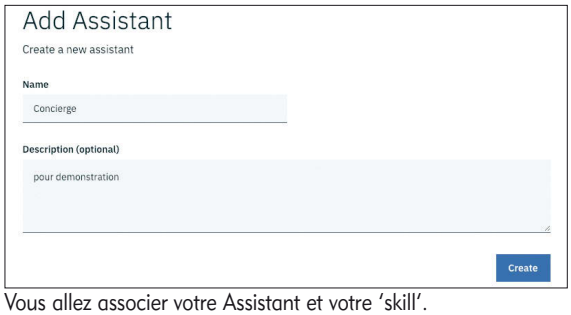
Nous allons finaliser notre assistant.

12 - Cliquez sur 'Deploy' (menu de gauche).



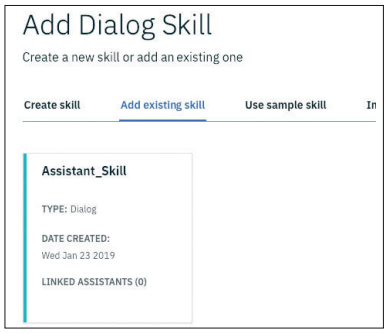


13 - Cliquez sur 'Add to an assistant'.
14 - Cliquez sur 'Create new'.
15 - Nommez votre Assistant 'concierge' et cliquez sur 'Create'.



Vous allez associer votre Assistant et votre 'skill'.

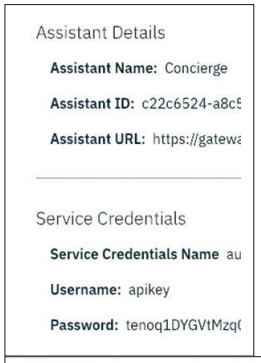
16 - Dans le cadre 'Dialog Skill', cliquez sur 'Add Dialog Skill'.
17 - Dans l'onglet 'Add existing Skill', selectionez votre 'Assistant_skill'.



Votre assistant est prêt. Pour l'intégration future, nous avons besoin de copier les clés de connections et l'ID de votre Assistant.

18 - Cliquez sur 'View API Details' (en haut à droite).

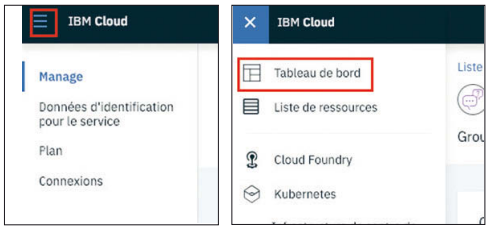
19 - Copiez localement 'Assistant ID'.



20 - Retournez sur IBM Cloud et la page d'accueil de Watson Assistant et copiez la Clé d'API et l'URL dans un éditeur de texte. Vous aurez besoin de ces informations ultérieurement.

Watson speech to text

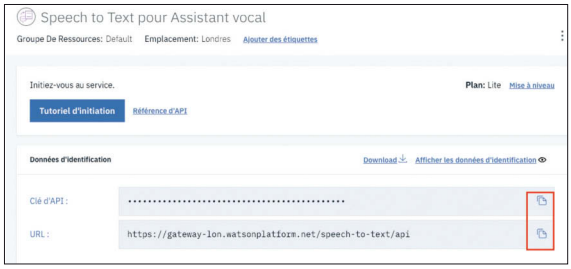
21 - Retournez sur votre Tableau de bord en cliquant sur le Hamburger menu (en haut à gauche) puis Tableau de bord.



22 - Comme précédemment, cliquez sur Créer une ressource **pour lancer le catalogue, puis sur** IA (15) **menu.**
23 - Cliquez sur la tuile 'Speech to Text'.

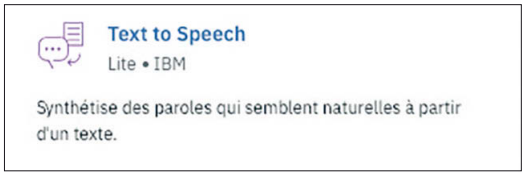
Je vous invite à définir un nom de service et à conserver les paramètres par défaut. Bien choisir le Plan de tarification simplifié qui est gratuit.

24 - Cliquez sur Créer **pour finaliser la création.**
25 - Comme pour le service Watson Assistant copiez la Clé d'API et l'URL dans un éditeur de texte. Vous aurez besoin de ces informations ultérieurement.



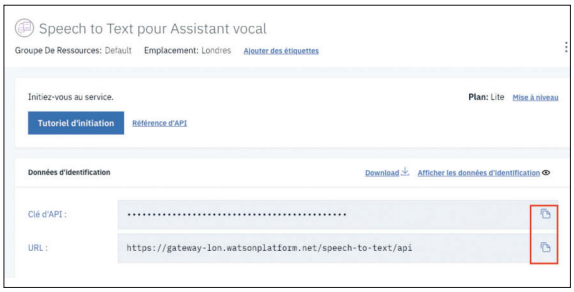
Watson text to speech

26 - Retournez sur votre Tableau de bord en cliquant sur le Hamburger menu (en haut à gauche) puis Tableau de bord.
27 - Comme précédemment, cliquez sur Créer une ressource **pour lancer le catalogue, puis sur** IA (15) **menu.**
28 - Cliquez sur la tuile 'Text to Speech'.



Je vous invite à définir un nom de service et à conserver les paramètres par défaut. Bien choisir le Plan de tarification simplifié qui est gratuit.

29 - Cliquez sur Créer **pour finaliser la création.**
30 - Comme pour le service Watson Assistant, copiez la Clé d'API et URL dans un éditeur de texte. Vous aurez besoin de ces informations ultérieurement.

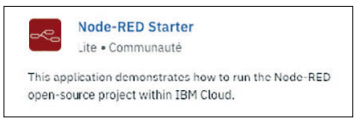


Le service Cloud : NodeRed

31 - Retournez sur votre Tableau de bord en cliquant sur le Hamburger menu (en haut à gauche) puis Tableau de bord.

32 - Comme précédemment, cliquez sur Créer une ressource **pour lancer le catalogue, puis sur** Kits de démarrage (2) **menu.**

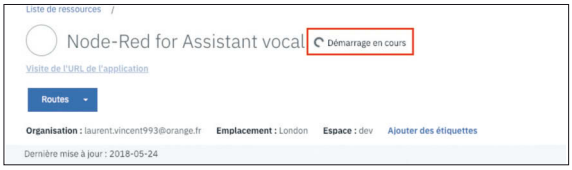
33 - Cliquez sur la tuile 'Node-RED starter'.



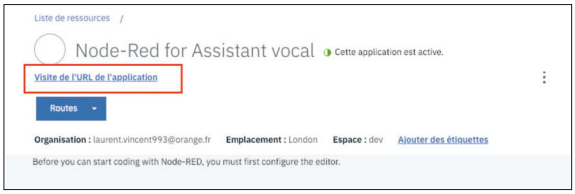
Je vous invite à définir un nom de service et à conserver les paramètres par defaut. Bien choisir le Plans de tarification simplifié qui est gratuit.

34 - Cliquez sur Créer **pour finaliser la création.**

La création peut prendre plusieurs minutes.



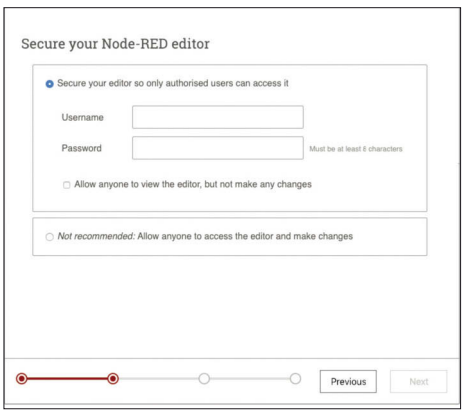
35 - Cliquez sur Visite de l'URL de l'application pour lancer Node-RED.



IBM Cloud ouvre la page de 'WELCOME' de Node-RED

36 - Cliquez sur 'Next'.

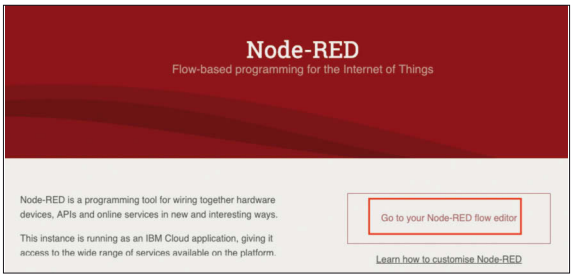
Vous êtes sur la page qui vous invite à sécuriser votre Node-Red. Pour le moment, je vous conseille l'option 'Allow anyone to access the editor'. Vous pourrez modifier les droits d'accès ultérieurement.



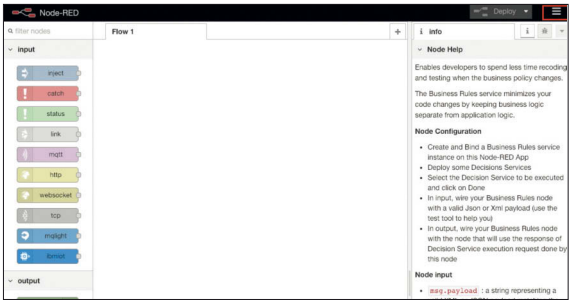
37 - Cliquez sur Next.

38 - Cliquez sur Next puis Finish.

Vous obtenez la page de confirmation de la création de votre Node-RED.



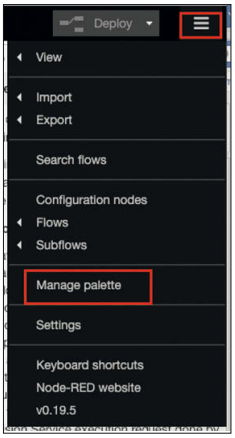
39 - Cliquez sur "Go to your Node-RED flow editor" pour ouvrir l'éditeur.



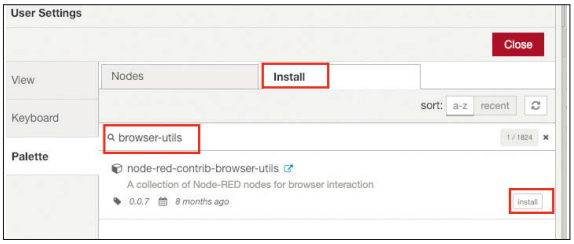
L'éditeur possède trois panneaux, le gauche où vous trouverez tous les nœuds, le central où les flux seront affichés, le droit où les informations sur le nœud sélectionné seront affichées et où nous pourrons trouver les logs d'exécution du flux.

Avant d'importer notre projet, nous allons télécharger les bibliothèques de nœuds nécessaires.

40 - En haut à droite, cliquez sur le 'hamburger' menu puis sur 'manage palette'.



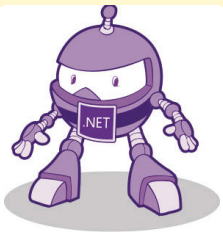
41 - Sélectionnez l'onglet 'Install' et dans le champ recherche, entrez le mot clé : *browser-utils*.



MEETUP PROGRAMMEZ! #3

SPÉCIAL

.NET CORE 3.0
VISUAL STUDIO 2019



14 mars à partir de 19h

Les intervenants de la soirée :

Andrés Talavera
(développeur, Ideastud.io)

Pierrick Blons
Team Leader Azure (SQLI)

Nicolas Gautier
Chef de Projet Microsoft (SQLI)

Où

Arolla
21, rue du bouloi
75001 Paris
Batiment D au fond à gauche

Métros / RER :
Métro 1 : station Louvre-Rivoli
Métro 4 / RER A, B & D : Châtelet - Les Halles

Inscriptions & infos pratiques : https://www.programmez.com/content/programmez-meetup-3-inscriptions

Nos prochains meetups :

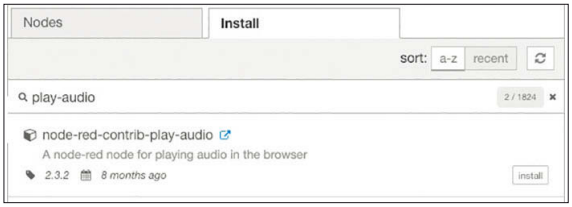
25 avril : *C++ et les dernières évolutions*
28 mai : *Java 12, le nouveau Java*
25 juin : *quantique, vers l'infini et au-delà*

Meetups organisés par [Programmez!] Le magazine des développeurs

42 - Cliquez sur 'install'. Cliquez sur 'install' dans la fenêtre de confirmation.

Trois nœuds ont été ajoutés dont celui pour gérer votre microphone.

43 - Sur ce même onglet, dans le champ recherche, entrez le mot clé : *play-audio*.



44 - Cliquez sur 'install' dans la fenêtre de confirmation.

Le 'Play audio' nœud est installé.

45 - Cliquez sur 'Close'.

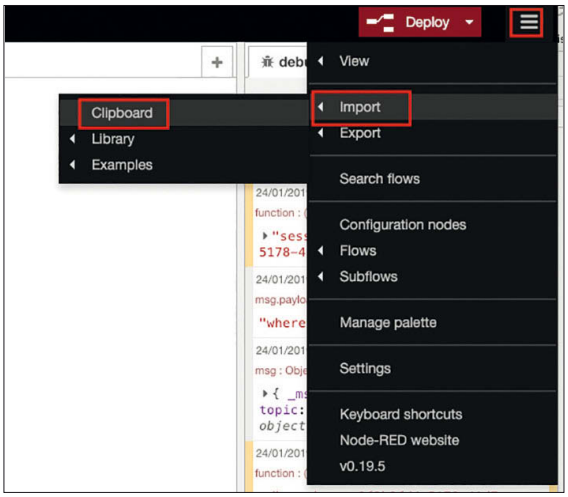
Votre interface Node-Red est prête.

Votre application

Nous allons importer le flux qui va orchestrer vos différents services Watson. Vous trouverez l'ensemble des sources nécessaires ici : https://github.com/laurentvinc/VoiceAssistant

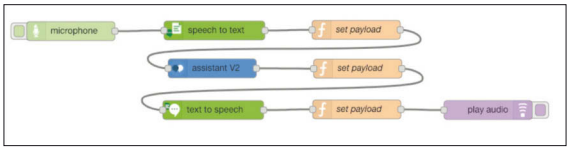
46 - Copiez le contenu du projet *AssistantNodeRED. Json.*

47 - Sur l'interface node-RED, cliquez sur le Hamburger menu (en haut à droite) puis sur'Import' et 'Clipboard'.



48 - Collez le contenu du projet puis cliquez sur 'Import'.

Vous visualisez le flux suivant dans l'éditeur.

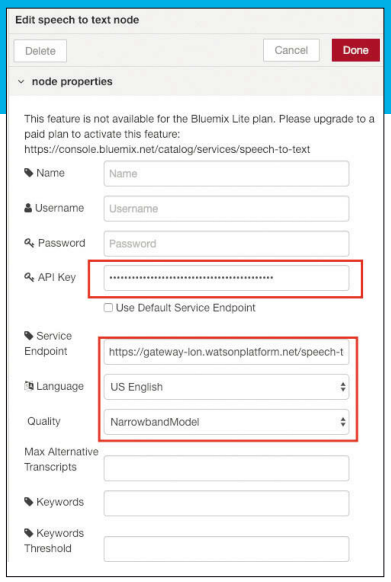


L'étape finale est de paramétrer vos nœuds 'Speech to Text', 'Assistant V2', 'Text to Speech'.

49 - Double cliquez sur le nœud 'Speech to Text'.

50 - Copier l'API Key, le service Endpoint URL.

51 - Le langage doit être US English et la qualité NarrowbandModel.



52 - Cliquez sur 'Done'.

53 - Comme précédemment, double cliquez sur le nœud 'Assistant V2'.

54 - Copiez l'API Key et l'Assistant ID.

55 - Cliquez sur 'Done'.

56 - Double cliquez sur le nœud 'Text to speech'.

57 - Copiez l'API Key, le service Endpoint URL.

58 - Le langage doit être US English.

59 - Cliquez sur 'Done'.

60 - Cliquez sur 'Deploy' (en haut à droite) .

Votre assistant est prêt à être testé.

Consultez votre assistant vocal

Pour lancer l'écoute de votre Assistant, il faudra cliquer sur le microphone et cliquer à nouveau pour interrompre l'écoute et cela pour chaque échange.

Lors de la première utilisation, vous devrez autoriser votre navigateur à utiliser le microphone de votre ordinateur.

Le démarrage d'une nouvelle conversation se fera avec le mot clé 'OK Watson'.

Vous pouvez demander des informations sur : 'the pool', 'the gym', 'the hotel restaurant', 'the sauna' ; ou commander une 'large mexicana'. Des exemples de scenarii :

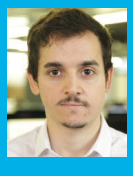
OK Watson
Where is the pool ?
When is it open?

OK Watson
I want a large Mexicana.

OK Watson
I want a pizza with more cheese.
A large margherita.

A vous de jouer !

Robin Guignard-Perret
(Product Manager)



Mennad Yami
(Junior software developer)

Créer votre assistant vocal 100% privé avec Snips

niveau 100

Snips est une plateforme vocale pour les objets connectés. Elle permet aux développeurs et aux fabricants d'objets d'ajouter un assistant vocal à leurs produits. Créée en 2013, la vision de Snips est : placer un assistant d'intelligence artificielle dans chaque objet, afin de rendre la technologie si intuitive qu'elle s'effacera en arrière-plan.

Snips est unique, car tout s'exécute localement sur l'objet auquel parle l'utilisateur, ce qui signifie qu'aucune donnée n'est jamais envoyée dans le cloud. C'est un concept qui garantit la confidentialité et la continuité du fonctionnement même en cas de panne d'Internet, et fait de Snips la première technologie vocale à suivre les normes générales sur la protection des données.
La plateforme Snips est gratuite pour quiconque souhaite tester notre technologie pour un usage non commercial. Nous avons créé une communauté de plus de 22 000 personnes - la plus grande communauté de développeurs d'applications vocales en dehors de Google et Amazon ! Snips fonctionne sous Linux (Raspberry Pi), Android, iOS, macOS et Debian. Tout le monde peut découvrir l'ensemble des outils et de la documentation permettant de créer ses propres applications vocales à l'adresse https://console.snips.ai. Snips est disponible en Français, Anglais, Allemand, Japonais, Espagnol et Italien.

Pourquoi nous avons choisi Rust ?

Snips permet de développer des interactions vocales à partir de n'importe quel langage de programmation (Python, Javascript...), cependant la plateforme elle-même est développée en Rust.

Performance : le code Rust est rapide et efficace. Il peut fonctionner sur des systèmes aux ressources limitées, et en tirer toute la performance possible. Le langage est construit autour d'abstractions sans coût, dans le même esprit que C ++, tout en maintenant le même niveau de sécurité qu'un langage à garbage collector. Ainsi Rust fournit des fonctionnalités de haut niveau sans pénalité d'exécution, exactement ce dont Snips avait besoin.

Portabilité : Rustc, le compilateur Rust, permet aux ingénieurs de Snips d'écrire du code une fois, et de le compiler pour de nouvelles architectures. C'est essentiel, car la société doit porter très régulièrement sa plateforme vers de nouveaux systèmes. Rustc est basé sur LLVM, un compilateur solide et éprouvé. Cela permet aux programmeurs de bénéficier de la cross-compilation vers la plupart des architectures matérielles modernes, des systèmes mobiles aux ordinateurs de bureau et aux serveurs.

Sécurité : le modèle unique d'"ownership" de Rust rend son code, une fois compilé, plus sûr que le C / C ++ et plus facile à maintenir dans la durée. Le langage introduit les concepts de propriété, de "move" et d'emprunt pour garder une trace des ressources, mémoire ou autres, et s'assurer qu'elles sont utilisées correctement. Ainsi, le compilateur Rustc valide l'usage des ressources et refuse du code qui ne les gère pas correctement. Il devient ainsi virtuellement impossible de produire accidentellement du code présentant des bugs de gestion de mémoire. Un effort particulier a été apporté aux messages d'erreur du compilateur afin d'aider à l'apprentissage des contraintes supplémentaires introduites par le langage et sa gestion des ressources. Dans la mesure du possible, le compilateur va suggérer des solutions. Celles-ci permettent aux nouveaux programmeurs débutant en Rust d'apprendre par la pratique, avec un risque minimum d'introduction de vulnérabilité.

Un cycle de développement rapide. En travaillant sur Rust, Snips a pu achever la première version de sa plateforme vocale en un temps record : il a fallu moins d'un an à Snips pour produire le code Rust et mettre son assistant vocal en production. La sécurité et l'inflexibilité du compilateur jouent un rôle important dans l'accélération du processus de développement de Snips. Dès l'écriture du code, les développeurs vont trouver et corriger des bugs qui n'apparaîtraient que bien plus tard dans le cycle de développement et de qualité si le code était en C ou C++. Ceci permet d'accélérer considérablement le processus de QA.

LES MAKER KITS

Snips et Seeed proposent deux types de kits :

- Le kit de base permet de créer facilement un puissant assistant vocal AI, de développer des applications personnalisées et de le déployer sur vos appareils connectés. Grâce au traitement sur l'appareil via le moteur ASR / NLP hors ligne de Snips, les données vocales des utilisateurs ne sont jamais envoyées dans le cloud, garantissant confidentialité et sécurité.
- Le kit satellite relaie les commandes vocales au kit de base pour traitement. Ainsi donc les développeurs peuvent prévoir plusieurs kits satellite dans toute la maison pour ajouter de nouvelles fonctionnalités au kit de base ou à tout autre haut-parleur intelligent, étendant ainsi le contrôle vocal dans plusieurs pièces.

Il n'y a pas de montages matériels, vous pouvez le faire tourner sur une RPI, un mac, un linux :

- Il vous faut une RPI 3+, une carte SD 4 Go minimum classe 10, un micro USB ou ReSpeaker, un haut-parleur (sinon un maker kit) ;
- Ou un Mac ;
- Ou une machine sous Linux.

Pour plus d'informations sur les kits de développement Snips Voice, veuillez consulter https://makers.snips.ai.

Snips a donc été en mesure d'intégrer rapidement de nouvelles fonctionnalités ou des correctifs à la production. Snips prend actuellement en charge une douzaine de plateformes différentes, notamment Raspberry Pi 3, DragonBoard, Sagemcom, Jetson TX2, IMX.8M et d'autres. Rust a permis à l'équipe de faciliter l'extension de la prise en charge de nouvelles cartes, car elles peuvent réutiliser la même base de code au lieu d'écrire des implémentations personnalisées pour chaque architecture.

Comment nos modèles fonctionnent sur des systèmes embarqués

Le principe de privacy-by-design met la confidentialité au coeur du processus de création du système. Cette confidentialité est d'autant plus cruciale que l'on déploie des assistants vocaux dans des contextes très variés, dont la sphère privée de ses utilisateurs. Ce choix garantit la protection des utilisateurs contre toute utilisation abusive, présente ou future, de leurs données. Chez Snips, le privacy-by-design se traduit par l'absence de toute communication de données d'utilisateur vers le cloud.

La plateforme vocale a été conçue avec comme priorité la portabilité et la bonne gestion des ressources. L'inférence intégrée de la plateforme fonctionne sur du matériel IoT commun aussi léger que le Raspberry Pi 3, un choix populaire parmi les développeurs. D'autres cartes Linux sont également prises en charge, et le SDK Snips pour Android fonctionne avec les périphériques dotés d'Android 5 et du processeur ARM, alors que le SDK iOS cible iOS 11 ou une version ultérieure. Pour des raisons d'efficacité et de portabilité, les algorithmes ont été ré-implémentés chaque fois que nécessaire en Rust.

Les assistants vocaux

Les assistants vocaux modernes peuvent se décomposer en plusieurs briques essentielles : la détection d'un mot clé (ou wakeword), la reconnaissance de la parole (Automatic Speech Recognition, ASR), la compréhension du langage naturel (Natural Language Understanding, NLU), et du code d'action qui va s'exécuter dès qu'une consigne est détectée.

Le *wakeword* est le mot clé qui permet d'indiquer à votre assistant le début d'une requête. Un assistant Snips démarre son écoute active dès qu'il entend "Hey, Snips!". Il commence alors à retranscrire les paroles de l'utilisateur, c'est le travail de la seconde brique qui est alors à l'oeuvre : l'ASR.

La brique ASR tente de transformer les signaux sonores captés par le micro en du texte intelligible par les humains. Lorsque l'utilisateur prononce "Hello world", le signal sonore enregistré traverse d'abord un premier modèle de deep learning, un modèle acoustique, qui le convertit en phonèmes, ces unités de représentation du son utilisées par les linguistes. "Hello World" sera d'abord perçu sous la forme "hɛˈləʊ wɜːld". Par la suite, un modèle spécifique à la langue de l'assistant est appliqué afin de transposer cette phonétique en vocabulaire que nous utilisons, c'est le modèle de langue. Si l'assistant est configuré en anglais, "Hello world" sera correctement retourné puis envoyé à la brique suivante, le NLU.

Le NLU permet d'extraire du sens dans un texte. Un assistant vocal étant conçu pour répondre à un éventail de requêtes limité, le travail du NLU est d'étiqueter une phrase avec une intention. Ces intentions viennent souvent avec des paramètres qu'il convient d'isoler pour pouvoir traiter la requête correctement. Par exemple, lorsque l'utilisateur a prononcé "Hey, Snips! Allume la lumière dans la chambre de Paul en rouge" et que le NLU reçoit la transcription de cette phrase, il distinguera l'intention "allumer la lumière" avec comme paramètre de lieu "chambre de Paul" et comme paramètre la couleur "rouge". Ce sont ces informations qui pourront être redirigées vers le code d'action.

La brique finale est donc le code d'action, qui attend, par le biais d'une boucle infinie, que son intention soit détectée afin de s'exécuter, en prenant en compte les paramètres reçus. Ce code est spécifique à chaque application et chaque appareil, et peut réaliser n'importe quelle action scriptable. Il peut servir à faire des appels sur des APIs distantes, commander un objet connecté, faire parler l'assistant vocal, ou tout ça à la fois. Le wakeword, l'ASR et le NLU servent de sens cognitifs à votre assistant, le reste est donc géré par le code d'action.

Les autres assistants vocaux du marché (Google Home, Alexa, Siri) hébergent leur moteur de détection de wakeword, leur ASR et leur NLU dans le cloud. Leurs modèles, très larges, nécessitent des puissances de calcul qui sont trop importantes pour de la technologie embarquée. Ce n'est pas le cas de Snips. Les modèles sont générés au cas particulier pour chaque assistant de chaque utilisateur, et sont spécifiques aux tâches pour lesquelles ils sont conçus. C'est ce qui permet de créer, grâce à Snips, des assistants ultra-légers, capables de tourner sur un Raspberry Pi 0 ou Pi 3, ou encore des assistants 100% offline. Tout cela est fait sans le moindre sacrifice du côté de la performance de la reconnaissance vocale.

La plateforme Snips

La plateforme est l'environnement qui rassemble tous les composants nécessaires. C'est elle qui, en l'installant, permet à chacun de profiter de son propre assistant vocal, respectueux de la vie privée, sur Linux, macOS, iOS et Android. Parmi ses éléments, aux côtés du détecteur de wakeword, de l'ASR et du NLU, on trouve le Dialog Manager qui est en charge de la coordination et de la succession des tâches que nous avons présentées. Le tts, ou "Text To Speech", est le programme en charge de la synthèse vocale, pour d'éventuelles réponses. Il reçoit du texte et le lit à haute voix, en adaptant son accent à la langue de l'assistant. Enfin, le Skill-server est le composant chargé d'exécuter le code d'action.

La communication interne à la plateforme se fait via le protocole Hermes, créé par Snips autour du protocole MQTT, léger et rapide. Grâce à Hermes, il est facile de personnaliser le fonctionnement de la plateforme pour, par exemple, communiquer avec des satellites de votre appareil principal, faire dire une phrase au TTS, ou commander l'un des composants de la plateforme séparément.

Snips, une fois installé sur un périphérique, peut accueillir un assistant. C'est sur la console web (console.snips.ai) que l'on crée cet assistant. La console permet de définir la langue de l'assistant, son champ de compréhension et ses fonctionnalités (ou Apps).

Par exemple, la plateforme, installée sur votre Raspberry Pi, pourrait accueillir un assistant en français possédant une App pour le contrôle de la lumière ainsi qu'une App météo.

Créer et installer son assistant

Passons maintenant à la création de votre assistant personnel, à déployer sur un Raspberry Pi. Votre Pi doit d'abord accueillir la plateforme Snips. Pour faciliter son installation et la manipulation des

assistants, nous avons créé SAM (Snips Assistant Manager), un outil de déploiement simple d'utilisation, en ligne de commande. SAM est un programme pour votre ordinateur, qui utilise SSH pour se connecter à votre Pi. Les deux appareils doivent donc être sur le même réseau, pensez à connecter votre Pi au réseau wifi ou à votre ordinateur par ethernet. Par ailleurs, votre machine principale doit supporter Node Js et NPM pour se servir de SAM. Un "sudo npm install -g snips-sam" fera alors l'affaire pour installer l'outil. La détection de votre Pi se fait ensuite par la commande "sam devices", qui liste les Pi présents sur le réseau. Lorsque vous obtenez l'adresse IP ou le hostname de votre appareil, connectez-le à SAM grâce à "sam connect <adresseIP/hostname>". Enfin, un "sam init" permet l'installation de la plateforme Snips. Pour connecter votre micro et votre sortie audio, vous pouvez utiliser "sam setup audio". La création de l'assistant se fait par la console web (console.snips.ai). Créez un compte, puis créez un nouvel assistant. Cet assistant pourra disposer d'une ou plusieurs fonctionnalité(s), vous pouvez en choisir parmi les Apps disponibles dans l'App store (météo, livre de recette, contrôle des lampes Philips Hue, ...) ou construire vos propres Apps. Vous pouvez également personnaliser une App de l'App store afin d'en produire une version unique grâce à la fonction de Fork.

Une App capture des intentions, ou "intents", grâce à l'entraînement du modèle de NLU. Elle en capture également les paramètres, ou "slots", s'ils existent. Dans le cas d'une App calendrier, lorsque l'utilisateur exprime "Ajoute un évènement anniversaire pour le 13 novembre", la demande est aiguillée par le NLU vers l'intent "addEvent", et ses paramètres avec comme slots un nom d'événement "anniversaire" et une date "13 novembre". Les équipes de Snips ont déjà créé certains slots courants, tels que les dates, pour faciliter leur détection.

Voici à quoi ressemblent les données envoyées par le NLU au code d'action sous format JSON :

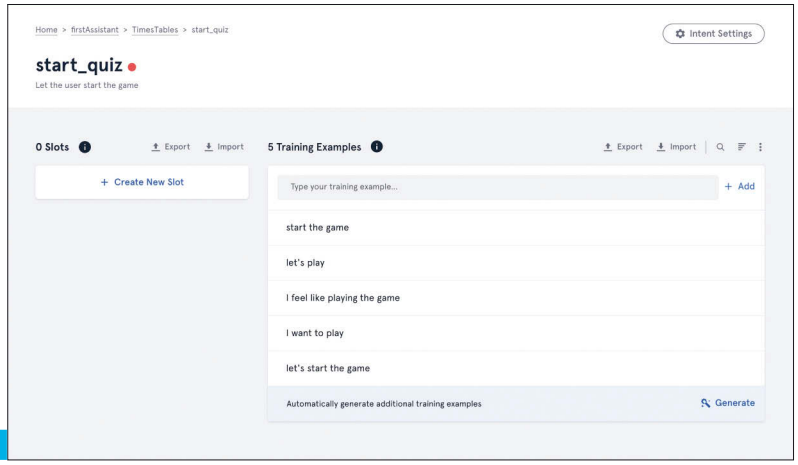
```
{
 "input": "ajoute un anniversaire pour le treize novembre",
 "intent": {
  "intentName": "userName:addEvent",
  "probability": 0.9662896
 },
 "slots": [
  {
   "rawValue": "anniversaire",
   "value": {
    "kind": "Custom",
    "value": "anniversaire"
   },
   "range": {
    "start": 10,
    "end": 22
   },
   "entity": "Noms communs_fr",
   "slotName": "summary"
  },
  {
   "rawValue": "pour le treize novembre",
   "value": {
    "kind": "InstantTime",
    "value": "2019-11-13 00:00:00 +00:00",
    "grain": "Day",
    "precision": "Exact"
   },
   "range": {
    "start": 23,
    "end": 46
   },
   "entity": "snips/datetime",
   "slotName": "start_datetime"
  }
 ]
}
```

Votre assistant a besoin de données d'entraînement pour entraîner ses réseaux de neurones. Vous pouvez vous-mêmes créer vos exemples grâce à la console. Pour augmenter les performances du NLU, vous pouvez identifier la position et la nature des slots dans une phrase en les surlignant à la souris. Par ailleurs, plus les exemples sont nombreux et variés et meilleures seront les performances de votre assistant. La console Snips vous propose également un service payant de génération de données d'entraînement. N'oubliez pas de sauvegarder, la console mettra alors à jour votre assistant. Vous pouvez utiliser l'interface de test à droite de la console pour observer les réponses de votre assistant aux requêtes orales ou écrites. Vous pouvez maintenant installer votre assistant sur votre Pi, grâce à SAM. La commande "sam login" vous permet de vous connecter sur votre compte utilisé dans la console. Vous pouvez ensuite exécuter "sam install assistant" et sélectionner votre assistant pour l'installer.

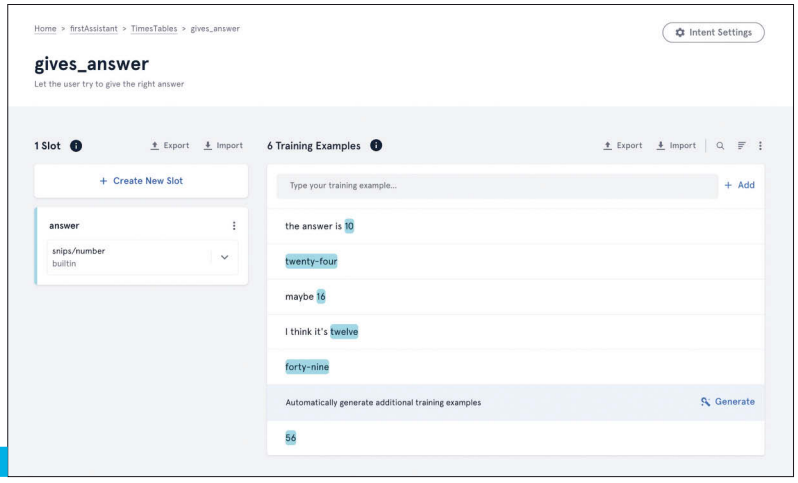
Le code d'action est composé de callbacks qui s'exécutent lorsque l'intent qui leur correspond est détecté. Il n'y a pas de limitation dans le langage utilisé, cependant Python et Javascript disposent d'une librairie, Hermes, qui simplifie les communications internes avec Snips. Ces deux langages sont donc recommandés (mais pas obligatoires). Vous pouvez exécuter votre code manuellement, mais le snips-skill-server peut également le faire automatiquement au démarrage du Pi. Pour cela, créez un répertoire au nom de votre choix dans /var/lib/snips/skills/, et placez votre script dans ce répertoire. Votre script doit alors avoir un nom commençant par "action-", et être exécutable. Son rôle est d'être à l'écoute des intents, via une boucle infinie. Il est possible de lier une App avec un répertoire Github, et laisser SAM copier vos fichiers au bon endroit lors de l'installation de l'assistant. Le repo doit être public, vous pourrez alors l'ajouter dans l'onglet Action de la console. Vous pourrez y placer, en plus de votre fichier d'action, un setup.sh qui sera exécuté lors de l'installation, par exemple pour compiler votre projet.

Un exemple d'assistant

Construisons ensemble un premier assistant. Cet assistant a pour but de vous faire réviser vos tables de multiplication via un jeu de questions-réponses. Depuis la console, créons un nouvel assistant en anglais nommé "FirstAssistant" qui comporte une App que vous allez nommer "TimesTables". Le joueur doit pouvoir lancer le jeu, puis répondre aux questions ou les passer lorsqu'il n'a pas la réponse. Nous avons là nos 3 intents, qui sont nommés "start_quiz",



1



2

"gives_answer" et "doesnt_know". Notons que votre code d'action recevra le nom de l'intent au format "userName:intentName".
L'intent "start_quiz" n'a qu'un seul effet voulu, lancer le jeu et n'a donc pas besoin de slot pour ajuster le comportement de votre assistant. Il doit comprendre des phrases du type "let's start the game" ou "I want to play". Il est important d'ajouter des exemples de phrases nombreux et variés pour rendre l'intent robuste, ajoutons par exemple "I feel like playing the game" ou "let's play" aux exemples avant de sauvegarder. 1

L'intent "gives_answer" doit, lui, permettre de comprendre une réponse qui est variable. Un slot est donc le bienvenu, on peut le nommer "answer" et lui donner le type prédéfini "number". Dans les phrases d'exemples, n'oubliez pas de surligner les nombres s'ils ne le sont pas automatiquement. Des réponses telles que "forty-nine" ou "I think it's twelve" sont attendues. Le JSON reçu par votre code d'action comportera bien la valeur du nombre exprimé par l'utilisateur parmi ses slots. 2

L'intent "doesnt_know" suit la même logique que "start_quiz" puisqu'il ne nécessite pas de paramètre. Il convient donc seulement de donner suffisamment d'exemples pour entraîner les modèles de votre assistant. "I don't know", "I have no idea" ou "I can't get the answer" sont des exemples pertinents.
L'assistant est prêt à être installé grâce à SAM mais avant cela, nous devons écrire le code d'action. Ici nous avons choisi un script écrit en Python et appelé "action-timestable.py", nous le plaçons dans un répertoire /var/lib/snips/skills/timestables/.

Le code d'action

Pour faire le code de l'action nous allons utiliser la librairie hermes_python.
La première étape est donc d'importer hermes_python ainsi que random (qui nous permettra de générer les questions).

```
#!usr/bin/env python2
# coding: utf-8

from hermes_python.hermes import Hermes
import random
```

Le shebang (1er ligne) est nécessaire pour que snips-skill-server puisse exécuter le code de votre assistant.
Maintenant nous allons définir les variables globales de notre action.

```
MQTT_HOST = "localhost"
MQTT_PORT = "1883" # Standard MQTT port
MQTT_URL = "{}:{}".format(MQTT_HOST, MQTT_PORT) # ==> "localhost:1883"

SNIPS_USER_NAME = "snips"
INTENT_STARTS_QUIZ = "{}:start_quiz".format(SNIPS_USER_NAME) # ==> "snips:start_quiz"
INTENT_GIVES_ANSWER = "{}:gives_answer".format(SNIPS_USER_NAME) # ==> "snips:gives_answer"
INTENT_DOESNT_KNOW = "{}:doesnt_know".format(SNIPS_USER_NAME) # ==> "snips:doesnt_know"

sessions_states = {}
```

Les trois variables "MQTT_..." définissent le point d'accès à notre serveur MQTT, dans notre cas on se connecte à "localhost" car le code sera exécuté sur le Raspberry sur lequel Snips est installé.
Les trois variables "INTENT_..." définissent les noms de vos intents préfixés du nom de l'utilisateur qui a créé l'assistant sur console.snips.ai .
La variable "sessions_states" permettra à l'assistant de se rappeler de la question posée à l'utilisateur.
Ici nous définissons la fonction qui sera appelée quand l'utilisateur demande à l'assistant vocal de lui faire réviser ses tables de multiplications.

```
def starts_quiz(hermes, intent_message):
  session_id = intent_message.session_id

  # Generating question
  a = int(random.random() * 10 + 1)
  b = int(random.random() * 10 + 1)
  tts = "What's {} times {} ?".format(a, b)
  sessions_states[session_id] = [a, b]

  # Define what the assistant is able to understand
  intent_filter = [INTENT_DOESNT_KNOW, INTENT_GIVES_ANSWER]

  hermes.publish_continue_session(session_id, tts, intent_filter)
```

Chaque fonction associée à un intent recevra deux objets en paramètres : "hermes" et "intent_message".
La première étape pour construire un dialogue avec Snips est de

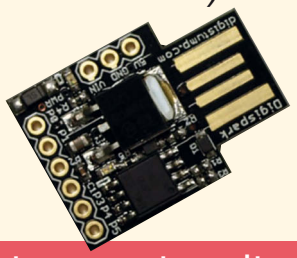
Tous les numéros de

Programmez! Le magazine des développeurs

sur une clé USB (depuis le n°100)

+

1 carte de prototypage Attiny85, compatible Arduino, offerte !



34,99 €*

Clé USB. Photo non contractuelle. Testé sur Linux, macOS, Windows. Les magazines sont au format PDF.

* tarif pour l'Europe uniquement.
Pour les autres pays, voir la boutique en ligne

Commandez-la directement sur notre site internet : www.programmez.com

Complétez votre collection

Prix unitaire : **6,50€**

Lot complet

1 CADEAU À OFFRIR !

48,90 €

(au lieu 58,90 €)

☐ 218	: __ ex	☐ 223	: __ ex
☐ 219	: __ ex	☐ 224	: __ ex
☐ 220	: __ ex	☐ 225	: __ ex
☐ 221	: __ ex	☐ 226	: __ ex

☐ Lot complet :
218 - 219 - 220 - 221
223 - 224 - 225 - 226
48,90 €

Commande à envoyer à :
Programmez!
57 rue de Gisors - 95300 Pontoise

soit __ exemplaires x 6,50 € = ____ € soit au **TOTAL** = ____ €

Prix unitaire : 6,50 €
(Frais postaux inclus)

☐ M. ☐ Mme ☐ Mlle Entreprise : ____ Fonction : ____

Prénom : ____ Nom : ____

Adresse : ____

Code postal : ____ Ville : ____

E-mail : ____ @ ____

Règlement par chèque à l'ordre de Programmez ! I Disponible sur www.programmez.com

Offres pouvant s'arrêter à tout moment, sans préavis **PROG 227**

récupérer "l'identifiant de la session de dialogue" (session_id). Ce "session_id" permet d'identifier le dialogue en cours de manière unique ; ce qui est important car il peut potentiellement avoir plusieurs personnes qui utilisent votre assistant en même temps (grâce aux satellites). Une session de dialogue commence quand un utilisateur prononce le wakeword ("hey snips") et se termine quand votre code envoie un "end_session".
Ensuite on génère une question aléatoire sur les tables de 1 à 10 et on stocke la question dans notre dictionnaire "sessions_states". Cela nous permettra de vérifier la réponse de l'utilisateur.
Enfin on peut publier un évènement à Snips pour continuer le dialogue en répondant à l'utilisateur et en écoutant sa réponse. Pour cela on fait un "hermes.publish_continue_session(...)" avec comme premier paramètre l'identifiant de la session que l'on veut continuer. Le deuxième paramètre est la phrase/question que l'assistant va poser à l'utilisateur (tts = text to speech). Le dernier paramètre est "l'intent filter" qui va permettre à l'assistant de savoir que l'utilisateur va répondre avec une phrase présente dans un des deux intents sélectionnés.
Ici les deux intents sont "INTENT_DOESNT_KNOW" et "INTENT_GIVES_ANSWER" ce qui veut dire que l'utilisateur ne peut que *donner une réponse* **ou** *dire qu'il ne sait pas*.
Il faut maintenant définir la fonction qui sera appelée quand l'utilisateur donne une réponse.

```
def gives_answer(hermes, intent_message):
  session_id = intent_message.session_id

  # get answer from user
  answer = intent_message.slots.answer
  answer = None if answer is None else answer.first().value

  # defines  TTS
  tts_good = "Good answer !"
  tts_bad = "Wrong answer ..."
  tts_no_answer = "I'm sorry I didn't get your answer"
  tts = ""

  # Check user's answer
  if answer is None:
    tts = tts_no_answer
  elif answer == sessions_states[session_id][0] * sessions_states[session_id][1]:
    tts = tts_good
  else:
    tts = tts_bad

  hermes.publish_end_session(session_id, tts)
```

Les deux lignes d'initialisations de la variable "answer" permettent de récupérer la valeur (sous forme d'un nombre entier) de la réponse de l'utilisateur. La deuxième ligne vérifie que l'utilisateur a bien donné une réponse avant d'accéder à la valeur.
Les lignes "tts_... =" définissent les phrases que l'assistant vocale répondra dans les trois différents cas :
1. l'utilisateur a donné la bonne réponse ;
2. l'utilisateur a donné une mauvaise réponse ;
3. la réponse de l'utilisateur n'a pas été comprise par l'assistant.

Le *if* permet de déterminer dans lequel de ces trois cas nous sommes.
Enfin on publie l'évènement end_session avec pour paramètre le "session_id" ainsi que la phrase que l'assistant va dire avant de fermer cette session de dialogue.
La fonction appelée quand l'utilisateur dit ne pas connaître la réponse est assez simple. On publie juste un end_session pour fermer le dialogue avec un TTS (text to speech) prédéfini.

```
def doesnt_know(hermes, intent_message):
  session_id = intent_message.session_id
  tts = "Ok, at least try next time !"

  hermes.publish_end_session(session_id, tts)
```

Cette fonction est la fonction qui sera appelée une fois qu'une session a fini de se fermer après un appel à "hermes.publish_end_session(...)".

```
def session_ended(hermes, session_ended_message):
  del sessions_states[session_ended_message.session_id]
```

Ici nous supprimons l'élément du dictionnaire qui était associé à la session qui vient de se fermer.
Finalement la dernière étape est de connecter tout ça ensemble.

```
with Hermes(MQTT_URL) as h:

  # Subscribe to intents
  h.subscribe_intent(INTENT_STARTS_QUIZ, starts_quiz)\
   .subscribe_intent(INTENT_GIVES_ANSWER, gives_answer)\
   .subscribe_intent(INTENT_DOESNT_KNOW, doesnt_know)

  # Subscribe endSession
  h.subscribe_session_ended(session_ended)

  # Start Listening
  h.loop_forever()
```

La première ligne crée une instance de l'objet "Hermes" dans un gestionnaire de contexte. On notera que son argument MQTT_URL est l'url que nous avions définie au début du fichier.
Les trois lignes après "# Subscribe to intents" vont définir que pour un intent donné (défini par nos variables globales) on associe une fonction donnée. C'est grâce à ces trois lignes que Snips saura quelle fonction appeler pour chacun des intents reçus.
Ensuite de la même manière on définit que la fonction "session_ended" sera la fonction appelée quand une session de dialogue se finit.
Et enfin on peut exécuter la fonction qui va se charger d'écouter les différents évènements Snips et d'appeler les fonctions associées à chaque intent. •

Le code complet de l'App sur programmez.com & github

François Tonic

niveau 100

Parlons matériels !

Nous avons évoqué les moteurs d'assistants vocaux. Et si nous passions un peu à la pratique ? Comme vous le verrez, nous avons deux catégories de matériels : les matériels pour créer un véritable assistant et des solutions de reconnaissance et synthèse vocale.

Il s'agit d'un petit échantillon des multiples modules et cartes pour créer un assistant vocal. Il est possible de créer un assistant pour moins de 50 € avec uniquement un Raspberry Pi, un microphone et un speaker. Seuls, ces modules ne vont pas servir à grand-chose, sauf quand ils sont capables d'interagir avec un moteur comme Snips, Alexa ou Google Assistant. A vous de jouer.

MOVI D'AUDEME

Plateforme : Arduino

Prix : 75 $ - Note : 12/20

Nous avons hésité à mettre cette plateforme compatible Arduino. Movi est une shield que l'on installe sur une carte Arduino classique. Il s'agit d'une carte de reconnaissance et de synthèse vocale. Il ne s'agit pas de créer un assistant en tant que tel.

La carte intègre un processeur Allwinner A13, des prises micro et speaker, un microphone. A noter qu'il faut impérativement alimenter l'Arduino sur secteur pour pouvoir utiliser Movi. Depuis le firmware 1.1, Movi supporte l'anglais, l'allemand et l'espagnol. La plateforme reconnaît plus de 150 commandes vocales.

Le modèle de développement est totalement intégré à Arduino IDE. La carte possède deux modes : entraînement et reconnaissance. L'entraînement est notamment fait quand vous changez les éléments vocaux dans le code. On peut ainsi définir le mot d'appel de l'assistant, par exemple « Jarvis ». On indique ensuite les séquences de mots utilisables : « Turn on », « light on », etc. Vous pouvez indiquer, en code, les réponses éventuelles à générer. Il faut éviter des mots trop longs et des phrases complexes.

Bon point : une interface bas niveau pour voir ce qu'il se passe dans les traitements, via le moniteur série.

Nous avons constaté une grande sensibilité à l'environnement, des latences parfois longues et une stabilité quelque peu aléatoire. Bref, spécificités prometteuses mais résultat un peu décevant.

GRASP.IO

Plateforme : Arduino 1

Prix : 26,50 € - Note : 15/20

Précisons que cette plateforme ne permet pas spécifiquement de créer un assistant. Mais nous pouvons manipuler à la voix des capteurs et objets, grâce à son smartphone (en installant l'app Grasp.io). La carte nécessite une Raspberry Pi. Nous avons été séduit par la mise en œuvre bien intégrée et l'app se révèle pratique même si l'ergonomie n'est pas toujours au rendez-vous. Les blocs permettent de construire très rapidement son usage. La possibilité de connecter des capteurs en Entrée et en Sortie est pratique. Nous avons eu des problèmes de compatibilité avec l'iPhone Xr, nous sommes repassés à un iPhone 6 pour la partie vocale. Pour un prix raisonnable, c'est une bonne solution à découvrir.

RESPEAKER 2-MICS PI HAT

Plateforme : Raspberry Pi 2

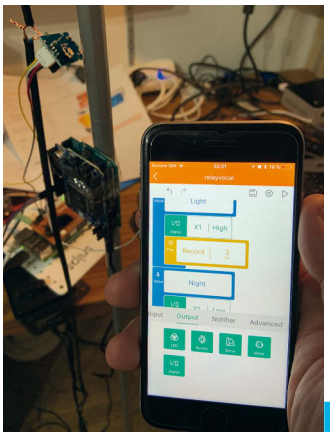
Prix : 15 € - Note : 15/20

ReSpeaker n'est pas plus grand qu'une Pi Zero. La carte possède deux micros, deux connecteurs Grove, une sortie audio (speaker) en prise jack et au format JST 2. Sur la partie Grove, nous sommes sur un port I2C et GPIO12 (en réalité 12 et 13). Bref, c'est du costaud.

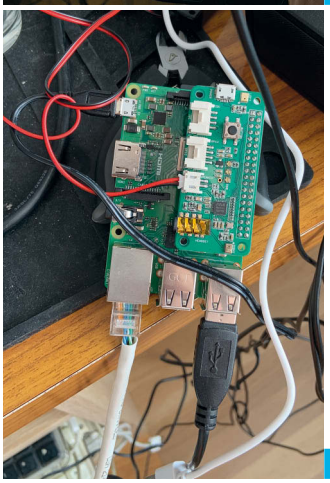
Par défaut, la carte fonctionne avec Google Assistant, Alexa est bien entendu utilisable. L'installation de Seeed mentionne mal la nécessité d'installer les tools Google pour pouvoir se connecter aux API, aux services et aux assistants que l'on crée dans la console. Attention : des conflits d'accès peuvent avoir lieu sur les API. Si le problème ne se résout pas : supprimer les projets et réactiver les API (enable). Faites attention aux périphériques micro et speaker. Vérifiez bien la configuration car parfois, ReSpeaker perd l'un ou l'autre ou les deux…

L'installation et l'initialisation de Google Assistant ne sont pas forcément très compliquées, il faut juste être rigoureux. Il faut environ 15-20 minutes pour tout mettre en place. Ensuite, on dispose d'un assistant.

Pour de petits IoT, la plateforme est sympathique. Nous retrouvons cependant les deux défauts de Seeed : une intégration matérielle – logicielle trop légère, une documentation technique limitée.



1



2

VOICE RECOGNITION MODULE DE GEEETECH

Plateforme : Arduino

Prix : env. 23 € - Note : 12/20

Un petit module sympathique qui existe en plusieurs déclinaisons et clones. Il doit permettre la capture vocale et la synthèse. Là encore, nous ne faisons pas un assistant de type Alexa mais plutôt une commande vocale. Nous avons rencontré beaucoup de soucis de reconnaissance en programmation. Par contre, en commande série (via AccessPort), le module était plus ou moins reconnu et utilisable.

Au final, nous ne savons pas trop quoi penser de ce module. En théorie, une solution intéressante, en pratique, c'est laborieux, et, là encore, une documentation limitée.

GROVE SPEECH RECOGNIZER DE SEEED STUDIO

Plateforme : Arduino

Prix : -20 € - Note : ?

Seeed propose de nombreux modules, plutôt bien faits et faciles à intégrer. Nous disposons d'un module de reconnaissance vocale. Théoriquement, nous pouvons l'utiliser pour créer un mini assistant vocal. Il reconnaît par défaut 22 commandes. Malgré plusieurs heures de tests sur plusieurs Arduino et shield Grove, nous n'avons jamais pu utiliser le module.

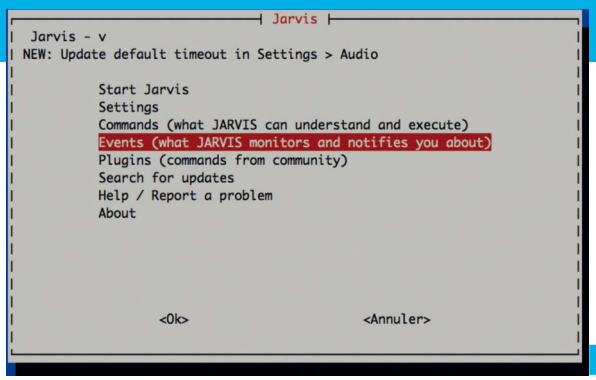
OPENJARVIS

Plateforme : Raspberry Pi 3

Prix : gratuit - Note : 15/20

Nous n'avons pas forcément besoin d'un shield dédié pour créer un assistant. Le projet Jarvis permet, avec un minimum de matériel, de créer un assistant léger : une Pi, un speaker, un microphone. L'installation est plutôt bien faite même s'il faut de nombreuses étapes et des packages externes, ainsi que les clés d'API pour pouvoir fonctionner. Mais franchement, c'est un joli travail !

Nous avons beaucoup galéré sur les premières configurations. Nous vous conseillons de partir sur un Raspbian clean sans aucune autre installation et rester sur du matériel standard. Tout se déroule en session SSH ce qui facilite bien les choses. L'un des intérêts de Jarvis est la diversité des plugins. Attention aux clés d'activations des services qui peuvent être sensibles.



3

La plateforme est extensible avec vos éléments déclencheurs et les commandes. La documentation est accessible et bien faite. La partie configuration mériterait d'être un peu explicite, notamment sur la gestion des clés. Il existe d'autres plateformes que Jarvis, telles que Kalliope. 4 Kalliope fonctionne de la même manière que Jarvis mais plus rudimentaire. Il est conseillé d'installer le starter.fr pour comprendre le fonctionnement de l'assistant : https://github.com/kalliope-project/kalliope_starter_fr

L'installation est très simple :

```
Sudo git clone https://github.com/kalliope-project/kalliope_starter_fr.git
cd kalliope_starter_fr
kalliope start
```

Pour la partie configuration : regardez les structures des fichiers settings.yml et brain.yml. Le dossier brains vous sera utile pour créer les interactions et comprendre celles-ci : la question et les réponses de l'assistant.

Les modules non testés

Nous n'avons pas pu tester deux cartes de Matrix : Creator et Voice. La pile logicielle était tellement instable, et la configuration aléatoire, que nous n'avons pas poussé les expériences.

Nous avions aussi un Google Voice Hat que nous avions testé il y a presque 2 ans. Faute de temps, nous n'avons redémarré le PoC. C'est une solution à regarder mais tournée vers Google Assistant.

```
pi@raspberrypi:/kalliope_starter_fr $ kalliope start
DEBUG:kalliope:[LIFOBuffer] LIFO buffer created
Starting REST API Listening port: 5000
Starting Kalliope
Press Ctrl+C for stopping
 * Serving Flask app "kalliope" (lazy loading)
Starting order signal
 * Environment: production
   WARNING: Do not use the development server in a production environment.
je suis prête
   Use a production WSGI server instead.
 * Debug mode: on
Waiting for trigger detection
Que puis-je faire pour vous?
[SpeechRecognition] Threshold set to: 4000
Say something!
Google Speech Recognition thinks you said quelle heure est-il
Order matched in the brain. Running synapse "say-local-date"
il est 10 heure et 53 minute
Waiting for trigger detection
Monsieur?
[SpeechRecognition] Threshold set to: 4000
Say something!
Google Speech Recognition could not understand audio
Waiting for trigger detection
Je vous écoute
[SpeechRecognition] Threshold set to: 4000
Say something!
Google Speech Recognition thinks you said quelle date sommes-nous
Order matched in the brain. Running synapse "say-local-date-from-template"
Nous sommes le lundi 18 février 2019
Waiting for trigger detection
```

4

CONFIGURATION AUDIO

La configuration audio est souvent un point sensible de votre montage et des assistants. Deux commandes sont essentielles :

```
arecord -l
aplay -l
```

Le premier donne la liste des périphériques de capture vocale (= microphone). Le second concerne le speaker. Deux éléments sont à donner : les cartes disponibles et les périphériques. Par exemple :

```
**** Liste des Périphériques Matériels PLAYBACK ****
carte 0: ALSA [bcm2835 ALSA], périphérique 0: bcm2835 ALSA [bcm2835 ALSA]
  Sous-périphériques: 7/7
  Sous-périphérique #0: subdevice #0
  Sous-périphérique #1: subdevice #1
  Sous-périphérique #2: subdevice #2
  Sous-périphérique #3: subdevice #3
  Sous-périphérique #4: subdevice #4
  Sous-périphérique #5: subdevice #5
  Sous-périphérique #6: subdevice #6
carte 0: ALSA [bcm2835 ALSA], périphérique 1: bcm2835 ALSA [bcm2835 IEC958/HDMI]
  Sous-périphériques: 1/1
  Sous-périphérique #0: subdevice #0
carte 2: seeed2micvoicec [seeed-2mic-voicecard], périphérique 0: bcm2835-i2s-wm8960-hifi wm8960-hifi-0 []
  Sous-périphériques: 1/1
  Sous-périphérique #0: subdevice #0
```

Ici, dans la partie playback (speaker), nous avons plusieurs interfaces de sortie audio. Nous allons prendre la carte 2, ReSpeaker. Le périphérique est 0. Nous aurons donc comme configuration : 2,0. Faisons la même chose pour le record, là, nous aurons 1,0.

Nous vérifions et configurons si besoin le fichier asoundrc :

```
sudo nano /home/pi/.asoundrc
```

```
pcm.!default {
 type asym
  playback.pcm {
   type plug
   slave.pcm "hw:2,0"
  }
  capture.pcm {
   type plug
   slave.pcm "hw:1,0"
  }
}
```

On redémarre le service alsa-utils : sudo /etc/init.d/alsa-utils restart. Si besoin, ajuster le volume et vérifier la prise en compte des cartes avec l'outil alsamixer.

Si besoin, faites un reboot de la Pi.

Christophe PICHAUD
Architecte Microsoft chez
'M Applications by Devoteam'
christophepichaud@hotmail.com
www.windowscpp.com

Détection faciale et reconnaissance faciale avec OpenCV4 en C++

niveau 300

Les services cognitifs ont le vent en poupe et la détection des visages et leur reconnaissance est un sujet à la mode. Il existe des services comme Azure Cognitive Services et Azure Computer Vision mais aussi des services open-source donc gratuits… à faire tourner en local sans passer par le cloud. On peut aussi y mixer du *machine learning* et de l'IA. C'est ce que nous allons mixer dans l'article de ce mois-ci.

Créée en 2000 par Intel, la librairie OpenCV (Open Source Computer Vision) est une bibliothèque C/C++ temps réel pour le traitement des images. La documentation et les packages Windows, Linux, Mac sont disponibles sur opencv.org. Cette bibliothèque est leader dans son domaine. Elle utilise massivement la STL (Standard Template Library) du C++. Il existe aussi des bindings pour Python, Java, Haskell, Perl, Ruby. Et aussi une version hybride EMGU pour .NET. Il existe aussi deux modes d'accélération matérielle :

- CUDA
- OpenCL

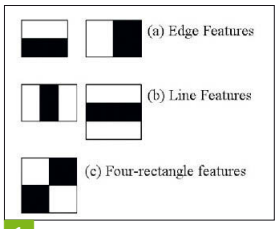
Opérations de bases

La gestion des images requiert des classes particulières. Le namespace cv contient de nombreuses classes C++ :

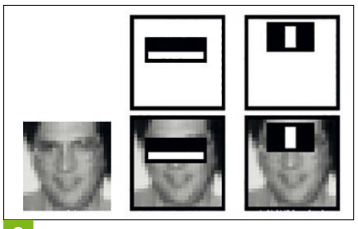
- Scalar pour la couleur ;
- Rect, Point, Size ;
- Mat pour les images.

Détection de visages via Cascades Haar

Commençons par la détection de visages. La détection d'objets à l'aide des classificateurs en cascade basés sur des fonctionnalités Haar est une méthode de détection d'objets efficace proposée par Paul Viola et Michael Jones dans leur article, « Détection rapide d'objets utilisant une cascade boostée de fonctionnalités simples » en 2001. C'est une approche basée sur l'apprentissage par machine où une fonction cascade est formée à partir de beaucoup d'images positives et négatives. Elle est ensuite utilisée pour détecter des objets dans d'autres images. Ici, nous allons travailler avec la détection de visages. Initialement, l'algorithme a besoin de beaucoup d'images positives (images de visages) et d'images négatives (images sans visages) pour former le classificateur. Ensuite, nous avons besoin d'extraire des fonctionnalités de celui-ci. Pour cela, les fonctions Haar affichées dans l'image ci-dessous sont utilisées. Ils sont comme notre noyau à convolution. Chaque fonction est une valeur unique obtenue en soustrayant la somme des pixels sous le rectangle blanc de la somme des pixels sous le rectangle noir. 1
Maintenant, toutes les tailles et les emplacements possibles de chaque noyau sont employés pour calculer beaucoup de dispositifs. Imaginez à quel point il y a besoin de calcul ! Même une fenêtre 24x24 donne des résultats de plus de 160000 fonctionnalités).



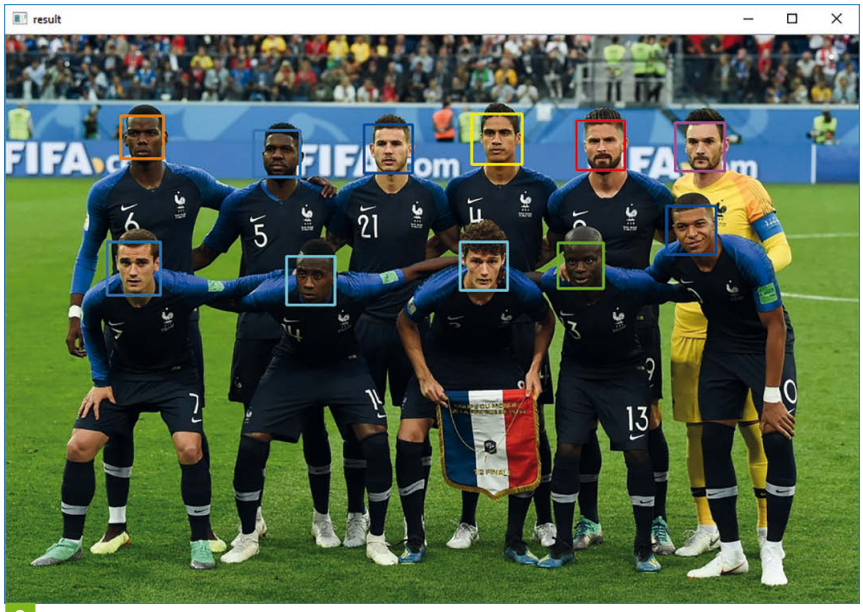
1


2

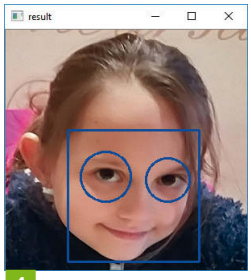
Pour chaque calcul de fonction, nous devons trouver la somme des pixels sous les rectangles blancs et noirs. Pour résoudre ce fait, ils ont introduit l'image intégrale. Quelle que soit la taille de votre image, elle réduit les calculs d'un pixel donné à une opération impliquant seulement quatre pixels. Bien, n'est-ce pas ? Ça rend les choses super rapides.

Mais parmi toutes ces caractéristiques, nous avons calculé : la plupart d'entre eux sont hors de propos. Par exemple, considérez l'image ci-dessous. La rangée du haut montre deux bonnes caractéristiques. La première caractéristique choisie semble se concentrer sur la propriété que la région des yeux est souvent plus sombre que la région du nez et des joues. La deuxième caractéristique choisie repose sur la propriété que les yeux sont plus foncés que le pont du nez. Mais les mêmes fenêtres appliquées aux joues ou à tout autre endroit ne sont pas pertinents. Alors, comment pouvons-nous choisir les meilleures caractéristiques de 160000 et + caractéristiques ? C'est réalisé par AdaBoost. 2

Pour cela, nous appliquons chaque fonctionnalité sur toutes les images de la formation. Pour chaque fonctionnalité, Adaboot trouve le meilleur seuil qui classe les faces positives et négatives. Évidemment, il y aura des erreurs ou des erreurs de classification. Nous sélectionnons les fonctionnalités avec des taux d'erreur minimum, ce qui signifie qu'elles sont les caractéristiques qui classent plus précisément le visage et les autres images. Néanmoins, le processus n'est pas aussi simple que cela. Chaque image se voit attribuée un poids égal au début. Après chaque classification, le poids des images mal classées sont augmentés. Alors le même processus est refait. De nouveaux taux d'erreurs sont calculés. Également de nouveaux poids. Le processus se poursuit jusqu'à ce que le taux d'exactitude ou d'erreur requis soit atteint ou le nombre requis de fonctionnalités sont trouvées.



3



4

Le dernier classificateur correspond à une somme pondérée de ces faibles classificateurs. Elle est qualifiée de faible parce que seul, il ne peut pas classer l'image, mais avec d'autres, il forme un classificateur fort. La documentation dit même que 200 fonctionnalités fournissent la détection avec une précision de 95 %. Leur configuration finale avait environ 6000 caractéristiques. (Imaginez une réduction de 160000 + caractéristiques à 6000 caractéristiques. C'est un gros gain).

Alors maintenant, vous prenez une image. Prendre chaque fenêtre 24 x 24. Appliquez-lui 6000 caractéristiques. Vérifiez si c'est le visage ou pas. Wow... N'est-il pas un peu inefficace et une perte de beaucoup de temps ? Oui. Les auteurs de OpenCV ont une bonne solution pour cela.

Dans une image, la plupart de l'image n'est pas la région du visage. Aussi est-il préférable d'avoir une méthode simple pour vérifier si une fenêtre n'est pas une région du visage. Si ce n'est pas le cas, jetez-le en un seul coup et il ne sera pas traité à nouveau. Au lieu de cela, se concentrer sur les régions où il peut y avoir un visage. De cette façon, nous avons passé plus de temps à vérifier les régions du visage possibles.

Pour cela, ils ont introduit le concept de **Cascade de classificateurs**. Au lieu d'appliquer toutes les 6000 fonctionnalités sur une fenêtre, les fonctions sont regroupées en différents stades des classificateurs et les appliquent un par un (normalement les premières étapes contiennent beaucoup moins de fonctionnalités). Si une fenêtre ne parvient pas à la première étape, jetez-la. Nous ne considérons pas les caractéristiques restantes à ce sujet. Si elle passe, appliquer la deuxième étape de fonctionnalités et ainsi de suite. La fenêtre qui passe toutes les étapes est une région du visage. Voilà le plan !

Codage de la détection

Il suffit de charger une image en mémoire et d'utiliser une routine qui se nomme CascadeClassifier::detectMultiScale. L'utilisation de cette classe doit être faite aussi en faisant appel à load() en lui passant un nom de fichier de cascades. OpenCV fournit ces fichiers de données en standard. Il y en a pour le visage, les yeux, le corps, etc.

```
string cascadeName = "haarcascade_frontalface_alt.xml";
string nestedCascadeName = "haarcascade_eye_tree_eyeglasses.xml";
scale = 1.3;
if (!nestedCascade.load(samples::findFileOrKeep(nestedCascadeName)))
{
    cerr << "Could not load classifier cascade" << endl;
    return 1;
}

if (!cascade.load(samples::findFile(cascadeName)))
{
    cerr << "Could not load classifier cascade" << endl;
    return 1;
}

double scale = 1.3;
string inputName = inputName = "image.jpg";
bool tryflip = false;

Mat image = imread(samples::findFileOrKeep(inputName), IMREAD_COLOR);
if (image.empty())
{
    if (!capture.open(samples::findFileOrKeep(inputName)))
    {
        cout << "Could not read " << inputName << endl;
        return 1;
    }
}

cout << "Detecting face(s) in " << inputName << endl;
if (!image.empty())
{
    detectAndDraw(image, cascade, nestedCascade, scale, tryflip);
    waitKey(0);
}
```

La routine imread() lit le fichier image pour le stocker dans un objet Mat. Ensuite la routine magique detectAndDraw fait le travail magique ! 3 4
Inspectons la routine detectAndDraw() :

```
void detectAndDraw(Mat& img, CascadeClassifier& cascade,
    CascadeClassifier& nestedCascade,
    double scale, bool tryflip)
{
    double t = 0;
    vector<Rect> faces, faces2;
    const static Scalar colors[] =
```

```
    {
        Scalar(255,0,0),
        Scalar(255,128,0),
        Scalar(255,255,0),
        Scalar(0,255,0),
        Scalar(0,128,255),
        Scalar(0,255,255),
        Scalar(0,0,255),
        Scalar(255,0,255)
    };
    Mat gray, smallImg;

    cvtColor(img, gray, COLOR_BGR2GRAY);
    double fx = 1 / scale;
    resize(gray, smallImg, Size(), fx, fx, INTER_LINEAR_EXACT);
    equalizeHist(smallImg, smallImg);

    t = (double)getTickCount();
    cascade.detectMultiScale(smallImg, faces,
        1.1, 2, 0
        //|CASCADE_FIND_BIGGEST_OBJECT
        //|CASCADE_DO_ROUGH_SEARCH
        | CASCADE_SCALE_IMAGE,
        Size(30, 30));
    t = (double)getTickCount() - t;
    printf("detection time = %g ms\n", t * 1000 / getTickFrequency());

    for (size_t i = 0; i < faces.size(); i++)
    {
        Rect r = faces[i];
        Mat smallImgROI;
        vector<Rect> nestedObjects;
        Point center;
        Scalar color = colors[i % 8];
        int radius;

        rectangle(img, Point(cvRound(r.x*scale), cvRound(r.y*scale)),
            Point(cvRound((r.x + r.width - 1)*scale),
cvRound((r.y + r.height - 1)*scale)),
            color, 2, 8, 0);

        if (nestedCascade.empty())
            continue;

        smallImgROI = smallImg(r);
        nestedCascade.detectMultiScale(smallImgROI, nestedObjects,
            1.1, 2, 0
            //|CASCADE_FIND_BIGGEST_OBJECT
            //|CASCADE_DO_ROUGH_SEARCH
            //|CASCADE_DO_CANNY_PRUNING
            | CASCADE_SCALE_IMAGE,
            Size(30, 30));

        for (size_t j = 0; j < nestedObjects.size(); j++)
        {
            Rect nr = nestedObjects[j];
            center.x = cvRound((r.x + nr.x + nr.width*0.5)*scale);
            center.y = cvRound((r.y + nr.y + nr.height*0.5)*scale);
            radius = cvRound((nr.width + nr.height)*0.25*scale);
            circle(img, center, radius, color, 2, 8, 0);
        }
    }

    imshow("result", img);
}
```

La routine fait le job en faisant appel à CascadeClassifier.detect MultiScale pour détecter le visage et ensuite pour détecter les yeux.

Reconnaissance faciale avec OpenCV4

Il est possible de faire trouver à qui appartient une photo donnée. Eh oui ! Pour cela, on va utiliser un module OpenCV qui est dans contrib sur Github et qui se nomme Face.

Le repository Github est ici : **https://github.com/opencv/opencv_contrib**

Dans le répertoire face, il y a du code pour reconnaître les visages suivant 3 techniques :

- Eigen faces ;
- Fisher faces ;
- Local Binary Pattern Histograms.

Utilisation de face

Pour faire les choses dans l'état de l'art, il faut recompiler OpenCV… ou bien incorporer les classes de Face dans votre outil. Comment marche Face ? C'est très simple, il y a trois étapes :

- Générer un modèle à partir de photos d'individus : c'est l'apprentissage ou *training* ;
- Sauvegarder le modèle ou le charger ;
- Faire une prédiction en fonction d'une image quelconque.

L'apprentissage

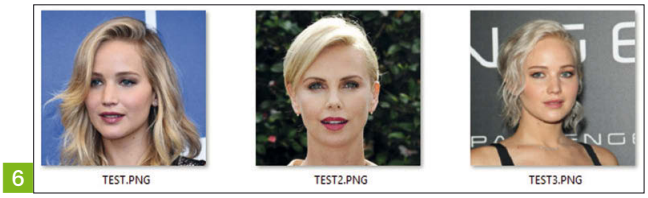
Il faut créer un fichier de configuration CSV dans lequel on met les data comme suit :

Chemin du fichier image ;index ;libellé.

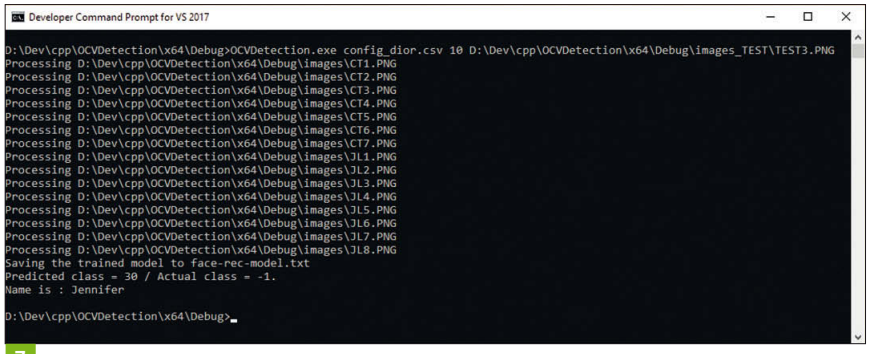
Exemple :

```
D:\Dev\cpp\OCVDetection\x64\Debug\images\CT1.PNG;20;Charlize
D:\Dev\cpp\OCVDetection\x64\Debug\images\CT2.PNG;20;Charlize
D:\Dev\cpp\OCVDetection\x64\Debug\images\CT3.PNG;20;Charlize
D:\Dev\cpp\OCVDetection\x64\Debug\images\CT4.PNG;20;Charlize
D:\Dev\cpp\OCVDetection\x64\Debug\images\CT5.PNG;20;Charlize
D:\Dev\cpp\OCVDetection\x64\Debug\images\CT6.PNG;20;Charlize
D:\Dev\cpp\OCVDetection\x64\Debug\images\CT7.PNG;20;Charlize
D:\Dev\cpp\OCVDetection\x64\Debug\images\JL1.PNG;30;Jennifer
D:\Dev\cpp\OCVDetection\x64\Debug\images\JL2.PNG;30;Jennifer
D:\Dev\cpp\OCVDetection\x64\Debug\images\JL3.PNG;30;Jennifer
D:\Dev\cpp\OCVDetection\x64\Debug\images\JL4.PNG;30;Jennifer
D:\Dev\cpp\OCVDetection\x64\Debug\images\JL5.PNG;30;Jennifer
D:\Dev\cpp\OCVDetection\x64\Debug\images\JL6.PNG;30;Jennifer
D:\Dev\cpp\OCVDetection\x64\Debug\images\JL7.PNG;30;Jennifer
D:\Dev\cpp\OCVDetection\x64\Debug\images\JL8.PNG;30;Jennifer
```

5



6



7

Il y a 7 photos de Charlize Theron. Son indice est 20.
Il y a 8 photos de Jennifer Lawrence son indice est 30. 5

Faire le training consiste à charger l'ensemble des images dans un vector<Mat> et utiliser la méthode train sur un modèle :

```
Ptr<EigenFaceRecognizer> model = EigenFaceRecognizer::create();
    for (int i = 0; i < nlabels; i++)
        model->setLabelInfo(i, labelsInfo[i]);
    model->train(images, labels);
    string saveModelPath = "face-rec-model.txt";
    cout << "Saving the trained model to " << saveModelPath << endl;
    model->save(saveModelPath);
```

Ensuite, on compare une image (passée en argument sur la ligne de commande) en la passant au modèle :

```
    Mat testSample = imread(argv[3], 0);
    int predictedLabel = model->predict(testSample);
    string result_message =
format("Predicted class = %d / Actual class = %d.",
predictedLabel, testLabel);
    cout << result_message << endl;

    auto it = labelsInfo.find(predictedLabel);
    string name;
    if (it != labelsInfo.end())
    {
        name = it->second;
        cout << "Name is : " << name << endl;
    }
```

Voici la liste des images de tests ; les deux premières sont simples mais la troisième n'est pas ressemblante. 6
Je confronte l'image TEST.PNG au modèle et la sortie est la suivante :

```
Predicted class = 30 / Actual class = -1.
Name is : Jennifer
```

Le modèle fait la prédiction que c'est l'indice 30 qui correspond à Jennifer.

Je confronte l'image TEST2.PNG au modèle et la sortie est la suivante :

```
Predicted class = 20 / Actual class = -1.
Name is : Charlize
```

Le modèle fait la prédiction que c'est l'indice 20 qui correspond à Charlize.

Je fais un dernier essai avec une photo peu prédictible de Jennifer, TEST3.PNG :

```
Predicted class = 30 / Actual class = -1.
Name is : Jennifer
```

Le système a quand même fonctionné. Il a prédit la bonne réponse. Magique ! 7

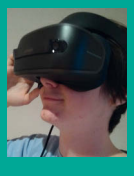
L'objet de l'article n'est pas de documenter l'ensemble des fonctionnalités d'OpenCV mais il est possible d'obtenir « une distance » de résultat. En effet, si je passe une photo de ma fille au module, il va me dire que le résultat est plus proche de telle ou telle personne mais avec une distance de plus de 13 000. Je ne sais pas quelle est l'unité à employer. Mais j'ai remarqué qu'à partir de 10 000, le facture de certitude est de 95%.

Pour rendre les choses ludiques, on couple ces fonctionnalités à une caméra et on fait le traitement pour chaque frame de la vidéo.

Conclusion

OpenCV est une librairie très puissante et passionnante à utiliser. Il y a de nombreuses options que nous n'avons pas couvertes comme la détection d'objets et de formes, les comportements de mouvements, etc.
Si vous êtes intéressé, une seule adresse : https://opencv.org/

•



Maxime Ellerbach
15 ans, lycéen

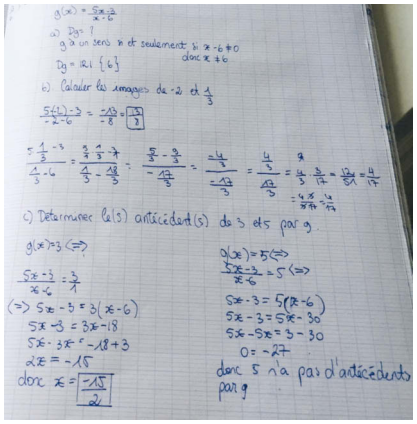
Python : faire une intelligence artificielle pour de la reconnaissance de caractères avec OpenCV et Keras

niveau 200

Ma prof de maths lors d'un cours, en regardant ma copie m'a dit que mon écriture était illisible. Du coup, en rentrant chez moi, je me suis lancé le défi de faire une IA en python qui sache reconnaître mon écriture mais aussi celle de mes amis.

Pour simplifier le code, j'ai décidé uniquement de reconnaître les différents chiffres et les groupes de lettres. Ayant déjà fait une IA reconnaissant les chiffres, j'avais déjà un projet sur lequel je pouvais me baser. Pour faire de la reconnaissance de caractères et de chiffres, il est nécessaire d'utiliser des images servant à entraîner le modèle. Les images du dataset MNIST (https://keras.io/datasets/) regroupent un ensemble de chiffres. Les images ressemblent à ça (chiffre blanc sur fond noir de dimension 28 par 28 pixels) :

Je ne vais pas utiliser ces images car je souhaite utiliser mes propres chiffres ainsi que ceux de mes amis. Je vais réutiliser la philosophie de MNIST sans les données. Le problème est que les images sources que j'ai ne sont que des photos de cahier de mathématiques généreusement fournies par mes amis !



Il me fallait donc trouver un moyen de détecter les caractères et ensuite de découper l'image pour avoir des images semblables à celles de MNIST. C'est là où entre en jeu OpenCV (https://opencv.org/), une librairie spécialisée dans la manipulation et l'analyse d'image.

Récolte de data et préparation des images

Avant toute chose, je suis allé sur le site de documentation d'OpenCV et je suis parti à la recherche d'une fonction magique qui me permettrait de trouver des contours à partir de contrastes dans une image (miracle, elle existe). Le problème c'est que mon image n'est pas assez « propre » pour que cette fonction soit efficace. Il me fallait donc trouver une fonction qui, cette fois-ci, me permette de mettre en évidence les contrastes de mon image (et donc les écritures en même temps).

```
import cv2

img_path = "input_image\\image.jpg"
img = cv2.imread(img_path,1)
```

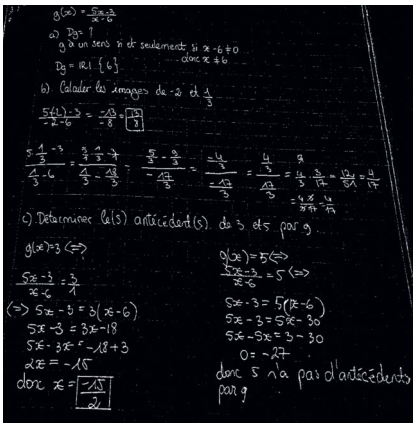
On ouvre notre image en BGR (normalement RGB mais OpenCV aime être diffèrent). On pourrait directement l'ouvrir en nuance de gris, mais par la suite, on utilisera l'image en couleur.

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

On convertit cette image en nuance de gris

```
th = cv2.adaptiveThreshold(img_gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV,31,24)
```

On applique notre fonction magique qui nous transforme notre image en image binaire en fonction des contrastes de l'image. En plus, on transforme en binaire inversé pour avoir les écriture en blanc et le fond en noir.



Notre image est enfin prête pour la détection de contours !

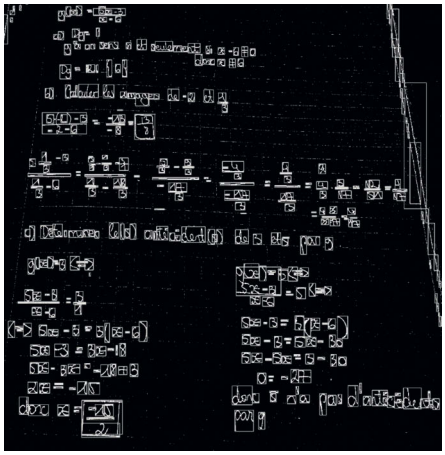
```
im2, cnts, hierarchy = cv2.findContours(th, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Cette fonction nous renvoie 3 variables, dans ce programme nous n'utiliserons que « cnts » qui est une liste contenant toutes les coordonnées de chaque contour.

Pour s'intéresser à chaque contour, on va parcourir la liste « cnts » avec une boucle « for » puis l'on récupère les coordonnés du contour en question. Attention, il nous faut supprimer les petits contours qui peuvent être les traits du cahier, des points, virgules, etc. Pour l'éviter, on met un simple « if » filtrant les contours suivant leur taille :

```
for cont in tqdm(cnts):

    x, y, w, h = cv2.boundingRect(cont)
    if h*w>50:
        cv2.rectangle(th, (x, y), (x+w, y+h), (200), 2)
```

La fonction cv2.rectangle nous permet de visualiser nos contours :



On découpe notre image avec les coordonnées du contour puis on la redimensionne et on l'enregistre avec la date comme nom, ces images composeront notre dataset.

```
img2= th[y-4:y+h+4, x-4:x+w+4]
img2 = cv2.resize(img2,(28,28))
cv2.imwrite('cropped_image\\'+str(time.time())+'.png',img2)
```

Pour que l'image soit plus lisible, on élargit de 4 pixels de chaque côté.

Labellisation

Dans la partie précédente, nous avons récupéré, à partir d'un cahier de mathématique des chiffres, lettres et autres.
Avant de programmer notre IA, il va falloir labelliser les images, c'est long et pénible mais c'est nécessaire au bon fonctionnement de notre IA. Une bonne IA commence d'abord par une bonne labélisation !
Ce que je fais en premier, c'est créer un dossier à part nommé « label » ; ce dossier contiendra nos images triées et labellisées. Dans notre cas, on aura 12 labels, (0,1,2,3,4,5,6,7,8,9 pour les chiffres), 10 correspondra au label « poubelle » contenant des contours indésirables et, enfin, 11 sera notre label écriture, caractères spéciaux et autres.
Dans mon cas, je labellise l'image en rajoutant le label au début de son nom par exemple : 3_1546345453.3818612.png ou encore 5_1546345268.2805524.png
Je trouve que c'est un moyen plus visuel, d'autres préfèreront faire des fichiers Excel ou autres. Dans mon cas j'ai labellisé 5-6 photos de cahier de mathématiques, cela représente environ 1000 images de chiffres, 1000 images d'images non désirées, et environ 500 lettres et caractères spéciaux. Pour augmenter mon nombre et la diversité de mes lettres, j'ai ajouté à mes labels une page de rédaction de français .
Gardez en tête que plus votre dataset sera grand, plus votre IA sera performante sur des images qu'elle ne connaît pas !
Lorsque notre labellisation est terminée, on peut enfin se lancer dans le code ! Mais tout d'abord, un peu de théorie.

Principe de base

Premièrement, pour entraîner un réseau de neurone, il nous faut des données ; dans notre cas nos données sont des valeurs de pixels. Une image de notre dataset correspond à un tableau de 28 par 28 avec comme valeur 0 ou 1 (noir ou blanc). Ensuite, il faut rassembler toutes les images en un tableau. Dans notre cas, le tableau aura comme dimensions : (notre_nombre_d'image, 28, 28,1). Ce tableau est souvent appelé « x ».
On ajoute une dimension à la fin pour avoir 4 dimensions de manière à ce que notre tableau soit compatible avec les fonctions que nous allons utiliser.
Il faut aussi créer un tableau contenant chaque label de chaque image, ce tableau ne contient au départ qu'une seule dimension, puis il aura deux dimensions suite à l'utilisation de la fonction « to_categorical() » dont je vais vous parler dans la partie code. Ce tableau est souvent appelé « y »
On peut donc imaginer deux tableaux « x (5000,28,28,1) » et « y (5000) ».

Note : *x[i] est l'image qui correspond au label y[i].*

Une fois nos tableaux créés, on va les séparer en deux groupes x_train, x_test et y_train, y_test. On prend souvent 10% des « trains », images d'entraînement, comme test. Les trains seront utilisés pour entraîner notre IA et les tests seront utilisés pour évaluer l'efficacité de notre IA sur des images inconnue à ses yeux. Par la suite, On va entraîner l'IA avec nos tableaux sur plusieurs époques. Une époque est un cycle d'entraînement. Faire un train sur 10 époques revient à dire que l'on entraîne notre IA 10 fois sur les mêmes images. Enfin, on enregistre notre IA pour pouvoir la réutiliser par la suite.

Code

Pour tout ce qui concerne notre intelligence artificielle, nous utiliserons « keras ». Keras est une API de réseau de neurones, Keras peut s'utiliser avec Tensorflow, Theano ou encore CNTK. Nous utiliserons Keras avec Tensorflow. En utilisant Keras, la syntaxe change mais le code, au fond reste le même que si nous utilisions Tensorflow !
Voici tous les modules que nous utiliserons pour ce programme :

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from sklearn.model_selection import train_test_split

from glob import glob
import cv2
import time
import numpy as np
```

Préparation de nos images

Création des deux tableaux :

```
x =[]
y =[]
```

```
path = glob('NumberRecognition\\label\\*')
```

Pour charger nos images, on utilise « glob » qui nous permet de créer une liste avec plusieurs chemins d'accès de fichiers, ici, on veut charger toutes les images qui sont dans le dossier « label ».

```
for img_path in dos:

  img = cv2.imread(img_path,0)
  img = cv2.resize(img,(28,28))
  img = np.reshape(img,(28,28,1))

  x.append(img)
  y.append((img_path.split('\\')[-1]).split('_')[0])
```

Pour chaque élément dans notre liste qui contient les chemins d'accès de nos images, on va lire notre image, on va la redimensionner (en principe elles le sont mais je préfère tout de même pour éviter quelques mauvaises surprises). On ajoute une dimension à notre image, puis on ajoute notre image au tableau « x ». Notre label étant dans le nom de notre image, on enlève la partie gauche (on enlève tout ce qui est avant le dernier « \\ ») puis on enlève tout ce qui est après le « _ » pour ne garder que le label. « NumberRecognition\\label\\5_154651 3993.3489306.png » se transforme

donc en « 5 ». On ajoute donc cette valeur à notre tableau « y ».

```
x = np.array(x_train).astype('float32')
x /= 255
```

On convertit le tableau « x » en array avec le module numpy puis on force l'utilisation d'un chiffre flottant. Ensuite on divise « x » par 255 pour normaliser nos valeurs, cela rend l'entraînement plus rapide.

```
y = keras.utils.to_categorical(y,12)
```

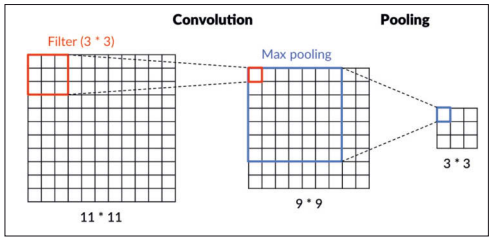
Cette fonction crée des labels sous forme de tableau (ex avec 3 labels : label 1 = [1,0,0] ; label 2 = [0,1,0] ; label3 [0,0,1]). Notre tableau est transformé par cette fonction, il a maintenant une dimension de : (nombre_d'image,12).

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)
```

Cette fonction, à partir de nos deux tableaux « x » et « y », nous crée « x_train », « y_train », « x_test » et « y_test ». Ces deux tableaux « test » sont composés de 10% de « x » et de « y ».
Comme vous l'aurez compris, les tableaux « train » se composent de 90% de « x » et de « y ».
Maintenant que tout est prêt, passons à la partie IA.

Architecture de l'IA

Dans cette partie, nous allons créer notre IA et l'entraîner sur nos propres images. Pour ce projet, nous allons utiliser un CNN (convolutionnal neural network). La convolution est très pratique pour repérer des concepts, des formes. La convolution consiste à « scanner » notre image avec des filtres.
Avec la convolution, on utilisera une couche de « maxpooling », le maxpooling sélectionne la valeur la plus grande dans une fenêtre qui parcourt notre image, cette couche nous permet de diminuer la taille de notre image qui sort de la convolution et donc de diminuer la puissance de calcul requise pour faire fonctionner notre réseau de neurones. Voici ce que fait typiquement une couche de convolution couplée avec une couche de maxpooling :



Source : https://saitoxu.io/2017/01/01/convolution-and-pooling.html
Après avoir appliqué des filtres et fait du maxpooling, on utilise « Flatten » qui nous permet de passer d'une image 2D à une image aplatie. Les colonnes de pixels sont mises bout à bout dans une seul liste. La couche de neurones « Dense » fonctionne uniquement avec une seule dimension. Après cela, nous utiliserons deux simples couches de neurones type « Dense » ce sont des couches de neurones interconnectés. Ces couches aboutiront à une dernière couche « Dense » d'activation « softmax », c'est notre dernière couche, elle sera dotée d'autant de neurones que l'on a de labels. Maintenant passons au code :

```
model = Sequential()
```

C'est le type de modèle le plus accessible et le plus simple à coder. Les couches s'ajoutent les unes après les autres.

```
model.add(Conv2D(32, (3, 3), activation='relu',input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(3, 3))
model.add(Conv2D(128, (2, 2), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))
```

On ajoute donc deux fois de la convolution et du maxpooling. La première fois, on applique 32 filtres différents et la taille du filtre est de 3x3 tout comme la taille du maxpooling. La seconde fois on utilise plus de filtres et la taille du filtre est légèrement plus petite. La première convolution permet de repérer les « grands traits » de l'image que l'on analyse plus en profondeur avec la seconde convolution.

```
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(12, activation='softmax'))
```

On aplatit notre image filtrée, puis comme indiqué précédemment, on ajoute les dernières couches en finissant par une couche « Dense » avec 12 neurones. Il faut toujours finir par une couche de neurones ayant le même nombre de labels que l'on désire ;nous faisons un classificateur.

Note : *J'ai une version GPU de Tensorflow et j'ai un GPU puissant. Si vous utilisez ce programme avec un CPU, vous pouvez diminuer ces différents éléments pour accélérer le calcul.*

On peut ainsi regarder la forme de notre modèle (nombres de neurones, formes de nos images à la sortie de chaque couche…) avec :

```
model.summary()
```

Qui nous donne :

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 128)         16512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               147584
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_3 (Dense)              (None, 12)                1548
=================================================================
Total params: 182,476
Trainable params: 182,476
Non-trainable params: 0
```

Il est temps d'entraîner notre IA sur notre dataset :

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(x_train, y_train, batch_size= 64, epochs=30, verbose=1, validation_data=(x_test, y_test))
```

Avant l'entraînement, on précise quelle fonction doit utiliser le programme pour calculer la « loss » (la perte ou de combien se trompe notre IA) et on doit préciser l'optimiseur. Plus d'informations sont disponibles sur les optimiseurs et algorithmes pour calculer la loss sur : https://keras.io/optimizers/ et : https://keras.io/losses/ . Pour entraîner, on utilise « model.fit », on précise nos données d'entraînement (x_train, y_train) et nos données de test (x_test, y_test). On précise le batch size (c'est le nombre d'images que l'on traite en une fois) et l'on choisit 30 époques, ce qui laisse le temps à notre IA de s'entraîner correctement. J'ajoute une dernière ligne de code qui nous permet d'enregistrer notre IA pour la réutiliser ultérieurement :

```
model.save('AI.h5')
```

On lance le programme ! Si tout se passe bien , on obtient au bout de 30 époques quelque chose de ce style-là :

```
Epoch 30/30
3120/3120 [==============================] - 0s 95us/step - loss: 0.0031 - acc: 0.9997 - val_loss: 0.1561 - val_acc: 0.9654
```

Sur ma machine, cet entraînement a pris une vingtaine de secondes à s'exécu-

ter. Sur un CPU, cela peut prendre plusieurs minutes. Ici, les résultats sont très satisfaisants, on s'intéressera uniquement aux résultats sur les images de test qui sont la « val_acc ». La val_acc (value accuracy) est le pourcentage de bons résultats sur les images de test (lorsque l'IA reconnaît le bon label). Ici on a 96% de bons résultats !

Utilisation de notre IA pour prédire nos chiffres

Maintenant que notre IA est entrainée, on peut l'utiliser !
On va donc reprendre le programme qui nous extrait nos chiffres/lettres à partir d'une image (vu dans la première partie) et ajouter quelques lignes pour tester les performances de notre IA !
Au début de notre programme (en dessous de nos imports par exemple), on ajoute cette ligne qui va charger notre IA afin de l'utiliser par la suite.

```
model = load_model('AI.h5')
```

Il faut ajouter la seconde ligne :

```
    img2 = cv2.resize(img2,(28, 28))
    img_pred = np.reshape(img2,(1, 28, 28, 1))
```

Et on prédit le label de notre image « img_pred » à l'aide de notre IA avec :

```
    predicted = model.predict(img_pred)
```

Lorsque l'on affiche la valeur de predicted, on obtient quelque chose comme cela : [[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
Il s'agit de la probabilité de chaque label. Pour obtenir le label, on utilise :

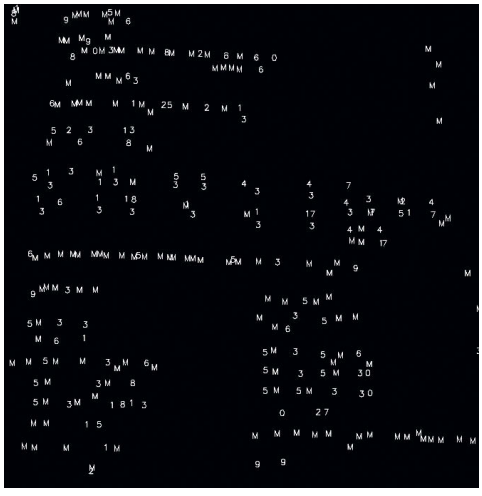
```
predicted = np.argmax(predicted)
```

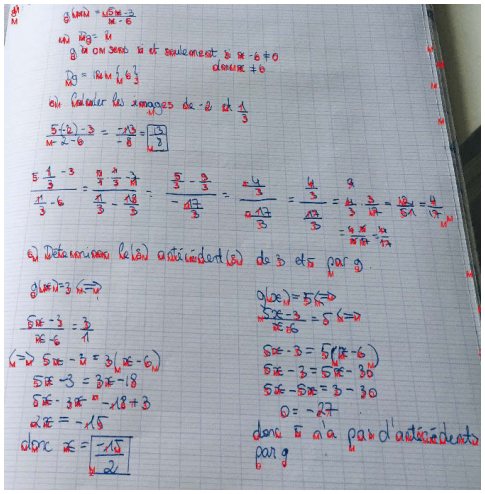
Cette fonction sélectionne la valeur la plus élevée c'est-à-dire, la prédiction la plus probable dans l'exemple précèdent, on obtient le label 2
Pour visualiser nos résultats, on peut enregistrer notre image sous le nom de sa prédiction et de la date

```
cv2.imwrite('cropped_image\\'+str(predicted)+'_'+str(time.time())+'.png',img2
```

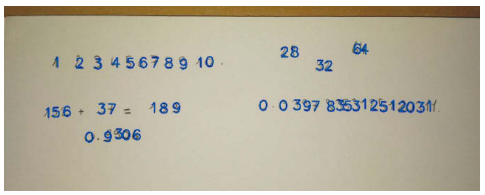
Et l'on obtient par exemple :

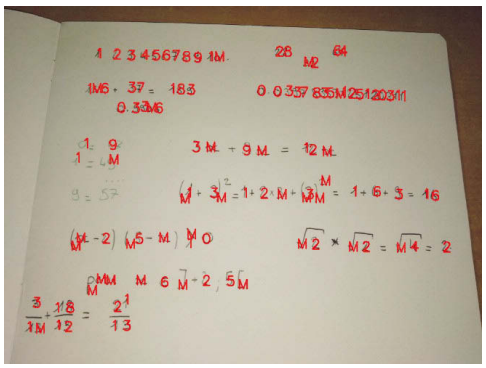
Avec comme noms respectifs 1_1546163109.7581706.png, 5_1546163161.4657226.png et 0_1546163160.984996.png
Pleins d'autres exemples de visualisation peuvent être imaginées comme écrire le label sur une nouvelle image de même taille que l'originale ou écrire directement sur l'image :





Voici d'autres résultats sur d'autres images :





Sur ces exemples, la lettre « M » représente un caractère, notre label 11. Je ne fais pas apparaître le label 10 car c'est notre poubelle !

Conclusion

Pour ce projet, la difficulté principale a été de trouver de bonnes fonctions pour « nettoyer » l'image de tous parasites tels que les carreaux ou la luminosité. Je suis conscient que cette intelligence artificielle n'est pas la meilleure en termes d'efficacité et de résultats. Je voulais vous proposer mon approche combinant un peu de computer vision et d'intelligence artificielle pour développer ma propre OCR en Python.
Si vous voulez aller plus loin dans l'utilisation des algorithmes d'IA et l'utilisation d'OpenCV, consultez la documentation d'OpenCV et Keras. Il y a de nombreux exemples et tutoriels permettant d'utiliser la plupart des fonctions. Je vous recommande également de lire un peu de théorie de façon à bien comprendre quand utiliser tel ou tel algorithme et leur domaine de spécialité.
Par la suite, j' ajouterai peut-être d'autres labels comme les signes opératoires, les parenthèses et autres. Le projet peut être retrouvé sur mon GitHub : https://github.com/Maximellerbach/NumberRecognition

Sébastien Angelloz-Nicoud,
Techleader JEE Osaxis

JDBI : se faciliter l'utilisation de JDBC

Partie 1

Lorsqu'il s'agit de récupérer des données depuis une base relationnelle et de manipuler les objets correspondants, il est souvent fait mention des ORM (Object Relational Mapping). L'utilisation d'un ORM n'est pas toujours la solution retenue ou la plus adaptée. Dans l'univers Java, la solution la plus directe est l'interface JDBC. Cependant, l'utilisation de cette API de bas-niveau peut s'avérer fastidieuse.

niveau 100

JDBI propose un accès pratique et idiomatique aux bases de données relationnelles par l'intermédiaire de JDBC. La version de JDBI présentée ci-après est la version 3, tirant profit des fonctionnalités de la version 8 de Java.

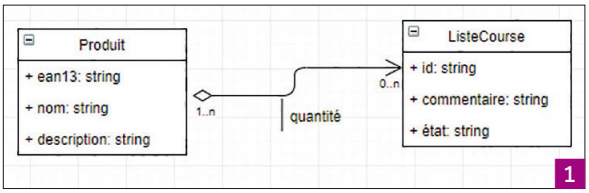
Utilisation par l'exemple

Prenons pour exemple une application permettant de gérer des listes de courses. Une liste de courses se caractérise par un identifiant, un état (ouverte ou terminée), ainsi qu'un commentaire. Elle est composée de produits à acheter en une certaine quantité. 1
Ce modèle se traduit par le schéma SQL suivant :

```
CREATE TABLE produit (
  ean13      CHAR(13) NOT NULL CHECK(length(ean13) = 13),
  nom        VARCHAR(100) NOT NULL,
  description VARCHAR(255) NOT NULL,
  PRIMARY KEY(ean13)
);

CREATE TABLE liste_course (
  id         CHAR(36) NOT NULL CHECK(length(id) = 36),
  commentaire VARCHAR(255) NOT NULL,
  etat       CHAR(8) NOT NULL CHECK(etat in ('OUVERTE', 'TERMINEE')),
  PRIMARY KEY(id)
);

CREATE TABLE liste_course_item (
  liste_course_id  CHAR(36) NOT NULL CHECK(length(liste_course_id) = 36),
  produit_ean13    CHAR(36) NOT NULL CHECK(length(produit_ean13) = 13),
  quantite         DECIMAL(10,2) NOT NULL,
  PRIMARY KEY(liste_course_id, produit_id),
  FOREIGN KEY(produit_ean13) REFERENCES produit(ean13),
  FOREIGN KEY(liste_course_id) REFERENCES liste_course(id)
);
```

Connexion et démarrage

La classe JDBI est le point d'entrée de la librairie. Cette classe permet d'abstraire la source de données JDBC faisant référence à la base de données à utiliser.
Il est possible de créer cette instance grâce à une référence directe à la base de données ou encore d'utiliser une source de données et ainsi de pouvoir également bénéficier d'un pool de connexions :



1

```
// Création d'une instance JDBI directement connectée à la base de données
JDBI jdbi = JDBI.create("jdbc:h2:mem:jdbi_article");

// Utilisation d'une source de données
final Context ctx = new InitialContext();
final DataSource ds = (DataSource) ctx.lookup("Java:comp/env/jdbi");
JDBI jdbi = JDBI.create(ds);
```

Premières requêtes

L'exécution de requêtes passe par la manipulation de l'objet « Handle », représentant une connexion active à la base de données.
Cet objet permet d'exécuter tous les types de requêtes ainsi que de gérer les transactions. Il fournit des méthodes chaînées (fluent API), permet d'associer les valeurs nécessaires aux arguments des requêtes, d'exécuter l'instruction SQL et d'en récupérer le résultat (sous forme d'objets ou types primitifs).
L'instruction « createUpdate » exécute une requête modifiant les données (Insert, Update ou Delete) et offre la possibilité de paramétrer les éléments de la requête.

```
// Insertion d'un enregistrement dans la table produit
this.jdbi.useHandle(handle ->
  handle.createUpdate("INSERT INTO produit (ean13, nom, description) VALUES (:ean13,
:nom, :description)")
    .bind("ean13", "1234567891011")
    .bind("nom", "Soda")
    .bind("description", "Soda rafraîchissant en bouteille")
    .execute()
);
```

La récupération de données se fait de façon similaire avec la méthode « createQuery » :

```
// Récupération d'un produit grâce à sa clef primaire
Produit soda = this.jdbi.withHandle(handle ->
```

```
handle.createQuery("SELECT ean13, nom, description FROM produit WHERE ean13 = :ean13")
    .bind("ean13", "1234567891011")
    .map((resultSet, ctx) -> new Produit(
      resultSet.getString("ean13"),
      resultSet.getString("nom"),
      resultSet.getString("description")))
    .findOnly()
);
```

La méthode « findOnly() » considère que le résultat de la requête doit être composé d'une seule et unique ligne. Dans le cas contraire, une exception sera levée.
La méthode « findOne() » retourne la première ligne du résultat de la requête (si elle existe) sous la forme d'un « Optional ». Si le résultat contient plusieurs lignes, seule la première ligne est traitée, les autres sont ignorées.
Enfin une méthode « list() » récupère l'ensemble des lignes du résultat sous forme de liste. Cette liste peut être vide si le résultat de la requête l'est également.
Notons ici la différence entre les méthodes « useHandle » et « withHandle ». La méthode « useHandle » prenant en paramètre un « Consumer » ne renvoie aucun résultat, alors que la méthode « withHandle » attendant en paramètre un « Callback » permet le retour d'un résultat.
A noter également qu'il aurait été possible pour l'instruction d'insertion d'utiliser un « Callback » afin de récupérer le nombre d'éléments affectés par l'instruction « createUpdate » (retour de la méthode« execute() »). Cette fonctionnalité est particulièrement utile dans le cas d'une requête de mise à jour pour connaître le nombre d'enregistrements modifiés par la requête.
L'objet « Handle » propose aussi des méthodes pour les traitements « batch » avec « createBatch » et « preparedBatch » et l'exécution de procédures stockées avec « createCall ».

Paramétrage des requêtes

JDBI permet de spécifier les arguments des requêtes de plusieurs manières. La plus courante est la possibilité de valoriser les arguments par leur position (index) dans la requête, ou par leur nom.

```
// Paramétrage des éléments de la requête par leurs positions
this.jdbi.useHandle(handle ->
  handle.createUpdate("INSERT INTO produit (ean13, nom, description) VALUES (?, ?, ?)")
    .bind(0, "1230565891412")
    .bind(1, "Eau gazeuse")
    .bind(2, "Bulles et fraîcheur")
    .execute()
);
```

JDBI est également capable de lier la valeur des attributs d'un objet aux paramètres nommés de la requête.

```
Produit gateau = new Produit("1458752200125", "Gâteau moelleux", "Pur beurre");

this.jdbi.useHandle(handle ->
  handle.createUpdate("INSERT INTO produit (ean13, nom, description) VALUES (:ean13,
:nom, :description)")
    .bindBean(gateau)
    .execute()
);
```

Un objet est fourni pour le paramétrage de la requête. Les paramètres sont valorisés par les attributs de l'objet.
Une autre variante est d'utiliser une simple « Map » ayant pour clef le nom des paramètres.

```
// Utilisation d'une map avec pour clef le nom des paramètres
Map<String, Object> params = new HashMap<>();
params.put("ean13", "1854785213654");
params.put("nom", "Dentifrice menthol");
params.put("description", "Prévient les carries");

this.jdbi.useHandle(handle ->
  handle.createUpdate("INSERT INTO produit (ean13, nom, description) VALUES (:ean13,
:nom, :description)")
    .bindMap(params)
    .execute()
);
```

Finalement, il est possible d'utiliser le résultat de l'appel des méthodes de l'objet, le nom du paramètre doit alors correspondre au nom d'une méthode de l'objet. Ce mécanisme est permis grâce à « bindMethods » et à la réflexion.
Ajouter à cela que les méthodes « bindBean » et « bindMethods » permettent d'accéder aux propriétés d'objets composant l'objet racine. En imaginant que l'objet « Produit » ait pour attribut un objet « Fabricant », il est possible de lier les paramètres aux valeurs de l'objet de cette manière :

```
// Utilisation de propriétés imbriquées à l'objet pour la définition de la valeur d'un paramètre
this.jdbi.useHandle(handle ->
  handle.createUpdate("INSERT INTO produit (ean13, nom, description) VALUES (:p.ean13, :p.nom,
            :p.fabricant.descriptionParLeFabricant)")
    .bindBean("p", produit)
    .execute()
);
```

Mappers

Lors de la récupération du résultat d'une requête, les enregistrements sont très souvent transformés en objet. Cette transformation est effectuée par l'instruction « .map() » comme illustré lors de la récupération de l'objet « soda » dans le chapitre « Premières requêtes ».
JDBI permet l'externalisation de ces opérations de mapping afin d'éviter de les définir à chaque instruction SQL.
JDBI distingue deux types de mapper, les « RowMapper » permettant de transformer une ligne du résultat de la requête en un objet et les « ColumnMapper » permettant de transformer une colonne de la ligne courante du résultat.

La suite de cet article dans le prochain numéro

Nicolas Lepage
Consultant chez Zenika Nantes
Développeur sur plusieurs langages et formateur Go.
Auteur de immutadot, une bibliothèque JS pour manipuler les structures immuables imbriquées.

zenika

Go 1.11: WebAssembly pour les Gophers

En février 2017, Brad Fitzpatrick, membre de l'équipe travaillant sur le langage Go chez Google, ouvre la question du support de WebAssembly sur le dépôt Github golang/go. Quatre mois plus tard, Richard Musiol, auteur de GopherJS, entre dans la discussion avec quelques idées "naïves" sur la manière d'ajouter WebAssembly à la liste des plateformes cibles pour le compilateur Go… pour enfin poster début novembre 2017 un court message : "fyi: I have started implementing this".

niveau 200

L'issue sera finalement fermée en juin 2018, et le tout nouveau support de WebAssembly fusionné dans la branche master du dépôt golang/go, pour être officiellement mis à disposition dans la version 1.11 de Go en août dernier.
C'est l'occasion de combiner deux de mes langages préférés. Jetons un oeil aux possibilités de cette première implémentation.
Tous les exemples de cet article peuvent être facilement exécutés en utilisant l'image docker nlepage/golang_wasm:examples :

```
$ docker container run -dP nlepage/golang_wasm:examples
ce04d7bb70387e5aa3ea668880b807e5c29700e580b3ea4e600fe8abda081e69
$ # Utiliser docker container port pour connaître le port d'écoute
$ docker container port ce04
80/tcp -> 0.0.0.0:32768
```

Puis en visitant http://localhost:32XXX/ et en suivant les liens vers les différents exemples. Les sources sont également disponibles sur le dépôt Github http://github.com/nlepage/golang-wasm.

Hello Wasm!

Commençons par un simple "Hello World!", ou plutôt "Hello Wasm!".
La première chose nécessaire est un toolkit Go à jour (1.11 ou supérieur), disponible sur https://golang.org/dl/. Il est également possible d'utiliser une image docker du dépôt officiel Go.
Une fois l'installation d'un toolkit à jour effectuée, il ne reste plus qu'à écrire un traditionnel helloworld.go :

```
package main

import (
  "fmt"
)

func main() {
  fmt.Println("Hello Wasm!")
}
```

Puis le compiler avec la commande suivante :

```
GOOS=js GOARCH=wasm go build -o test.wasm helloworld.go
```

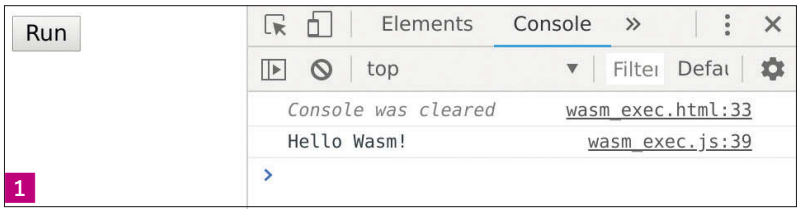
Ou encore en utilisant un Dockerfile comme celui-ci :

```
FROM golang:1.11
COPY ./ src/hello/
ENV GOOS=js GOARCH=wasm
RUN go build -o test.wasm hello
```

Ce sont les variables d'environnement GOOS et GOARCH utilisées pour la cross-compilation qui nous permettent de demander au compilateur de produire un binaire wasm.
L'étape finale consiste à utiliser les fichiers wasm_exec.html et wasm_exec.js, disponibles dans le répertoire misc/wasm du toolkit, pour exécuter test.wasm dans le navigateur (wasm_exec.js s'attend à ce que le binaire soit nommé test.wasm, d'où l'utilisation de ce nom).
Une fois les 3 fichiers servis, en utilisant Nginx par exemple, wasm_exec.html affiche un bouton "Run", actif uniquement si le chargement de test.wasm a réussi.
Attention, le fichier test.wasm doit être servi avec le type MIME "application/wasm", sans quoi le navigateur peut refuser de le charger (nginx par exemple nécessite une adaptation du fichier mime.types).
L'image docker nlepage/golang_wasm:nginx dispose du type MIME correct, et des fichiers wasm_exec.html et wasm_exec.js dans le répertoire /usr/share/nginx/html/ .
Après un clic sur le bouton "Run", la console du navigateur devrait afficher le message "Hello Wasm!" (wasm_exec.js capture la sortie standard du binaire wasm et appelle console.log() pour chaque nouvelle ligne). 1
Les sources de cet exemple sont disponibles à l'adresse https://github.com/nlepage/golang-wasm/tree/master/examples/hello.

Appeler JS depuis Go

Maintenant que nous avons réussi à exécuter notre premier binaire WebAssembly compilé à partir de sources Go, explorons plus avant



les capacités de cette première version du support de wasm par Go. Un nouveau package "syscall/js" a fait son entrée dans la bibliothèque standard de Go, jetons un oeil à l'API de ce package (https://golang.org/pkg/syscall/js/).

Le package "syscall/js" est centré autour d'un nouveau type js.Value, qui représente une référence JavaScript. Ce type porte un ensemble de méthodes permettant d'interagir de manière simple avec une référence JavaScript :

- js.Value.Get() et js.Value.Set() lisent et écrivent des propriétés sur une référence de type Object ;
- js.Value.Index() et js.Value.SetIndex() lisent et écrivent des valeurs dans une référence de type Array ;
- js.Value.Length() lit la taille d'une référence de type Array ;
- js.Value.Call() appelle une méthode sur une référence (de type Object, Array, ou autre...) ;
- js.Value.Invoke() effectue un appel sur une référence de type Function ;
- js.Value.New() exécute l'opérateur new sur une référence représentant un type JS ;
- js.Value.InstanceOf() vérifie si une référence est du type JS donné en paramètre ;
- js.Value.Type() renvoie le type de la référence, équivalent de l'appel à l'opérateur JS typeof ;
- Quelques méthodes supplémentaires, telles que js.Value.Int() ou js.Value.Bool(), permettant de "caster" une référence JS en type natif Go.

Enfin, quelques fonctions intéressantes :

- js.Global() renvoie la js.Value donnant accès au scope global JS ;
- js.Undefined() renvoie la js.Value correspondant au undefined JS ;
- js.Null() renvoie la js.Value correspondant au null JS ;
- js.ValueOf() accepte n'importe quel type natif Go et renvoie la js.Value correspondante.

À la place d'envoyer notre message sur la sortie standard, affichons le dans un dialogue en utilisant window.alert() de JS.

Comme nous sommes dans un navigateur, le scope global est la fenêtre, donc nous devons commencer par récupérer alert() depuis le scope global :

```
alert := js.Global().Get("alert")
```

Nous avons maintenant une variable alert typée js.Value qui est une référence au window.alert() de JS, nous pouvons utiliser js.Value.Invoke() pour l'appeler :

```
alert.Invoke("Hello Wasm!")
```

Il n'est pas nécessaire d'utiliser js.ValueOf() sur les paramètres envoyés à Invoke(), celle-ci accepte un variadic de interface{} et passe les valeurs par js.ValueOf() à notre place.

Notre programme final est le suivant :

```
package main

import (
  "syscall/js"
)

func main() {
  alert := js.Global().Get("alert")
  alert.Invoke("Hello Wasm!")
}
```

Pour compiler et exécuter, la procédure est strictement la même que pour notre premier exemple.

Le clic sur le bouton "Run" devrait maintenant ouvrir un dialogue avec notre message.

Les sources de cet exemple sont disponibles à l'adresse https://github.com/nlepage/golang-wasm/tree/master/examples/js-call.

Appeler Go depuis JS

Appeler JS depuis Go est assez facile, maintenant explorons plus avant l'API du package "syscall/js".

Le type js.Callback représente une fonction Go encapsulée afin de pouvoir être utilisée comme une fonction JS.

La fonction js.NewCallback() prend en paramètre une fonction Go acceptant en paramètre d'entrée une slice de js.Value (sans paramètre de retour) et renvoie un js.Callback.

Le type js.Callback porte également une méthode Release() qui doit être appelée quand un callback ne sera plus utilisé afin de libérer les ressources associées.

En bonus, on trouve une fonction js.NewEventCallback() facilitant la réception d'événements JS.

Donnons-nous un objectif simple, déclencher un fmt.Println() depuis le "côté JS".

Nous allons devoir commencer par quelques ajustements dans wasm_exec.html, nous devons être en mesure de réceptionner le callback envoyé depuis Go afin de pouvoir ensuite l'appeler.

Avant tout, ajoutons un input pour personnaliser le message à afficher dans la console :

```
<input id="messageInput" type="text" value="Hello Wasm!">
<button onClick="run();" id="runButton" disabled>Run</button>
```

Pour le moment la fonction run() qui exécute le binaire wasm lors de l'appui sur le bouton "Run" ressemble à ça :

```
async function run() {
 console.clear()
 await go.run(inst)
 inst = await WebAssembly.instantiate(mod, go.importObject)
}
```

Elle démarre le binaire et attend que celui-ci termine son exécution, puis elle le ré-instancie en vue du prochain appui sur le bouton "Run".

Adaptons la fonction run() afin de réceptionner le callback Go avant de l'appeler :

```
var setPrintMessage
async function run() {
 console.clear()
 // Créer la promesse et stocker sa fonction de résolution
 const printMessagePromise = new Promise(resolve => {
```

```
  setPrintMessage = resolve
})
const run = go.run(inst)
// Attendre la réception du callback
const printMessage = await printMessagePromise
// Appeler le callback !
printMessage(document.querySelector('#messageInput').value)
await run
inst = await WebAssembly.instantiate(mod, go.importObject)
}
```

Préalablement à l'exécution du binaire nous créons la promesse de réception du callback, puis suite au démarrage du binaire nous attendons la résolution de cette promesse pour réceptionner et appeler le callback.
L'utilisation du mot clé var directement dans la balise <script> rend la référence setPrintMessage accessible dans le scope global et donc pour le binaire wasm.
Et c'est bon pour la partie JS !
Maintenant côté Go, nous devons créer le callback, l'envoyer côté JS, et attendre que celui-ci soit appelé.
Tout d'abord déclarons un channel permettant de signaler que notre callback a été appelé :

```
var done = make(chan struct{})
```

Ensuite nous devons écrire la fonction Go printMessage() qui servira de callback :

```
func printMessage(args []js.Value) {
  message := args[0].String()
  fmt.Println(message)
  done <- struct{}{} // Signaler que printMessage a été appelée
}
```

Les arguments sont reçus dans une slice de js.Value, nous devons donc appeler js.Value.String() sur le premier élément de la slice afin de récupérer le message sous forme d'une chaîne Go.
Nous pouvons maintenant encapsuler cette fonction dans un callback :

```
callback := js.NewCallback(printMessage)
defer callback.Release()
```

L'utilisation de defer pour remettre le "nettoyage" du callback à la fin de l'exécution de la fonction main() est une bonne pratique.
Ensuite nous devons appeler la fonction JS setPrintMessage(), de la même manière que lorsque nous avons appelé window.alert() :

```
setPrintMessage := js.Global().Get("setPrintMessage")
setPrintMessage.Invoke(callback)
```

Enfin il ne nous reste plus qu'à attendre l'appel du callback :

```
<-done
```

Cette dernière étape est importante, en effet les callbacks sont exécutés dans une goroutine dédiée, il faut donc mettre la main goroutine "en attente", sans quoi l'exécution du binaire wasm se terminera prématurément.
Le résultat final est le suivant :

```
package main

import (
  "fmt"
  "syscall/js"
)

var done = make(chan struct{})

func main() {
  callback := js.NewCallback(printMessage)
  defer callback.Release()
  setPrintMessage := js.Global().Get("setPrintMessage")
  setPrintMessage.Invoke(callback)
  <-done
}

func printMessage(args []js.Value) {
  message := args[0].String()
  fmt.Println(message)
  done <- struct{}{}
}
```

Comme pour nos exemples précédents, il faut compiler un binaire nommé test.wasm, mais cette fois le fichier wasm_exec.html doit être remplacé par notre version modifiée, et nous pouvons utiliser le fichier wasm_exec.js d'origine tel quel.
Maintenant au clic sur le bouton "Run", comme pour notre premier exemple le message s'affiche dans la console du navigateur, cependant il est possible de le personnaliser puisque sa valeur est fournie par l'input HTML.
Les sources de cet exemple sont disponibles à l'adresse https://github.com/nlepage/golang-wasm/tree/master/examples/go-call.

Exécution prolongée

Un appel de Go depuis JS est un peu plus compliqué qu'un appel de JS depuis Go, en particulier du côté JS.
Cela est en partie dû au fait que les callbacks ont une exécution asynchrone.
Essayons une approche un peu différente : pourquoi ne pas avoir un binaire wasm qui ne s'arrête pas juste après l'appel du callback, mais qui continue à s'exécuter afin de recevoir d'autres appels ?
Cette fois commençons par la partie Go, comme dans notre exemple précédent nous avons besoin d'un callback à envoyer côté JS.
Nous allons ajouter un compteur d'appels afin de garder en mémoire combien de fois le callback a été appelé.
Notre nouvelle fonction printMessage() va afficher sur la sortie standard le message reçu et la valeur du compteur d'appels :

```
var no int

func printMessage(args []js.Value) {
  message := args[0].String()
  no++
  fmt.Printf("Message no %d: %s\n", no, message)
}
```

La création du callback et son envoi côté JS est strictement identique à notre exemple précédent :

```
callback := js.NewCallback(printMessage)
defer callback.Release()
setPrintMessage := js.Global().Get("setPrintMessage")
setPrintMessage.Invoke(callback)
```

Mais cette fois nous n'avons pas le channel `done` pour signaler à la main goroutine quand se terminer. Une solution pourrait être de la bloquer indéfiniment avec un `select` vide :

```
select {}
```

Ce n'est pas très satisfaisant, notre binaire wasm ne s'arrêtera jamais "proprement" et son exécution sera probablement interrompue de force au déchargement de wasm_exec.html par le navigateur.
Nous allons plutôt attendre l'évènement `beforeunload` de la page, il nous faut donc un second callback pour recevoir l'évènement et le signaler à la main goroutine via un channel :

```
var beforeUnloadCh = make(chan struct{})

func beforeUnload(event js.Value) {
  beforeUnloadCh <- struct{}{}
}
```

Cette fois notre fonction `beforeUnload()` ne prend pas une slice de `js.Value`, mais un seul paramètre `js.Value`, en effet nous allons utiliser `js.NewEventCallback()` pour créer le callback et l'enregistrer côté JS :

```
beforeUnloadCb := js.NewEventCallback(0, beforeUnload)
defer beforeUnloadCb.Release()
addEventListener := js.Global().Get("addEventListener")
addEventListener.Invoke("beforeunload", beforeUnloadCb)
```

Enfin nous devons remplacer le `select` vide par une réception sur le channel `beforeUnloadCh` :

```
<-beforeUnloadCh
fmt.Println("Bye Wasm !")
```

Le résultat final est le suivant :

```
package main

import (
  "fmt"
  "syscall/js"
)

var (
  no          int
  beforeUnloadCh = make(chan struct{})
)

func main() {
  callback := js.NewCallback(printMessage)
  defer callback.Release()
  setPrintMessage := js.Global().Get("setPrintMessage")
  setPrintMessage.Invoke(callback)

  beforeUnloadCb := js.NewEventCallback(0, beforeUnload)
  defer beforeUnloadCb.Release()
  addEventListener := js.Global().Get("addEventListener")
  addEventListener.Invoke("beforeunload", beforeUnloadCb)

  <-beforeUnloadCh
  fmt.Println("Bye Wasm !")
}

func printMessage(args []js.Value) {
  message := args[0].String()
  no++
  fmt.Printf("Message no %d: %s\n", no, message)
}

func beforeUnload(event js.Value) {
  beforeUnloadCh <- struct{}{}
}
```

Maintenant côté JS, le chargement du binaire wasm était jusque-là effectué comme suit :

```
const go = new Go();
let mod, inst;
WebAssembly.instantiateStreaming(
  fetch("test.wasm"),
  go.importObject).then((result) => {
    mod = result.module;
    inst = result.instance;
    document.getElementById("runButton").disabled = false;
  }
);
```

Adaptons le afin de démarrer le binaire tout de suite après son chargement :

```
(async function() {
  const go = new Go()
  const { instance } = await WebAssembly.instantiateStreaming(
```

```
    fetch("test.wasm"),
    go.importObject
  )
  go.run(instance)
})()
```

Et remplaçons le bouton "Run" par un bouton pour déclencher printMessage() :

```
<input id="messageInput" type="text" value="Hello Wasm!">
<button
  onClick="printMessage(document.querySelector('#messageInput').value)"
  id="printMessageButton"
  disabled
>
  Print message
</button>
```

Enfin, il nous faut cette fois une fonction setPrintMessage() pour stocker le callback qui sera réutilisé plusieurs fois :

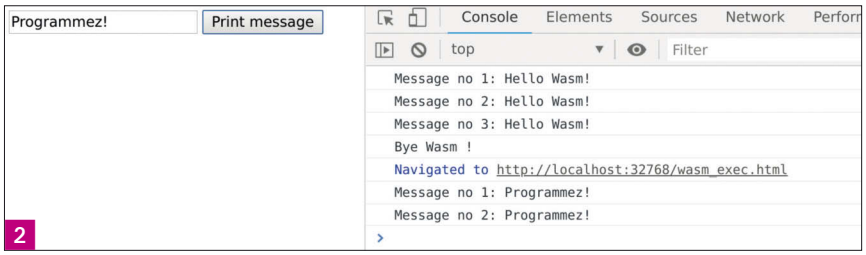
```
let printMessage
function setPrintMessage(callback) {
  printMessage = callback
  document.querySelector('#printMessageButton').disabled = false
}
```

Maintenant au clic sur le bouton "Print message", le message de notre choix et le compteur d'appel s'affichent dans la console du navigateur.
En cochant l'option "Preserve log" de la console du navigateur et en actualisant la page, on peut voir le message "Bye Wasm!" dans la console prouvant que le binaire wasm a bien reçu l'événement beforeunload et pu s'arrêter correctement. 2
Les sources de cet exemple sont disponibles à l'adresse https://github.com/nlepage/golang-wasm/tree/master/examples/long-running.

Bilan

Comme on peut le voir, l'API de "syscall/js" va droit au but et gère ces exemples simples avec assez peu de code. Elle reste pour le moment une API expérimentale et donc possiblement sujette à de grosses évolutions.
C'est aussi une API bas niveau, qui n'est pas réellement destinée à des utilisateurs finaux, mais plutôt à servir de base à d'autres APIs. Des bibliothèques facilitant l'interaction avec le navigateur existent déjà, https://github.com/dennwc/dom fournit par exemple une API plus haut niveau pour interagir avec JS, ainsi que le DOM, le LocalStorage ou encore les WebSockets...
Globalement, ce premier support de WebAssembly par Go est assez réussi, puisque toutes les fonctionnalités coeur du langage sont supportées, y compris la concurrence.
Quelques limitations / points de vigilance sont néanmoins à garder en tête pour le moment :

- Les callbacks (fonctions Go appelables par JS) sont nécessairement asynchrones et ne peuvent donc renvoyer de valeur directement ;



- Attention ! Tous les callbacks s'exécutent dans la même goroutine ; donc si vous effectuez une opération bloquante dans un callback, n'oubliez pas de créer votre propre goroutine, sans quoi toute exécution d'un autre callback sera bloquée ;
- WebAssembly ne supportant pas le multi-threading pour le moment (une proposition est en cours), toutes les goroutines s'exécutent sur le même thread ;
- De même WebAssembly ne possédant pas de garbage collector (là encore une proposition est en cours), c'est celui du runtime Go qui est embarqué dans les binaires compilés, dégradant probablement significativement les performances ;
- Les binaires compilés à partir de Go ont une taille minimum de 2MB ! Cela est dû en grande partie au runtime et aux packages standards Go qui sont embarqués, en plus du code utilisateur…

À savoir : les binaires peuvent aussi être exécutés dans un environnement NodeJS.
Pour aller plus loin, je vous invite à consulter la page du wiki Go consacrée à WebAssembly (https://github.com/golang/go/wiki/WebAssembly), c'est une excellente ressource de départ maintenue régulièrement à jour.
Elle contient notamment quelques exemples d'utilisation plus concrets.

Et après ?

Go 1.12, prévu en février, apporte surtout des corrections de bugs, mais aussi :

- Le renommage de js.Callback en js.Func et de js.NewCallback() en js.FuncOf() (et la disparition de js.NewEventCallback()) ! Le support de WebAssembly étant expérimental, la compatibilité descendante n'est ici pas respectée ;
- Les ~~callbacks~~ fonctions encapsulées seront maintenant exécutées de manière synchrone, et supportent donc une valeur de retour ;
- Une nouvelle interface js.Wrapper permet d'indiquer à js.ValueOf() comment obtenir la valeur JS pour un type Go ;
- Une nouvelle méthode js.Value.Truthy() permet d'obtenir le résultat de la coercition d'une valeur JS en booléen.

Have fun !

bug ou feature

Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : rédaction de ZDnet, Hier
Nos experts techniques : O. Denoo, B. Homes, M. Hage ChahineKey, O. Bouzereau, L. Duport, H. Iqbal, X. Detant, A. Giretti, L. Yovanovitch, S. Stormacq, L. Vincent, R. Guignard-Perret, M. Yami, C. Pichaud, M. Ellerbach, S. Angelloz-Nicoud, N. Lepage
Couverture : assistant rond : AlexLMX - vague : monsitj - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@programmez.com.
Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34

Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, mars 2019

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

INFORMER
pour transformer l'entreprise

La dématérialisation, le Cloud,
les communications unifiées,
les nécessités de la cybersécurité
transforment le travail et toute l'entreprise,
les services publics.
Le magazine, le site, ses newsletters
vous informent sur cette actualité mouvante
et vous aident à décoder les tendances

Abonnez-vous

www.solutions-numeriques.com/abonnement/

4 sites thématiques, pour répondre à vos besoins d'information

- Vous êtes responsable informatique ou bien dirigeant ou cadre d'entreprise ? *2 sites répondent à votre profil*
- La cybersécurité vous concerne ? *Cliquez sur l'onglet. Vous trouverez les infos, l'annuaire, le lexique, etc*
- L'emploi, les salaires, les formations, les offres vous intéressent ? *Le site sur l'Emploi dans le numérique est à votre disposition*

www.solutions-numeriques.com

.NEXT® ON TOUR

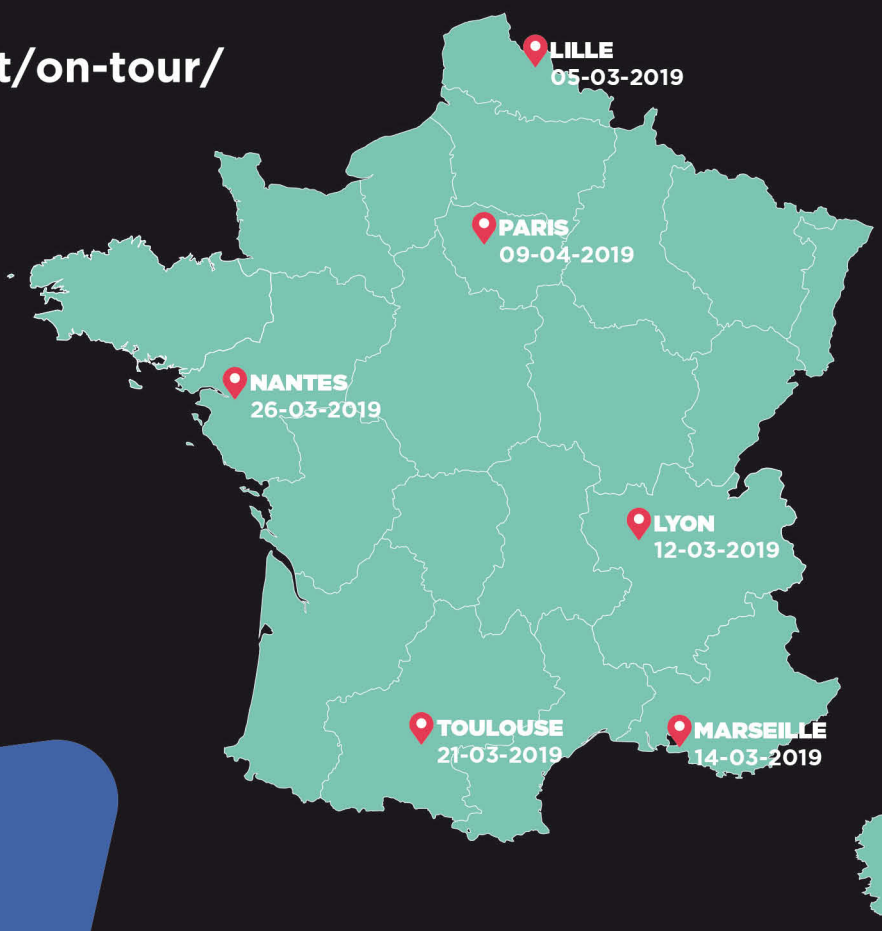
Imagine what's .NEXT

Vous avez le sentiment que votre infrastructure est complexe et dépassée ?
Vous êtes perdu et pensez que l'innovation dans le cloud va trop vite ?

Prenez le contrôle de votre transformation numérique.
Participez au .NEXT on Tour et venez découvrir les dernières technologies cloud hybrides de pointe, témoignages clients à l'appui.

Eliminez la complexité de votre infrastructure : le .NEXT On Tour est un moment dédié pour fixer le cap de votre migration vers le cloud.

nutanix.com/next/on-tour/



NUTANIX™

Lenovo · rubrik · HYCU · CLOUDIAN · EXAGRID · Bitdefender · VERITAS · SUSE · control up