PYTHON | JAVA | SÉCURITÉ | COBOL | CPU | FAIRPHONE | CLOUD

[Programmez!] Le magazine des développeurs
programmez.com
233 OCTOBRE 2019

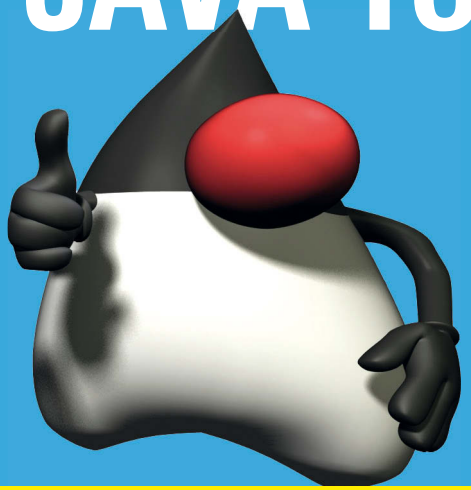
QUOI DE 9 ANGULAR ?

WINDOWS SUBSYSTEM LINUX 2.0

Installation
Utilisation

TKINTER CRÉER DES INTERFACES EN PYTHON

JAVA 13



```
000100 * I am a comment
000200   IDENTIFICATION DIVISION.
000250   PROGRAM-ID. RESEL-WORLD.
000300   PROCEDURE DIVISION.
000350   A-FIRST-PARA.
000400        DISPLAY "Hello World".
000400   STOP RUN.
1       7 8    12
```

COBOL a 60 ans !

“Il ne peut en rester qu’un”

M 04319 - 233 - F: 6,50 € - RD

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

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

Olymp

La formation nouvelle génération

Le logiciel indispensable aux centres de formation pour profiter des avantages de la formation présentielle, et de la formation en ligne.

Edité par

IdeaStudio

© Valérie Turmel

EDITO

Bah, si les études le disent ! (PG13)(1)

Dans les journées bien merdiques où tout va mal et que rien ne fonctionne (sauf les bugs et les warnings, toujours des valeurs sûres), il y a des études, rapports, sondages qui nous font rêver. Mode licorne activé.

Prenons Fed IT qui publie une jolie étude avec 385 pros de l'IT. Et là, c'est le chiffre qui matche bien : 97 % veulent que leur entreprise les forme aux nouvelles technologies et aux tendances du secteur. Par contre, on retombe vite en déprime : 40 % veulent de la gestion de projet et de la sécurité IT, et disons qu'il ne faut pas non plus trop en faire : 1 fois par trimestre ou mieux 1 fois par semestre. Par contre le développement n'intéresse que 27 % (trop nerd ?). Le salaire reste toutefois le 1er choix pour un poste (ouf !(2)). Mais 53 % disent tout de même que l'intérêt de la mission doit aussi être excitant. Bah ouais, on ne va passer 6 mois au fin fond des Vosges juste pour les arbres. Par contre, seuls 6 % parlent de l'environnement technique.

Ah. Tu es prêt à coder sur un vieux PC 486 avec 4 Mo de RAM, un disque dur de 100 Mo (qui fait un bruit bizarre à chaque utilisation), avec un écran 14'' VGA 640x480, connexion RTC ? En soi, c'est un défi (pas sûr que Android Studio ou Visual Code s'installe correctement).

Mais y'a aussi de bonnes nouvelles. Selon Tech.Rocks, un CTO et un Tech Lead valent 82 000 € en moyenne. Mais à ce tarif-là, tu as droit un petit bonus annuel qui peut monter à 5 000 € … Réflexion instantanée : pourquoi n'est-ce pas les dévs qui ont ce salaire en France ?

PageGroup n'est pas en reste. Un développeur peut espérer 38 000 € en mode gentil Padawan. Mais tu vas rire (?) quand tu verras qu'un Product Owner ou un Scrum Master gagnera autant que toi.

Bientôt il suffira d'écrire en gros « je suis développeur » sur son t-shirt pour trouver des propositions de postes et simplement en traversant la rue(3).

Un point est trop rarement évoqué dans ces études et dans le buzz qui va bien de « on manque de compétences en informatique » : le chômage des informaticiens (tout âge, tout profil). Entre 2009 et 2018, il n'a cessé de progresser, dépassant les 57 000 demandeurs d'emploi. Depuis 2018, ce chiffre a marqué un recul, mais il n'en est pas moins important. Mais tout ceci ne fait pas rêver dans une Startup Nation comme la France qui en veut(4).

Bon allez, je vous laisse, j'ai quelques livres à ouvrir : Programmer avec Rust (O'Reilly) et Hacking côté pentest (éditions First)(5). Décidément le RTFM(6) me poursuivra toute ma vie.

François Tonic
ftonic@programmez.com

(1) Cet édito contient des idées (?) et du vulgaire

(2) Ça se voit que je suis cynique ou pas ?

(3) Tiens cela me rappelle vaguement quelque chose, peut-être la fameuse pochette Abbey Road.

(4) Yep, c'est plus sympa de parler d'un Netflix à la française (pourquoi cela me fait rire quand je le dis ?) ou de ressortir le « cloud souverain à la française » que de chômage.

(5) Tu sais le truc qui pèse lourd, avec du papier à l'intérieur et des lettres imprimées dessus.

(6) Lis ce putain de manuel : c'est tout de même la base, non ?

SOMMAIRE

Dans le prochain numéro !
Programmez! #234, dès le 31 octobre 2019
Sécurité & hacking
FMX Linux :
Delphi revient sur Linux

Les PDG américains veulent aussi leur RGPD

51 PDG américains de plusieurs industries ont cosigné une lettre ouverte réclamant une loi fédérale sur la protection des données personnelles. Selon eux, le patchwork de différentes lois sur la protection des données dans chaque état américain complexifie leurs affaires, en les obligeant à se conformer à des textes différents. Ils proposent donc une loi fédérale visant à unifier les différents textes. Le modèle retenu pour donner un exemple de ce qu'ils aimeraient ? Le RGPD européen. Comme quoi, on finit par s'y faire, même outre-Atlantique.

Richard Stallman donne une conférence chez… Microsoft

Richard Stallman est l'un des pères du logiciel libre et l'initiateur du projet GNU n'a jamais été le dernier à fustiger Microsoft. Son site personnel dispose d'ailleurs d'une page listant les raisons de ne pas utiliser les services et appareils de l'éditeur. L'idée d'une conférence de Stallman chez Microsoft a donc de quoi faire lever un sourcil, mais c'est pourtant ce qu'il s'est passé mercredi 4 septembre : Richard Stallman était invité par Microsoft à présenter ses idées sur l'importance du logiciel libre et sur le danger que représentent les logiciels propriétaires. Le genre de discours qu'on n'aurait pas pu entendre chez Microsoft dans les années 2000, mais l'éditeur semble aujourd'hui prêt à tout pour enterrer la hache de guerre.

Teams sur… Linux

Les rumeurs allaient bon train depuis cet été, mais Microsoft a finalement officialisé l'annonce : ses équipes travaillent bien sur un client Linux pour son application de messagerie Teams. Teams est actuellement disponible pour Windows, macOS, iOS, Android et en version web, mais Linux manquait à l'appel. Microsoft ne donne pas de date précise sur la sortie de cette version, mais promet que le sujet est sur la table. En attendant, son concurrent principal, Slack, propose de son côté un client natif pour Linux.

Le cofondateur d'Alibaba tire sa révérence

Une page de l'histoire d'Alibaba se tourne avec le départ de son cofondateur, Jack Ma. Aujourd'hui première fortune de Chine, Jack Ma a fondé le site d'e-commerce en 1999. Comme un symbole, son départ est intervenu le jour anniversaire des 20 ans d'Alibaba. Daniel Zhang aura la lourde responsabilité de remplacer Jack, pour gérer la plus grande plateforme mondiale d'e-commerce.

1 000 000 000, ça en fait des zéros…

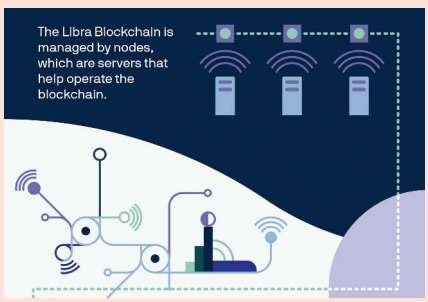
Le géant américain Google est parvenu à un accord avec les autorités françaises afin d'effacer son ardoise. Google France SARL et Google Ireland ltd étaient dans le collimateur des autorités, qui les accusaient depuis 2015 d'avoir négligé de payer des sommes dues au titre de l'impôt sur les sociétés.
La firme américaine devra verser 500 millions d'euros auprès du parquet national financier et payer la modique somme de 465 millions d'euros à la DGFIP. En payant cette somme, Google s'évite donc un procès et une possible condamnation. On trouve toujours matière à s'arranger avec un aussi gros chéquier.

La marque à la pomme est de retour… Avec 3 smartphones

Le mardi 10 septembre était marqué par la traditionnelle keynote d'Apple. Lors de cette conférence la Pomme a eu l'occasion de présenter 3 nouveaux smartphones, rien que ça… Nommés respectivement iPhone 11, iPhone 11 Pro et 11 Pro Max, ces smartphones se voient dotés de nouvelles caméras, pour le plus grand plaisir des instagrammeurs. Apple a également annoncé un nouvel iPad ainsi qu'une nouvelle version de son Apple Watch. Mais l'éditeur tente de tirer son épingle du jeu grâce à de nouveaux services : Apple TV+ viendra marcher sur les plates-bandes de Netflix, tandis qu'Apple Arcade sera une première (véritable) incursion d'Apple dans la distribution de jeux vidéo, avec une centaine de titres au lancement, en novembre (ce service proposé à -5 € ne sera pas encore un service de jeux en streaming, mais à télécharger, NDLR).

Une petite brève…
côté Programmez !

La France voudrait-elle interdire Libra en Europe ?



Le ministère des finances français a évoqué son mécontentement face à la cryptomonnaie annoncée par Facebook, Libra. La France n'en veut pas sur le sol européen. Mais pour suivre la tendance de « tout le monde fait sa cryptomonnaie », le ministère ne serait pas contre une monnaie numérique publique. La position du ministre n'est que la position de la France et non de l'Europe.
« Libra serait une monnaie globale détenue par un seul acteur qui a plus de 2 milliards d'utilisateurs sur la planète. La souveraineté monétaire des États est en jeu. […] Dans des États qui ont des monnaies faibles […] Libra se substituera aux monnaies souveraines et remettra en cause l'indépendance des États. Cette privatisation éventuelle d'une monnaie soulève des risques d'abus de position dominante. Elle soulève des risques de souveraineté. Elle soulève des risques pour les consommateurs et pour les entreprises. […] Toute défaillance dans le fonctionnement de cette monnaie, dans la gestion de ses réserves, pourrait créer des désordres financiers considérables. Enfin je ne vois pas pourquoi nous ferions autant attention depuis des années à éviter toute utilisation d'une monnaie pour le blanchiment ou pour la lutte contre le financement du terrorisme, et qu'une monnaie digitale comme Libra échapperait à ces obligations. […] Dans ces conditions, nous ne pouvons pas autoriser le développement de Libra sur le sol européen. » dixit le ministre.
Petits détails : quand un Etat défaillit sur sa monnaie, le résultat n'est pas meilleur et les conséquences sont tout autant désastreuses. Les États n'ont jamais hésité à manipuler leur monnaie. La domination de certaines monnaies existe depuis que la monnaie existe. Cette réalité n'a pas attendu la cryptomonnaie.

FIRST EUROPEAN FREE & OPEN SOURCE EVENT

opensourcesummit.paris

#OSSPARIS19

PARIS REGION SMART WEEKS

La FRENCH TECH

PARIS OPEN SOURCE SUMMIT

5e ÉDITION

10 & 11 DECEMBRE 2019

Show and Congress
Dock Pullman

Pour toute demande d'informations complémentaires :
Email : cjardon@weyou-group.com – Tel : 01 41 18 60 52

sponsors platinium

alter way, SMILE I.T IS OPEN

sponsors gold

actency, BlueMind, jahia, redhat, SensioLabs Créateur de Symfony, Vorteks

sponsors silver

AXELTI ZABBIX, Capgemini

un événement

Systematic Paris Region Deep Tech Ecosystem, weyou group

en partenariat avec

îledeFrance, CNLL

Octobre

A partir du 30 septembre : .Net Challenge 2019

Le Microsoft DotNet Challenge de SoftFluent est un concours fait pour inspirer les professionnels et les amateurs de la technologie .NET résidant en France. Le premier tour de la compétition est joué en ligne. L'enregistrement est gratuit. Il est ouvert sur toute la durée du concours. A l'issue du tour joué en ligne, les meilleurs participants de la phase en ligne seront invités à l'épreuve finale. Si tu veux relever le défi : http://tinyurl.com/yxbj5vbc

3 : ReBuild 2019 / Nantes

La grande conférence des technologies Microsoft revient à Nantes. L'agenda s'annonce chargé : 50 sessions, 50 speakers, des ateliers techniques. Un rendez-vous à ne pas manquer si vous êtes à Nantes et dans l'ouest.
Site : http://www.communautes-microsoft.fr/rebuild/

10-12 : Paris Web / Paris

Mauvaise nouvelle : les inscriptions ont été suspendues courant septembre suite à un problème de salles pour héberger l'évènement. A l'heure où nous imprimons, nous n'avions pas d'information supplémentaire. Site : https://www.paris-web.fr/actualites/

15 : Red Hat Forum / Paris

La grand'messe de Red Hat reviendra sur ces derniers mois et les nombreuses sorties produits depuis janvier. Deux solutions seront particulièrement mises en avant : Enterprise Linux 8 et OpenShift 4, sans oublier l'écosystème de l'éditeur.

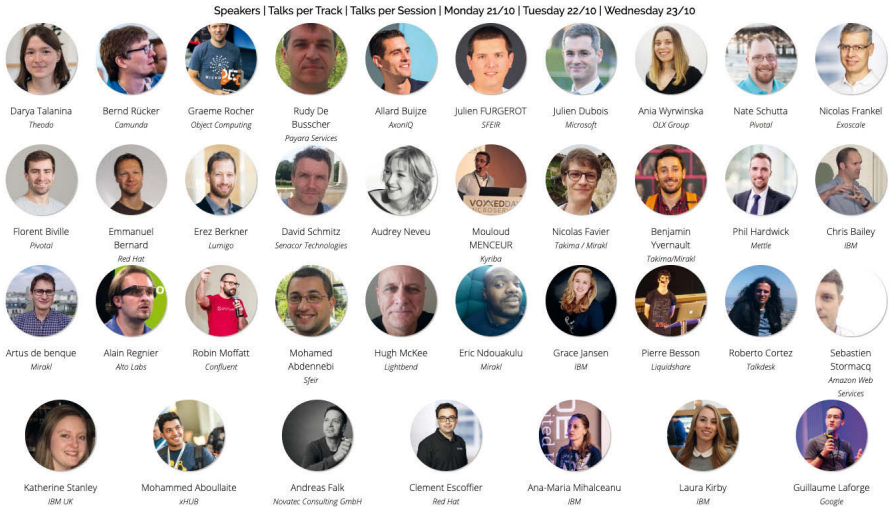
17 : DevOps REX / Paris

Devops REX est la conférence devops 100% retour d'expérience (REX). Des speakers reconnus traitent des applications concrètes de la méthodologie devops en entreprise, avec ses bénéfices mais aussi ses contraintes et ses limites. Le tout se déroule dans un univers prestigieux et haut de gamme.
Site : http://www.devopsrex.fr

21-22 : DevFest / Nantes

Le DevFest, ou 'Developer Festival', est une conférence technique destinée aux développeurs. Elle s'adresse aussi bien aux étudiants, aux professionnels ou tout simplement aux curieux technophiles.
Site : https://devfest.gdgnantes.com/fr/

21-23 : Voxxeddays Paris



Scala io

DEVFEST 3019

Voxxed Days Microservices est une conférence professionnelle qui s'adresse aux développeurs européens concernés par les Microservices :

- 2 jours de conférences, 40 présentations et 38 orateurs
- 1 workshop optionnel d'une journée complète à choisir parmi une liste de 4 (Quarkus / Istio & Kubernetes / Securité / Migration de Monolith)

38 experts internationaux interviendront en anglais sur des thématiques telles que cloud, containers & infrastructure, data store, messaging, architecture microservice, monitoring, integration, securité… Vous pourrez écouter par exemple : Ana-maria Mihalceanu, Sebastien Stormacq, Emmanuel Bernard, Audrey Neveu, Graeme Rocher, David Schmitz, Nate Schutta...

Tout ce que vous voulez savoir sur les Microservices sans oser le demander… Venez rencontrer les sponsors, les participants et les orateurs autour de ce sujet devenu incontournable.

Informations complémentaires :
Lieu : Espace Charenton (Paris 12e)
Tickets : https://tickets.voxxeddays.com/event/vxdms2019
Speakers & programme :
https://voxxeddays.com/microservices/#speakers

DevCon #8

Technologies vocales avec Amazon Alexa

24 OCTOBRE 2019 À 42

Les technologies
Comment créer et déployer des skills
Usages & opportunités
Les modèles économiques
Des ateliers pratiques

2 plénières
4 sessions
+ pizza beer party

NE RATEZ PAS CETTE CONFÉRENCE DÉVELOPPEUR

Une conférence Programmez! avec la participation d'Amazon Alexa

Informations & inscriptions sur **www.programmez.com**

24-25 : Forum PHP
Pour en savoir plus : https://event.afup.org

29-31 : Scala IO / Lyon
6e édition de la grande conférence dédiée à Scala et à son écosystème.
Site : https://scala.io

Novembre
4-8 : Devoxx Belgium / Belgique
L'évènement développeur belge de l'année : 200 speakers, +200 sessions, + 3200 personnes !
Pour en savoir plus : https://devoxx.be

5 : JFIE 2019 / Paris
Cet évènement, qui réunit en moyenne plus de 200 participants, est une occasion unique pour les professionnels du test de réfléchir et d'échanger sur les problématiques de la gestion des exigences, composante indispensable de la gestion de la qualité des tests de logiciels et systèmes d'information. Ces échanges et démonstrations s'appuieront sur 6 conférences illustrant la mise en œuvre de l'ingénierie des exigences dans des contextes variés : projets classiques, agiles, lignes de produits ou systèmes complexes… Site : www.cftl.fr

6 : DevFest / Strasbourg
Le DevFest, ou "Developers Festival", est une conférence technique destinée aux développeurs. Elle s'adresse aussi bien aux étudiants, aux professionnels ou tout simplement aux curieux, passionnés de technologies.
Site : https://devfest.gdgstrasbourg.fr

13-14 : Microsoft Ignite Tour
Microsoft expériences n'aura pas lieu cette année. L'évènement qui rassemblait l'écosystème et les communautés se transforme en deux évènements :
13 &14 : Microsoft Ignite The tour. Deux jours pour se former à travers des parcours, des workshops et des rencontres… Mais aussi pour explorer les innovations qui seront derrières les usages de demain.
13 : Microsoft Envision The tour. Cet évènement parlera IA, de compétences, d'opportunités marchés, etc. Journée plus orientée entreprise et transformation digitale.
Site : https://www.microsoft.com/en-us/ignite-the-tour/

16 & 17 : Capitole du libre / Toulouse
Le logiciel libre rencontre le grand public et les développeurs en novembre prochain à Toulouse. Ce grand évènement avait rencontré un beau succès en 2018 avec +1500 personnes. Cette année ce sont 100 conférences et 25 ateliers qui seront proposés. Côté développement, il y aura du choix : C++, développement web, embarqué, les jeux, le DevOps.
Site : https://capitoledulibre.org

21 : Codeurs en Seine
Codeurs en Seine est une journée de conférences gratuites qui se déroule à Rouen, pour découvrir, apprendre et partager autour du monde du développement.
Site : https://www.codeursenseine.com

23 : Campus du libre / Lyon
Lyon accueille la 2e édition du Campus du Libre sur le logiciel libre, de l'open source. Il est organisé par des personnes issues du milieu universitaire (étudiants et personnels) pour les étudiants lyonnais. L'objectif est de partager différents aspects du libre et des communs, allant par exemple du logiciel libre (Linux, Firefox, etc.) aux espaces communs gérés collaborativement (Wikipedia, OpenStreetMap).
Site : http://www.campus-du-libre.org

26-27 : VoiceTech Paris
Les technologies vocales se répandent peu à peu. Ce salon met en avant les usages, les enjeux du vocal. Les deux jours seront animés par des conférences et ateliers. Si le sujet vous intéresse, c'est l'évènement à ne pas rater.
Site : https://www.voicetechparis.com/2019/

Décembre
5 & 6 : La nuit de l'info 2019
La nuit de l'informatique revient. Pour rappel, il s'agit d'une grande compétition nationale pour les étudiants, enseignants et entreprises pour développer un projet durant la nuit. Les équipes doivent choisir le défi à relever ! +4 000 participants ont tenté de ne pas dormir en 2018… Site : https://www.nuitdelinfo.com

Girls Can Code :
des week-ends pour s'initier au code
Un « Girls Can Code! week-end » est une initiation de deux jours à la programmation informatique pour collégiennes et lycéennes. On y découvre tranquillement le langage Python, facile à prendre en main, très utilisé notamment par YouTube ou encore la NASA !

Les prochaines dates :
- Montpellier - Google Atelier Numérique, le 18 et 19 octobre 2019
- Nancy - Google Atelier Numérique, le 25 et 26 octobre 2019

Pour s'inscrire : https://gcc.prologin.org/

2020

Janvier
22-25 : SnowCamp / Grenoble
SnowCamp est une grande conférence rassemblant dévs, ops et architectes. On y trouve une journée de formation, 2 jours de conférences et une journée d'échanges sur les pistes.
Site : http://snowcamp.io/fr/

Février
1-2 : FOSDEM
L'évènement incontournable des développeurs open source. C'est l'occasion de découvrir des projets et surtout d'autres développeurs. On échange, on discute, on oppose les arguments. Un lieu passionné et passionnant.
Site : https://fosdem.org/2020/

3 : dotSwift / Paris
La conférence Swift revient à Paris. Avec les dernières évolutions du langage, le nouveau framework UI, il y aura de nombreuses choses à découvrir.

Merci à Aurélie Vache pour la liste 2019/2020, consultable sur son GitHub :
https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

Prix Bernard Novelli 2019-2020

tangente CASIO SIF Société Informatique de France programmez! le magazine des développeurs

Concours de projets informatiques pour collégiens et lycéens.
Participez individuellement, en équipes ou dans le cadre scolaire.

Le trophée d'art numérique offert au vainqueur

- Vous êtes collégien ou lycéen amateur d'informatique ?
- Vous êtes parent d'élève et voulez développer le goût de vos enfants pour l'algorithmique et l'informatique ?
- Vous êtes enseignant et vous voulez donner un objectif à votre enseignement à la rentrée ?

Participez !
Faites participer des jeunes au Prix Bernard Novelli !

Le but des participants :
réaliser un programme informatique autour du jeu et des mathématiques.

- **Le calendrier est dorénavant celui de l'année scolaire. Pour 2019-2020 :**
- **Inscription avant 30 octobre 2019**
- **Remise du projet avant le 30 avril 2020.**
- **Palmarès en juin 2020**

Un jury composé de représentants des partenaires (Tangente, CASIO, Programmez!, Société Informatique de France) désignera le vainqueur et les mentions liées au niveau scolaire.
Les lauréats gagneront de nombreux prix : œuvre d'art numérique, calculatrice CASIO haut de gamme, déplacement à Paris pour la remise des prix (depuis la métropole).

Réglement sur **www.tropheestangente.com**
Inscription des candidats sur **tropheestangente@yahoo.fr**

LES TROPHÉES tangente

NOUVEAU : organisation dans les établissements scolaires

Depuis cette année, les professeurs de collèges et lycées peuvent organiser le Prix Bernard-Novelli dans leur établissement.
Ils décident des procédures (thème, langage de programmation, participation individuelle ou par équipes...)
Inscription des établissements : **tropheestangente@yahoo.fr**

Roadmap des langages

Chaque mois, Programmez ! vous propose un panorama des agendas de sorties des versions des langages, frameworks, etc.

Attention ! Python 2.7 sera déprécié dès le 1er janvier 2020. A partir de cette date, plus aucune mise à jour.

DÉJÀ DISPONIBLE

Flutter 1.9

Date de sortie : disponible
Les principales nouveautés attendues / annoncées : Google sort très rapidement les versions de Flutter. La 1.9 a été déployée le 10 septembre dernier. Cette version inclut plus de 1500 corrections et modifications ! Elle supporte le prochain macOS et iOS 13, les nouveautés du langage Dart et les nouveaux composants Material. Grande nouvelle aussi, l'intégration de Flutter web dans le référentiel principal de Flutter ! L'éditeur annonce aussi le support de 24 nouvelles langues.

Dart 2.5

Date de sortie : disponible
Les principales nouveautés attendues / annoncées : bien que Dart soit plus discret que Go, il évolue lui aussi régulièrement. La v2.5 inclut une pré-version du Foreign Function Interface qui permet d'appeler du code C directement depuis Dart. Cette version comporte pas mal de nouveautés : code complétion en utilisant des modèles de machine learning (preview), amélioration sur les const. Des fonctionnalités à suivre.

SWIFT 5.1

Date de sortie : disponible
Les principales nouveautés attendues / annoncées : ce sera la première mise à jour de SWIFT 5. Cette version doit apporter des corrections de bugs et terminer la stabilité des librairies et des modules. Bien entendu, la 5.1 est rétrocompatible avec la 5.0. On doit s'attendre à quelques changements dans ce que les équipes appellent la « module stability » comme l'apparition du .swiftinterface pour le fichier d'interface à la place de .swiftmodule. Quelques éléments ici : https://swift.org/blog/5-1-release-process/

Kotlin 1.3.40

Date de sortie : disponible
Les principales nouveautés attendues / annoncées : cette version du langage améliore plusieurs éléments. L'équipe s'est focalisée sur le support de Gradle pour NPM / Yarn / Webpack, meilleurs tests dans les projets multiplateformes, amélioration des performances et interopérabilité pour Kotlin / Native. Les détails : https://blog.jetbrains.com/kotlin/2019/06/kotlin-1-3-40-released/
La version 1.3.50 apportera un convertisseur Java – Kotlin réécrit. L'équipe annonce de nombreux bug fix ! Kotlin Native sera plus complet sur macOS. Bref de belles choses pour les dévs !

TypeScript 3.6

Date de sortie : disponible
Les principales nouveautés attendues / annoncées : cette version introduit une vérification plus stricte des itérateurs et dans la génération des fonctions. Par exemple, il va vérifier que le type soit correct pour curr.value. La partie Promises a été améliorée. Attention : l'équipe annonce aussi un peu de casses sur le code. Regardez bien les notes de version. Quelques nouveautés : import.meta supporté dans SystemJS, nouveaux playground.
Pour en savoir plus : https://devblogs.microsoft.com/typescript/announcing-typescript-3-6/

Ruby on Rails 6

Date de sortie : disponible
Les principales nouveautés attendues / annoncées : cette version propose la fonctionnalité Action Mailbox permettant de router des mails entrants vers des boîtes mails se comportant comme des contrôleurs. On notera également, grâce à l'arrivée d'une nouvelle API, la prise en charge de connexions à plusieurs bases de données simultanément. L'équipe de Ruby on Rails souligne que ceci offre l'opportunité de segmenter certains enregistrements dans leurs propres bases de données à des fins de dimensionnement ou d'isolation. La v6 propose un nouveau chargeur de code et propose désormais Webpacker comme bundle JavaScript par défaut.

FIN 2019

React 16.9

Date de sortie : automne / hiver
Les principales nouveautés attendues / annoncées : la 16.9 est en développement depuis plusieurs mois. Durant l'été, les développeurs ont été appelés à tester les préversions. Parmi les nouveautés, on notera la dépréciation de UNSAFE_* ou encore de Javascript :URLS, des composants Factory, pas mal de corrections de bugs. Parmi les nouveautés : Async act(), mesures de performances avec React.Profiler. Une nouvelle version de React DevTools a été déployée en août dernier.
Pour suivre les sorties : https://github.com/facebook/react/releases

Goland 1.13

Date de sortie : disponible.
Les principales nouveautés attendues / annoncées : cette version doit activer le mode module par défaut (le mode par défaut d'auto à activer) tout en déconseillant le mode GOPATH. On aura aussi des nouveautés sur les génériques, la gestion des erreurs.
Au-delà : Go 2 sera LA grosse évolution du langage Go, même si les équipes ne veulent pas faire une version de rupture, mais des évolutions au fil des versions. Un des changements sera la manière de définir les évolutions et comment la gouvernance fonctionne. L'idée est que la communauté soit plus impliquée. La compatibilité avec Go 1.x sera un des aspects cruciaux. Pour le moment, le planning de Go2 reste à préciser.
Notez que la 1.13 sera aussi la dernière version à s'exécuter sur le Native Client. Le langage tournera aussi sur Illumos, NetBSD sur arm64, OpenBSD sur arm64.
Pour en savoir + : https://blog.golang.org/go1.13

C# 8.0

Date de sortie : été / automne
Les principales nouveautés attendues / annoncées : cette nouvelle évolution du langage phare .Net doit arriver avec .Net Core 3.0. Parmi les nouveautés attendues :
- Les types de références nullables doivent en finir avec les exceptions null. Pour cela, elles vous empêchent de mettre null dans des types de référence ordinaire comme string. Par défaut, ces types seront non nullables ! Cela pourrait avoir un impact sur les codes existants ;
- Les flux asynchrones ;

- Types range et index ;
- Expressions Switch.

Au-delà : il faut s'attendre à des itérations périodiques comme pour C# 7. Pas de détails pour le moment.

Python 3.8

Date de sortie : octobre

Les principales nouveautés attendues / annoncées : plusieurs nouveautés devraient plaire aux développeurs. Tout d'abord, on bénéficie du nouveau paramètre pythonpycacheprefix. Il permet d'utiliser le cache bytecode dans une branche séparée du système de fichiers parallèle. Il sera à préférer au __pycache__ présent dans les sous-répertoires des dossiers sources. On disposera aussi de la méthode as_integer_ratio(), dans le type int. Le contrôle-C, SIGINT, sera modifié pour éviter les problèmes liés à l'exception KeyboardInterrupt. Plusieurs modules auront droit à des améliorations comme asyncio avec ProactorEventLoop ou encore dans gc avec de nouveaux paramètres dans le get_objects().
Pour plus de détails : https://docs.python.org/dev/whatsnew/3.8.html .

Au-delà : en toute logique, la prochaine version majeure sera la 3.9. Pour le moment, pas grand-chose n'a été communiqué. Cette version ne devrait pas arriver avant 2021.

Symfony 4.4

Date de sortie : novembre

Les principales nouveautés attendues / annoncées : Symfony 4.4 doit arriver courant novembre. Elle répond à la politique de mise à jour annuelle : 1 en mai, 1 en novembre. Cette version sera LTS.

PHP 7.4

Date de sortie : fin novembre

Les principales nouveautés attendues / annoncées : la 7.4 doit apporter le préchargement (preloading) au cœur de PHP pour pouvoir améliorer les performances. Ce mécanisme permet de charger l'ensemble des fichiers PHP dès le démarrage et ainsi améliorer les accès pour assurer une disponibilité constante de ceux-ci. En revanche, en cas de changement des fichiers, il faudra redémarrer le serveur. On notera aussi la disponibilité des propriétés typées, l'extension FFI (Foreign Function Interface) pour appeler du code C, le nouvel opérateur Null Coalescing, l'apparition de nouvelles méthodes (__serialize & __unesrialize). On notera aussi le retrait de PEAR ou la dépréciation de ext/wwdx.

Au-delà : la prochaine version majeure devrait être la 8.0. Une grosse nouveauté connue sera la présente d'un compilateur JIT. Cette nouveauté permettra de se passer du Zend VM.

Ruby 2.7

Date de sortie : décembre (?).

Les principales nouveautés attendues / annoncées : le langage Ruby continue d'évoluer. La 2.6 est sortie fin 2018, notamment avec un nouveau compilateur JIT. On bénéficie aussi d'un nouveau module RubyVM ::AbstractSyntaxTree. Il analyse une chaîne de caractères et retourne les nœuds d'arbre syntaxique. Un travail d'optimisation a été réalisé.

COURANT 2020

C++20

Date de sortie : 2020.

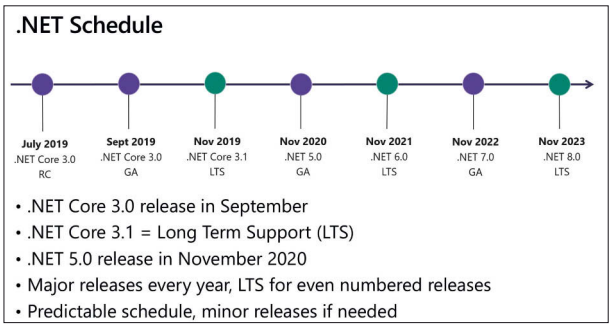
Les principales nouveautés attendues / annoncées : les spécifications de C++20 sont désormais figées ; un premier document complet sera disponible cet été. Cette future version du langage mettra en avant deux nouveautés : les modules et les coroutines. Les modules constituent une nouvelle alternative aux fichiers d'en-tête qui apportent un certain nombre d'améliorations clés, notamment en isolant les effets des macros et en permettant des compilations évolutives, explique Herb. Il ajoute qu'en 35 ans c'est la première fois que C ++ ajoute une nouvelle fonctionnalité permettant aux utilisateurs de définir une limite d'encapsulation nommée.
Les coroutines sont elles aussi une fonctionnalité à remarquer. Une coroutine est une unité de traitement qui s'apparente à une fonction (ou routine), avec cette différence que si une sortie du corps d'une fonction met fin à l'exécution de celle-ci, la sortie de la coroutine suspend seulement son traitement qui peut ensuite reprendre, l'état du traitement à la sortie étant conservé. Une coroutine peut être vue comme un morceau de programme qui conserve son état, mais qui n'a pas de thread d'exécution. Les coroutines peuvent notamment être utilisées pour des itérateurs et des générateurs.

.Net 5

Date de sortie : novembre 2020.

Les principales nouveautés attendues / annoncées : l'annonce en a été faite à la dernière conférence BUILD. Il s'agit de .Net Core vNext, donc au-delà de .Net Core 3.0. Il s'agit de réunifier les noms. L'ambition est d'être disponible sur Windows, Linux, macOS, iOS, Android, tvOS, webassembly, etc.

Au-delà, Microsoft a annoncé une ambitieuse roadmap :



LTS

On parle souvent de LTS et de non-LTS. LTS signifie support long terme. C'est-à-dire que cette version est supportée officiellement durant x années et recevra les mises à jour et les patchs de sécurité nécessaires. Une version non-LTS a une durée de vie très courte, quelques mois.

Seules les versions issues d'une forte communauté, d'une fondation, d'un éditeur sont LTS. Chacun fait un peu ce qu'il veut sur le support. Exemple : Angular est sur un cycle de 6 mois pour les versions majeures. Le support se fait sur une version.

VERSIONING

On parle de versions majeures et de versions mineures. Ces dernières sont souvent des versions de bug fix, de sécurité, introduisant peu ou pas de nouveautés. React explique aussi la structure des versions avec x.y.z :
X = une version majeure introduisant une rupture ;
Y = nouvelle fonction, version mineure (ex. : 15.6) ;
Z = bug fix. On corrige les bugs, les problèmes importants.
N'oubliez pas de lire les notes de versions (release notes). Elles indiquent les changements, les modifications, les nouveautés, la migration entre les versions.

Comment Nutanix Calm facilite la mise en place des pipelines CI/CD

Pensé pour automatiser le cycle de vie des applications, Nutanix Calm peut s'avérer très utile pour simplifier la mise en place de stratégie d'intégration continue pour les développeurs.

Dans sa volonté de proposer des infrastructures toujours plus transparentes pour les utilisateurs, Nutanix a présenté ces dernières années de nombreux services venant compléter son offre historique d'hyperconvergence. La société qui s'affiche aujourd'hui comme un leader du cloud computing d'entreprise a ainsi présenté des solutions comme Leap (Back-up-as-a-Service), Era (gestion des bases de données), XiIoT (IoT et Edge computing) ou encore Calm, qui nous intéresse aujourd'hui. Cette solution d'automatisation de la gestion du cycle de vie applicatif apporte de nombreux outils aux développeurs, notamment dans la mise en place de modèles de développement DevOps et CI/CD. Concrètement, Calm automatise, à travers un système de blueprint, un certain nombre de tâches au niveau des infrastructures. Dans une logique de DevOps, cette solution permet notamment de faire le pont entre la partie Dev, et la partie Ops en mettant à disposition des développeurs des ressources similaires à ce qu'ils auront en production. Pour les processus CI/CD, les tests sont très souvent faits sur des environnements dédiés qui ne correspondent pas forcément à ceux mis en production. Calm permet de gommer cette lacune en déployant les applications rapidement sur des infrastructures identiques à celles en production, que ce soit en termes de dimensionnement ou des paramètres réseau, et ce en s'intégrant aux outils de CI/CD.

« En tant que développeur, vous êtes probablement conscient de la valeur que l'intégration continue apporte, mais vous êtes également conscient du défi que représente la configuration de l'automatisation des tests et du déploiement, qui est essentielle à la réussite d'un modèle CI/CD, en particulier quand il faut intervenir sur des composants d'infrastructure en production », déclare Christophe Jauffret, Solution Architect Specialist Automation & DevOps de Nutanix. « Calm réduit une grande partie de cette complexité, permettant aux développeurs de se concentrer sur leurs applications. »

Calm aide donc tout le monde, des administrateurs d'opérations informatiques de haut niveau aux développeurs en codage actif. Ils peuvent déployer les mises à jour plus rapidement, tout en tirant parti de la puissance des tests automatisés pour maintenir la qualité en interagissant directement avec les infrastructures en production. Pour un développeur, il est par exemple possible de mettre en place des pipelines de test and dev pour servir tous types d'applications du plus simple au plus complexe. Voici un exemple d'architecture pour une application web conteneurisée (Fig 1).

Reste maintenant à voir comment se déroule en pratique la mise en place d'un pipeline CI/CD avec Calm. Dans l'exemple suivant, voici comment assurer la mise en place d'un tel processus sur un cluster Nutanix, avec Karbon (distribution Kubernetes de Nutanix) et Jenkins.

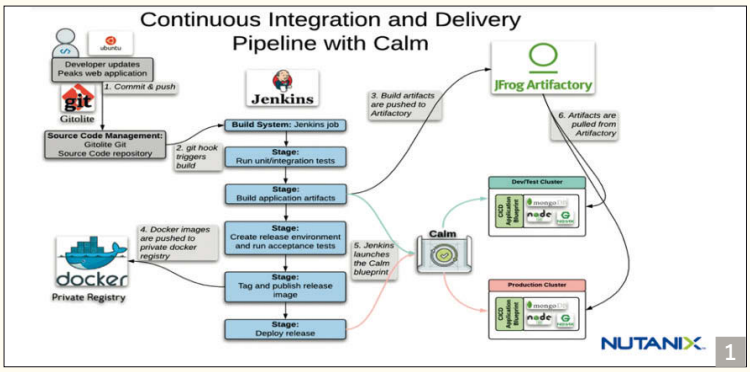
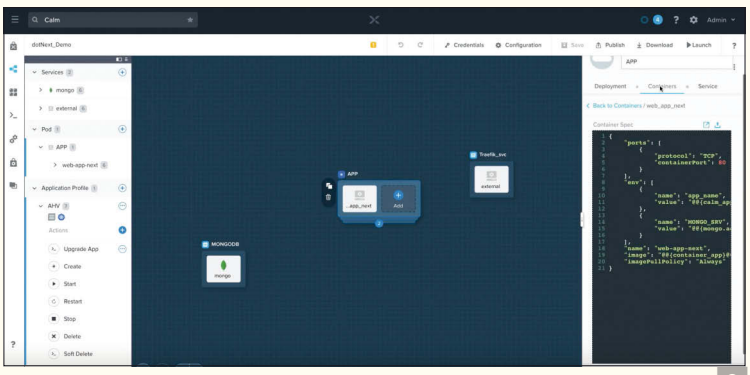


Schéma de déploiement d'un processus d'intégration continue pour une application web avec Calm.

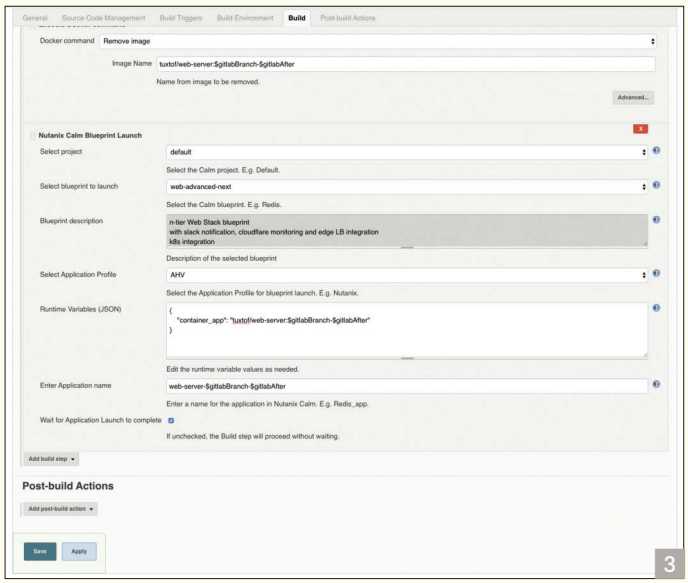


Paramétrage de kubernetes via le blueprint et la vue API.

Configuration d'une application et d'un pipeline CI/CD avec Nutanix Calm

Dans cet exemple, c'est une application déjà développée, ici un site web de démo qui sera utilisé et dont le code est hébergé sur GitLab. Elle sera déployée sur un cluster Kubernetes déjà paramétré grâce à Karbon et intégré à Calm.

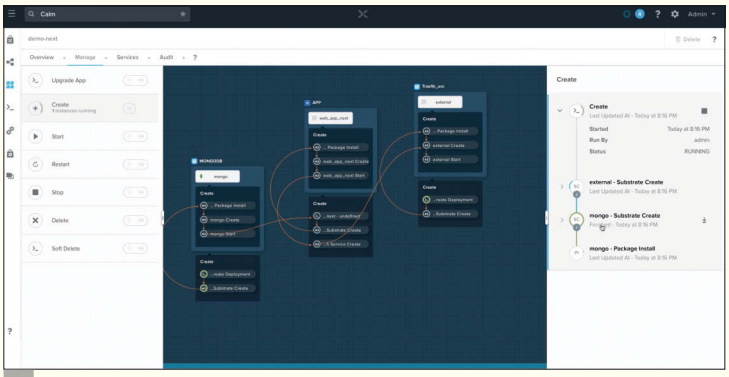
Les DevOps ont créé un blueprint qui compose le cœur de l'utilisation de Calm. Dans cet exemple, ce dernier regroupe tous les services nécessaires à l'application. On y retrouve donc l'application conteneurisée, une base de données MongoDB et un service qui permet de configurer les interactions avec les environnements extérieurs. Ce dernier va notamment permettre de mettre en place le DNS, de paramétrer un load balancer externe, d'intégrer l'application à des systèmes de monitoring existants, et même de notifier les utilisateurs via Slack de la mise en



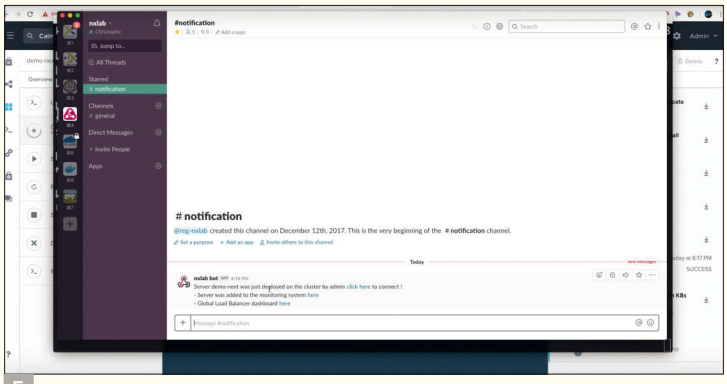
Jenkins va automatiquement appeler le Blueprint de Calm qui va alors mettre en place les ressources nécessaires et assurer le déploiement de l'application automatiquement sur l'infrastructure choisie.

place d'une nouvelle release sur l'application.
La première chose à savoir c'est qu'il est possible de gérer l'ensemble des paramètres Kubernetes de l'application directement depuis Calm. Il est aussi possible de passer en vue API pour avoir un maximum de possibilités de personnalisations (Fig 2). C'est via ces fonctionnalités que les utilisateurs vont pouvoir relier les différents services au cluster, notamment la base MongoDB et paramétrer le load balancer pour exposer le container aux outils externes.
Une fois ces étapes de paramétrage terminées, il faut passer au déploiement de l'application via Jenkins. Le projet GitLab dans lequel se trouvent toutes les informations nécessaires à son déploiement est alors associé à l'environnement Jenkins où se trouve un projet dédié. Ce projet est configuré pour builder automatiquement l'application lors d'un nouveau commit sur le projet GitLab et pousser cette nouvelle image sur le hub docker prévu à cet effet. Dans Jenkins, il faut alors ajouter une nouvelle étape qui va permettre de déployer l'application depuis Calm (Fig 3).
Calm va ensuite gérer directement le déploiement de l'application sur l'infrastructure définie préalablement, du déploiement de l'image docker sur le cluster K8s aux paramétrages des politiques de réseaux, en passant par la mise en place du load balancer ou encore des DNS (Fig 4).
Une fois l'application déployée, elle est directement accessible en ligne et un message Slack est envoyé aux utilisateurs (Fig 5). Ici, il s'agit d'un site de démonstration directement publié sur internet avec un nom de DNS public reprenant dynamiquement le nom de l'application.
En cliquant sur le lien envoyé dans Slack, il est possible de se rendre directement sur l'application. À noter qu'il est évidemment possible de le faire directement depuis Calm. (Fig 6).
Pour réaliser des mises à jour sur l'application, la procédure sera ensuite toujours la même. À partir du moment où un nouveau commit sera poussé sur GitLab, Jenkins lancera automatiquement Calm pour déployer la nouvelle version de l'application sur l'infrastructure définie par les administrateurs, qu'elle soit en production ou similaire à celle de production dans les logiques de CI/CD.
Voici donc un aperçu des capacités de Calm. Depuis son lancement, Nutanix Calm a déjà permis à de nombreux utilisateurs de simplifier la gestion du cycle de vie de leurs applications.

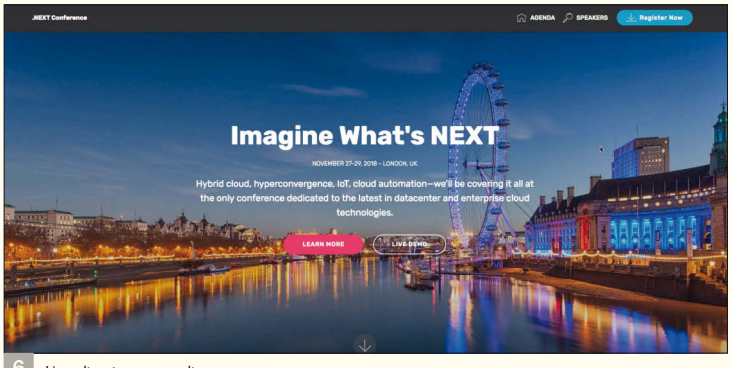
Ressources complémentaires :
https://www.nutanix.com/products/calm
https://www.nutanix.dev/2019/02/14/creating-a-ci-cd-pipeline-with-nutanix-calm-and-jenkins/



4 *Calm permet de suivre toutes les tâches et les sous-tâches nécessaires au lancement de l'application. Elles sont détaillées à droite de l'écran et les tâches qui se déroulent correctement sont notifiées en vert et s'il y a une erreur elle est notifiée en rouge.*



5 *L'intégration de Slack permet de notifier directement les utilisateurs dès le lancement de l'application.*



6 *L'application est en ligne.*

CALM, C'EST COMME UNE RECETTE DE CUISINE

Par Christophe Jauffret, Solution Architect Specialist Automation & DevOps de Nutanix

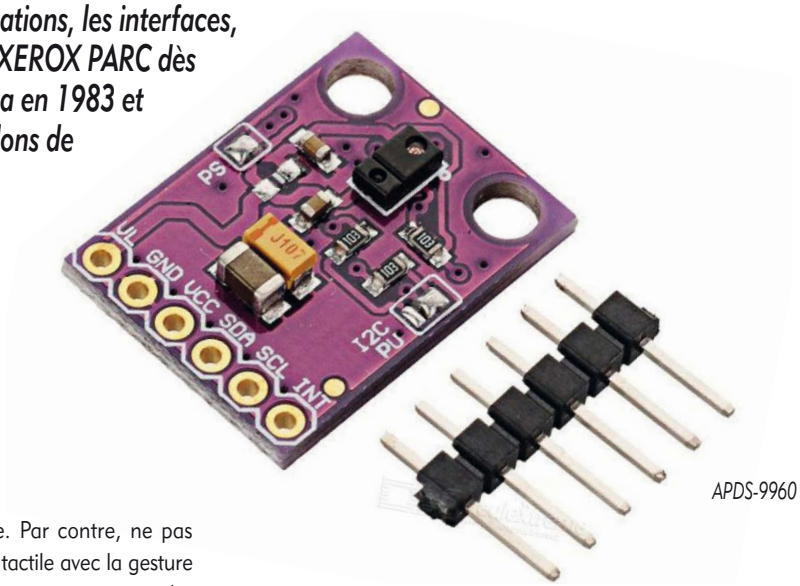
Calm a été conçu pour répondre à trois problématiques distinctes qui sont la gestion du cycle de vie des applications, de pouvoir proposer aux utilisateurs de consommer les applications sur une approche self-service, et, enfin, d'orchestrer ces applications dans les environnements actuels, notamment multicloud. L'idée était d'enlever un maximum de contraintes dans la gestion des applications, de leur développement à leur déploiement à l'échelle en passant par le test, les mises à jour, etc.

Si on apparente la création d'une application à une recette de cuisine, Calm permet déjà de rassembler et de mélanger ensemble tous les ingrédients à travers un blueprint. Les développeurs vont pouvoir spécifier les VMs, les outils, les frameworks et les différentes configurations réseau dont ils ont besoin, ou choisir des blueprint mis en place par les Ops correspondants, traits pour traits, aux environnements de production. On va pouvoir y ajouter également les différentes politiques que l'on veut appliquer à cette application, que ce soient les politiques de sécurité, la façon dont l'application communique avec les autres et aussi la gestion des utilisateurs et de leurs droits d'accès. Toutes ces informations vont être intégrées au blueprint de Calm pour ensuite être automatisées. Cela concerne un peu moins les développeurs, mais Calm va également permettre de gérer le quotidien des applications, leur mise à l'échelle, les mises à jour ou toutes les tâches répétitives. L'autre atout de Calm réside dans sa capacité à gérer les applications et leurs déploiements sur des environnements éparses, on premise, cloud privé, public, hybride et multicloud.

François Tonic

Air interface : la révolution de l'interface et de l'ergonomie

On parle beaucoup d'interfaces, d'interactivités homme-machine et d'ergonomie. Personnellement, ces questions m'intéressent depuis les années 90. Les Newton et autres Palm avaient créé une nouvelle approche dans la manière de manipuler, d'interagir avec les données, les applications, les interfaces, un peu comme la révolution de la GUI, inventée par XEROX PARC dès 1973, mais réellement démocratisée avec l'Apple Lisa en 1983 et surtout avec le Macintosh en 1984. Aujourd'hui, parlons de l'Air Interface.



APDS-9960

La notion d'Air Interface est un bricolage inspiré du fameux Air Guitare. Je trouve cette idée passionnante, car cela induit une nouvelle manière d'interagir avec un objet électronique (typiquement un IoT), une machine (disons un ordinateur, une table, etc.).

Dans le n°198 de Programmez!, j'avais évoqué une très intéressante conférence durant la GitHub Satellite 2016 à Amsterdam autour des interfaces et des IoT. 4 principales interactions (modèles d'interaction) sont souvent citées :

- CLI : ligne de commande
- GUI : interface graphique classique
- NUI : interface dite naturelle
- OUI : interface dite organique

En NUI, les entrées sont diverses : le toucher, les gestes (gesture), la voix, les mouvements du corps / yeux / tête. En sortie, les réponses aux actions que nous avons en entrée peuvent se présenter sous une forme visuelle, sonore ou haptique. Par micro-interaction, comprenons qu'à un moment donné, l'objet tournera autour d'un seul cas d'usage et n'aura alors qu'une seule tâche principale.

Comment définir une Air Interface ?

Pour Air Interface, j'entends toute interface sans contact physique entre l'Homme et la machine proprement dite. C'est-à-dire, qu'aucun contact n'est nécessaire pour interagir / agir. Typiquement, le mouvement (sans matériel à porter sur soi) sera l'interaction directe. Leap Motion, Hololens sont dans cette catégorie. Par contre, ne pas confondre la gesture tactile avec la gesture dans l'espace. Ces gestes sont principalement produits par les bras et les mains. On peut rajouter les mouvements du corps, des yeux, les expressions du visage, etc. Mais là, on utilise d'autres technologies et capteurs.

Ces interfaces se rangent dans les NUI et OUI.

Comment définir une OUI ? On parlera d'une combinaison d'interface naturelle et tangible. On dit souvent que le modèle OUI repose sur le modèle d'interaction SMaG, Speech Manipulation air-Gesture. L'OUI ne définit pas forcément une nouvelle génération d'interface, mais reprend les interfaces existantes.

Dans l'approche SMaG, nous avons, pour simplifier les choses :

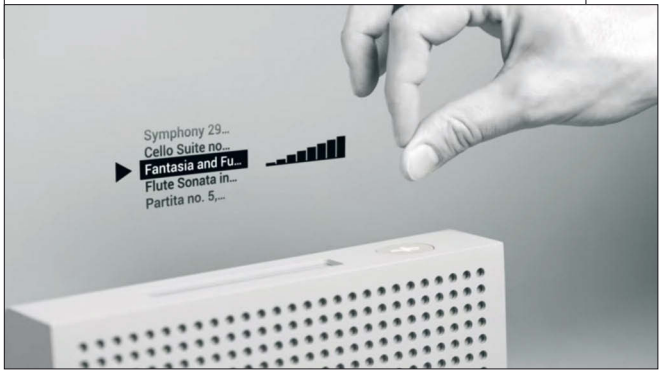
- La voix ;
- Les manipulations : tactile dite discrète, tactile continu (par exemple pour dessiner, écrire), la nécessité d'avoir une ou deux mains pour manipuler l'objet. La notion de tangible s'exprime par la nécessité d'avoir un stylet ou tout autre pointeur ;

PROJET SOLI DE GOOGLE

Depuis plusieurs années, le lab R&D de Google, ATAP, travaille sur Soli. Il s'agit d'un système de reconnaissance des gestes grâce à l'utilisation de minuscules radars embarqués dans une puce. Les ondes sont ensuite analysées pour être interprétées. Le système utilise du machine learning pour apprendre et comprendre.

Cela rappelle les technologies comme Kinect, Leap Motion, mais occupant une surface extrêmement réduite. Soli est fait pour les IoT, le wearable.

Pour le moment aucune date de commercialisation n'a été donnée par Google. Les démos donnent une idée des possibilités.



- Gestes : principalement les mains, mais cela peut être le mouvement du corps, le tracking des yeux, le visage, etc.

La NUI fournit des interactions intuitives, et donc naturelles, dans le monde réel.

Les capteurs de gesture de type APDS-9960

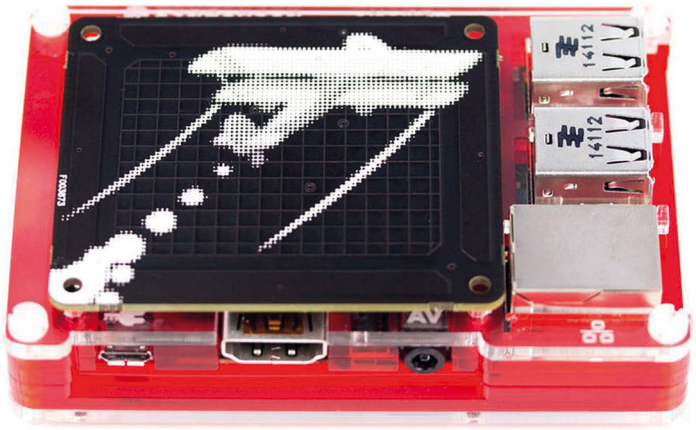
Pour expérimenter facilement cette interface entièrement dématérialisée, testons un petit capteur très simple : l'APDS-9960. Il existe de nombreux capteurs sous différentes marques. Fondamentalement, ils sont identiques : capteur RGB et capteur de gestes, le plus souvent sur une carte prête à l'emploi. Les headers sont parfois soudés. Parmi les usages possibles : reconnaissance des couleurs (basés sur le RGB) et reconnaissance des gestes.
On dispose des pins I2C (SDA, SCL), INT, GND, VCC (3,3 V généralement), VL (pour la partie RGB).

L'APDS-9960 utilise 4 photodiodes pour détecter les gestes. Le capteur convertit les mouvements physiques en information numérique. Il permet de détecter les mouvements simples (haut, bas, gauche, droite). Selon la capacité du capteur embarqué l'APDS-9960, les fonctions disponibles varient. On dispose aussi des gestes proches et loin (near / far) permettant des interactions simples.

Gestes plus complexes (cercle, zigzag) : oui en théorie, en pratique, cela dépend du capteur. Par exemple, le Grove Gesture autorise par défaut les cercles (sens horaire et anti-horaire), la vague. Grove utilise un capteur PAJ7620U2. Les capteurs APDS-9960 à bas prix ne supportent pas ces gestes en standard.

Par défaut, plusieurs usages sont privilégiés : détection des mouvements, détection des couleurs, lumière ambiante, remplacer un composant physique, contexte IoT / wearable. La calibration est faite en usine. Généralement, son utilisation se fait à une distance de 10 à 20 cm. La distance varie selon la sensibilité du capteur.
Pour en savoir plus : https://cdn.sparkfun.com/assets/learn_tutorials/3/2/1/Avago-APDS-9960-datasheet.pdf

SKYWRITER HAT : AIR INTERFACE POUR RASPBERRY PI !



Pimoroni propose une carte HAT pour Rapsberry Pi, la Skywriter. Le principe est simple : détecter les mouvements des mains sur 3 positions (X, Y, Z). Il promet donc bien plus que les capteurs vus plus hauts. La carte se programme en Python (librairie compatible). Il fonctionne selon le champ électrique ce qui permet une grande souplesse.

L'installation tient en une ligne :

```
curl -sSL get.pimoroni.com/skywriter | bash
```

L'installation est un peu longue, car il y a beaucoup de dépendances et de librairies à installer. D'autre part, certains usages / exemples nécessitent de faire un startx pour avoir un environnement dédié. Si l'expérience de la souris avec le skywriter est intéressante, à l'usage, c'est trop imprécis, mais on peut en voir tout le potentiel.

L'exemple donné par MagPi est intéressant : usage du skywriter pour faire télécommande (magic_presentation.py) pour les présentations Impress. Un autre projet montre le shield en télécommande d'une TV : http://frederickvandenbosch.be/?p=1554

Une version Arduino existe, mais nous ne l'avons pas testée.

Le branchement est très simple :

Capteur	Arduino
GND	GND
VCC	3,3 V
SDA	A4
SCL	A5
INT	2
VL	Pas utilisé

Nous utilisons la librairie Adafruit_APDS9960. Elle permet de manipuler les différentes fonctions du capteur. Plusieurs exemples sont fournis notamment pour les gestes et les couleurs.
Les gestes se codent très facilement :

```
apds. enableProximity (true);
apds. enableGesture (true);

uint8_t gesture = apds.readGesture();

gesture == APDS9960_DOWN
gesture == APDS9960_UP
gesture == APDS9960_LEFT
gesture == APDS9960_RIGHT
```

Attention, nous avons remarqué que la détection est sensible à la lumière, donc éviter une lumière directe sur le capteur. Attention aussi à la distance, car au-delà de 20-25 cm (au-delà de 10-15 cm, le mouvement est souvent mal reconnu), le capteur ne reconnaît plus rien.

Personnellement, nous voyons parfaitement ces gestes / interactions dans le wearable, dans les transports (par exemple en vélo, moto), contexte industriel contraint, les IoT du quotidien (miroir, lampe, etc.). La question se pose tout de même sur la sécurité, car nous ne pouvons pas vérifier l'identité de la personne qui fait le geste, pas avec les capteurs standards.

À vous ensuite d'imaginer les interactions et les capteurs et matériels supplémentaires. •

François Tonic

Fairphone 3, le téléphone modulaire et réparable : réalité ou simple promesse ?

Nous avons eu l'occasion de prendre en main la dernière itération, le Fairphone 3, sorti il y a quelques semaines.

Commençons tout d'abord, par le classique, les spécifications :

- **Système : Android 9**
- **SoC : Snapdragon 632 avec 4 Go de RAM**
- **64 Go de stockage par défaut + port SD pour du stockage supplémentaire**
- **2 logements SIM**
- **Batterie 3000 mAh**
- **Écran full HD 5,7 pouces**
- **Connecteur USB-C**
- **Lecteur d'empreinte**
- **Prise mini-jack pour l'audio**
- **Finition noire**

Côté utilisation, le Fairphone 3 ne diffère pas d'un autre smartphone. La prise en main est bonne, pas trop lourd, un écran amplement suffisant. La robe noire est réussie. L'écran LCD est honnête, inutile ici, de vouloir chercher la comparaison avec les OLED et AMOLED. Le constructeur a choisi un SoC relativement ancien (dans le monde mobile). On peut sentir ça et là des latences dans la réactivité, mais rien d'handicapant. Le Fairphone se positionne plutôt dans le milieu de gamme, entrée de gamme supérieure et pour des usages courants, hors jeux 3D.

Ce qui nous intéresse ici c'est son argument modulaire et sa capacité à être réparé par l'utilisateur. Ces promesses de départ sont plutôt excitantes : facilité d'ouverture, accès aux composants internes, possibilité de changer des modules, batterie amovible.

Un démontage facile

Effectivement, on retire facilement la coque arrière. Le smartphone se compose de deux éléments : le bloc écran et le bloc arrière. Il suffit de retirer les vis avec un petit tournevis (fourni). L'opération ne prend que quelques minutes. On retire la batterie et

on écarte (délicatement) les deux blocs. La connexion se fait par un gros connecteur. Celui-ci doit être fermement maintenu par deux vis, sinon le contact ne se fait pas.

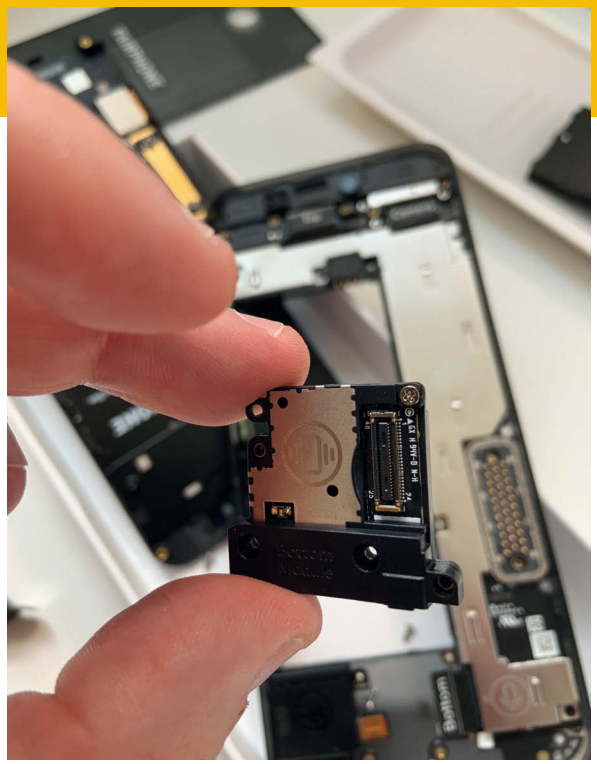
Les modules se situent sur la coque arrière. Là encore, il suffit de retirer les vis, de déconnecter le câble (là encore délicatement). C'est tout. Pour remonter, on fait la démarche inverse. Attention : ne forcez pas quand vous remettez le connecteur de la nappe. Vous risqueriez de tordre le cerclage métallique, autour du connecteur. Celui-ci est assez fragile.

Les modules que l'on peut changer sont : la caméra, le haut-parleur, module inférieur (contenant le moteur haptique, le connecteur USB-C, le microphone), le module supérieur (incluant la caméra frontale et divers capteurs). Et c'est tout. Par exemple, le bloc SIM + SD reste solidaire.

Pas de rétrocompatibilité

Une question naïve vient immédiatement en tête : peut-on utiliser des modules d'une version 1 ou 2 dans le Fairphone 3 et inversement ? La réponse est non : il n'y a pas de rétrocompatibilité. Le design des modules et des composants utilisés diffèrent en effet

entre les itérations du téléphone. Mais l'absence de rétrocompatibilité est une faiblesse de ce smartphone et répond donc partiellement à l'obsolescence programmée.
Le Fairphone aurait été particulièrement intéressant si la compatibilité des modules d'une itération à une autre était assurée et réelle. Pour nous, c'est l'avantage d'une approche modulaire. Et c'est la différence entre la notion de réparable et d'évolutivité. Aujourd'hui, Fairphone répond plutôt bien à la première idée, mais pas à la seconde. Espérons que les futurs modules seront compatibles avec la version 3 et donneront accès à de nouvelles fonctionnalités. Si un Fairphone 4 sort, la rétrocompatibilité sera attendue au tournant. A voir si le constructeur saura sortir des évolutions des modules.
D'autre part, les changements des modules ne se font pas à chaud. Il faut éteindre et redémarrer le mobile. De plus, s'il y a bien une app hardware, il est dommage de ne pas disposer d'une app dédiée pour voir l'intégrité du smartphone et des modules. Cela aurait été un plus.
Fairphone 3 est-il open source ? Le firmware est open source. Cette version est dépourvue des services Google. Par contre, il n'est pas open hardware. Et c'est peut-être là un défaut non négligeable pour l'avenir. L'ouverture du matériel pourrait inciter la communauté et des constructeurs à créer de nouveaux modules, imaginer d'autres usages. Est-ce que le constructeur osera franchir ce Rubicon ?
Aujourd'hui, Fairphone communique beaucoup sur une filière durable des métaux et terres rares utilisés dans les smartphones. C'est un véritable défi : comment utiliser des métaux recyclés tels que l'or, le cobalt, l'étain, le lithium ? Le traitement des déchets électroniques est un défi colossal, loin d'être efficace. Le constructeur veut que 40 % de 8 matériaux proviennent des filières durables / équitables.
Fairphone 3 est vendu 450 €, expédition à partir de mi-octobre.
Les modules seront vendus à partir de 19,95 €. Le module caméra est vendu 49,95 €, l'écran, 89,95 €. La question se pose sur les tarifs des modules : trop cher ou bon marché ? Nous sommes partagés. Pour être réellement attractif, des tarifs un peu moins élevés auraient été un argument. L'avantage est que l'on peut changer soi-même des composants sans risquer de casser une coque ou de mal remonter le téléphone. Fairphone est un petit constructeur par rapport à Samsung ou Oppo.
Nous remercions les rédactions de Les Numériques et de ZDNet. •



Les +
- Concept modulaire
- Smartphone adapté aux usages courants
- Batterie amovible
- Facilité d'ouverture

Les –
- Pas open hardware
- Tarif et tarifs des modules
- Ergonomie discutable sur certains éléments
- Pointe du tournevis trop grosse
- Modularité limitée

VR

HP VR BACKPACK G2

Vous vous souvenez peut-être du sac à dos MSI VR One. Il embarquait toute la puissance nécessaire pour y connecter un casque VR, le tout avec 2 grosses batteries pour tenir jusqu'à 90 minutes…

Depuis le printemps dernier, HP propose un concurrent, moins imposant et anguleux que le MSI : HP VR BackPack G2. Pour le nom, ils auraient trouvé plus fun.
Comme sur d'autres solutions, il embarquait un Core i7, Windows, le stockage, les batteries. Étonnement, les batteries ne sont pas directement intégrées mais sur des supports placés sur la ceinture. Sur le MSI, elles s'enfoncent directement sur le boîtier. Cela évite des câbles supplémentaires. Mais elles sont changeables à chaud. Ce qui est un plus. Ce marché semble tout de même très limité, surtout avec des casques VR qui deviennent autonome. A voir dans les prochains mois. Prévoyez minimum 3 900 €.

Maker / IoT

NOUVELLE CARTE NANOPI M4V2

Les NanoPI sont de bonnes alternatives à Raspberry Pi, avec une large gamme. Le constructeur a dévoilé une nouvelle carte, la NanoPi M4V2. Il s'agit d'une nouvelle itération de la M4. Le SoC est un classique RK3399. La grosse nouveauté est la nouvelle mémoire : LPDDR4. Le constructeur simplifie la gamme, ce qui est une bonne chose, avec uniquement une version 4 Go à 70 $. Pour le reste des specs, pas de changement entre la 1ere M4 et cette v2. Notons qu'elle possède 4 ports USB 3, USB-C pour l'alimentation.

Vintage

THE C64 VERSION RETROGAMING POUR 120 €

Le mythique C64 va revenir dans une version moderne et de dimensions plus réduites. Il ne s'agit pas de recréer un C64 avec la même électronique, dommage. Ce modèle proposera une prise HDMI avec possibilité d'avoir un mode VIC 20. Le firmware pourra être mis à jour et un slot SD sera disponible. 64 jeux seront installés par défaut. Aurait pu mieux faire. Mais au moins, on retrouve un vrai clavier, le fameux Basic.
Pour le même prix, vous trouvez un vrai Commodore 64 complet avec lecteur de disquettes !

Formation tout au long de la vie d'un informaticien : le mentorat

Jean-Noël GERARD, Ingénieur en informatique depuis 20 ans, freelance, éternel étudiant et mentor chez OpenClassrooms

Prolonger sa formation, recommencer un nouveau cycle, entrer dans une nouvelle carrière ou tout simplement rester à jour, autant de raisons de continuer à se former tout au long de sa vie professionnelle. Des dispositifs divers et variés ont vu le jour et les pouvoirs politiques ont renforcé les droits des travailleurs avec la création d'un compte formation (le CPF).

Se former : une réalité bien tangible, retour d'expérience

Diplômé d'un MASTER 2 FEDE en informatique, j'ai changé de travail en informatique toutes les 4 à 5 années. A chaque palier ou progression, des nouveaux besoins sont arrivés, des nouvelles technologies, nouvelles méthodes, nouveaux concepts et pas des moindres. Entre 1990 et maintenant, Internet s'est démocratisé, marginalisant les systèmes centraux, puis les systèmes legacy, Java est apparu face à Visual Basic, C, C++, COBOL, et Pascal. Certains langages ou systèmes sont bel et bien morts en route, d'autres existent toujours. Il a fallu se former sur des nouvelles technologies, puis sur des nouvelles méthodes, passant du cycle en V, aux méthodes dites en cascades, et ensuite aux méthodes Agiles. Changements de pensées, changements de paradigmes et changements sociétaux avec l'internet d'abord, puis l'IoT. Pourtant, pour continuer d'exister et rester un professionnel de l'informatique il faut continuer à être à jour. Mais comment ? A quel rythme ? A quel coût ?

Au-delà de cursus bien connus, en mode intra entreprise ou en formation scolaire, d'autres modes d'apprentissage commencent à émerger et se répandre, y compris dans les entreprises. Les entreprises doivent mettre en place une réponse à l'obligation générale de formation des salariés qui leur est faite. Dans les systèmes de Ressources Humaines, on appelle cela la GPEC, la Gestion Prévisionnelle de l'Emploi et des Compétences. Tout un programme ! Et bien sûr, pas question de répondre qu'on va se voir un petit tuto sur une chaîne internet ! Vous l'avez bien entendu ces dernières années le « petit tuto » qui vous apprend juste ce que vous voulez savoir et qui est réalisé au fond du garage avec un smartphone. Preuve que tout le monde est concerné et sous toutes ces formes !

Mais les entreprises doivent-elles se substituer à des organismes de formation ? Certes elles ont des besoins spécifiques mais elles ne sont pas les spécialistes de la formation. Alors déployer un système pour cela, est-ce réellement nécessaire ? Et comment suivre les tendances ? Et comment créer ces cours ? Très rapidement on voit que la transmission des connaissances se fait par divers moyens, que c'est l'affaire de tous et que vouloir y répondre d'une manière unique et dirigée est inepte. Place donc aux spécialistes de l'apprentissage.

Gamifier pour mieux apprendre

Un des axes les plus répandus pour faciliter l'apprentissage est la gamification des contenus. En effet, plus que de long, trop longs cours magistraux, la tendance est à de petits morceaux de savoir mis en scènes : vidéos, présentations cliquables, quizz… L'interactivité est le moteur, et, le jeu, l'essence qui vous permet d'apprendre sans vous en rendre réellement compte. Ce sont les « serious games » où on peut réviser, ré-apprendre, compléter ses connaissances. Ce genre d'apprentissage est disponible sur tout un tas de LMS (Learning Management Systems) qui gèrent tous types de contenus.

Rapides à mettre en œuvre, souvent proposés en modules SaaS, à la demande et disponibles 24h/24, les serious games sont de vrais atouts dans une politique de ressources humaines et évitent à l'entreprise des coûts de déploiement et de maintenance d'un système de type LMS. Certains permettent également d'ajouter des modules spécifiques à l'entreprise et complètent les cours déjà proposés. C'est le principe de Pareto appliqué à l'apprentissage : la société fournit les 20% de spécifique et le système les 80% restants.

Limites de la gamification

Les limites de cette approche sont nombreuses car la gamification du contenu nécessite de gros moyens en terme de présentation, scenarii, tests et comme tous les contenus multimédias, de leur investissement dépend directement leur qualité mais pas forcément leur pérennité. Avec des contenus rapidement obsolètes, des sources externes moultes et variées, sauf à avoir une audience rapide et suffisante, aucune possibilité de réellement rentabiliser les budgets alloués aux créations.

Comme tout support de communication, ils subissent aussi les tendances et doivent être disponibles sur tous supports, en tous lieux, en quantité, en qualité et en formats différents en durée, en graphisme…

Alors, la gamification des contenus : effet de mode ?

Il n'est pas sûr et certain que la gamification soit un effet de mode même si cela est arrangeant ces dernières années. En effet, la gamification a été bien vendue aux DRH. J'ai eu l'occasion de discuter avec un client DSI sur l'évolution de son système d'information dédié aux ressources humaines. Entre l'évolution du SI interne, long, progressif et dont les ressources humaines ne sont pas la priorité, et la disponibilité du contenu multimédia et immédiat, distribué par SaaS, les serious games à faible prix ont été plébiscités. Ils posent des soucis de sécurité informatique, car il faut le plus souvent, pour pouvoir suivre et justifier des coûts, être capable de lier le système de l'entreprise (un Active Directory par exemple), avec celui du prestataire en SaaS. Et ce prestataire n'est pas forcement en Europe, Canada voire au *safe harbour** états-unien. Autrement dit, après la CNIL, le RGPD oblige les entreprises à savoir où sont hébergées les don-

**Safe Harbour : c'est le nom donné aux US aux 4000 entreprises qui ont signé une charte de protection des données équivalentes à celle de l'Europe. Au delta près que la Patriot Act permet à l'état fédéral des Etats-Unis un accès total en tous temps.*

nées. La rapidité d'achat, une simple carte bleue suffit, puis déploiement dans un service peut tout simplement être à l'encontre des obligations légales de l'entreprise et ce malgré les économies budgétaires importantes qu'on peut réaliser en gamifiant certains pans d'un système informatique. J'ai vécu ce casse-tête entre efficacité, rapidité, simplicité d'une part, et sécurité, fonctionnalité, légalité d'autre part. Et les contenus toujours plus riches, toujours plus attirants l'ont emporté sur le système plus rigide, et plus lourd à faire bouger.

Jusqu'où aller dans la gamification ?

Chez un autre de mes clients, il a été procédé à la mise en place d'un système complet d'académie interne. Avec grand renfort d'équipes dédiées sur l'évangélisation du système, un plan de communication sur 3 années et des opérations bien visibles : des corporate awards. La grosse équipe interne et externe de consultants pour s'assurer que chacun des employés du groupe pouvait avoir accès aux contenus multimédias tous gamifiés. Et tout y est passé : badges de réussite, contenus dédiés, contenus débloqués en fonction des réussites, mise en avant des performances des acteurs, ouvertures à un maximum de collaborateurs externes etc., etc.

Mais n'est-ce pas aller trop loin ? Si je suis prestataire ai-je vraiment le droit de passer ma vie sur l'outil de coorporate academy pour me former ? Certes j'ai besoin d'être à jour de mes connaissances, de connaître mon client pour mieux le servir. Mais est-ce que cela m'autorise à passer plusieurs heures par jour sur ce système d'apprentissage ? A continuer à m'y connecter le soir, le week-end, à la pause café ? C'est addictif et c'est fait pour.

Tiens la newsletter corporate vient de tomber dans ma boîte aux lettres ! Voyons voir qui est le premier au classement par étoiles, par badges et combien j'ai gratté de places ?

La question est évidente si on compare le temps de travail effectif, la valeur ajoutée proposée, supposée, vendue et celle réelle : est-ce à l'entreprise cliente de former les collaborateurs externes ? Panum, ludum, du pain et des jeux… le nouveau paradigme où l'entreprise offre tout ce que le bon peuple a toujours voulu avoir ?

Plus sérieusement, la vraie question est de savoir comment faire une bonne approche par le jeu sans que cela tombe dans le n'importe quoi. Il y a des limites à cet ensemble de nouvelles méthodes d'apprentissage permanent et à la réelle pédagogie qui en ressort. Avoir libre accès à tous contenus, en tous lieux, est-ce ce que Rabelais appelait l'abbaye de Thélème et sa devise célèbre : « fay ce que vouldra » ?.

L'academy externe est peut-être un mix souhaitable entre liberté et guidage pédagogique, un juste milieu nécessaire entre le ludique et le professionnel. En bref, la dernière évolution en matière de digitalisation des contenus de connaissance.

26 Academy, par exemple, utilise dans ses modules de cours l'ensemble des types de contenus pour vous préparer à diverses certifications telles que le SCRUM MASTER. A votre rythme, avec un tuteur joignable en cas de soucis et avec des devoirs à rendre.

J'ai été personnellement amené à découvrir ces nouvelles façons d'apprendre et de se certifier sur un sujet suite à la demande d'un client pour une mission : l'application directe des connaissances acquises. Etant en portage salarial chez BENETT PORTAGE, c'est mon employeur qui m'a proposé cette formation dans le cadre de mon compte personnel de formation (CPF).

Pourquoi cette expérience m'a plu ? Par ce qu'il faut bien matérialiser des compétences plus que des connaissances. Car là est tout le but de l'apprentissage tout au long de la vie professionnelle : la mise en application réelle des connaissances. Mais également parce que l'on n'apprend pas de la même façon à 20 ans qu'à presque 50 ans. Et enfin parce que toutes les parties prenantes, client, employeur, employé et société de prestation, se sont mises autour de la table pour une solution commune.

Mais pour moi, il existe une autre voie d'apprentissage que ces dernières évolutions digitales que nous venons de parcourir.

Apprendre à Apprendre

Apprendre c'est d'abord être en condition pour apprendre, avoir un plan de formation et finalement ce qu'on oublie souvent, il faut commencer par le début et se poser les bonnes questions. Pourquoi dois-je apprendre ? Quel but est-ce-que je poursuis ? Etc., etc.

Chez OpenClassrooms, Apprendre à Apprendre est le titre du premier cours que vous devriez faire, avant même d'entamer vos formations. Preuve s'il en est que ce n'est pas trivial.

OpenClassrooms, est l'ex « site du zéro », site bien connu qui a évolué dans son approche de l'apprentissage, et propose soit l'apprentissage libre, soit du mentorat, avec obtention possible de diplômes reconnus par l'état (inscrits au RNCP).

Des connaissances aux compétences : le mentorat chez OpenClassrooms

D'abord, le mentorat c'est quoi ? Et comment est-il mis en musique ? Le principe du mentorat est l'accompagnement d'une ou plusieurs personnes sur des situations ou des projets réels, afin de développer ses compétences.

Le mentor est un professionnel de la discipline, exerçant ou ayant exercé dans celle-ci. Le mentor met à disposition ses connaissances, son réseau, ses compétences auprès de la communauté des mentorés.

Plus qu'un apprenant, le mentoré est, quant à lui, amené à tester ses capacités sur des projets, en toute autonomie. Autonomie dans sa gestion du temps, dans le choix de ses outils de travail, dans sa vélocité de progression, son temps libre…. Il a des points réguliers avec son mentor à qui il explique son avancement, ses doutes, ses questions et échange régulièrement avec son mentor mais aussi la communauté, au travers des outils mis à disposition.

Contrairement à un enseignement classique, le mentorat nécessite une relation construite, comme un partenariat, entre les deux personnes avec une prépondérance à la gestion de l'humain. Le mentoré doit pouvoir se tester et développer ses compétences sans être jugé mais en étant accompagné pédagogiquement et professionnellement.

L'équipe pédagogique d'OpenClassrooms est chargée de tout ce qu'on ne voit pas : préparer cours, tutoriels, gérer les mentors, les mentorés, les parcours, les relations publiques, y compris avec les autorités et ministères de tutelle. Et j'en oublie !

C'est un vrai soutien d'artillerie avec pour but que tout fonctionne, un rouage essentiel souvent invisible et garant d'une qualité certaine.

Au jour le jour, le mentor doit être à l'état de l'art mais également à l'écoute des propositions du mentoré. Durant son parcours, le mentoré doit rendre des travaux qu'il réalise à son rythme, ce qui suppose un travail fourni, récurrent, continu de la part du mentoré et un travail également du côté du

mentor qui doit aider le mentoré tout au long du parcours. Et c'est dans l'échange et l'écoute, deux qualités impératives dans ce métier que le mentor peut exercer sereinement ses prestations.

En effet, soyons humble, le mentor ne sait pas tout, il est là pour accompagner et jauger des progrès et compétences, interroger, proposer des cours complémentaires pour limiter les lacunes, renforcer les connaissances, guider dans l'apprentissage de manière générale et bienveillante. Ce qui veut dire une certaine appétence à l'apprentissage permanent, une curiosité sur tous sujets et un réel échange avec son écosystème. Au quotidien , le mentor peut participer aux diverses actions proposées par l'équipe pédagogique et en relation avec les parcours, les mentorés, etc., etc. Que ce soit en termes d'amélioration continue, de nouveaux parcours, de refonte de parcours, d'échanges sur les réseaux sociaux ou internes, il est toujours possible de participer à la vie globale de la structure. Le mot de communauté n'est pas galvaudé car les débouchés sont aussi possibles : il arrive fréquemment que les divers acteurs partagent des offres d'emploi, annoncent avoir trouvé un emploi, ce qui est la finalité, au-delà de la formation, pour beaucoup.

Etre utile aux mentorés, avec qui on monte une vraie relation professionnelle, pouvoir les aider et les guider pour la suite est quelque chose de très gratifiant. Les mentorés sont heureux de partager leurs réussites, nouveaux emplois mais également leurs produits ou livrables tout au long de leur parcours de formation. En favorisant les échanges, Openclassrooms arrive à fonder une vraie entraide dont les mentorés sont une des pièces principales. En tant que mentor, nous sommes tous sensibles à cela car notre motivation principale n'est pas que l'argent issu de la prestation, c'est pour beaucoup, le passage de connaissance dans un parcours guidé. Non seulement on peut développer sa propre façon de transmettre mais on peut également échanger et partager des points de vue différents, avec des acteurs de tous horizons. Tout cela demande à la fois du travail supplémentaire autre que les sessions hebdomadaires, du recul et de la prise de conscience de l'environnement global dans lequel nous vivons. Cela a été et reste ma motivation personnelle primordiale : aider, transmettre et échanger. J'ai eu la chance d'avoir un parrain, un ami qui m'a proposé de rejoindre le réseau, j'ai donc pu, avant même de rejoindre Openclassrooms, tester et observer en direct le déroulement sur plusieurs semaines. Cela m'a plu et j'ai passé le pas. Je ne le regrette pas et le feedback des mentorés m'encourage à continuer. Merci à eux.

Le métier de mentor permet également de voir et d'observer les mutations du monde du travail dans l'informatique, les changements sur les technologies, le vocabulaire ou les méthodes utilisées. Il permet aussi de s'interroger profondément sur les évolutions à venir sur la sphère informatique.

Aussi, puisque c'est l'une des questions de ce dossier, si je devais donner mon sentiment sur le métier de développeur aujourd'hui, il serait très contrasté. D'une part nous avons un métier qui a complètement été modifié et où, finalement, le code n'est plus qu'une infime partie du temps passé par le développeur. Entre les IDE qui font tout pour soi-disant simplifier le métier, entre les externalisations de personnel vers les pays moins chers, ou encore la philosophie DEVOPS où l'on passe plus de temps à tenter de travailler ensemble vers un but commun, il est difficile de savoir ce que sera le métier de développeur demain. Certes les ultras-spécialistes arriveront toujours en renfort sur des points ultra pointus, à fort tarif. Mais pour 80% des développeurs d'il y a 20 ans, s'ils ne passent pas au DEVOPS, il sera difficile de s'intégrer aux équipes et peu de solitaires vont pouvoir vivre encore longtemps de leur code. Car la profession a tendance à aller vers le moins disant salarial, de plus en plus. Plus de formation, plus de connaissance, mais moins bien payés et avec une rotation plus rapide des équipiers, des missions… Qui aujourd'hui pourra prétendre rester 10 ans chez le même client en étant DEVOPS ? Gageons qu'il y aura toujours un contre-exemple pour justifier des discussions à n'en plus finir. Laissons cela aux comptoirs des bistrots, à ce jour, plus que l'individu, le DEVOPS est avant tout un membre d'une équipe. La meute sera la seule solution pour survivre et être performant, mais là encore des chausses trappes existent.

Comment intégrer dans des équipes plusieurs générations de travailleurs ayant des philosophies différentes, des cadres de référence différents, des comportements différents. A ce jour, pour moi, le principal enjeu est la cohabitation de personnes junior, trentenaires, avec les personnes de la génération précédente, les cinquantenaires, au sein d'équipes de type DEVOPS. Avec l'expérience et le recul nécessaire de celui qui a programmé en mode contraint sur 4 Ko de mémoire, tout en assembleur, en Cobol ou en C++ « on core on metal » comme on le dit souvent, et avec la jeunesse, les nouvelles technologies, nouveaux langages, on peut monter des équipes très compétentes et très professionnelles. Et comme l'équipe est faite d'individus, l'humain devra être la première priorité dans le management de ces équipes dont les développeurs. Et pour durer tous devront se former dans le mode de formation qui lui conviendra.

Conclusion

Dans la première partie de ce dossier, voir Programmez! de septembre 2019, nous avions pu constater que le marché de la formation en informatique était vaste, adressé par de multiples écoles et formations mais que dans tous les cas le volume est tel qu'il faut d'autres manières d'adresser celui-ci. Le modèle de mentorat proposé par OpenClassroom est un autre paradigme qui s'adresse à tous ceux qui ont besoin de se former tout au long de leur carrière et accompagné par des professionnels. C'est un autre angle pour aborder cette problématique qui peut répondre à la multiplicité des profils, des besoins, tout en restant épaulé, accompagné si nécessaire. •

VINCENT HAGEN, 29 ANS,

Chef de projet
Digital Freelance, Mentoré chez OpenClassrooms

« J'avais très envie de faire la formation Chef de projet digital de chez OpenClassrooms, car c'est grâce à leur cours que j'avais commencé la programmation. Je me forme assez souvent pour acquérir des compétences et répondre aux besoins des entreprises. Ce qui m'a poussé à passer à l'action, c'est qu' OpenClassrooms met en place ses formations en restant à l'écoute de ces besoins, chose que j'ai réussi à vérifier rapidement.

Nous avons accès à un certain nombre de cours que nous pouvons appliquer sur des missions que nous devons valider par des soutenances face à un professionnel. Nous avons aussi un mentor (également un professionnel) qui répond à nos questions et qui nous donne des conseils pour assurer le bon déroulement de cette mission.

L'avantage de cette formation, c'est bien que nous acquérons des connaissances avec les cours, nous développons aussi nos compétences. Je sais que je serai plus efficace face à la demande des entreprises. »

Cobol : toujours d'actualité !

Difficile de faire plus vintage que le mainframe et le langage Cobol. Il ne faut jamais oublier l'importance de ces plateformes dans l'industrie, la banque et l'assurance. Les entreprises seront nombreuses à dire : oui je veux changer, oui je veux m'en débarrasser. Dans certaines études, on peut voir une large majorité d'entreprises voulant sortir du mainframe, mais dans le même temps, une majorité relativise : il est difficile de modifier, de migrer des applications Cobol et de sortir du mainframe proprement dit.

Cobol a réussi à se moderniser sur plusieurs points :

- On peut utiliser un IDE moderne : ce n'est pas nouveau, mais c'est bien de le rappeler
- Le langage reste lisible dans sa syntaxe et efficace dans les usages qui sont les siens
- On peut moderniser de plusieurs manières les codes Cobol : recompilation avec adaptation/optimisation de certains codes, réécriture de codes/replatforming

On parle beaucoup de modernisation, de transformation, de migration, de replatforming. Les raisons de sortir du mainframe sont nombreuses (le coût revient souvent), mais ces projets restent sensibles, à l'aboutissement incertain.

Une des questions qui se pose : mon code vieux de 30 ans fonctionne, ne plante jamais. Est-ce que le nouveau code sera aussi fiable, fonctionnant de la même manière, etc. Réécrire des programmes entiers n'est jamais simple.

Applications mainframe : quels sont les défis de leur développement ?

LzLabs a créé la solution Software Defined Mainframe pour une migration des applications historiques existantes vers des environnements modernes sur x86 ou dans le cloud, sans recompilation ou reformatage de données.

Didier Durand, Chief Product Officer, LzLabs

Régulièrement, nos clients nous expliquent que leurs applications mainframe historiques ne sont pas en mesure d'évoluer assez vite pour répondre aux besoins métier évoluant eux-mêmes de plus en plus rapidement dans leur entreprise.

Ce constat semble être la toute première raison qui sous-tend l'essor de l'intérêt que portent les entreprises à la migration de leurs applications sur d'autres plateformes (une forme de modernisation), soit en réécrivant les codes des applications ou dans le cas de nos clients, en les redéployant dans leur forme courante sur une autre plateforme (le replatforming).

Au-delà des questions d'agilité qui émergent avec la volonté de modernisation des applications, un détail d'importance mérite examen : pourquoi est-il apparemment à ce point difficile de moderniser les applications COBOL / mainframe ? Le problème ne vient pas du COBOL et des autres langages historiques. Tout bon programmeur qui se respecte est « polyglotte », donc habitué à jongler entre plusieurs langages. Le COBOL n'est certes pas forcément le langage le plus « tendance » mais il est couramment utilisé dans certains secteurs économiques.

Afin de comprendre ce qui est au cœur des défis de modernisation des applications historiques, nous avons conduit des discussions avec nos clients, des prestataires extérieurs et des intégra-

teurs système. À chaque fois, ce sont les quatre mêmes grands enjeux qui émergent.

Outils de développement : une faible productivité sur les applications mainframe

L'environnement traditionnel d'un programmeur COBOL ou PL/1 sur mainframe dispose de peu de fonctionnalités par rapport aux environnements actuels : références croisées et renvois intelligents, vérification immédiate de la syntaxe, auto-complétion pour les variables y sont absents. Plusieurs tentatives ont été menées afin de rapprocher le modèle mainframe et un environnement de type Java. Mais le travail s'effectuera en majeure partie sur le back-end mainframe. Les langages de programmation mainframe sont ainsi confinés dans leur environnement d'origine et dépendent directement des outils dont dispose le développeur.

Indisponibilité de l'open source : un frein à l'innovation

La corne d'abondance des projets open source (100 millions de référentiels de données disponibles sur GitHub[1]) répondant aux exigences métier, est l'une des principales raisons pour lesquelles les processus de développement sont devenus globalement plus agiles. Concrètement, moins la quantité de code « interne » nécessaire à un besoin métier donné est importante, plus vite cette exigence sera satisfaite. Le nombre de projets open source qui ont été mis en œuvre et testés sur des mainframes historiques est très faible. Plus troublant encore peut-être, il reste difficile d'intégrer de manière naturelle des solutions orientées Linux au sein d'applications historiques.

Intégration en pipeline : un pipeline et des processus distincts pour le développement des applications mainframe

Les pipelines de développement modernes ne s'adaptent pas naturellement au développement d'applications sur mainframe ni à leur déploiement. Les processus de build, les tests, la gestion du code source et la mise en production sont intégrés et rationalisés au sein d'un pipeline de développement moderne. Plus précisément, les outils modernes permettent aux développeurs de déployer des changements testés en local et éprouvés via un pipeline de test automatisé, avec des niveaux d'intégration en augmentation constante avec les applications et données existantes. Un tel pipeline sera facilité par l'intégration étroite d'outils tels que Git, Eclipse, Jenkins, des conteneurs Docker, etc., qui permettent de tirer parti des meilleures pratiques d'intégration et de déploiement continus (CI/CD). L'environnement mainframe ne bénéficie pas, de par sa nature, d'une telle agilité.

Test – Autonomie et automatisation, des compétences limitées

Les tests sont, généralement, automatisés, planifiés et évolutifs. S'ils sont bien conçus, ils pourront réduire de 63 % les retards dans le processus de production logicielle, représentés par les tests/assurance qualité au sein des environnements obsolètes/historiques[2]. La capacité à lancer en un seul clic des ensembles entiers de conteneurs, comprenant l'ensemble des composants d'une application et de son environnement technique, lors de n'importe quelle étape du test, est l'une des principales raisons pour lesquelles le développement est aujourd'hui plus rapide qu'à l'ère du mainframe.

Les attentes concernant la phase de test impliquent désormais la capacité à lancer un environnement de test complet sur le poste de travail du développeur, sans avoir recours aux administrateurs système, pour les tests unitaires récurrents qui sont étroitement liées au développement lui-même. En raison de la dépendance matérielle physique du développement mainframe, une telle autonomie de tests, qui apporte une agilité optimale, est tout simplement impossible pour les applications historiques.

Les tests menés à grande échelle peuvent être planifiés régulièrement, automatiquement et à moindre coût dans des environnements de cloud modernes basés sur des conteneurs. Pour les applications historiques, de tels tests exigent un budget et une préparation importants.

Par conséquent, tous les tests dans le cadre du développement d'applications historiques sont rallongés dans leur durée par le besoin d'impliquer les administrateurs système mainframe, et alourdis par des ressources coûteuses liées au mainframe.

Comment LzLabs Software Defined Mainframe® (LzSDM®) résout ces problèmes ?

La solution LzSDM casse la dépendance logicielle entre une application historique et un mainframe « physique ». L'objectif est d'injecter des services Linux ou d'autres projets open source à côté des bibliothèques LzLabs.

Ces bibliothèques proposent des services qui recréent les comportements du système d'exploitation, des bases de données, de la sécurité et de l'exécution du langage qui auparavant créaient un lien très fort entre l'application et le matériel mainframe.

Une fois l'application mainframe et son environnement technique représentés sous la forme d'un ensemble de bibliothèques logicielles, comme toute autre dépendance applicative plus conventionnelle, le problème du développement d'applications historiques est résolu dans son intégralité. Tout poste de travail ou serveur d'entrée de gamme peut virtualiser en totalité l'environnement mainframe sur plusieurs instances en parallèle, si nécessaire, grâce aux conteneurs. Et lorsque cette opération est réalisée sur le cloud, l'élasticité est totale.

Par cette transformation, le développeur pourra maintenant exécuter un environnement de test complet pour des applications historiques sur son poste de travail. Cette opération est la même que s'il la menait pour un programme Java ou C#. La totalité du cycle de développement des applications historiques peut suivre le même pipeline que s'il s'agissait d'autres applications. Les mêmes modèles agiles et mêmes méthodologies DevOps CI/CD avec la même importance accordée aux tests réalisés très en amont (dits « shift-left testing ») peuvent être employés. Les mêmes tests évolutifs automatisés sont également possibles. Il est en outre possible de réaliser une intégration aisée de projets open source.

En modernisant ainsi les applications historiques, on pourra retirer le terme « historique ». Elles restent des applications COBOL ou PL/1 dans un environnement moderne.

(1) https://github.blog/2018-11-08-100m-repos/

(2) https://www.continuoustesting.com/2019/02/12/continuous-testing-what-why-and-how/

François Tonic

Cobol : 60 ans et toujours en forme

Cobol est LE langage increvable de l'informatique. C'est le 18 septembre 1959 que la première version du langage apparaît officiellement. Le chantier du langage démarre quelques mois auparavant, le 8 avril. Durant une réunion, les premières spécifications sont discutées sur un nouveau langage qui s'appelera COBOL (Common Business-Oriented Language). Il s'inspire de FLOW-MATIC. Le projet fut renforcé fin mai avec des ressources provenant aussi bien d'éditeurs et de constructeurs privés (notamment IBM, RCA, Honeywell) et des agences gouvernementales (US Air Force, US Navy et le National Bureau of Standards).

Le nom COBOL fut acté en septembre 59. Les spécifications complètes du langage furent définitivement figées en décembre. Il faut néanmoins attendre 1960 pour voir les premiers compilateurs. En décembre 60, la compilation du même code COBOL sur deux ordinateurs différents fut réalisée avec succès. Une première dans l'histoire de l'informatique naissante.

Un langage simple répondant aux besoins

COBOL est, par conception, un langage orienté métier, gestion. Il a un grand mérite : il décrit précisément les choses que l'on souhaite qu'il fasse. Jusqu'à présent, les langages étaient difficiles à maîtriser, proche du langage machine ou encore des cartes perforées (eh oui pas de clé USB ni de SSD en 1959-60). L'un des objectifs des travaux autour d'un nouveau langage était de créer quelque chose de facile à écrire et à lire. C'est pour cela que COBOL a une syntaxe et une structure proche du langage naturel. Et naturellement, il utilise une syntaxe anglaise.

Il faut comprendre que COBOL répondait à un besoin des entreprises et des administrations : un langage capable de traiter les données et d'en fournir les résultats (via un affichage). COBOL sera une véritable révolution en ce domaine. Comme le big data, il y a quelques années. Et son concept d'origine répond parfaitement à ce besoin orienté données. Il gère les données, les traite et permet d'afficher les informations demandées. Ce n'est pas pour rien si COBOL a été massivement utilisé dans les banques, assurances, mutuelles, RH, comptabilité et les administrations publiques.

Mais, cette simplicité rencontre sa limite avec les besoins de transactions qui explosent et la masse de données qui ne cesse d'augmenter. Les temps de traitement s'allongent et il faut adapter les vieux codes. En COBOL, il n'est pas possible d'exprimer, ou très mal, des algorithmes très complexes, ni de faire des expressions lambda ou d'utiliser la notion de classe ou de coroutines. Mais si COBOL est un langage très structuré, on ne peut pas le comparer à un langage natif comme C++ ni à un langage dynamique. Il n'est pas fait pour ça. La dernière version normalisée est le Cobol 2014.

Une structure qui peut étonner

COBOL ne gère pas la sensibilité à la casse, donc majuscule et minuscule, c'est la même chose. Les 3 opérateurs booléens n'étaient pas symbolisés dans la syntaxe. Par conception, il reconnaît 52 caractères (chiffres, lettres, caractères spéciaux), une instruction peut tenir sur plusieurs lignes, mais il faut respecter le colonnage, etc.

COBOL étant très strucuté, il impose une logique hiérarchique. Le langage n'est pas permissif. Il existe 4 divisions (grosso modo des blocs de code ayant chacun une utilité précise dans le programme COBOL) qui elles-mêmes contiennent des sections qui sont formées de paragraphes. Chaque paragraphe commence par une étiquette et contient des phrases se terminant par un point (comme le ; que l'on oublie souvent). Chaque phrase est une ligne de code contenant des instructions et elle commence forcément par un verbe avec, si besoin, des clauses (en quelque sorte des paramètres).

La division identification est obligatoire. C'est la « fiche » informative du programme avec son nom, des commentaires, etc. La division « environnement » décrit l'environnement d'exécution : système monétaire, signe décimal, etc.

On dispose de boucles conditionnelles, tellement pratiques : if, then, else, end-if. On peut faire appel à des sous-programmes et des boucles. On gère aussi les Entrées/Sorties : par exemple avec l'instruction display. Par exemple :

DISPLAY formulaire-fiche : on affiche le formulaire ayant pour nom « formulaire-fiche ». L'instruction ACCEPT « accepte » les données saisies dans le formulaire et les garde en mémoire.

À l'époque de la création du COBOL, les cartes perforées étaient utilisées pour les programmes. Elles font 80 colonnes. C'est pour cela qu'un code COBOL se limitait à 80 colonnes, pour ne pas déborder de la carte perforée.

Ainsi, on comprend pourquoi le COBOL s'organise en colonnes :

1-6 : pour le numéro de la page et les numéros de lignes. Obsolète.

7 : ligne réservée pour * (commentaire), - pour les phrases sur plusieurs lignes, / pour stopper l'impression, D pour le debug…

8-72 : c'est là que l'on écrit le code proprement dit. Les colonnes 8-11 sont pour les divisions, sections et paragraphes. C'est que l'on appelle l'AREA A. Les colonnes 12 à 72, c'est tout le reste : les phrases. C'est l'AREA B.

73-80 : obsolète. Ne doit absolument pas contenir de code. Le compilateur ne prenait pas en charge ces colonnes. C'est là que l'on trouvait le nom du programme, de l'écran, du serveur, le nom de la macro, etc.

GRACE HOPPER

Grace Hopper est une des pionnières de l'informatique dans les années 50 et 60. Elle va jouer un rôle central dans la création de Cobol. Mais son travail va bien au-delà :

- Supervision de la création du langage Math-Matic qui est en développement dès 1955.
- Flow-Matic : en développement à partir de 1955 pour les environnements Univac.

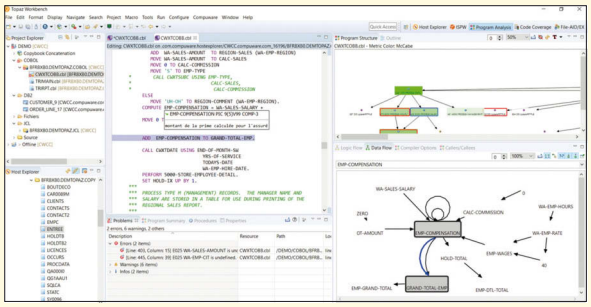
L'automatisation et les outils modernes boostent l'efficacité des développeurs mainframe

Avec des données et programmes majoritairement Cobol, le mainframe reste le cœur de l'IT de nombreuses grandes entreprises, particulièrement dans le secteur bancaire et toutes ne sont pas prêtes à quitter leur plateforme historique. Avec plus de MIPS mainframe en cours d'exécution qu'à aucun autre moment dans l'histoire, le mainframe doit devenir aussi agile que les autres plateformes pour répondre aux besoins des métiers sans compromettre la qualité du développement applicatif. Pour ce faire, les entreprises dotent leurs équipes techniques d'outils modernes, de processus automatisés et d'agilité. C'est dans ce contexte que l'éditeur Compuware les accompagne dans une démarche de modernisation et de transition agile.

On parle beaucoup de transformation numérique, de replatforming, de migration, de recompilation. « Beaucoup de projets impliquant la réécriture d'un grand volume d'applications mainframe hautement performantes et sécurisées en applications Cloud ont, dans la pratique, échoué. Les entreprises sont conscientes de la nécessité de capitaliser sur leurs applications stratégiques mainframe existantes tout en modernisant le front-end.» Constate Benjamin Beurier, Technical Account Manager chez Compuware.

Si l'image du mainframe n'est pas toujours flatteuse : écrans verts, ergonomie des moyens de développements dépassée, technologies peu connues pour beaucoup de jeunes développeurs, il reste une réalité bien ancrée dans les grandes entreprises. Compuware aide les équipes techniques à moderniser les outils en permettant aujourd'hui d'ajouter de l'agilité et d'adopter une culture DevOps. Cela passe notamment par l'automatisation des builds et des tests.

« Notre rôle est d'accompagner les entreprises dans leur transition aux méthodes agiles sur le mainframe. Selon nos propres observations, celles qui déploient de l'agilité à l'échelle ont accéléré leur innovation jusqu'à 80 % ! On entend beaucoup parler de replatforming mais nous ne le voyons pas sur le terrain, l'entreprise voulant capitaliser sur son patrimoine technique. Le mainframe est une infrastructure pérenne qui fonctionne de manière sécurisée, fiable et performante. » Rappelle Benjamin Beurier.



SOCIÉTÉ GÉNÉRALE : ZDEVOPS

Société Générale est l'un des premiers groupes européens de services financiers depuis plus de 150 ans. La DSI de la Banque de détail en France du Groupe s'est engagée dans une démarche de transformation DevOps et de Continuous Delivery axée sur la valeur et l'innovation. Cette transformation passe notamment par l'acquisition de nouveaux outils performants pour moderniser l'environnement IT incluant le mainframe.

zDevOps fait partie intégrante du projet IDEM (Integrated Development Environment for Mainframe) de la DSI. Il vise à moderniser les applications et les pratiques mainframe en capitalisant sur le développement agile et l'approche DevOps.

Le choix de la DSI s'est porté sur Topaz de Compuware, un pilier de l'approche DevOps mainframe. Cette solution conçue spécifiquement pour les équipes Agile/DevOps aide les développeurs à visualiser les structures des programmes et s'intègre parfaitement avec les outils DevOps mainframe et non-mainframe leaders du marché tels que Sonar Source et Jenkins.

« En mois de deux ans, plus de 90 % de nos collaborateurs IT sont engagés aujourd'hui dans le DevOps mainframe » se félicite Gatien Dupré, responsable du projet IDEM — zDevOps — au sein de la DSI de la Banque de détail en France du groupe Société Générale.

Des outils modernes pour les développeurs

Il faut reconnaître que de nombreuses équipes mainframe utilisent encore des outils et des processus vieillissants, la plupart du temps manuels, sans aucune automatisation. « Nous proposons un environnement de développement complet basé sur Eclipse. Comme pour les développements Java, les développeurs Cobol bénéficient d'un IDE moderne avec les outils de tests unitaires, de gestion de projets, d'autocomplétion, de vérification dynamique de la syntaxe, de puissantes fonctions de debugging, un explorateur de données, etc. » Poursuit Benjamin Beurier.

Dans les approches agile et DevOps, l'automatisation est la colonne vertébrale de l'accélération des développements. Les développeurs Cobol peuvent mettre en place le build et l'intégration continue, automatiser les batteries de tests et notamment les tests unitaires et les tests fonctionnels. Beaucoup d'équipes mainframe / Cobol réalisent encore aujourd'hui les tests manuellement. L'automatisation est réellement une réponse concrète à la demande d'efficacité du mainframe par la direction générale et la DSI.

« Après les audits techniques et la mise en place d'outils adéquats, nous constatons des gains de productivité de 10 à 20 %, en moyenne. » Constate Benjamin Beurier.

Une des stratégies adoptées par les DSI est de rationaliser les langages et les technologies utilisées, notamment de sortir de certaines technologies exotiques, coûteuses et obsolètes sur mainframe pour rationnaliser sur du Cobol, toujours sur mainframe. Même si Cobol est un langage qui a beaucoup évolué en 60 ans, les équipes utilisent principalement la forme « classique » du langage, faisant abstraction des dernières évolutions.

« L'un des défis est de gérer au mieux le renouvellement générationnel. Les développeurs partent progressivement en retraite, emportant trop souvent avec eux leur savoir-faire. Notre mission est d'aider les nouvelles générations à mieux apprendre, à mieux comprendre et donc à mieux coder en Cobol notamment, en leur donnant des outils de développement modernes qu'ils ont l'habitude d'utiliser, comme elles le feraient en Java » Précise Benjamin Beurier.

Cette connaissance est primordiale dans la maintenance des codes existants ainsi que pour les futurs développements. •

Comment Openframe veut sortir le mainframe du monolithique ?

Le mainframe est une plateforme matérielle et logicielle, reconnu pour sa stabilité et sa maturité mais qui reste une machine monolithique. Malgré les derniers signes d'ouverture technologique vers Linux, le DevOps et le cloud, et malgré la criticité des applications s'y exécutant, le mainframe reste prisonnier de son modèle économique.Comment donc passer d'un monde monolithique à un monde ouvert et distribué en proposant un replatforming complet ?

Pour rappel, une modernisation mainframe par replatforming consiste à conserver la richesse fonctionnelle et la fiabilité applicative, en déplaçant les programmes en minimisant la réécriture et les données sur une plateforme standard. L'enjeu global étant de réduire significativement le TCO et d'en faire un socle pour l'innovation.

Dans ce sens, TmaxSoft a été le premier à proposer une approche logicielle pour « porter » les services mainframes et les déployer sur une infrastructure ouverte et distribuée. Ce qui ne va pas de soi : un replatforming de cette envergure nécessite des alternatives complètes puisqu'un service mainframe équivaut à un service dans le monde distribué.

Une architecture complète

OpenFrame propose un environnement complet. Il s'agit d'une suite logicielle, comportant au cœur de la solution, le moniteur transactionnel Tmax qui prend en charge aussi bien les traitement Batch que le TP sur lequel se sont greffés les équivalents des moniteurs CICS, IMS-DC et JES.

La solution ne s'arrête pas là puisqu'elle est capable de prendre en charge l'ensemble des fonctionnalités mainframe utilisées par les entreprises : base de données DB2 et IMS-DB grâce à leur base de donnée Tibero, l'exécution des applications, Cobol, PLI, ASM avec leur propre compilateur et l'exécution des JCL avec un interpréteur identique à celui du Z/OS. Pour compléter la couverture fonctionnelle, la solution est livrée avec un équivalent RACF pour faciliter la migration des droits ainsi qu'un émulateur d'affichage 3270 afin d'optimiser le TCO. Enfin, l'un des avantages de cette architecture moderne est que chaque bloc fonctionnel est un module découplé du reste de l'architecture ouvrant sur une infinité de possibilités d'intégration.

On profite aussi des avantages du monde moderne : machine virtuelle / conteneur, montée en charge infinie (ou presque) load balancing & haute disponibilité. Les deux derniers sont importants. L'un des intérêts des infrastructures cloud et virtualisées est cette capacité de résilience et de montée en charge. Ceci tout en étant capable de répartir la charge entre plusieurs datacenters / sites physiques. Ainsi, il est possible de mettre en place des plans de reprise d'activé et surtout, contrairement aux mainframes qui coûtent très cher, on peut déplacer les workloads mainframe/cobol (=openframe) au plus près des utilisateurs…

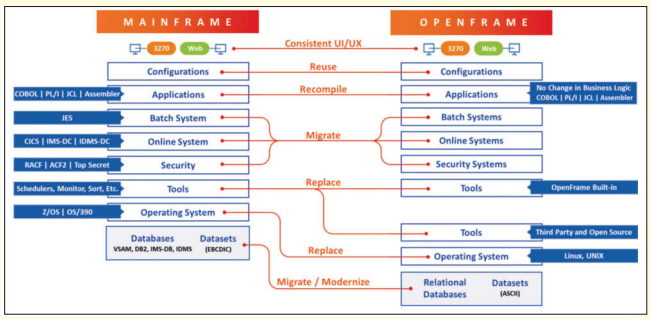


Schéma openframe your mainframe avec reuse, recompile, migrate

Bienvenue dans le merveilleux monde du rehosting

Le replatforming répond à plusieurs problématiques et réflexions des DSI / directions générales : la transformation numérique des entreprises et des usages, une meilleure efficacité économique de l'IT, mieux gérer les risques, et, de facto, moderniser l'IT. Voilà pour le contexte général. Mais, attention, on ne transforme pas le mainframe et les apps historiques comme cela, simplement pour faire comme tout le monde et être à la dernière mode technologique.

Le replatforming / rehosting n'est pas réellement réversible, sauf à dépenser de gros budgets et énormément de temps pour remonter l'infrastructure historique.

Dans un projet de rehosting, qui peut s'étaler d'un an à parfois plusieurs années pour les plus grands comptes, il y a fondamentalement 4 étapes :

- Le business case : on positionne cette modernisation dans le cadre de la transformation de système d'information afin d'en évaluer les bénéfices.
- L'audit : on étudie l'architecture, on cartographie l'ensemble des services mainframes et les applications, on analyse le code, on commence à créer des scénarios de migration et de transformation.
- Livraison : on migre les applications et les données. On intègre les services historiques dans la nouvelle architecture comme les scheduling et l'éditique qui seront des projets à mener en parallèle. Puis on teste.
- Production & maintenance : tout projet doit inclure ces trois phases. Le replatforming / rehosting est aussi l'occasion de définir sa nouvelle plateforme d'exécution on premise ou dans le Cloud, ainsi que l'utilisation de nouveau service autour du devops et de l'analytics. Cette vision est importante car ces applications continueront à être utilisées sachant que certaines ont 20, 30 ans, voire plus.

Quelles motivations pour réaliser ce genre de projet ? C'est souvent au cas par cas mais il y a toujours le coût de l'informatique historique. Le MIPS mainframe coûte cher. Il faut maintenir le matériel, voire déployer une machine plus récente (ce qui prend du temps et mobilise un budget important). Il faut faire évoluer les applications et surtout être capable d'en reprendre le contrôle. Car trop souvent, plus personne ne connaît le code Cobol et la cartographie des applications…

L'autre question qui peut aussi se poser : votre mainframe est-il exploité comme il le devrait ? Est-ce que les services qu'il délivre nécessitent réellement une telle infrastructure ?

Le risque, bien réel, est que l'entreprise utilise deux réalités IT : une IT historique et une IT « moderne ». Et les deux n'utilisent pas les mêmes technologies, ni les mêmes méthodes.

Mais un projet de transformation du mainframe ne s'improvise pas. Il faut au préalable déterminer les services et les applications à moderniser / migrer, la faisabilité du chantier. La phase d'audit est cruciale. TmaxSoft a bati avec ses partenaires français toute une démarche dans ce sens.

Vous comprendrez très vite qu'un replatforming vers une solution moderne de type Openframe peut devenir complexe si vous n'avez pas une équipe de sachants en interne et les bons interlocuteurs en face. De plus, si vos applications ont beaucoup d'interactions entre elles, cela signifie qu'il faudra migrer soit l'ensemble des applications, qui n'est pas sans risque, soit être capable de définir des périmètres homogènes à migrer tout en laissant d'autres applications sur le legacy sans casser les liens. •

Wassim CHEGHAM
Senior Cloud Advocate at Microsoft
Angular Core Team GDE nominé par Google en Web, Assistant et GCP Membre invité de la fondation Node.js Auteur de xlayers.dev, ngx.tools, autocap.cc.
Twitter: **@manekinekko**
Dev.to: **dev.to/wassimchegham**

Quoi de neuf Angular ?

Il est inutile de présenter Angular en 2019. En effet, Angular est déjà utilisé par plusieurs millions de développeurs et d'entreprises à travers le globe, et cette adoption ne fait que croître. Dans ce dossier, nous allons découvrir ensemble les dernières nouveautés apportées par la version 8 et nous aborderons aussi ce que nous réserve le futur d'Angular.

niveau 100

Une adoption croissante

Pour avoir un ordre d'idée du taux d'adoption d'Angular, au sein de l'équipe Angular, nous utilisons le nombre de visiteurs uniques sur la documentation officielle[1]. La raison à cela est que nous avons constaté qu'il y a une grande majorité d'entreprises qui utilisent Angular pour développer des application destinées à être utilisées en Intranet ou dans des environnements non exposés directement sur Internet (par exemple, en tant que dashboard pour des voitures connectées !). Nous n'avons donc aucun moyen fiable de mesurer cette adoption. De ce fait, le nombre de personnes qui accèdent à la documentation de manière régulière est un bon indicateur.

Il y a quelques mois, nous avons annoncé avoir dépassé les **1.5 million de visiteurs uniques** et ce chiffre ne fait qu'augmenter bien évidemment. 1

Un framework complet et intégré

Pourquoi les développeurs préfèrent-ils Angular ? L'une des raisons qui revient le plus souvent est qu'Angular est un framework complet. Que ce soit pour la gestion des routes, de vos formulaires et leurs validations, de l'internationalisation (*i18n*), le nécessaire pour le SEO, la gestion des tests unitaires, les outils pour les développeurs, les animations, ou encore les composants graphiques. Angular intègre de base les briques et APIs nécessaires pour vous aider à mener à bien le développement de vos web apps.

Depuis la version 6, nous avons mis la plupart de nos efforts dans l'amélioration de l'expérience développeur (DX). Proposer la meilleure DX pour nos développeurs est quelque chose de très important pour nous. Cela fait également partie de nos trois valeurs essentielles :

- Des applications qui soient agréables à créer par les développeurs.
- Des applications qui soient agréables à utiliser.
- Une communauté où tout le monde se sente la/le bienvenue.

Rappel : cycle des releases

Avec Angular, nous avons adopté un cycle de releases régulier et basé sur le standard *Semantic Versioning*. Cela vous permet de planifier et anticiper vos mise à jour. Le cycle classique ressemble à :

- Une version majeure tous les 6 mois.
- 1 à 3 versions mineures pour toute version majeure.
- Des versions patch avec des correctifs de bugs environ toutes les semaines.

Vous aurez ainsi la possibilité de bénéficier des dernières nouveautés du framework. Si vous souhaitez expérimenter les mises à jour, ou lire la documentation d'une version suivante d'Angular, vous pouvez accéder à next.angular.io.

A noter également que nous mettons en place une politique de support et LTS (*Long-Term Support*). Nous offrons :

- Un support actif durant les 6 mois pour une version majeure.

(1) http://angular.io

1

• Un support (correctifs de bugs critiques et patch de sécurité) de 12 mois de LTS.

Pour en savoir plus sur le processus des releases : https://angular.io/guide/releases

Procédure de mise à jour automatisée

Pour plus d'information détaillées concernant la procédure de mise à jour vers la dernière version d'Angular, veuillez lire https://update.angular.io. Vous pouvez également utiliser le Angular CLI pour exécuter la mise à jour automatisée via la commande :

```
ng update @angular/cli @angular/core
```

LES NOUVEAUTÉS DE LA VERSION 8

Chargement différentiel (Differential Loading)

Le chargement différentiel est une technique où le navigateur choisit le standard de JavaScript (ES2015 ou ES5) à charger, en fonction de son propre support des *ES Modules*. Dit autrement, les navigateurs modernes pourront charger les fichiers "*transpilés*" en ES2015 ; les anciens navigateurs quant à eux, auront toujours accès à la version "*transpilée*" en ES5 de l'application (en y ajoutant les *polyfills* nécessaires).

Pour cela, lors du build avec la CLI, nous inspectons le fichier *tsconfig.json* pour vérifier si le niveau du JavaScript que vous ciblez est bien ES2015. Si tel est le cas, alors nous générons 2 versions du build de votre application :

• Un build ES2015 (JavaScript moderne).

• Un build ES5 (JavaScript Legacy).

Par exemple, pour cette configuration *tsconfig.json* :

```
{
 "compilerOptions": {
  "module": "esnext",
  "moduleResolution": "node",
  "target": "es2015"
 }
}
```

Vous aurez ainsi 2 ces builds :

```
<!-- For modern browsers -->
<script type="module" src="polyfills-es2015.js"></script>
<script type="module" src="runtime-es2015.js"></script>
<script type="module" src="style-es2015.js"></script>
<script type="module" src="vendor-es2015.js"></script>
<script type="module" src="main-es2015.js"></script>

 <!-- For Legacy browsers -->
<script nomodule src="polyfills-es5.js"></script>
<script nomodule src="runtime-es5.js"></script>
<script nomodule src="style-es5.js"></script>
<script nomodule src="vendor-es5.js"></script>
```

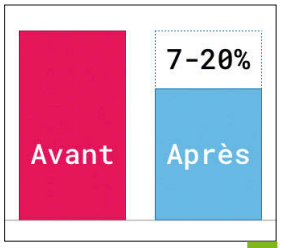
> **A noter** que si vous utilisez la commande ng update décrite précédemment, nous mettons à jour automatiquement votre configuration tsconfig.json et générons les 2 builds !

Ce chargement différentiel est maintenant activé par défaut depuis la version 8 d'Angular et vous n'avez rien à modifier.

En savoir plus sur le Chargement Différentiel(2).

Nous avons appliqué cette technique sur angular.io et nous avons réussi à économiser plus de 40kb de la taille des fichiers à charger sur les navigateurs modernes. Aussi, d'après les retours que nous avons eu de la communauté, le gain constaté est de l'ordre de 7 à 20%. 2

Les Imports Dynamiques dans les routes



2

Nous vous avons toujours recommandé de recourir au chargement à la demande (*Lazy Loading*) de certaines parties de vos applications, via le *Router*, afin d'améliorer le temps de chargement de vos apps. Ce chargement se fait via l'utilisation de l'attribut *loadChildren* du *Router* comme ceci :

```
{
 path: 'home',
 loadChildren: 'home/home.module#HomeModule'
}
```

Cette syntaxe était propre à Angular et n'était donc pas standard. Cependant, avec la démocratisation du support des Imports Dynamiques ES2015, et à partir de la version 8 d'Angular, nous allons migrer vers ce nouveau standard. La nouvelle syntaxe ressemble à ceci :

```
{
 path: 'home',
 loadChildren: () => import('home/home.module')
                .then(m => m.HomeModule)
}
```

Les éditeurs tels que VSCode ou WebStom pourront désormais valider ces imports pour vous.

> *Si vous utilisez la commande ng update, nous mettrons automatiquement à jour votre code avec cette nouvelle syntaxe.*

Builder API

A l'instar des Schematics qui nous permettent de personnaliser le code généré par le CLI via les commandes *ng new, ng generate, ng add* et *ng update*, ou d'intégrer de nouvelles commandes (voire bien plus(3)), l'API *Builder* nous permet de personnaliser le comportement des commandes *ng build, ng test* et *ng run* afin d'exécuter des tâches liées au build ou au déploiement. Certains *Cloud Providers* ont déjà commencé à écrire des commandes pour faciliter le déploiement de vos applications Angular dans leurs environnements ; par exemple : déployer sur Firebase avec *@angular/fire*(4) ou sur Microsoft Azure avec *@azure/ng-deploy*(5) (vous pouvez utiliser ce lien(6) pour créer un compte gratuit sur Azure)

En savoir plus sur les Builders(7).

---

(2) https://angular.io/guide/deployment#differential-loading

(3) https://angular.io/guide/schematics

(4) https://github.com/angular/angularfire2#readme

(5) https://aka.ms/ng-deploy-azure

(6) https://aka.ms/programmez-free-azure

(7) https://blog.angular.io/introducing-cli-builders-d012d4489f1b

Support des Web Workers

Si vous avez souvent recours à beaucoup de calculs intensifs ou gourmands en temps CPU, l'utilisation des *Web Workers*(8) est une très bonne solution standard vous permettant de déléguer ces calculs vers un thread séparé du thread principal dans lequel s'exécute le code de votre application ainsi que la gestion du rendu et du DOM, etc. Vous pouvez maintenant générer simplement un Web Worker directement via le CLI :

```
ng generate web-worker app
```

Cette commande va :

- Configurer votre projet pour pouvoir utiliser les *Web Workers*.
- Générer et ajouter un fichier *src/app/app.worker.ts* avec le code suivant :

```
addEventListener('message', ({ data }) => {
 const response = `worker response to ${data}`;
postMessage(response);
});
```

Mettre à jour le composant app.component.ts pour utiliser le Worker :

```
if (typeof Worker !== 'undefined') {

 // Create a new
 const worker = new Worker('./app.worker', { type: 'module' });
 worker.onmessage = ({ data }) => {   console.log(`page got message: ${data}`);
 };
 worker.postMessage('hello');

} else {
 // Web Workers are not supported in this environment.
 // You should add a fallback so that your program still executes correctly.
}
```

En savoir plus sur le support des *Web Workers* dans le Angular CLI(9).

> *Le code qui s'exécute dans le Web Worker est le vôtre. Le runtime Angular quant à lui s'exécute dans le thread principal !*

Nouvelle API pour les Workspaces de la CLI

Précédemment, lorsque vous écriviez des Schematics pour la CLI, la manipulation du fichier *angular.json* n'était pas prise en charge. Depuis la version 8, une nouvelle API a été introduite pour cet effet notamment grâce à ces deux fonctions :

```
export function readWorkspace(
 path: string,
 host: WorkspaceHost,
 format?: WorkspaceFormat,
): Promise<{ workspace: WorkspaceDefinition; }>;
```

```
export function writeWorkspace(
 workspace: WorkspaceDefinition,
 host: WorkspaceHost,
 path?: string,
 format?: WorkspaceFormat,
): Promise<void>;
```

Voici un exemple d'utilisation :

```
import { NodeJsSyncHost } from '@angular-devkit/core/node';
import { workspaces } from '@angular-devkit/core';
async function demonstrate() {
  const host = workspaces.createWorkspaceHost(new NodeJsSyncHost());
  const workspace = await workspaces.readWorkspace('path/to/workspace/directory/', host);
  const project = workspace.projects.get('my-app');
  if (!project) {
    throw new Error('my-app does not exist');   }
  const buildTarget = project.targets.get('build');
  if (!buildTarget) {
    throw new Error('build target does not exist');
  }
  buildTarget.options.optimization = true;
  await workspaces.writeWorkspace(workspace, host);}
demonstrate();
```

En savoir plus sur cette nouvelle Workspace API(10).

Améliorations de la migration depuis AngularJS

Si vous utilisez le service *$location*(11) dans une application AngularJS, Angular fournit désormais un module *LocationUpgradeModule* qui vous propose un service unifié permettant de transférer la gestion des URLs depuis le service *$location* d'AngularJS vers le service Location d'Angular. Cela devrait améliorer la vie des applications utilisant *ngUpgrade* qui ont besoin d'un routage à la fois dans les parties AngularJS et Angular de leur application.
En savoir plus sur le service Unified Angular Location(12).

Télémétrie

La version 8 de la CLI intègre une nouvelle fonctionnalité permettant l'envoi de données anonymes de télémétrie. Ces données sont à l'unique destination de l'équipe Angular. Cela nous permettra de diagnostiquer rapidement d'éventuels problèmes et d'améliorer les performances des outils que nous proposons. A noter qu'il vous est également possible de connecter votre propre service d'analytics et de collecter les mêmes données pour vos propres usages.
Cette fonctionnalité est désactivée par défaut, et nous comptons sur votre entière collaboration. Vous pouvez à tout moment l'activer (ou la désactiver), et cela se fait par projet. Les données collectées sont, entre autres, le temps d'un build, la taille des fichiers d'un bundle,

(8) https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

(9) https://v8.angular.io/guide/web-worker

(10) https://github.com/angular/angular-cli/blob/master/packages/angular_devkit/core/README.md#workspaces

(11) https://docs.angularjs.org/api/ng/service/$location

(12) https://v8.angular.io/guide/upgrade#using-the-unified-angular-location-service

la quantité de la RAM et le temps CPU utilisés… etc. Voir la liste complètes des métriques(13).

Angular Element et les Web Components

Les Angular Elements sont simplement des composants Angular packagés en tant que *Custom Elements* (les CE sont une des 4 spécifications des *Web Components*).

Il y a toujours eu un certain degré d'interopérabilité(14) entre Angular et les Web Components. Mais Angular proposait sa propre implémentation des Custom Elements, comparée à celle proposée par le WHATWG(15).

Les cas d'usage des Angular Elements sont nombreux :

- Intégrer des composants Angular avec des technologies serveur (e.g. ASP.net).
- Intégrer des composants Angular dans des sites/pages static, type CMS.
- Intégrer des composants Angular au sein d'autres technologies front-end (e.g. Vue.js).

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, Injector } from '@angular/core';
import { createCustomElement } from '@angular/elements';
import { HelloComponent } from './hello.component';

@NgModule({
 declarations: [AppComponent],
 imports: [BrowserModule],
 providers: [],
 entryComponents: [HelloComponent]
})
export class AppModule {
 constructor(injector: Injector) {
  const element = createCustomElement(
                HelloComponent, { injector });
  customElements.define('x-hello', element);
 }
}
```

Quant à l'élément Angular, il est instancié comme d'habitude dans le template :

```
<x-hello></x-hello>
```

En savoir plus sur les Angular Elements(16).

Bazel : un nouvel orchestrateur de build

Actuellement, dans la Angular CLI, nous utilisons WebPack pour lancer les builds de vos applications – et ng-packagr pour vos librairies. Cela ne devrait pas changer de sitôt. Cependant, nous avons travaillé sur l'intégration d'un nouveau système d'orchestration des builds, afin d'en améliorer les performances. Cet outil s'appelle Bazel.

Pour la petite histoire, Google utilise un seul et unique *monorepo* pour y stocker le code source de plusieurs de ses projets (quelques 86TB de données !). Si vous vous posez la question : Non, Google n'utilise pas Git pour gérer son *monorepo* mais un outil fait maison nommé Piper. En ce qui concerne la gestion des builds et tests, Google utilise un outil nommé Blaze qui a pour responsabilité d'orchestrer de manière distribuée les builds et les tests d'un codebase partagé entre plus de 25000 ingénieurs répartis dans une dizaine de bureaux à travers le monde. Blaze traite donc plus de 500000 requêtes par secondes dans une journée normale de travail !

Google a rendu open source une partie de Blaze sous le nom de Bazel(17), en 2015. Voici quelques projets qui utilisent Bazel en tant qu'orchestrateur de build : 3

Les avantages de Bazel sont multiples. Par exemple :

- La prise en charge de plusieurs technologies backend et frontend. Bazel est capable de lancer les tâches de compilation de ces technologies en parallèle et en même temps.
- Le support de la compilation incrémentale (ne recompile que la partie du code qui a été modifiée).
- Une herméticité des packages et une gestion des dépendances assez strictes.

En savoir plus sur Bazel via une de mes présentations(18) sur le sujet. Dans le contexte d'Angular, il y a environ un an, nous avions fait le choix de migrer le framework Angular vers Bazel, faisant passer le temps de build et tests sur notre CI de plus d'une heure à environs 8 minutes !

Nous souhaitons maintenant faire bénéficier la communauté Angular de ce boost de performances et avions décidé d'intégrer Bazel dans la CLI.

Dans la version 8 de la CLI, Bazel est encore en phase d'expérimentation. Pour le tester, vous pouvez soit créer une nouvelle application en utilisant la configuration Bazel :

```
npm install -g @angular/bazel
ng new my-app --collection=@angular/bazel
```

(13) https://github.com/angular/angular-cli/blob/master/docs/design/analytics.md

(14) http://slides.com/wassimchegham/angular-web-component-polymer-interoperability#/

(15) https://html.spec.whatwg.org/multipage/custom-elements.html

(16) https://angular.io/guide/elements

(17) https://bazel.build/

(18) https://slides.com/wassimchegham/bazel-for-angular-developers-angular-in-depth-2019-ukraine#/

3

Ou bien, ajouter la configuration Bazel à votre projet existant :

```
ng add @angular/bazel
```

Si vous utilisez la CLI, toute la configuration de Bazel devrait être entièrement prise en charge par la CLI. Vous ne verrez même pas les fichiers de configuration. Une fois que l'intégration de Bazel dans la CLI sera finalisée (prévue normalement pour la v9), la CLI utilisera Bazel par défaut et cette transition sera transparente pour vous.

> Gardez à l'esprit que cette intégration est encore expérimentale. Si vous rencontrez des bugs, merci de nous en informer.

Améliorations de la migration AngularJS

Depuis la version 8, Angular propose un module *LocationUpgradeModule* pour les applications AngularJS utilisant le service *$location(19)*. Grâce à ce nouveau module, les responsabilités du service *$location* sont maintenant gérés par le service *Location* d'Angular. En savoir(20) plus sur cette unification des services de gestion d'URL.

Ivy : un nouveau compilateur et moteur de rendu

Depuis quelques mois maintenant, l'équipe en charge du coeur du framework s'est penchée sur la réécriture intégrale du moteur de rendu historique d'Angular (appelé *ViewEngine*). Le nouveau moteur de rendu et de compilation (nom de code *Ivy*) est presque achevé et le but de cette réécriture est de :

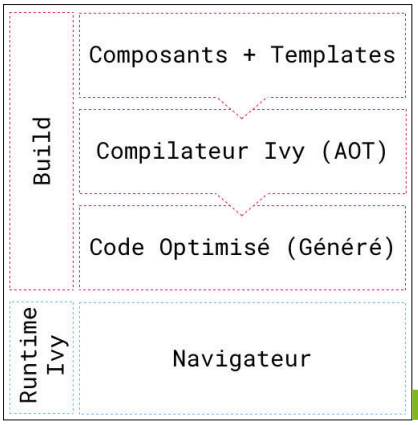
- Permettre un code généré qui soit beaucoup plus simple à lire et à déboguer.
- Permettre un temps de re-build plus court.
- Réduire de manière significative la taille des bundles générés.
- Améliorer le *type-checking* dans les templates HTML (à travers la réécriture du *Langage Service* pour TypeScript).

La réduction des tailles (et de ce fait, l'amélioration du temps de chargement) des bundles générés par Ivy est rendue possible grâce à l'application de la technique du *Tree Shaking(21)*. Toutes les APIs publiques peuvent maintenant être nettoyées du bundle final si ces dernières ne sont pas utilisées par l'application métier :

(19) https://docs.angularjs.org/api/ng/service/$location

(20) https://v8.angular.io/guide/upgrade#using-the-unified-angular-location-service

(21) https://en.wikipedia.org/wiki/Tree_shaking



4

- Les *Hooks* du cycle de vie des composants (NgOnInit, NgOnChanges...) ;
- Les Pipes ;
- Les directives structurelles ;
- Les Queries ;
- Le système d'injection de dépendances ;
- Etc.

A noter également qu'avec Ivy, le mode de compilation en JIT (en *ng build* et en *ng serve*) va disparaître en faveur du mode AOT(22).

4

Voici un exemple d'un code compilé et généré par Ivy, pour un simple composant *AppComponent* :

```
import {
 Component}
 from '@angular/core';
@Component({
 selector: 'app-root',
 template: `
  <h1>{{ title }}</h1>
 `
})
export class AppComponent {
 title = 'ivy';
}
```

Voici le code généré par Ivy :

```
import * as tslib_1 from "tslib";
import { Component } from '@angular/core';
import * as i0 from "@angular/core";
var AppComponent = /** @class */ (function () {
 function AppComponent() {
  this.title = 'ivy';
 }
 AppComponent.ngComponentDef = i0.edefineComponent({
  type: AppComponent,
  selectors: [["app-root"]],
  factory: function AppComponent_Factory(t) {
   return new (t || AppComponent)();
  },
  consts: 2,   vars: 1,
  template: function AppComponent_Template(rf, ctx) {
   if (rf & 1) {
    i0.eelementStart(0, "h1");
    i0.etext(1);
    i0.eelementEnd();    } if (rf & 2) {
   i0.etextBinding(1, i0.einterpolation1("", ctx.title, ""));
   }
  }, encapsulation: 2
 });
 AppComponent = tslib_1.__decorate([
  Component({
   selector: 'app-root',
   template: "\n\t<h1>{{ title }}</h1>   \n  "
```

(22) https://slides.com/wassimchegham/demystifying-ahead-of-time-compilation-in-angular-2-aot-jit#/

```
  })
 ], AppComponent);
 return AppComponent;
}());
export { AppComponent };
```

Clair et concis, n'est-ce pas ?
Dans la version 8, Ivy est encore en phase expérimentale. Cependant la couverture des tests unitaires a dépassé les 97%. Ce qui est une très bonne nouvelle ! Pour tester Ivy dans vos projets, vous devez le flag *--enable-ivy* :

```
ng new shiny-ivy-app --enable-ivy
```

Si vous souhaitez tester Ivy avec vos applications existantes(23), il vous faudra modifier ces deux fichiers de configuration :

- Assurez-vous que vous avez bien mis à jour vos dépendances vers la version 8 (Angular et le CLI).
- Activer Ivy dans le fichier *tsconfig.app.json* :

```
{
 "compilerOptions": { ... },
 "angularCompilerOptions": {
  "enableIvy": true
 }
}
```

Assurez-vous que la compilation AOT est bien activée dans le fichier *angular.json* :

```
{
 "projects": {
  "votre-projet": {
   "architect": {
    "build": {
     "options": {
      ...        "aot": true,
     }
    }
   }
```

(23) https://angular.io/guide/ivy

```
  }
 }
}
```

LE FUTUR D'ANGULAR

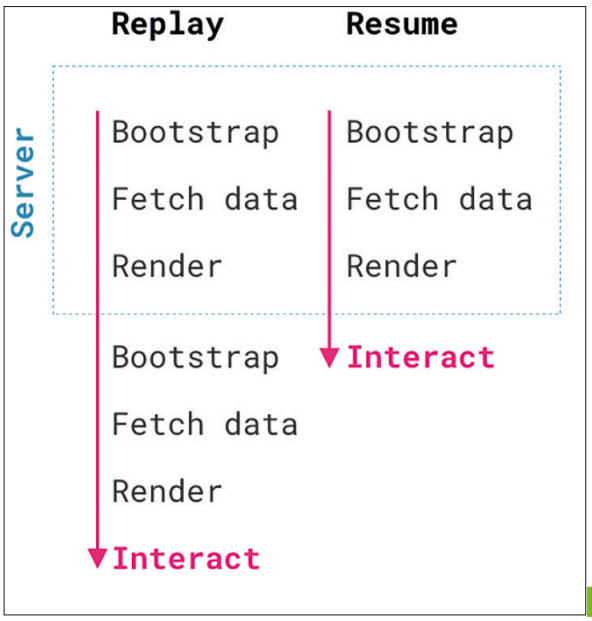
Ce qui suit n'est qu'une projection dans le futur d'Angular et n'est que mon avis personnel. 5

Avec l'arrivée de la nouvelle architecture autour d'Ivy et la façon dont le coeur d'Angular a été revu, cela va permettre de tester de nouvelles techniques d'optimisation afin de réduire encore plus le temps de chargement des applications Angular, réduire le *Time To Interactive*, etc. Nous pouvons imaginer par exemple rendre les *NgModules* optionnels et ainsi charger les composants directement et individuellement (restera la dépendance vers Zone.js à résoudre !).
Une autre piste à explorer est de pousser encore plus loin le *Server-Side Rendering*, en permettant d'avoir des applications qui seraient rendues côté serveur puis envoyées, avec leur State créé côté serveur, au client (navigateur). Ainsi, du côté navigateur, Angular pourra simplement "reprendre" la main en se basant sur le DOM et le State générés par le serveur. Cela éviterait donc à Angular de "refaire" ce qui a déjà était fait par le serveur et de réhydrater le DOM (ce que fait l'implémentation actuelle du SSR, aka *Universal(24)*.
Grâce à cette nouvelle technique, il sera ainsi possible de booster radicalement le temps de chargement des applis Angular.
Le futur d'Angular sera donc marqué par un runtime de plus en plus léger, des outils de dev de plus en plus rapides, et des applications de plus en plus performantes.

Stay tuned.

PS : Un grand merci à Vincent Ogloblinsky(25) pour la relecture de ce dossier.

(24) https://medium.com/google-developer-experts/angular-universal-for-the-rest-of-us-922ca8bac84
(25) https://twitter.com/vogloblinsky



5

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

Windows Subsystem for Linux 2 (WSL2)

Faire tourner des binaires Linux 64 bits sur Windows, c'est possible.

Avant, il y avait Nutcracker dans les années 1990. Il fallait recompiler. Avec Windows NT 3.51, on avait un subsystem POSIX. Ensuite, il y a eu Interix. Et enfin WSL et maintenant WSL2. Du côté des solutions open-source, il y a toujours eu Cygwin et MinGW (Minimal GNU Windows). Le meilleur est certainement MinGW et sa version 64 bits. Utiliser les application Unix/Linux sous Windows est un vieux rêve.

niveau 100

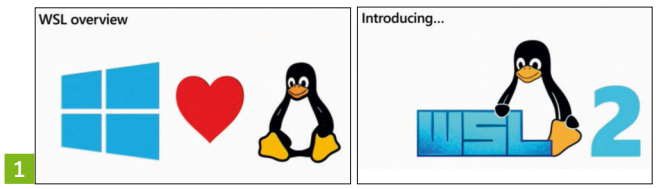
WSL2 a été annoncé officiellement le 6 mai dernier et disponible dans Windows 10 Juin 2019. L'équipe en charge de cette fonction mettait en avant deux améliorations importantes :

- Les performances globales du système de fichiers ;
- La compatibilité avec tous les appels systèmes.

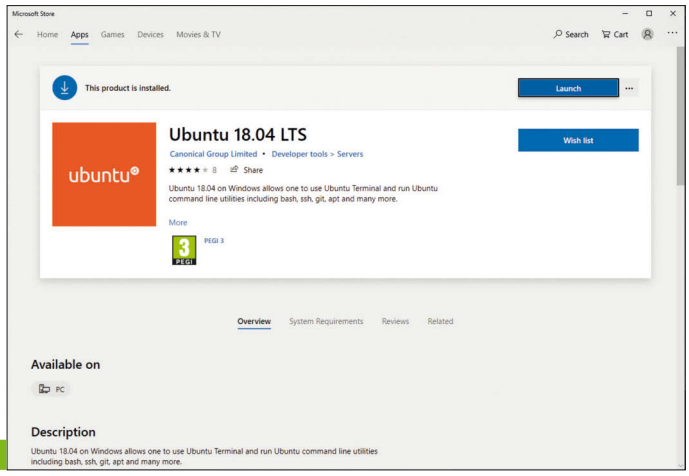
A cela, se rajoute, comme nous le verrons dans ce court article : une nouvelle architecture, un meilleur support des binaires Linux, une expérience utilisateur identique à la version précédente.

On dispose aussi de nouvelles commandes :

- wsl –set-version <Distro> <Version> : pour convertir une distribution utilisant WSL1 ou WSL2. Distro est le nom de la distribution, Version est la version de WSL.
- wsl –set-default-version <Version> : change la version par défaut pour les nouvelles distributions que l'on déploie.
- wsl –shutdown : on termine sans délai les processus et les distributions ainsi que la VM WSL2.
- wsl –list –quiet : liste le nom des distributions. L'option –verbose à la place de –quiet affiche l'ensemble des informations des distributions.



1



2

A noter que WSL2 utilise un disque VHD pour stocker les fichiers. WSL1 n'est pas dépréciée. Pour le moment l'accès matériel depuis WSL2 est limité. Microsoft prévoit, dans le futur, de supporter les ports séries, l'USB et le GPU. 1

Installation

L'installation de WSL se fait par l'activation de fonctionnalités Windows (Windows Features On/Off). Ensuite, on installe une distribution au travers le Windows Store. Exemple, Ubuntu 18.04 LTS. La distribution fait 220 MB. Le seul prérequis est un Windows 10 version 16237.0 ou plus. La suite est classifiée comme Developer Tools / Utilities. Quand on parle de distribution Linux, on parle en fait de l'environnement ligne de commandes.

Vous devez disposer d'un Windows build 18917 minimum, l'activation du programme Windows Insider et l'option Fast doit être activée. En plus, vous devez activer la fonction Windows : Virtual Machine Platform.

Depuis la console Powershell (en mode Administrateur) :

```
Enable-WindowsOptionalFeature -Online -FeatureName VirtualMachinePlatform
```

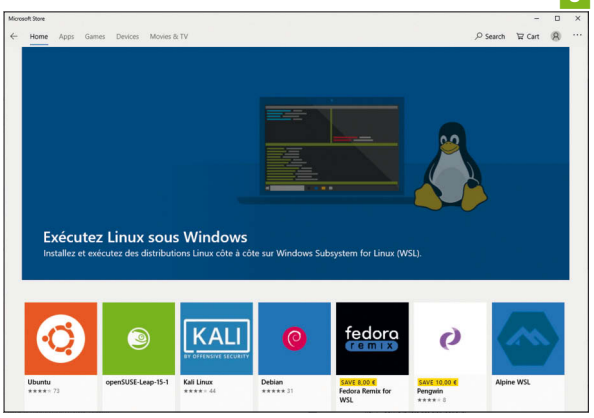
2

Il existe plusieurs distributions comme Ubuntu, openSUSE, Kali Linux, Debian, Fedora, Alpine. 3

Par exemple, Debian ne requiert que 80 MB et Windows 10 version 16215.0.

Ensuite, on se retrouve à avoir l'application bash… On est réelle-

3



ment dans le bash Ubuntu : ce n'est pas une émulation. Je suis sous Linux ! Je décide donc d'installer gcc, g++ et cmake via :

- sudo apt update
- sudo apt install gcc
- sudo apt install g++
- sudo apt install cmake

Depuis le bash, je tape « gcc -v » et voici le résultat. 4

Essayons GCC

Je décide de lancer VS Code et d'entrer un petit programme « Hello Ubuntu ! » :

- Ouverture du dossier \\wsl$\Ubuntu\home\christophep
- Création du fichier hello.cpp

```
#include <iostream>
#include <string>

int main()
{
  std::cout << "Hello Ubuntu !" << std::endl;
  return 0;
}
```

```
christophep@DESKTOP-7VJOH39:~$ g++ hello.cpp -std=c++17
christophep@DESKTOP-7VJOH39:~$ ls
a.out  hello.cpp
christophep@DESKTOP-7VJOH39:~$ ./a.out
Hello Ubuntu !
christophep@DESKTOP-7VJOH39:~$
```

WSL2 c'est quoi ?

C'est la possibilité d'avoir un kernel Linux directement inclus dans Windows. De plus, il est possible d'avoir plusieurs distributions : Kali, Debian, Ubuntu, etc. WSL2 est plus performant que WSL1, comme nous avons pu le constater. Par exemple :

- git clone : 2.5 x plus rapide
- npm install : 4.7 x plus rapide
- cmake : 3.1 x plus rapide

La plupart du temps, cela vient des améliorations du système de fichiers. L'autre considération est la compatibilité des appels système. Surtout pour les tips & tricks de debugging. WSL1 avait des limitations.

« wsl -l » donne la liste des distributions. 5

Architecture de WSL

WSL est un Windows driver et un Linux Kernel en *translation layer* au-dessus du Windows NT Kernel. 6

Au cœur de WSL v1

Les translations sont lentes par exemple sur les opérations de fichiers car les systèmes sont différents… Parfois, la translation n'est pas possible. Exemple : renommer un fichier ouvert sous Windows donne ERROR_ACCESS_DENIED alors que sous Linux, ça donne SUCCESS.

Dans WSL2, la donne change : 7

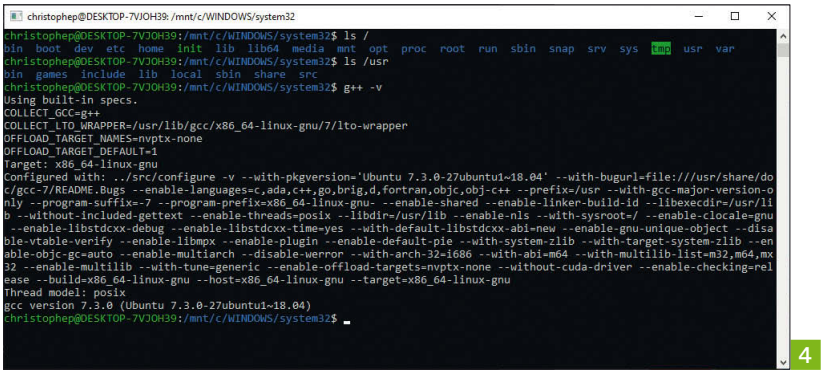
Il y a un Kernel Linux construit à partir des sources Linux qui tourne avec la virtualisation Hyper-V.

Ce n'est pas de la VM traditionnelle. C'est de la virtualisation légère ; Native x64 bits Linux support !

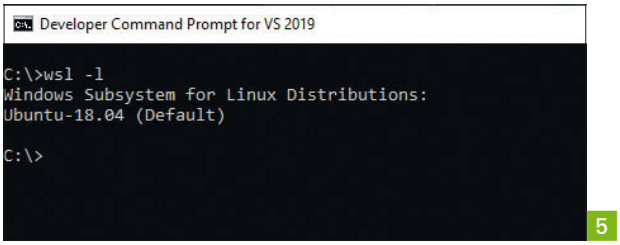
Développée initialement pour Windows Server, c'est la technologie Hyper-V des containers isolés qui est utilisée. Il peut y avoir de nombreux containers sur le même host (la même machine). Le temps de boot est très rapide et la création de nouveau container est une activité anodine. Les applications Windows comme Windows Defender Guard (WDAG) et Windows Sandbox sont des applications clients.

C'est de la virtualisation légère, comparée au monde des VM. 8

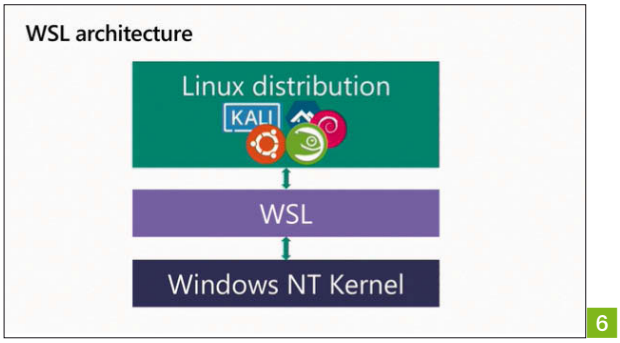
Le diagramme montre la comparaison entre les deux mondes.



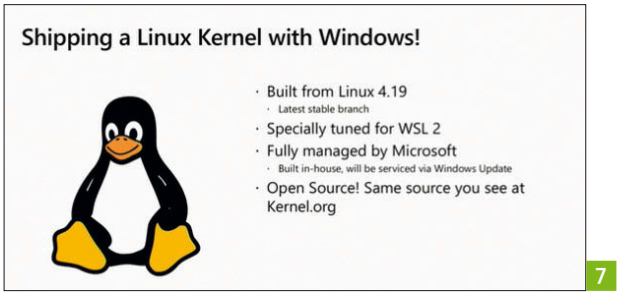
4



5



6

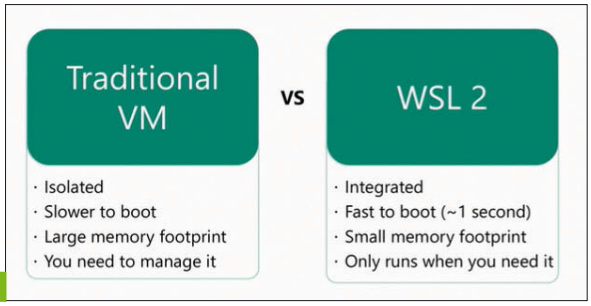


7

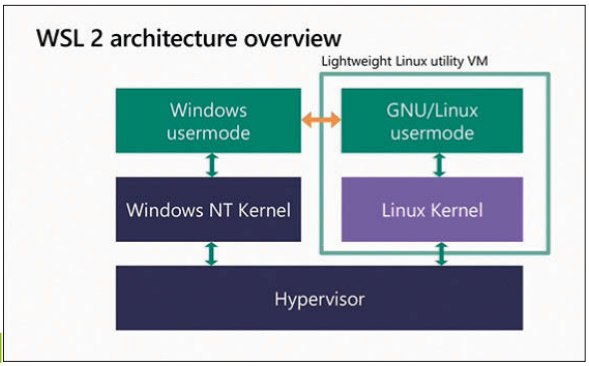
Architecture WSL2 9

Il y a donc un vrai Kernel Linux et ça, ça change tout ! Car, il ne s'agit plus d'émuler à travers une couche de translation mais bien d'être compatible. Le kernel Linux est issu des sources de kernel.org. 10

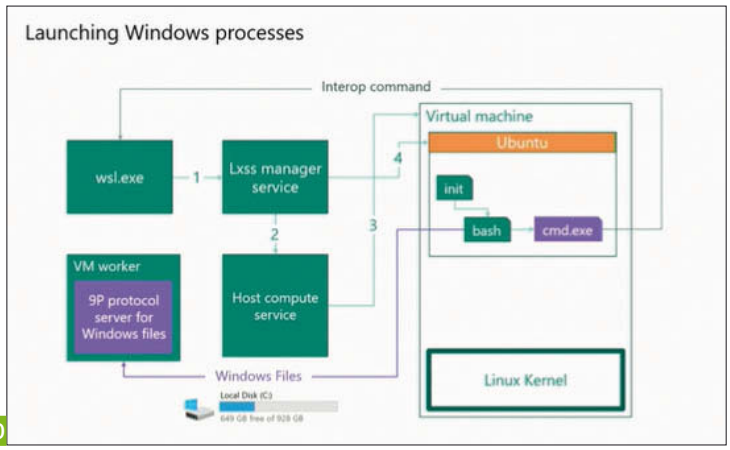
Il est possible de lancer des processus Windows depuis le bash Linux. 11



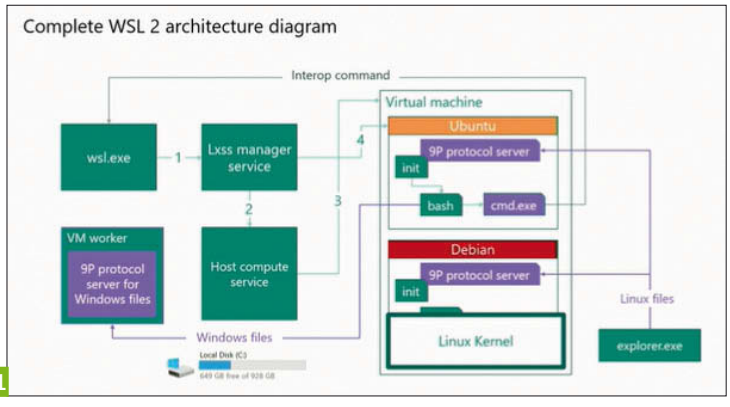
8



9



10



11

Accès aux fichiers

L'explorateur de fichiers Windows peut accéder au filesystem Linux. Il faut spécifier \\wsl2$\<distro> 12 13

Dans ce schéma on y voit la coexistence du Noyau Linux au sein d'un environnement Windows sous forme de VM légère avec la technologie Hyper-V. On y voit le protocole d'échanges entre Windows et Linux pour les fichiers.

Pour quels usages ?

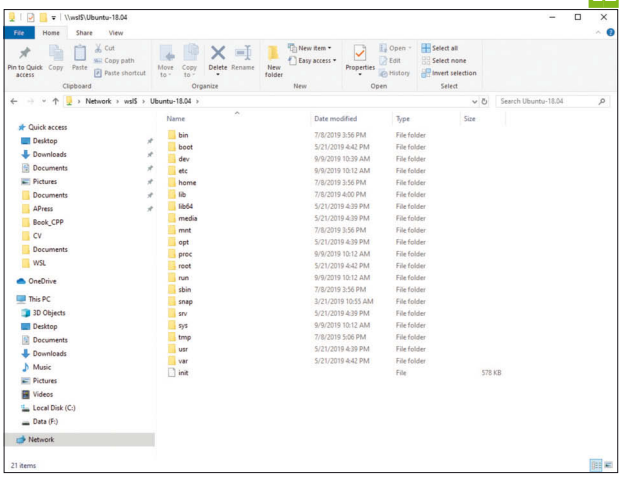
Soyons franc, il s'agit d'un mode console donc c'est orienté Serveur. On pourra y faire tourner NGINX, Apache mais quid des application X fenêtrées ? il existe dans le Windows Store une application X Server pour Windows 10. Est-ce fiable, est-ce compatible WSL2, je ne peux pas répondre à cette question.

Le truc c'est que chaque distribution fournit son environnement de commandes qui est relativement commun. Pour le développeur qui veut faire du bash avec toute la série des outils comme sed, awk, grep, tail, vi & co, c'est parfait. Pour celui qui veut builder avec le dernier GCC c'est parfait aussi. Il faut noter aussi que la consommation mémoire est relativement faible.

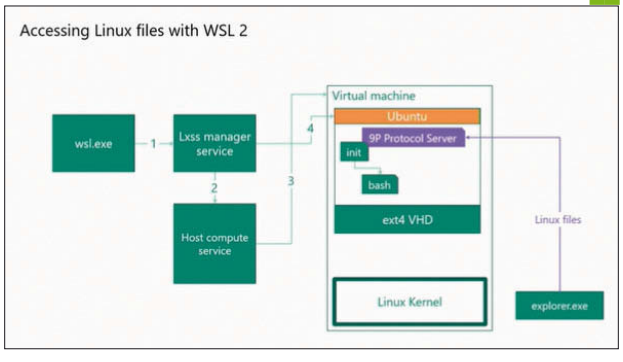
Conclusion

WSL2 est la solution pour faire du Linux sous Windows. C'est du vrai Linux avec un vrai Kernel Linux. Plus besoin d'avoir deux portables en réseau, tout est sous Windows 10, c'est génial ! •

12



13



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

Java 13 : une version principalement technique

Comme prévu, la nouvelle version de Java est disponible tout juste 6 mois après la précédente. Ce nouveau calendrier resserré décidé par Oracle depuis Java 9 nous permet d'avoir à disposition deux versions dites majeures de Java par an. Néanmoins, ces versions n'ont de majeures que le nom, comme nous allons le voir dans cet article en détaillant les nouvelles fonctionnalités apportées par Java 13.

Java 12 est sorti en mars 2019 et Java 13 est également à l'heure puisque cette version est sortie en septembre 2019 comme prévu. Il y a des nouveautés mais pas forcément majeures.

Un support long terme qui fait la différence

À présent, pour différencier les versions majeures des versions mineures de Java, il faut plutôt regarder du côté de la date de support long terme annoncée par Oracle. Ainsi, la date de support long terme de Java 8 court jusqu'à Mars 2022 et peut être étendue jusqu'à mars 2025 moyennant finances. Pour Java 10, cette date correspondait à septembre 2018 sans possibilité d'extension. Du côté de Java 11, on a une date de support long terme s'étalant jusqu'à septembre 2023 avec une extension possible jusqu'à septembre 2026. 1

La version 13 du JDK qui vient de sortir bénéficiera d'un support long terme ne dépassant pas mars 2020 et donc, la sortie de la prochaine mouture du JDK. Tout ceci nous permet de déduire que les dernières versions majeures du JDK sont la 8 et la 11.

Du côté des entreprises, on a bien intégré ce nouveau mode de fonctionnement. Ainsi, la plupart sont restées en Java 8 alors que les plus avancées sont passées en Java 11 et attendent la prochaine version avec support long terme digne de ce nom pour éventuellement migrer. Tout ceci n'est évidemment pas sans poser des problèmes de compatibilité avec des bibliothèques de code open source qui ne peuvent suivre la cadence imposée par Oracle.

Ceci étant dit, je vous propose de découvrir les nouvelles fonctionnalités apportées par Java 13 afin de vous faire une meilleure idée sur ce que cette nouvelle mouture va vous apporter.

Introduction des Text Blocks

La grande nouveauté de Java 13 s'adressant aux développeurs est l'introduction des Text Blocks. Là encore, les architectes en charge des évolutions de Java ont décidé de procéder avec précaution puisque les Text Blocks ne sont disponibles qu'en tant que fonctionnalité expérimentale dans Java 13.

Rien d'extraordinaire cependant puisque les Text Blocks sont simplement un nouveau type de String Literal permettant d'écrire des chaînes de caractères sur plusieurs lignes. Afin de simplifier leur usage, il a été décidé qu'un Text Block commencerait par le délimiteur """ et un retour à la ligne. Ceci permettant d'utiliser des simples quotes au sein d'un Text Block sans avoir besoin de les échapper.

Oracle Java SE Support Roadmap*†

Release	GA Date	Premier Support Until	Extended Support Until	Sustaining Support
6	December 2006	December 2015	December 2018	Indefinite
7	July 2011	July 2019	July 2022*****	Indefinite
8**	March 2014	March 2022	March 2025	Indefinite
9 (non-LTS)	September 2017	March 2018	Not Available	Indefinite
10 (non-LTS)	March 2018	September 2018	Not Available	Indefinite
11 (LTS)	September 2018	September 2023	September 2026	Indefinite
12 (non-LTS)	March 2019	September 2019	Not Available	Indefinite
13 (non-LTS)	September 2019***	March 2020	Not Available	Indefinite

1 *Roadmap de support de Java SE*

On peut donc déclarer son premier Text Block de la sorte :

```
System.out.println("""
  Bonjour,
  "Mon premier Text Block"
  pour Programmez !""");
```

Cette utilisation du Text Block étant équivalente à la construction de l'objet *String* suivant :

```
System.out.println("Bonjour,\n\"Mon premier Text Block\"\npour Programmez !");
```

On comprend clairement l'intérêt des Text Blocks et la simplification que cela va permettre lorsque l'on souhaite imprimer un code HTML ou du JSON :

```
String json = """
{
  date: "17/09/2019",
  title: "Java 13",
  version: "Majeure"
}
  """;
```

Les Text Blocks sont également puissants parce qu'ils conservent l'indentation définie par l'utilisateur au moment de la déclaration dans le code source. Seule l'indentation faite avant la première lettre est supprimée, car elle est considérée comme accidentelle. Il faut également noter qu'un retour à la ligne défini avant le délimiteur de fin """ sera considéré comme volontaire et sera donc bien pris en compte dans la chaîne de caractères ainsi construite à l'exécution.
Enfin, cette introduction des Text Blocks a mené à l'ajout de 3 nouvelles méthodes utilitaires au sein de la classe String qu'il est bon de connaître, car elles pourraient vous être utiles un jour ou l'autre :

- String::stripIndent() utilisée pour enlever l'espace blanc ajouté accidentellement au contenu d'un Text Block ;
- String::translateEscapes() utilisée pour traduire les séquences d'échappement ;
- String::formatted(Object… args) qui doit simplifier la substitution de valeur dans un Text Block.

Modification des Switch Expressions

Les Switch Expressions ont été introduites en tant que fonctionnalité expérimentale dans Java 12. Une modification a été proposée pour Java 13. Pour le moment, les Switch Expressions restent toujours en mode preview dans cette nouvelle mouture du JDK.
Java 13 introduit donc le nouveau mot-clé *yield* qui doit être utilisé pour retourner une valeur dans une Switch Expression dont le côté droit de la syntaxe "case L ->" n'est pas qu'une expression simple.
Une fois la fonctionnalité Switch Expression de Java 13 activée, le code suivant est valide :

```
int j = switch (day) {
  case MONDAY -> 0;
  case TUESDAY -> 1;
  default -> {
    int k = day.toString().length();
    int result = myMethod(k);
    yield result;
  }
};
```

Bien sûr, il est toujours possible d'utiliser la syntaxe "case L :" des blocs switch traditionnels au sein des Switch Expressions en la combinant à l'utilisation du mot-clé *yield* :

```
int result = switch (s) {
  case "Texte1":
    yield 1;
  case "Texte2":
    yield 2;
  default:
    System.out.println("Ni Texte 1, ni Texte 2, valeur par défaut à retourner …");
    yield 0;
};
```

Il est important de savoir que les cases d'une Switch Expression doivent être exhaustives. Ainsi, pour toutes les valeurs possibles en entrée, il doit y avoir un label switch qui correspond. Naturellement, cette obligation ne concerne pas les déclarations switch classiques.
Concrètement, cela revient à toujours définir une clause *default* au sein d'une Switch Expression. Les modifications apportées aux Switch Expressions dans Java 13 ont pour but de préparer le terrain à la future introduction du pattern matching dans le JDK. Il n'y a pas de version cible connue pour cette dernière fonctionnalité. Comme toujours avec Java, il faudra donc faire preuve de patience.

Réécriture de l'API Socket

L'implémentation de l'API Socket de Java est très vieille puisqu'elle date du JDK 1.0 ! Elle s'articule principalement autour des classes *java.net.Socket* et *java.net.ServerSocket*. Une réécriture de ces classes et plus généralement de l'API Socket s'imposait donc afin de pouvoir doter le JDK d'une implémentation plus moderne.
Il faut dire que l'implémentation originale a fait son temps. Elle était basée sur un mix de code C et Java très délicat à faire évoluer. De plus, cette implémentation de l'API Socket avait un autre problème majeur : une structure de données native afin de prendre en charge les fermetures asynchrones. Tout ceci entraînant des problèmes de fiabilité et de portabilité ainsi que des problèmes de concurrence.
Pour les développeurs Java, cette réécriture de l'API reste transparente en termes d'utilisation. Les codes s'appuyant sur l'ancienne API restent donc compatibles sans aucun travail de réécriture. Au niveau du JDK cependant, cette nouvelle implémentation de l'API Socket permettra d'avoir un code plus simple et plus moderne, ce qui rendra le travail de debugging et de maintenance plus aisé pour les développeurs du JDK.
En outre, cette nouvelle implémentation vise à favoriser son interopérabilité avec les Fibers, qui sont des User-Mode Threads, en cours d'exploration au sein du projet Loom. Pour rappel, ce projet prévoit d'introduire au sein du JDK un modèle de concurrence plus léger.
Sur les derniers prototypes disponibles au sein d'OpenJDK, une nouvelle classe *Fiber* a été ajoutée, fonctionnant en parallèle de la classe *Thread*. La future API mise à disposition des développeurs Java devrait être semblable à celle de la classe *Thread* à deux différences près :

- Un Fiber va encapsuler n'importe quelle tâche dans un contexte de travail User-Mode. Cela permettra de suspendre et de reprendre une tâche directement au sein du runtime Java plutôt que via le kernel.
- Un planificateur de tâches User-Mode, de type API *ForkJoinPool*, sera utilisé.

Pour le moment, aucune version cible du JDK n'a pas été communiquée pour le projet Loom.
Pour conclure sur cette réécriture de l'API Socket, il est bon de pré-

ciser que les implémentations originales n'ont pas été supprimées du JDK pour le moment et qu'il est possible de continuer à les utiliser en lançant la JVM avec l'option suivante :

```
-Djdk.net.usePlainSocketImpl
```

Améliorations du Garbage Collector ZGC

Pas toujours maîtrisé ni bien connu de nombreux développeurs Java, le Garbage Collector constitue une partie essentielle du fonctionnement de la machine virtuelle Java. Au fur et à mesure de l'évolution du JDK, pour faire face aux nouveaux besoins des applications Java, de nouvelles implémentations ont été proposées par Oracle ou même par des contributeurs tiers tels que Red Hat par exemple.

Java 11 avait ainsi vu l'introduction d'un Garbage Collector à faible latence nommée ZGC. Ce Garbage Collector était proposé à titre expérimental et ciblait uniquement les environnements Linux 64 bits. Nous vous avions d'ailleurs présenté son fonctionnement en détail au sein d'un article dédié dans le numéro 225 de Programmez!.

Avec Java 13, ZGC a été amélioré afin de lui permettre de rendre la mémoire du tas non utilisée au système d'exploitation sous-jacent. Actuellement, ZGC ne le faisait pas et ce comportement était clairement non optimal quel que soit le type d'applications et d'environnements ciblés. Cela s'avérait même encore plus problématique dans les cas où l'empreinte mémoire est un problème majeur. Par exemple, pour :

- Les environnements de type conteneur où les ressources sont payées à l'usage.
- Les environnements où une application peut rester inactive pendant de longues périodes et rentrer en compétition avec d'autres applications pour l'obtention de ressources.
- Une application qui peut avoir des besoins en espace de stockage qui vont varier pendant son exécution. Ainsi, la mémoire du tas nécessaire au démarrage peut être supérieure à ce qui est nécessaire ensuite au runtime lorsque l'application est lancée. Il faut donc pouvoir s'adapter au cours de la vie de l'application à ces besoins différents.

D'autres Garbage Collectors, tels que G1 ou Shenandoah de Red Hat, fournissent déjà cette capacité aujourd'hui. L'ajout de cette capacité à ZGC dans Java 13 était donc une nécessité afin de favoriser son adoption dans le futur.

Dynamic CDS Archives

La dernière grande nouveauté introduite dans Java 13 ne concerne également pas directement les développeurs Java puisqu'il s'agit de la JEP 350 qui vise à étendre le partage de données entre les classes d'une application s'exécutant sur la JVM.

Plus connue sous l'appellation AppCDS, cette fonctionnalité doit rendre possible l'archivage dynamique de classes à la fin de l'exécution d'une application. Ainsi archivées, les classes incluraient toutes les classes de l'application et des bibliothèques de codes chargées au cours de leur exécution et qui ne sont pas présentes dans la couche d'archive de base CDS.

L'ajout des Dynamic CDS Archives à la JVM a pour but d'améliorer l'utilisabilité de AppCDS et d'éliminer la nécessité pour les utilisateurs de faire différents essais afin de créer une liste de classes pour chaque application.

À la lecture de ces quelques lignes, vous aurez tout de suite compris que cette nouveauté concernera une minorité des développeurs Java. Néanmoins, il est toujours intéressant de se tenir au courant de ce type de nouveautés ajoutées au JDK.

Nouveautés attendues pour Java 14

Alors que Java 13 n'était pas encore sorti, les premières rumeurs concernant les nouveautés attendues pour Java 14 ont commencé à fleurir au cœur de l'été. Parmi ces rumeurs, on a déjà une quasi-certitude puisque la JEP 352 a été incluse dans le JDK 14 officiellement. Cette API vise à ajouter au JDK des fichiers de mapping modes pour que l'API *FileChannel* puisse être utilisée ensuite pour créer des instances d'objets *MappedByteBuffer* référant à de la mémoire non volatile. Il s'agit là encore d'une fonctionnalité qui ne vous fera pas rêver en tant que développeur Java.

L'outil de packaging *jpackage* permettant de créer des packages d'applications Java autonomes était prévu pour être intégré de manière expérimentale dans Java 13. Finalement retiré du scope du JDK 13, *jpackage* devrait probablement faire partie de Java 14.

Pour le reste, on n'en sait pas beaucoup plus encore, mais on en apprendra plus au cours des semaines à venir, car il ne faut pas oublier que le JDK 14 sera mis à disposition en mars 2020, c'est-à-dire dans moins de 6 mois.

Conclusion

La nouvelle cadence de sortie des versions du JDK se poursuit sans accroc depuis plus de 2 ans désormais. Quand on se rappelle de ce que l'on a connu en matière de report durant la décennie menant de Java 6 à Java 9, il s'agit déjà d'une victoire en soi pour Oracle et l'ensemble de la communauté Java. La contrepartie à ces mises à jour plus fréquentes tient dans des nouvelles fonctionnalités beaucoup moins excitantes pour les développeurs Java et à des difficultés pour suivre les mises à jour du JDK.

La solution à ce problème consiste à bien avoir conscience que les versions majeures du JDK sont désormais celles ayant un support long terme étendu alors que les versions majeures du JDK n'en bénéficiant pas sont plutôt des mineures visant principalement à faire évoluer le fonctionnement interne du JDK. Pour le développeur Java, elles contiennent de petites améliorations bonnes à connaître, mais qui ne bouleverseront pas leur quotidien.

•

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

Les nouveautés d'ASP.NET Core 3, à la découverte de gRPC

ASP.NET CORE PREVIEW 6, 7, 8 ET 9

Eh oui, nous sommes déjà à la preview 9 d'ASP.NET Core 3. Nous approchons de la release finale ! Aux dernières nouvelles la preview 9 serait finalement la RC ou la version avant la RC, soit la version quasi finale ! En attendant regardons les dernières améliorations apportées à ces previews.

niveau 200

En premier lieu, il semblerait qu'ASP.NET MVC / API soit considéré comme mature et stable, car il n'y a plus beaucoup d'évolution. Il semble en revanche que Microsoft mette les bouchées doubles sur Razor Pages et Blazor.

ASP.NET CORE 3 en général

- Les templates ASP.NET Core s'affichent maintenant en tant que modèles de niveau supérieur dans Visual Studio dans la boîte de dialogue "Créer un nouveau projet".
- Simplification des templates, suppression des morceaux code fréquemment supprimés tels que le bloc de consentement de cookie, ainsi que les scripts et ressources statiques qui sont dorénavant référencés en tant que fichiers locaux au lieu d'utiliser des CDN basés sur l'environnement actuel.
- Les templates Angular sont mis à jour (Angular 8).
- Simplification des templates Blazor.
- Le template de bibliothèque de classes Razor remplace le template de bibliothèque de classes Blazor.
- **EventCounters** : à la place des compteurs de performances Windows, .NET Core 3 introduit une nouvelle méthode d'émission de métriques via **EventCounters**.
- Introduction de nouvelles primitives de services réseaux vous permettant d'ajouter un support pour les protocoles non HTTP.
- Prise en charge de socket de domaine Unix pour le transport de sockets Kestrel (options **ListenUnixSocket()** de la méthode **ConfigureKestrel** du builder d'hote).

Blazor

- On note l'introduction de la directive **@attribute** permettant d'ajouter des attributs dans les fichiers Razor.
- Introduction de la directive **@code** permettant l'ajout de blocs de code dans les fichiers Razor.
- Introduction de la directive **@key** utilisée pour identifier les composants dans une collection de composants identiques.
- Introduction enfin de la directive **@namespace** permettant de spécifier le namespace des classes générées.
- Support de l'authentification et de l'autorisation.
- « Attribute splatting » pour les composants.
- Prise en charge du Databinding pour les **TypeConverters** et les **Generics**.
- Clarification des attributs de directive attendus HTML vs C #, les gestionnaires d'évènements (**@onclick** et **@bind**) attendent maintenant les valeurs C #, de sorte qu'un caractère @ n'est plus obligatoire pour spécifier la valeur du gestionnaire d'évènements.
- Les composants dans les fichiers .razor sont désormais insensibles à la casse.
- Amélioration de la logique de reconnexion pour les applications serveur Blazor.
- Support de la culture pour le data binding. Exemple : **<input @bind="date" @bind:culture="CultureInfo.InvariantCulture" />**.
- Les event handlers et les attributs de data binding ont été migrés dans l'assembly**Microsoft.AspNetCore.Components.Web.**
- Amélioration du Routing avec le nouveau composant **RouteView.**
- Ajout du composant **LayoutView** pour permettre un rendu particulier, exemple quand une page est non trouvée.
- Le routing est désormais découplé de l'autorisation avec l'ajout d'un composant**AuthorizeRouteView** qui permet de gérer l'affichage d'un composant si l'utilisateur est authentifié.
- Support de composants provenant de différentes assembly pour le routeur.
- Ajout des classes **OwningComponentBase** et **OwningComponentBase<TService>** permettant d'exposer dans un composant la propriété **ScopedServices** de type **IServiceProvider** pouvant être utilisée pour résoudre les services étendus à la durée de vie du composant.

JSON

Il est désormais possible dans le fichier **Startup.cs** de préciser l'utilisation de JSON.NET à la place de **System.Text.Json** utilisé par défaut.

SECURITE

ASP.NET Core 3 supporte désormais l'authentification Kerberos et par certificats.

SIGNALR

La librairie **@aspnet/signalr@next** (coté client) permet désormais la reconnexion automatique après déconnexion.

GRPC

- gRPC dispose maintenant d'un client gRPC permettant de se connecter a des endpoints gRPC en .NET. Il est donc possible

désormais de développer des solutions gRPC entièrement en .NET.
- gRPC supporte désormais HTTPS.
- Outil CLI pour la gestion de la génération de code gRPC **dotnet-grpc**.
- Prise en charge de **CallCredentials** permettant une interopérabilité avec des bibliothèques existantes telles que **Grpc.Auth** qui reposent sur **CallCredentials**.
- Amélioration des diagnostics : ajout du support de classe **Activity** accessibles aux outils de télémétrie (à travers la propriété **Baggage**).
- Les fournisseurs **Grpc.AspNetCore.Server** et **Grpc.Net.Client** nouvellement introduits émettent maintenant les compteurs d'évènements suivants: *total-calls*, *current-calls*, *calls-failed*, *calls-deadline-exceeded*, *messages-sent*, *messages-received*, *calls-unimplemented*.
- Ajout d'un outil de référence de service dans Visual Studio.
- Nouveau pour gRPC : amélioration de la consommation des streams dans le client gRPC avec l'ajout de l'interface **IAsyncStreamReader** afin d'itérer sur les réponses de manières asynchrones.

DÉCOUVERTE DE GRPC

Introduction

Avant de commencer à décrire les fonctionnalités gRPC d'ASP.NET Core 3, nous allons d'abord revoir ensemble ce qu'est gRPC.

Tout d'abord gRPC est un framework RPC (Remote procedure call) et non pas un framework Microsoft.

gRPC est développé par Google. Il utilise le protocole HTTP2 pour le transport des données et Protocol Buffers comme format de sérialisation également développé par Google. Il s'agit donc d'un descripteur d'interface tout comme le WSDL pour SOAP si on veut faire une analogie.

Avec gRPC, on peut s'authentifier, transmettre des données de manière bidirectionnelle, annuler la communication, paramétrer des délais d'attente.

gRPC n'est pas non plus nouveau, il a été open sourcé par Google en 2015 et supportait déjà Java, C, C++, Node.js , Python et Ruby. Autant dire que Microsoft était en retard, mais le retard est rattrapé et nous allons voir comment construire un service gRPC et un client gRPC en .NET.

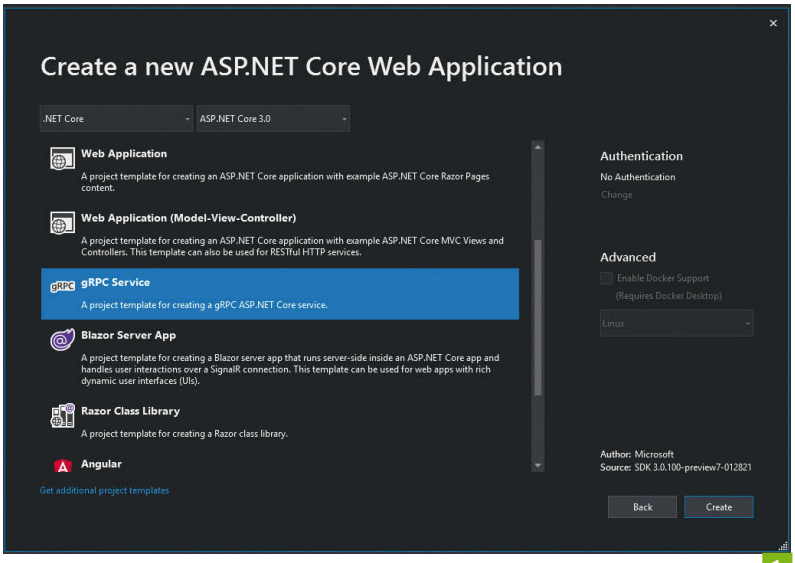
Pourquoi utiliser gRPC ? Pour des raisons de performance essentiellement dans le cas où on voudrait créer des microservices légers où l'efficacité est primordiale.

Création d'un service gRPC

Après avoir correctement installé le SDK en preview et Visual Studio 2019, créez un projet ASP.NET Core puis sélectionnez le template « gRPC Service » : 1

Anatomie d'un service gRPC

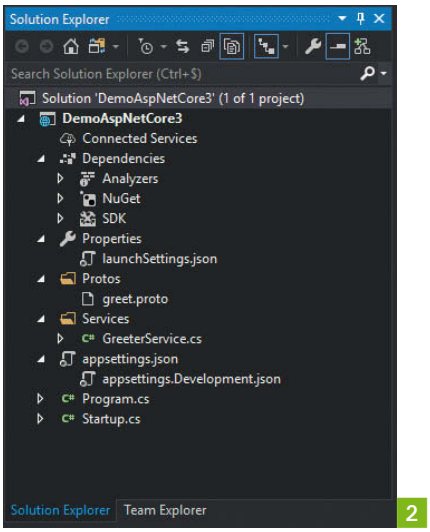
On retrouve les composantes habituelles d'un projet ASP.NET Core comme :
- Le launchSettings ;
- Les fichier appsettings.json ;
- Le Program.cs ;



1

- Le Startup.cs.

Dans un projet ASP.NET Core 3 de service gRPC nous découvrons :
- Un répertoire **Services** avec un service gRPC, nommé dans cet exemple **GreeterService** héritant d'une classe autogénérée nommée **Greeter.GreeterBase**, avec son constructeur et sa / ses méthode(s). Notez que l'injection dépendance built-in d'ASP.NET Core est disponible avec un service gRPC.
- Un répertoire **Protos** contenant les **Protocols buffers**. Autrement dit les contrats de service. 2



2

A quoi ressemble un fichier Protocol Buffer ?

Ils sont suffixés avec l'extension **.proto,** et ils contiennent :
- La définition de la syntaxe utilisée (**syntax = proto3**) ;
- Le namespace auquel ils appartiennent, déclaré à l'aide du mot clé **option csharp_namespace = namespace ;**
- Le nom de la définition du service à l'aide du mot clé **package ;**
- Le nom du service est défini à l'aide du mot clé **service ;**
- Et les modèles entrants / sortants via **message.**

```
syntax = "proto3";

option csharp_namespace = "DemoAspNetCore3";

package Greet;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}
```

```
// The request message containing the user's name.
message HelloRequest {
 string name = 1;
}

// The response message containing the greetings.
message HelloReply {
 string message = 1;
}
```

A quoi ressemble un fichier de service gRPC ?

Comme indiqué plus haut, c'est avant tout un fichier C# avec sa classe de service héritant d'une classe autogénérée, d'un constructeur et de ses méthodes de service. Chaque méthode de service prend en paramètre un modèle un entrée de type **message** tel que défini dans le fichier **.proto** et d'un contexte de type **ServerCallContext**

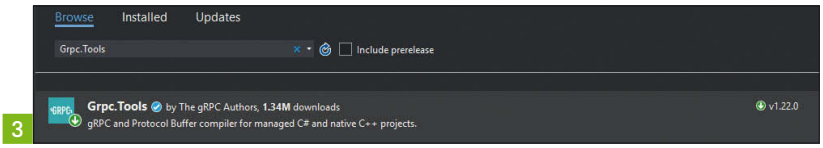
```
using System.Threading.Tasks;
using Grpc.Core;
using Microsoft.Extensions.Logging;

namespace DemoAspNetCore3
{
  public class GreeterService : Greeter.GreeterBase
  {
    private readonly ILogger<GreeterService> _logger;
    public GreeterService(ILogger<GreeterService> logger)
    {
      _logger = logger;
    }
```

```
    public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)
    {
      return Task.FromResult(new HelloReply
      {
        Message = "Hello " + request.Name
      });
    }
  }
}
```

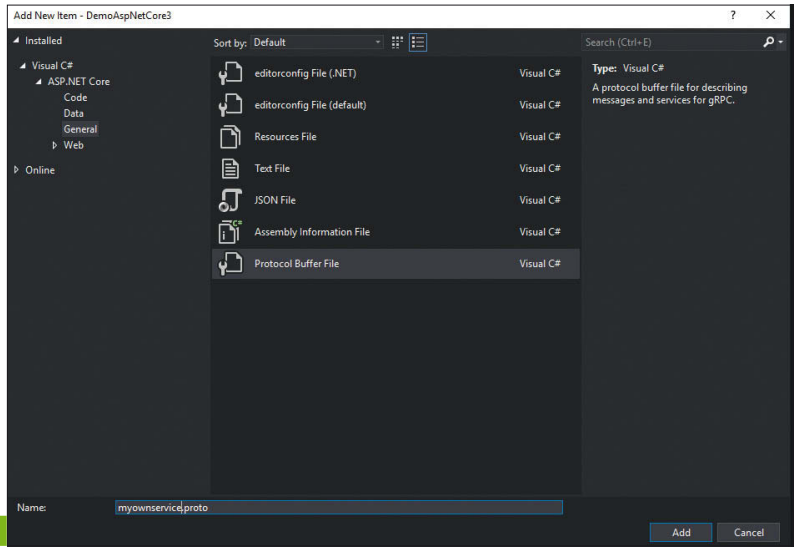
A quoi ressemble le fichier du projet ?

Dans sa version la plus simpliste (après génération du projet par Visual Studio) nous pouvons observer la présence habituelle du **TargetFramework** mais surtout le nécessaire pour inclure les fichiers **.proto** et également la dépendance Nuget gRPC nommée **Grpc.AspNetCore** :

```
<Project Sdk="Microsoft.NET.Sdk.Web">

 <PropertyGroup>
  <TargetFramework>netcoreapp3.0</TargetFramework>
 </PropertyGroup>

 <ItemGroup>
  <Protobuf Include="Protos\greet.proto" GrpcServices="Server" />
 </ItemGroup>

 <ItemGroup>
  <PackageReference Include="Grpc.AspNetCore" Version="0.1.22-pre2" />
 </ItemGroup>

</Project>
```



3



4

Comment créer un nouveau service gRPC ?

Tout d'abord téléchargez le package **Grpc.Tools**. Cela va vous permettre de générer les classes de votre service gRPC défini dans vos fichiers Protocol Buffer que vous créerez manuellement. Ces classes seront autogénérées lorsque vous builderez votre solution dans Visual Studio. 3

Ensuite créez votre propre fichier Protocol Buffer comme suit et implémentez vos contrats : 4

```
syntax = "proto3";

option csharp_namespace = "DemoAspNetCore3";

package My;

service MyOwnService {
 rpc Whols (MyRequest) returns (MyReply) {}
}

message MyRequest {
 string name = 1;
}
message MyReply {
```

```
 string message = 1;
}
```

- Buildez votre solution et normalement, si tout se passe bien, les classes seront générées dans ce répertoire : **..\obj\Debug\netcoreapp3.0** 5

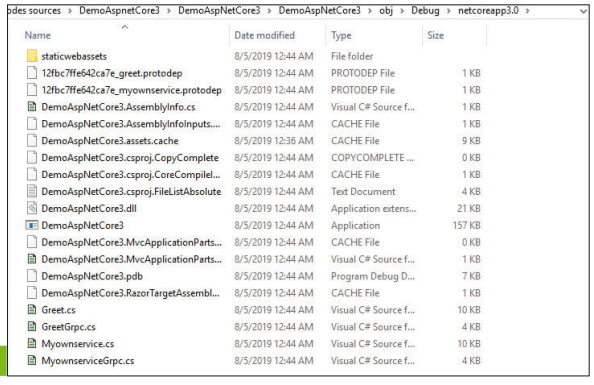
- Créez ensuite votre service gRPC en créant une nouvelle classe C# :

```
using Grpc.Core;
using Microsoft.Extensions.Logging;
using System.Threading.Tasks;

namespace DemoAspNetCore3.Services
{
  public class MyOwnGRpcService : MyOwnService.MyOwnServiceBase
  {
    private readonly ILogger<MyOwnGRpcService> _logger;
    public MyOwnGRpcService(ILogger<MyOwnGRpcService> logger)
    {
      _logger = logger;
    }

    public override Task<MyReply> WhoIs(MyRequest request, ServerCallContext context)
    {
      return Task.FromResult(new MyReply
      {
        Message = "Hello " + request.Name
      });
    }
  }
}
```

Comme dans l'exemple auto généré par Visual Studio, la classe dont il faut hériter est le nom de votre service déclaré à l'aide du mot clé service dans votre fichier Procotol Buffer avec le pattern suivant : **MyOwnService.MyOwnService**Base.

- N'oubliez-pas dans Startup.cs d'ajouter vos endpoints, (**GreeterService** est ajouté automatiquement) comme suit :

```
    app.UseEndpoints(endpoints =>
    {
```



5

```
        endpoints.MapGrpcService<GreeterService>();
        endpoints.MapGrpcService<MyOwnGRpcService>();

        endpoints.MapGet("/", async context =>
        {
          await context.Response.WriteAsync("Communication with gRPC endpoints must be made through a gRPC client. To learn how to create a client, visit: https://go.microsoft.com/fwlink/?linkid=2086909");
        });
      });
```

- Finalement si vous utilisez https, n'oubliez pas de truster votre certificat ssl local avec la commande suivante : *dotnet dev-certs https --trust*

Compilez l'ensemble du projet et tout devrait bien se passer !

Consommation d'un service gRPC avec une application console

Au moment où j'écris ces lignes, la documentation de Microsoft concernant la création d'un client gRPC n'est pas à jour. Dans le tutoriel qui suit, je vais me baser sur la preview 9 sortie la première semaine de septembre, tandis que la documentation actuelle de Microsoft date du 25 août 2019.

Nous allons créer une application console en .Net Core 3 avec la preview 3 pour l'exemple.

Les dépendances à installer sont les suivantes :

- Grpc.Net.Client -prerelease
- Google.Protobuf -prerelease
- Grpc.Tools -prerelease

Notez bien qu'au moment où j'écris ces lignes, ces packages sont encore en prerelease.

- Pour consommer votre service dans un client, créez comme pour la partie serveur un fichier **.proto** contenant le Protocol Buffer de votre service (voir plus haut), puis ajouter la référence de votre fichier **.proto** dans votre **.csproj**, avec cette fois-ci comme valeur pour la propriété **« GrpcServices = Client »** :

```
<Project Sdk="Microsoft.NET.Sdk">

 <PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>netcoreapp2.2</TargetFramework>
 </PropertyGroup>

 <ItemGroup>
  <None Remove="Protos\myownservice.proto" />
 </ItemGroup>

 <ItemGroup>
  <PackageReference Include="Google.Protobuf" Version="3.10.0-rc1" />
  <PackageReference Include="Grpc.Net.Client" Version="0.2.23-pre2" />
  <PackageReference Include="Grpc.Tools" Version="2.24.0-pre1">
   <PrivateAssets>all</PrivateAssets>
   <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
  </PackageReference>
```

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

[Programmez!] Le magazine des développeurs

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

Offres 2019

1 an 49€
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 79€
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 €)

eni

* Offre limitée à la France métropolitaine

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

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

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

☐ Abonnement 1 an : 49 €
11 numéros + 1 vidéo ENI au choix :
☐ Abonnement 2 ans : 79 €
22 numéros + 1 vidéo ENI au choix :

☐ Vidéo : Symfony 3
☐ Vidéo : Raspberry Pi

PROG 233
Valable jusqu'au 31 octobre 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

NOUVEAU ! OFFRES 2019

1 an
11 numéros

+ Histoire de la micro-informatique 1973 à 2007

+ clé USB Programmez!

69€*

2 ans
22 numéros

+ Histoire de la micro-informatique 1973 à 2007

+ clé USB Programmez!

+ pack Maker 2 cartes
Attention : quantité limitée

99€*

1 an
11 numéros

+ 1 an de PHARAON *Magazine* (Histoire / Archéologie) 4 numéros

69€*

2 ans
22 numéros

+ 2 ans de PHARAON *Magazine* (Histoire / Archéologie) 8 numéros

99€*

OFFRES SPÉCIALES D'ABONNEMENT DISPONIBLES SUR WWW.PROGRAMMEZ.COM

*(*Offre limitée à la France métropolitaine. Pour l'étranger ; nous consulter)*

```
</ItemGroup>

<ItemGroup>
 <Protobuf Include="Protos\myownservice.proto" GrpcServices="Client" />
</ItemGroup>

</Project>
```

- Ensuite compilez. Et, comme pour la partie serveur, vous devriez retrouver vos classes générées dans le répertoire **« obj »** de votre projet console : 6

Ecrivez le client gRPC simplement ainsi, (bien sur conformément aux service exposé) :

```
using DemoClientAspNetCore3;
using Grpc.Net.Client;
using static DemoClientAspNetCore3.MyOwnService;

namespace ConsoleAppGRPC
{
  class Program
  {
    static void Main(string[] args)
    {
      var channel = GrpcChannel.ForAddress("https://localhost:5001");
      var client = new MyOwnServiceClient(channel);

      var reply = client.WhoIs(new MyRequest { Name = "Anthony" });
    }
  }
}
```

Exécutez votre console en mode debug : 7

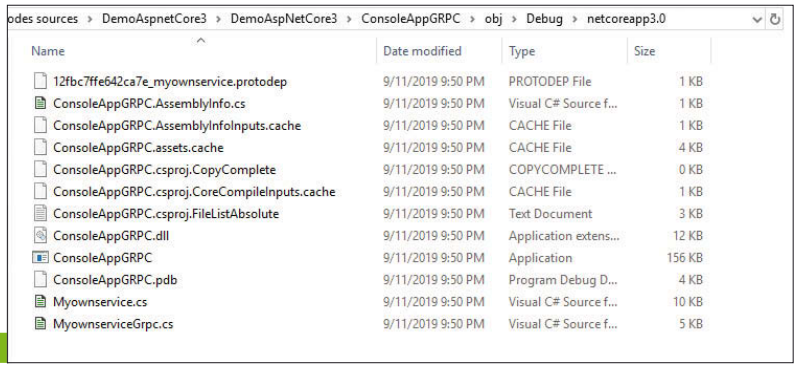
Tout en vérifiant les logs d'information de votre service coté serveur : 8

Conclusion

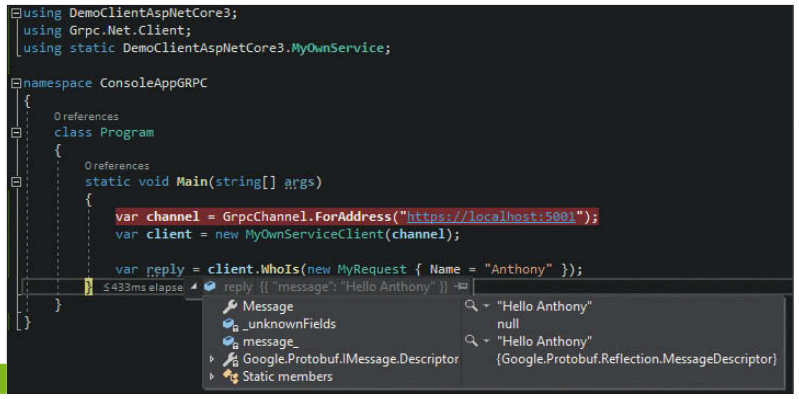
Dans cet article dedié à la découverte ded'ASP.NET Core 3 et gRPC nous avons vu comment scaffolder un projet gRPC à l'aide du template gRPC service de Visual Studio mais aussi comment créer notre propre service gRPC à l'aide des outils adéquat tel que le package Grpc Tools.

Nous avons également vu comment consommer un service gRPC à l'aide d'un client .NET.
A noter que pour le moment un client supportant http2 comme un client .NET, POSTMAN, etc. sont les seuls moyens de consommer un service gRPC. La librairie client **grpc-web** pour les SPA tels qu'Angular, React, etc., n'est pas compatible, car, pour le moment, les navigateurs ne supportent pas le niveau de contrôle requis sur les requêtes Web pour prendre en charge un client gRPC.
Pour en savoir plus vous pouvez vous référer à cette page ici : https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-3.0#limited-browser-support
Si vous souhaitez utiliser malgré tout **grpc-web,** cela reste possible, mais il faudra utiliser un proxy tel que **Envoy** pour transformer les requêtes HTTP 1 en HTTP 2 et vice versa : 9

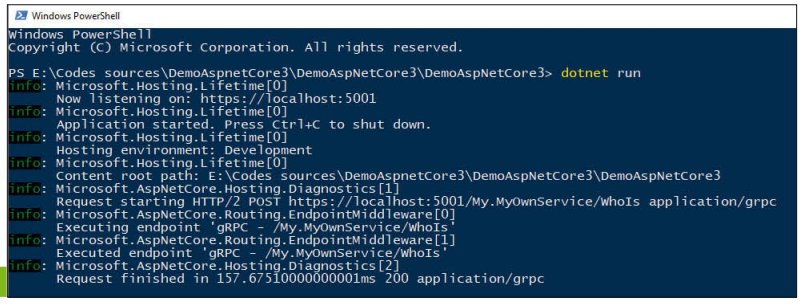
A noter également que **TestServer** n'est pas compatible avec gRPC, vous ne pourrez donc utiliser ce dernier pour faire des tests d'intégration sur vos services gRPC.
Je vous propose de vous retrouver un prochain numéro pour approfondir gRPC lors de la release finale d'ASP.NET Core 3 et .NET Core 3 •



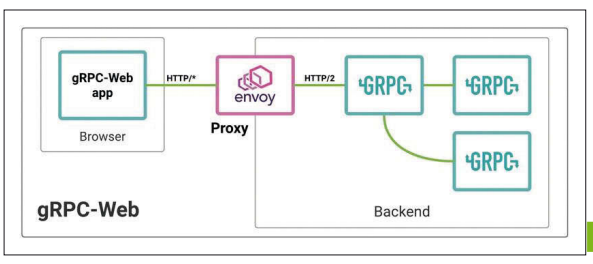
6



7



8



9

Michael Bacci
michael.bacci.software@gmail.com - https://www.linkedin.com/in/michaelbacci
Senior Software Engineer R&D, Expert en HPC, C/C++, J2EE, Python
Freelance, Michael travaille dans des projets R&D où les enjeux de performance, scalabilité et algorithmique sont particulièrement sensibles.

niveau 200

La vectorisation dans les CPU modernes

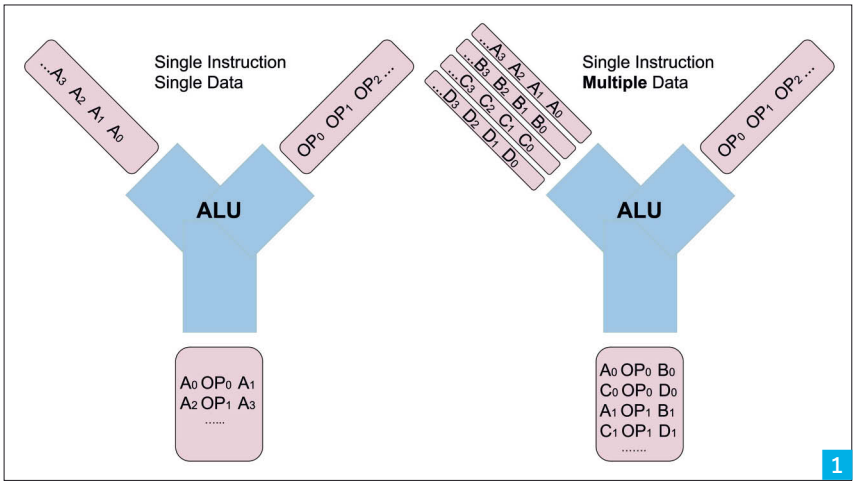
Pour faire face à des demandes toujours croissantes de puissance de calcul, les nouveaux CPU ont augmenté leurs capacités matérielles comme le nombre de coeurs par CPU, la quantité de cache et leur niveau ainsi que la fréquence d'horloge. Depuis 1997, une technique appelée Vectorisation a rendu possible l'exécution d'une même instruction sur un vecteur des données : elle permet, à la condition d'écrire un code ad-hoc, d'accélérer l'exécution des programmes les plus variés, en passant du domaine purement scientifique au simple fait, par exemple, de regarder un film.

Dans cet article, on explorera ces techniques de programmation et les ressources matérielles disponibles dans les CPU modernes pour réécrire nos programmes dans l'idée d'une accélération d'exécution.

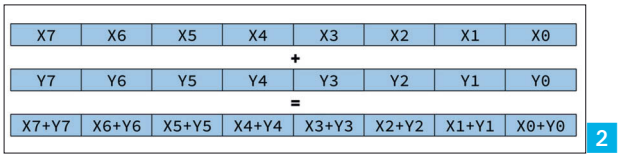
1 – Qu'est-ce que la Vectorisation ?

La vectorisation est une opération du CPU qui permet d'exécuter en un seul cycle d'horloge des instructions arithmétiques et/ou en logique booléenne sur plusieurs données à la fois.
Ces données doivent être enregistrées dans des registres spéciaux présents dans chaque cœur des CPU.
La Fig [1] compare une opération d'addition "classique" binaire et une opération "moderne" vectorielle. La différence est très simple : dans une opération "classique" de somme de deux nombres (qu'ils soient entiers ou décimaux), ces derniers sont mémorisés dans deux registres bien différents de l'ALU (Arithmetic Logic Unit - élément de calcul du CPU). La somme est mémorisée dans l'un des deux registres ou bien directement dans une adresse en mémoire (RAM). Dans une opération "moderne" de somme de deux vecteurs de données, ces derniers sont mémorisés de façon contiguë dans des registres spéciaux de taille variable selon l'architecture : 80 bits pour le standard MMX, 128 bits pour SSE, 256 bits et 512 bits pour AVX.
Toutes les valeurs contenues dans les deux registres sont additionnées en tenant compte de l'ordre des valeurs individuelles, et la somme est placée (toujours en fonction de l'architecture) dans l'un des deux registres ou bien dans un troisième.
La Fig [2] montre une somme avec le registre __mm256 en float (décimale à précision simple), le calcul du bénéfice est vite fait : une fois les données dans les registres, il est possible d'additionner seize nombres décimaux en un seul cycle d'horloge.

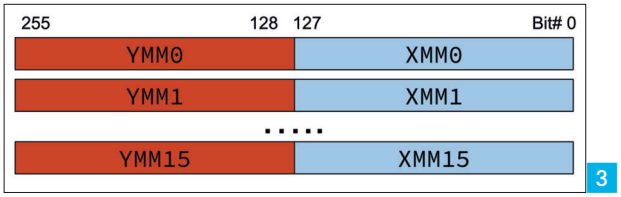
#Note technique :
*Il ne faut pas commettre l'erreur de calcul suivante : le nombre total des cycles d'horloge qu'il faut pour additionner 2*X nombres en utilisant la technique "moderne", c'est-à-dire, en utilisant deux registres de 256 bits, ne correspond pas à multiplier par huit le nombre des cycles d'horloge qu'il faudrait pour la même technique "classique" avec des registres à 32 bits. L'inverse n'est pas vrai non plus, c'est-à-dire que le total des cycles utilisés pour additionner 2*X nombres décimaux en technique "classique" n'équivaut pas (supposant le standard AVX 256bit) à réduire d'un facteur de 1/8 en technique "moderne".*
L'explication est simple et s'appelle <<overhead des transferts de données>> : pour chaque opération "classique" ou "moderne", les données doivent être copiées de la mémoire RAM (ou du cache) dans les registres, puis les résultats sont ensuite transférés des registres vers la RAM. Parfois des opérations de synchronisation entre les différents niveaux de caches s'ajoutent au coût de l'opération.



1



2



3

2 - Les registres de la CPU

La Fig.[3] montre les registres pour SSE et AVX; chacune utilise des noms et des tailles différents et les instructions varient en fonction de chaque famille.
Chaque standard SIMD a une nomenclature propre de ses registres, par exemple sur AVX on trouve les registres de YMM0 à YMM15. Avec **gdb** on peut facilement voir leur contenu:

```
(gdb) print/d $ymm0
$1 = {
 v8_float = {0, 0, 0, 0, 0, 0, 0, 0},
 v4_double = {0, 0, 0, 0},
 v32_int8 = {47 <repeats 16 times>, 0 <repeats 16 times>},
 v16_int16 = {12079, 12079, 12079, 12079, 12079, 12079, 12079, 12079, 0, 0, 0, 0, 0, 0, 0, 0},
 v8_int32 = {791621423, 791621423, 791621423, 791621423, 0, 0, 0, 0},
 v4_int64 = {3399988123389603631, 3399988123389603631, 0, 0},
```

```
 v2_int128 = {62718710765820030520700417840365121327, 0000000000000000000000000000000}
}
```

Quand on travaille avec ces registres on peut les utiliser dans leur représentation décimale ou en nombre entier, cet affichage est du coup très utile lorsque l'on veut voir les informations mémorisées dans l'utilisation que l'on a choisie.

3 - Parlons technique : {single, multi, many}-cores

Pour une vision d'ensemble de la vectorisation, on doit faire un petit rappel de certains concepts liés aux paradigmes de programmation parallèle :

- La technique de vectorisation est une implémentation d'un concept plus large nommé SIMD (Single Instruction Multiple Data). Cette technique de parallélisation en single-core permet donc d'exploiter les maxima des capacités d'instructions de l'ALU (contenue dans chaque cœur du CPU).
- Pour exploiter au maximum les capacités d'un CPU moderne, le multi-cores est le paradigme qui permet d'utiliser tous (ou un sous ensemble) des cœurs d'un CPU en même temps. Cette technique est aussi communément appelée <<programmation parallèle>>. Il existe plusieurs façons d'atteindre ces capacités, notamment en utilisant des librairies comme les thread POSIX ou le Threading Building Blocks (TBB) d'Intel. Toutefois, il existe un standard qui s'appelle OpenMP.
- Encore plus en profondeur, un 3e paradigme, le many-cores, permet d'exploiter les capacités de calculs de plusieurs CPU et/ou clusters en même temps. Le standard le plus connu est MPI - Multi Passing Interface. La librairie OpenMPI (open source) et Intel MPI library sont les plus utilisées.

Nous n'avons pas énuméré toutes les techniques de programmation parallèle. D'autres techniques comme la programmation des GPGPU (General-Purpose computing on Graphics Processing Units) ou des coprocesseurs (comme le Xeon Phi d'Intel) font partie des techniques dites de "programmation parallèle hybride".

On peut maintenant avancer dans la lecture des concepts de la vectorisation, en sachant où elle se positionne par rapport à d'autres techniques et quel impact elle peut avoir.

4 - Langage et Librairies : les fonctions intrinsèques

Pour utiliser les instructions SIMD plusieurs chemins sont possibles :

- Code assembleur : écrire des directives en assembleur directement dans son code C/C++ en faisant appel aux instructions SIMD.
- Fonction intrinsèque : une cartographie des opérations SIMD en C. La correspondance est de 1-à-n entre une fonction intrinsèque et le nombre de sous-opérations SIMD. Un exemple est la fonction intrinsèque _m256_fmadd_ps qui exécute une addition et une multiplication (ret[i] = a[i]*b[i] + c[i]).
- Wrapper: en C++, Python et même JavaScript, utiliser des wrappers (extensions de haut niveau - qui nous évitent les détails techniques des opérations) qui à leur tour utilisent des fonctions intrinsèques. Par exemple, en C++, il existe des librairies qui nous permettent d'écrire directement une opération telle que `auto R = A*B + C` où A, B et C sont des objets (qui représentent des vecteurs numériques) et qui ont une overloading des opérateurs {+, - , *, /, %} capable de simplifier des opérations vectorielles complexes dans une syntaxe très proche de l'écriture mathématique.
- Directive du compilateur : ajouter des directives simples <<#pragma>> dans un code existant afin d'indiquer au compilateur quelle portion de code optimiser. Exemple :

```
#pragma omp simd
for (i=0; i < n; ++i)
  Ret[i] = A[i]*B[i] + C[i]
```

- Options de compilation : les compilateurs modernes sont capables d'optimiser des boucles for comme la précédente en totale autonomie. Mais plus les expressions mathématiques sont complexes et font référence à des index non contigus et moins les compilateurs sont capables d'optimiser le code.

Du chemin 1 au chemin 5 la complexité du code diminue, sa lisibilité et maintenabilité augmentent, mais les performances peuvent aussi se dégrader.

5 - Premier pas : mon système supporte-t-il la vectorisation ?

En utilisant un compilateur gcc sur un système Linux, voici la commande pour énumérer les standards SIMD acceptés :

```
$ gcc -march=native -dM -E - < /dev/null | egrep "AVX|MMX|SSE" | sort
#define __AVX__ 1
#define __AVX2__ 1
#define __MMX__ 1
#define __SSE__ 1
#define __SSE2__ 1
#define __SSE2_MATH__ 1
#define __SSE3__ 1
#define __SSE4_1__ 1
#define __SSE4_2__ 1
#define __SSE_MATH__ 1
#define __SSSE3__ 1
```

L'output montre les deux versions existantes du AVX, la version pour MMX et 8 extensions pour SSE. Nous explorerons le standard AVX créé en 2010 qui a évolué en AVX2 ces dernières années.

Connaître le type de processeur, et pas seulement un sous-ensemble de ses capacités, est fondamental pour prédire toutes les optimisations possibles à implémenter dans le code. Voici une des façons pour extraire ces informations :

```
$ lscpu && cat /proc/cpuinfo | egrep "(flags)" | sort | uniq
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              8
On-line CPU(s) list: 0-7
Thread(s) per core:  2
```

```
Core(s) per socket:   4
Socket(s):            1
NUMA node(s):         1
Vendor ID:            GenuineIntel
CPU family:           6
Model:                94
Model name:           Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
Stepping:             3
CPU MHz:              3882.602
BogoMIPS:             6818.02
Virtualization:       VT-x
L1d cache:            32K
L1i cache:            32K
L2 cache:             256K
L3 cache:             8192K
NUMA node0 CPU(s):    0-7
flags                 : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush
dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perf
mon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes
64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic mov
be popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch ida
arat epb pln pts dtherm hwp hwp_noitfy hwp_act_window hwp_epp tpr_shadow vnmi fle
xpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx rdseed adx
 smap clflushopt xsaveopt xsavec xgetbv1 xsaves
```

Avoir une référence de cpu sur laquelle construire et tester son code est important, mais si le produit que l'on conçoit est destiné à être utilisé sur des machines hétérogènes, il faut utiliser des options de compilation pour garantir le support à l'extension SIMD utilisée dans le programme. Avec gcc, il est possible de générer un code binaire compatible avec certaines classes de CPU et/ou des extensions. Par exemple, on peut indiquer des extensions précises en utilisant le préfix '-m' pour '-mavx' (pour le support AVX) si le CPU qui exécute le code le supporte, ou bien un type de CPU particulier avec '-march=*cpu-type*'.

Il faut savoir qu'en utilisant l'option de compilation '-march', votre code pourrait ne pas fonctionner sur des CPU différents de celui indiqué en paramètre, c'est pourquoi il existe une autre option de compilation '-mtune=*cpu-type*' destinée à générer du code optimisé pour l'architecture demandée, mais qui sera toujours interprété par d'autres CPU.

6 - SIMD Multiversioning

Quand on écrit du code SIMD, on doit toujours choisir entre deux chemins :

- Écrire un code pour une architecture précise : on est dans ce cas dans le domaine du calcul scientifique pour la plupart des cas que je connais. Il y a un code de simulation numérique qui a nécessité d'être optimisé pour une tâche bien particulière.
- Écrire un code optimisé qui bénéficie des avantages du SIMD, mais qui soit exploitable pour le plus grand nombre d'architectures (même les ARM). Ce cas prévoit généralement le traitement des flux vidéo (conversion de format et affichage), les jeux vidéo (opérations d'algèbre linéaire sur vecteur et matrice pour les affichages 2D/3D et simulation des effets physiques, par exemple, les collisions), la cryptographie (connexions réseaux, modules de messagerie cryptée). Voici un exemple :

[Code 1]

```
__attribute__ ((target ("default")))
void simd_code() {
   //Code générique: exécuté sur les CPU dont l'architecture n'est pas dans la liste des
 cibles (target) prévues
}

__attribute__ ((target ("sse4.2")))
void simd_code() {
   //Pour les CPU qui supportent le standard SSE4.2
}

__attribute__ ((target ("avx")))
void simd_code() {
   //Pour les CPU qui supportent le standard AVX
}

__attribute__ ((target ("arch=amdfam10")))
void simd_code() {
   //Code spécifique aux processeurs de la famille AMD 0x10
}
```

En langage C++, cet overloading des fonctions ne peut pas être accepté, car les fonctions (ou méthodes si définies dans une classe) ont toutes la même signature. Du coup le compilateur ne pourrait pas savoir quelle fonction appeler. Avec l'attribut de fonction '__attribute__' le compilateur génère pour chaque "target" une seule et unique version qui est conforme C/C++ ISO. Ensuite, chaque fonction spécifique est ajoutée au même binaire, et au run-time, un code introduit par le compilateur choisira quelle version appeler.

Regardons de plus près ce mécanisme : chaque target a un nom particulier en langage Assembleur, par exemple, la target 'avx' s'appelle "simd_code.avx", celle par défaut s'appelle "simd_code.default" etc. Lorsque la fonction est appelée, le compilateur ajoute une couche de détection ou resolver "call simd_code.resolver" en Assembleur :

[Code 2]

```
simd_code() [clone .resolver]:
    sub    rsp, 8
    call   __cpu_indicator_init  #charge dans __cpu_model les informations relatives
 à l'architecture.
    test   BYTE PTR __cpu_model[rip+13], 2 #Est-ce cette architecture AVX ?
    je     .L14
    mov    eax, OFFSET FLAT:simd_code() [clone .avx]
.L13:
    add    rsp, 8
    ret
.L14:
    test   BYTE PTR __cpu_model[rip+13], 1 #Est-ce cette architecture SSE4.2 ?
    je     .L15
    mov    eax, OFFSET FLAT:simd_code().sse4.2
    jmp    .L13
[...]
```

L'ordre des tests est très important: ils sont en fait ordonnés à partir du support vectoriel du plus performant au moins performant.
Le code précédent peut être réécrit de cette façon :

[Code 3]

```
void simd_code_default() {...}
void simd_code_avx() {...}
void simd_code() {
    if (CHECK_CPU_HAS(AVX_SUPPORT) { simd_code_avx(); } else
        if (CHECK_CPU_HAS(SSE42_SUPPORT) { simd_code_sse42(); } else
        if (...) {...} else
        { simd_code_default(); }
}
```

Le pseudo-code ci-dessus doit en réalité être substitué par un code inline en Assembleur qui doit faire les mêmes vérifications du [Code 1] généré automatiquement par le compilateur.

7 - Base de la programmation AVX

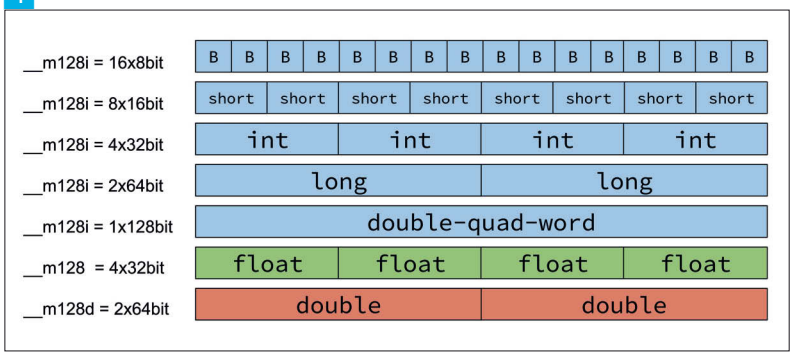
Comme tous les éléments de programmation, commençons par les types de données :

Type	Description
__m128	Vecteur de 128 bits contenant 4 float
__m128d	Vecteur de 128 bits contenant 2 doubles
__m128i	Vecteur de 128 bits contenant des entiers : char,short, int,uint,unsigned long long
__m256	Vecteur de 256 bits contenant 8 float
__m256d	Vecteur de 256 bits contenant 4 doubles
__m256i	Vecteur de 256 bits contenant des entiers

La Fig. 4 montre les types pour standard SSE
Ces nouveaux types permettent de définir des variables avec lesquelles exécuter des opérations grâce aux fonctions intrinsèques qui se composent principalement en :

- Arithmétique : somme, soustraction, multiplication, division ;
- Arithmétique composée (extension FMA) : composition des différentes opérations arithmétiques de base; exemple ret[i] = a[i]*b[i] + k*c[i] ;
- Mathématiques : trigonométrie (sin, cos, tan…) et fonctions communes comme exp, log, sqrt, pow... ;
- Permutation : la permutation des éléments dans un vecteur permet, par exemple, le calcul matriciel et les opérations sur les nombres complexes ;
- I/O: lecture et écriture avec la mémoire centrale (RAM) ;
- Shuffling : la capacité de sélectionner un sous-ensemble des éléments d'un vecteur pour ensuite exécuter une opération uniquement sur les éléments sélectionnés ;
- Cryptographie: algorithmes AES et SHA1.

4



Il existe une convention pour le nom des fonctions intrinsèques :
_mm<nombre_de_bits>_<nom>_<type_de_donnée>

Voici l'interprétation:

- **<nombre_de_bits>** c'est tout simplement la taille du vecteur en bits. Pour le vecteur de 128 bits, ce champ est vide ;
- **<nom>** décrit l'opération exécutée par la fonction ;
- **<type_de_donnée>** exprime quels types de données sont acceptés :
- **<ps>** float (single-precision),
- **<pd>** double (double-precision),
- **<epi8/epi16/epi32/api64>** signed integers,
- **<m128/m128i/m128d/m256/m256i/m256d>** type de donnée accepté quand le vecteur de retour n'est pas le même que celui en entrant.

Par exemple __mm256 vect = _mm256_set_ps(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0) initialise le vecteur 'vect' avec huit nombres en float.

8 - Let's code

L'approche que je vous propose, est de vous montrer comment mesurer le bénéfice empirique "(complexité du code)/(temps exécution)" sur un même algorithme écrit en utilisant des approches d'optimisation différentes, dans le but de trouver la meilleure.
On considère deux vecteurs ***a*** et ***b*** de taille N (telle que N soit multiple de 64) et on veut calculer le temps d'exécution moyen de ***c = a + b***:

```
// file <<sum.hpp>>
#pragma GCC optimize("O3","unroll-loops","omit-frame-pointer","inline","tree-vectorize",
"opt-info-vec-all")
#pragma GCC option("arch=native","tune=native","no-zeroupper")
#pragma GCC target("avx")

#include <x86intrin.h> //Fonctions intrinsèques

namespace simd {

namespace utils {
void alloc_vect(float **vec, const int SIZE) {  // Ce check est "pédagogique": on travaille avec
 AVX sur des variables alignées sur 256 bits
  if (SIZE % 32) throw std::runtime_error("SIZE not aligned in 32bytes = 256 bits");
  *vec = (float *) _mm_malloc(SIZE * sizeof(float), 32);
}

void assign_vect(float *vec, const int SIZE, const float base, const float f = 0.03f) {
  for (int i = 0; i < SIZE; ++i)
    vec[i] = base + f * i;
}
} //namespace utils

__attribute__((optimize("no-tree-vectorize"))) //désactiver l'auto-vectorisation du compilateur
inline void __sum_no_vect__(const float *a, const float *b, float *c, const int SIZE) {
```

```
  for (int i = 0; i < SIZE; ++i)
    c[i] = a[i] + b[i];
}

// GCC nous informe de la vectorisation automatique de la boucle ici présente avec le message suivant:
// <<sum.hpp:30:23: note: loop vectorized>>
inline void __sum_auto_vect__(const float *a, const float *b, float *c, const int SIZE) {
  for (int i = 0; i < SIZE; ++i)
    c[i] = a[i] + b[i];
}

// On considère les vecteurs virtuellement regroupés en sous-vecteurs de 8 éléments chacun
// a,b,c = [(0,1,2,3,4,5,6,7),(8,9,10,11,12,13,14,15),(16,...)(..SIZE-1)]
inline void __sum_implicit_vect__(const float *a, const float *b, float *c, const int SIZE) {
  for (int i = 0; i < SIZE; i += 8) { //l'index avance de 8 éléments à la fois
    __m256 qa1 = _mm256_load_ps(a + i); //chargement de 256 bytes (ou 8 float) depuis
 l'offset <<i>> de <<a>>
    __m256 qa2 = _mm256_load_ps(b + i); //idem pour <<b>>
    __m256 qsum = _mm256_add_ps(qa1, qa2); //exécution de l'opération de somme
    _mm256_store_ps(c + i, qsum); //copie dans <<c>> sur l'offset <<i>>
  }

  /* même algorithme, mais avec un usage différent des index
  for (int i = 0; i < SIZE / 8; ++i) {
    _mm256_store_ps(c, _mm256_add_ps(_mm256_load_ps(a), _mm256_load_ps(b)));
    a += 8; b += 8; c += 8;
  }*/
}

// L'unroll permet de réduire le nombre d'itérations (dans ce cas précis d'un facteur 4, mais on
 aurait pu en ajouter davantage)
inline void __sum_unroll_vect__(const float *a, const float *b, float *c, const int SIZE) {
  for (int i = 0; i < SIZE / (8 * 4); ++i) {
    _mm256_store_ps(c, _mm256_add_ps(_mm256_load_ps(a), _mm256_load_ps(b)));
    a += 8;b += 8;c += 8;
    _mm256_store_ps(c, _mm256_add_ps(_mm256_load_ps(a), _mm256_load_ps(b)));
    a += 8;b += 8;c += 8;
    _mm256_store_ps(c, _mm256_add_ps(_mm256_load_ps(a), _mm256_load_ps(b)));
    a += 8;b += 8;c += 8;
    _mm256_store_ps(c, _mm256_add_ps(_mm256_load_ps(a), _mm256_load_ps(b)));
    a += 8;b += 8;c += 8;
  }
}

} //namespace simd
// file <<sum.cpp>>
#include <benchmark/benchmark.h>
#include "sum.hpp"

// Pour mesurer les performances on va utiliser le framework google-benchmark
class SIMDFixture : public benchmark::Fixture {
public:
  void SetUp(const ::benchmark::State& state) {
    simd::utils::alloc_vect(&a, state.range(0));
    simd::utils::alloc_vect(&b, state.range(0));
    simd::utils::alloc_vect(&c, state.range(0));
    simd::utils::assign_vect(a, state.range(0), 3.14f);
    simd::utils::assign_vect(b, state.range(0), 1<<10, 0.314f);
  }

  void TearDown(const ::benchmark::State& state) {
    _mm_free(a);_mm_free(b);_mm_free(c);
  }

public:
  // On informe les compilateurs que les variables suivantes sont alignées
  float * __attribute__((__aligned__(8))) a;
  float * __attribute__((__aligned__(8))) b;
  float * __attribute__((__aligned__(8))) c;
};

BENCHMARK_DEFINE_F(SIMDFixture, BM_SumNoVect)(benchmark::State& state) {
  for (auto _ : state)
    simd::__sum_no_vect__(a, b, c, state.range(0));
}
BENCHMARK_DEFINE_F(SIMDFixture, BM_SumAutoVect)(benchmark::State& state) {
  for (auto _ : state)
    simd::__sum_auto_vect__(a, b, c, state.range(0));
}
BENCHMARK_DEFINE_F(SIMDFixture, BM_SumImplicitVect)(benchmark::State& state) {
  for (auto _ : state)
    simd::__sum_implicit_vect__(a, b, c, state.range(0));
}
BENCHMARK_DEFINE_F(SIMDFixture, BM_SumImplicitUnrollVect)(benchmark::State& state) {
  for (auto _ : state)
    simd::__sum_unroll_vect__(a, b, c, state.range(0));
}

// Le Arg(1L<<18) définit la variable SIZE=2^18
BENCHMARK_REGISTER_F(SIMDFixture, BM_SumNoVect)->Arg(1L<<18);
BENCHMARK_REGISTER_F(SIMDFixture, BM_SumAutoVect)->Arg(1L<<18);
BENCHMARK_REGISTER_F(SIMDFixture, BM_SumImplicitVect)->Arg(1L<<18);
BENCHMARK_REGISTER_F(SIMDFixture, BM_SumImplicitUnrollVect)->Arg(1L<<18);
BENCHMARK_MAIN();
```

Ce qui nous donne :

```
Run on (8 X 4000 MHz CPU s)
CPU Caches:
  L1 Data 32K (x4)
  L1 Instruction 32K (x4)
  L2 Unified 256K (x4)
  L3 Unified 8192K (x1)
Load Average: 1.16, 0.36, 0.66
-----------------------------------------------------------------------------------------------
Benchmark                                     Time           CPU         Iterations
-----------------------------------------------------------------------------------------------
BM_SumNoVect/262144                      165204 ns     160315 ns         4653
BM_SumAutoVect/262144                     69634 ns      67628 ns         9802
BM_SumImplicitVect/262144                 58072 ns      57982 ns        12095
BM_SumImplicitUnrollVect/262144           55105 ns      55019 ns        11838
```

Tout d'abord, on doit apprendre à lire les données :

- Time : c'est le temps total d'exécution de l'algorithme dans sa

boucle for(auto _ : state). Ce temps comprend aussi les syscall du noyau kernel et les I/O avec la mémoire.
- CPU : cela correspond au seul temps que le CPU a employé pour exécuter l'algorithme, sans interruption avec d'autres éléments extérieurs au CPU.
- Itérations : ce sont le nombre de fois que l'algorithme a été appelé dans la boucle for(auto _ : state) laquelle permet de faire les statistiques finales.

Interprétations:
- Entre BM_SumNoVect/262144 (la fonction sum_no_vect avec N=262144=2^18) et BM_SumAutoVect, il y a une amélioration de 237% (sur le 800% théorique) en faveur de la vectorisation automatique générée par le compilateur, avec juste une option pour le compilateur.
- On atteint une amélioration de 300% avec BM_SumImplicit UnrollVect qui se paye dans la complexité du code.
- Entre les différentes versions qui utilisent l'AVX implicite (BM_SumAutoVect) ou explicite (BM_SumImplicitVect, BM_SumImplicitUnrollVect) le bénéfice n'est pas si significatif.

Pour une opération simple comme ***c = a + b*** la seule vectorisation automatique est la meilleure solution. C'est dans la résolution des formules plus complexes que l'AVX explicite peut atteindre son bénéfice maximal.

9 - OpenMP 4.0

Avec la dernière version d'OpenMP sont disponibles des directives de compilation qui permettent la vectorisation automatique du code combinées avec la puissance de la programmation parallèle. Je vous laisse comme exercice de rajouter la fonction suivante à la suite des tests analysés:

```
inline void __sum_omp_simd_vect__(const float *a, const float *b, float *c, const int SIZE)
{#pragma omp for simd aligned(a,b,c : 32)
  for (int i = 0; i < SIZE; ++i)
    c[i] = a[i] + b[i];
}
```

Avec cette seule directive #pragma on peut atteindre un bénéfice de 1400% par rapport à la version sans optimisation.

10 - Quel avenir pour la vectorisation ?

L'intelligence artificielle (IA) et le Deep Learning utilisent des équations devenues "standard" dans ces domaines. C'est par exemple les cas de l'opération de convolution et la descente du gradient. Dans le standard AVX-512 VNNI (Vector Neural Network Instructions) ont été rajoutées des opérations pour accélérer la convolution dans des réseaux neuronaux. Mon pari c'est que les futurs standards de vectorisation auront un set d'instructions enrichi d'opérations mathématiques orientées IA.

11 - Conclusion

Ceux qui seront capables d'exploiter la totalité des ressources mises à disposition par le hardware seront les vainqueurs des batailles digitales actuelles et futures. La vectorisation est une arme très puissante que l'on ne peut pas ignorer si l'on veut être compétitif. *Le coût de cette arme se paie en codes*, mais pour certaines applications il peut aussi se résoudre par quelques options au niveau du compilateur et de simples directives d'optimisation.

A vous de jouer.

Robin Huart
Consultant sénior
INVIVOO

Réaliser un Mastermind avec Tkinter

Il existe plusieurs outils pour créer des interfaces graphiques en Python, parmi lesquels on peut citer par exemple Kivy, PyQt, wxPython et Tkinter, même s'il en existe bien d'autres. À l'exception de Kivy, tous ceux-ci sont des interfaces pour des bibliothèques graphiques bien connues et utilisables dans d'autres langages (Qt, wxWidgets, Tk). Les principaux avantages de Tkinter sont sa facilité de prise en main, le fait que la bibliothèque soit installée par défaut avec Python et, en conséquence, la très grande quantité de ressources disponibles pour aider les développeurs confirmés comme ceux qui débutent dans la programmation d'interfaces graphiques.

niveau 200

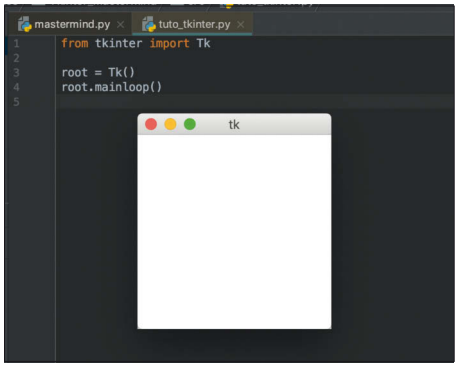
Tkinter est une interface (le nom signifiant « *Tk Interface* ») pour manipuler les objets de la bibliothèque Tk, créée par John Ousterhout à la fin des années 90. Il s'agissait à l'origine d'une extension pour son nouveau langage de scripts Tcl. Tant et si bien que la combinaison des deux est depuis célèbre sous le nom de Tcl/Tk. Tkinter n'est en réalité qu'une fine couche orientée objet au-dessus de Tcl/Tk, la bibliothèque Python embarquant son propre interpréteur Tcl. Ce tutoriel se divisera en deux parties principales. Nous ferons d'abord une revue des concepts mis en œuvre dans tout projet basé sur Tkinter, avec une présentation succincte (et non-exhaustive) des principaux *widgets* disponibles. Nous étudierons ensuite un exemple d'implémentation du célèbre jeu *Mastermind*. Tous les tests seront effectués à partir de la version 3.6 de Python et le rendu visuel sera obtenu à partir d'un MacBook Pro.

1. Premiers pas

À grand traits, une application graphique consiste en une fenêtre principale (voire plusieurs) dans laquelle on peut disposer divers composants avec lesquels l'utilisateur pourra interagir, les placer selon ses vœux, à partir de laquelle d'autres fenêtres et boîtes de dialogue pourront apparaître, et où l'on peut définir des réactions du programme en fonction des différentes actions effectuées par l'utilisateur. D'emblée, cela implique une façon de programmer très différente des programmes à but non interactif, où le flux peut être connu dès le départ en fonction des données d'entrée qui seront injectées. Ici, c'est un humain (a priori) qui choisira quelles actions mener, qui pourront être considérées comme autant de nouveaux points d'entrée du programme.

Voyons tout de suite comment écrire un programme minimal faisant apparaître la fenêtre principale de notre future application.

```
from tkinter import Tk

root = Tk()
root.mainloop()
```

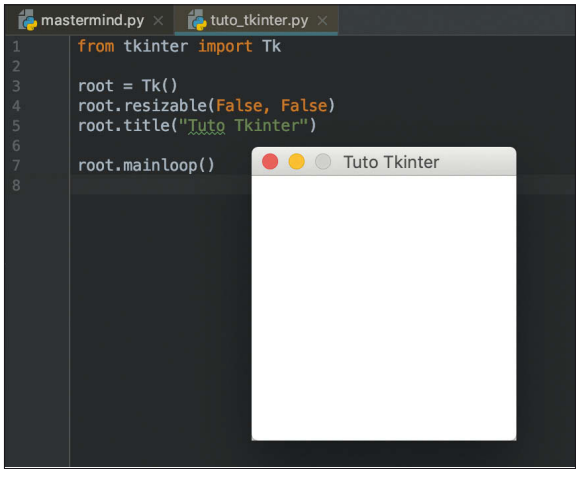


L'objet racine de toute application Tkinter est une instance de la classe Tk. Si nous exécutons ces 3 lignes de code, nous voyons bien apparaître une fenêtre vide, avec laquelle il n'est pas encore possible d'interagir. La dernière instruction est essentielle. Sans elle, la fenêtre disparaît aussitôt l'application lancée. Il s'agit en effet d'une boucle infinie servant à capter les signaux émis par les actions de l'utilisateur. On la placera généralement à la toute fin du programme.

Comme beaucoup d'autres objets que nous verrons, il est possible de modifier *a posteriori* la fenêtre principale. Par exemple, la fenêtre créée par défaut avec le code précédent est étirable. Nous pouvons figer ses dimensions à l'aide de la méthode *resizable*. Ajoutons-lui également un titre. Le code devient alors :

```
from tkinter import Tk

root = Tk()
root.resizable(False, False)
root.title("Tuto Tkinter")

root.mainloop()
```

Voici ce que nous constatons maintenant à l'exécution :



Nous pouvons constater que la fenêtre n'est plus étirable et que son titre a effectivement changé.

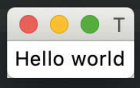
2. Rajouter des éléments

2.a Quelques exemples de base détaillés

En l'état, nous sommes bien d'accord que l'application n'a pas de

grande utilité. Mais nous pouvons à présent y ajouter des composants. Ces éléments, aussi appelés *widgets* en anglais (contraction de *window* et *gadget*), seront des instances d'autres classes disponibles dans le module tkinter. Le plus simple d'entre eux est probablement le *Label*, un objet dont le seul but est d'afficher du texte à l'écran. Voyons comment l'insérer dans notre fenêtre vide.

```
mastermind.py | tuto_tkinter.py
1  from tkinter import Label, Tk
2
3  root = Tk()
4  root.title("Tuto Tkinter")
5
6  Label(root, text='Hello world').pack()
7
8  root.mainloop()
9
```



La taille de la fenêtre s'est automatiquement adaptée à son contenu, une seule zone de texte assez petite, raison pour laquelle la fenêtre est elle-même très petite. Afin de pouvoir l'étirer, retirons dès à présent l'instruction « root.resizable(False, False) ».

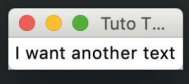
Il y a plusieurs choses intéressantes à relever sur la manière dont nous venons de rajouter notre premier *widget*, qui seront également vraies pour tous les autres :

- Un *widget* est créé simplement en instanciant une classe, sans obligation de récupérer cette instance dans une variable et de faire autre chose avec.
- Le premier argument sera toujours la fenêtre (pas un identifiant, l'objet lui-même) dans laquelle le *widget* doit s'insérer, suivi par des arguments nommés optionnels de configuration (il en existe beaucoup).
- Une fois le *widget* créé par instanciation, il est **indispensable** de dire à Tkinter où le placer. C'est le rôle ici de la méthode *pack*. Attention, si cette étape n'est pas effectuée et bien qu'il existe, le *widget* **ne s'affichera tout simplement pas**.

La méthode *pack* fait appel à ce qu'on appelle un *geometry manager* chargé de répartir les éléments selon les demandes du développeur. On reviendra un peu plus tard sur cet aspect essentiel de la programmation d'interfaces graphiques.

Tous les paramètres de configuration pouvant être affectés à la création d'un *widget* sont modifiables a posteriori via la méthode *config* ou *configure* (les deux sont identiques). Exemple :

```
mastermind.py | tuto_tkinter.py
1  from tkinter import Label, Tk
2
3  root = Tk()
4  root.title("Tuto Tkinter")
5
6  my_label = Label(root, text='Hello world')
7  my_label.pack()
8  my_label.config(text='I want another text')
9
10 root.mainloop()
11
```



Pour effectuer cette opération, il a suffi de récupérer l'instance du *widget* dans une variable, de bien penser à n'appeler *pack* que dans un second temps (car *pack* modifie l'objet « sur place » et renvoie *None*), puis d'appeler *config* ou *configure* avec les mêmes mots-clés qu'on aurait employés à l'instanciation.

Parmi les paramètres de configuration les plus courants, on peut citer :

- *bg* (ou *background)* qui prend comme valeur un code couleur (sous forme de chaine de caractères) remplissant le fond du *widget ;*
- *fg* (ou *foreground)* qui fait la même chose pour le reste du *widget* (typiquement, le texte affiché sur un *Label* ou un *Button*) ;
- *font* pour préciser la police de caractères ;
- *cursor* pour modifier la forme du curseur de la souris quand on passe sur le *widget ;*
- *state*, spécifique à certains *widgets*, pour les rendre inactifs (grisés) ou non ;
- …

Le pendant en lecture du *widget Label* est l'*Entry*, qui, comme son nom l'indique, permet d'entrer du texte (en français, on dit aussi « une zone de saisie »). Il est courant d'utiliser un *Label* à proximité d'une *Entry* pour indiquer à l'utilisateur le sens qui sera donné par l'application au texte attendu. Exemple :

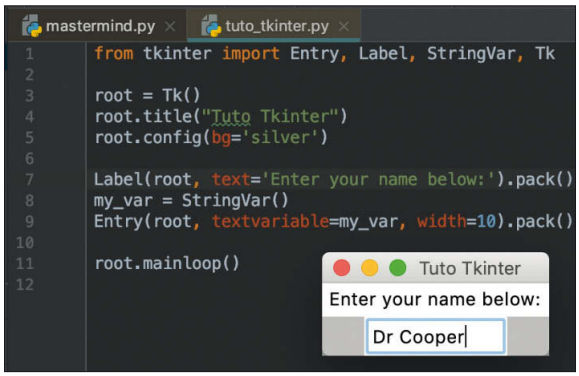
Avant édition *Pendant l'édition*

Ici, nous avons créé une zone de saisie de largeur 10 puis inséré un texte (« John Doe ») en partant du premier caractère (0) de la zone. La largeur se mesure en nombre de caractères. Toutefois, notre exemple ne signifie pas qu'on ne pourra saisir que 10 caractères, mais plutôt que seulement 10 caractères seront assurés d'être visibles quel que soit le texte entré.

Ce texte entré par l'utilisateur peut être récupéré à n'importe quel moment par l'instruction my_entry.get(). Une autre manière très classique de l'obtenir, partagée par quasiment tous les autres *widgets* pouvant se voir attribuer une valeur (à la différence de la méthode *get* que tous les objets Tkinter ne possèdent pas), consiste à déclarer au préalable une variable d'un type spécial qui recevra la valeur retournée par l'action de l'utilisateur (dans ce cas, le texte tapé dans la zone de saisie). Il existe trois types de telles variables :

- IntVar (pour recevoir des entiers) ;
- DoubleVar (pour recevoir des nombres flottants) ;
- StringVar (pour recevoir des chaînes de caractères) ;
- BooleanVar (pour recevoir des booléens sous forme entière, c'est-à-dire 0 ou 1).

Pour notre cas, nous voudrons récupérer du texte. Il faut donc créer une variable de type *StringVar* et l'associer à un paramètre précis du *widget* : *textvariable*. Exemple :

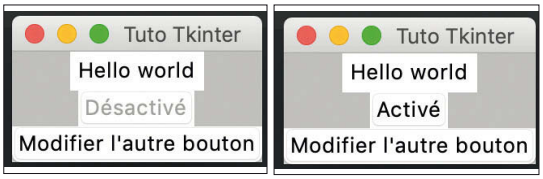


Avec cette technique, plutôt que faire my_entry.get() lorsque nous souhaiterons récupérer le texte entré, nous pourrons faire my_var.get() (la variable *my_entry* n'ayant au passage plus de raison d'être dans ce code).

Un autre exemple de *widget* simple est le bouton (*Button*), qui se résume à afficher un texte, sur lequel on peut appuyer, qui déclenche une action en réponse. Nous allons en créer deux, le premier ne faisant rien et le second modifiant le premier.

```
from tkinter import Button, DISABLED, Label, Tk

root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Hello world').pack()

button1 = Button(root, text='Activé', command=lambda: None)
button2 = Button(root, text='Modifier l\'autre bouton',
        command=lambda: button1.config(text='Désactivé',
                        state=DISABLED,
                        cursor='watch'))
button1.pack()
button2.pack()
root.mainloop()
```

Voyons le résultat avant et après avoir appuyé sur le second bouton :



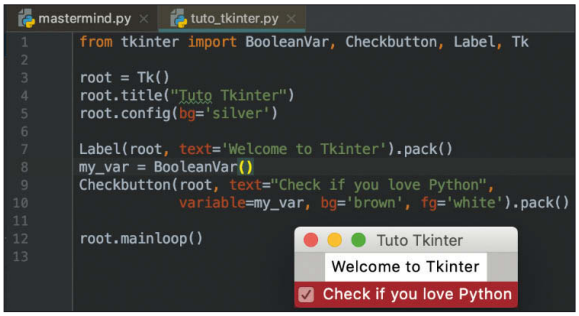
La partie un peu plus complexe par rapport aux précédents *widgets* consiste en la définition de l'action (*command*) à effectuer en cas d'appui. La valeur de cet argument est nécessairement une fonction qui n'accepte aucun argument, comme nous le voyons ici. Les fonctions anonymes (*lambda*) seront souvent très utiles à cet endroit, mais rien n'empêche de définir des fonctions classiques complexes ailleurs ou dans d'autres modules et de les utiliser à la place. On appelle généralement les fonctions ainsi passées en argument des *callbacks* (fonctions appelées non pas directement mais en cascade depuis un autre appel). C'est une technique très largement utilisée par Tkinter.

2.b Quelques autres objets disponibles

Comme nous pouvons nous en douter, il existe beaucoup plus de *widgets* que ces trois-là. Bien comprendre comment utiliser ces trois premiers permet néanmoins de saisir les notions essentielles mises en œuvre pour tous les autres.

Checkbutton

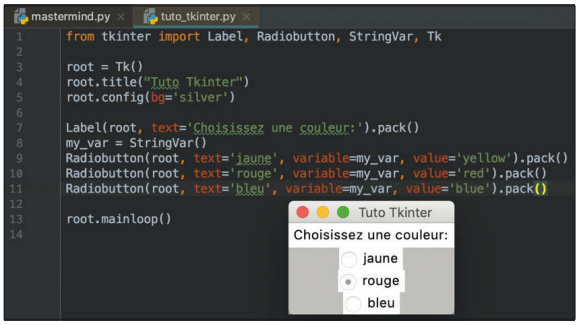
Il s'agit d'une case à cocher avec du texte. Le résultat se récupère via la technique de la *textvariable*, sauf que le paramètre s'appelle cette fois-ci *variable*. Exemple :



Le statut « coché ou non » se récupère à tout moment via l'instruction my_var.get().

Radiobutton

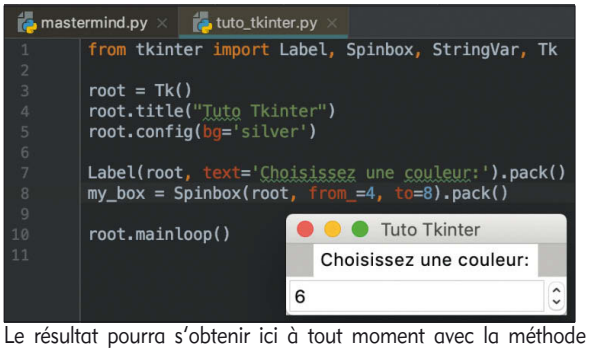
Ce *widget* sert à représenter un choix multiple. Illustration :



Dans cet exemple c'est la couleur « rouge » qui est sélectionnée. Le texte et la valeur récupérée sont décorrélés. Ici, le résultat de my_var.get() renverra « red ». Le lien d'exclusion mutuelle entre ces différents choix n'existe que par le fait que les trois *widgets* partagent la même variable.

Spinbox

Cet objet définit une entrée dans laquelle récupérer des valeurs qui peuvent être incrémentées à la hausse ou à la baisse en respectant des bornes. Illustration :



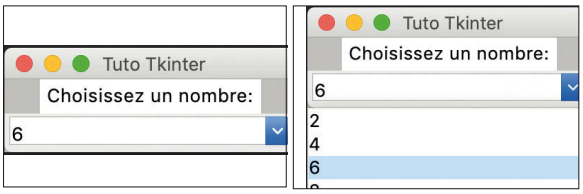
Le résultat pourra s'obtenir ici à tout moment avec la méthode my_box.get().

Combobox

La *Combobox* est un *widget* inclus dans une extension de Tkinter (disponible nativement) appelée *ttk* (*Tk themed widgets*), qui est un sous-module de Tkinter. Cette extension redéfinit un certain nombre de *widgets* de Tkinter (comme *Button* ou *Spinbox*) avec un rendu visuel et une configuration légèrement différents, et en définit aussi de nouveaux. Nous ne nous attarderons pas sur les spécificités de *ttk*, mais il est nécessaire de mentionner son existence dans un tutoriel sur Tkinter. La *Combobox* est typiquement un widget intéressant disponible uniquement dans *ttk*. Son principe rejoint celui de la *Spinbox*.

```
from tkinter import Label, Tk
from tkinter.ttk import Combobox

root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Choisissez un nombre:').pack()
my_box = Combobox(root, values=(2, 4, 6, 8))
my_box.current(2)
my_box.pack()

root.mainloop()
```

À partir de ce code, nous obtenons le rendu suivant :



Avant la sélection *Pendant la sélection*

On spécifie une liste de valeurs à parcourir via l'argument *values*. La méthode my_box.current(i) permet de placer le choix sur une des valeurs (i) de la liste à partir de son indice (en partant de 0, comme toujours en Python). Le résultat pourra être récupéré à tout moment par l'instruction my_box.get().

3. Les gestionnaires de positionnement

Il existe trois gestionnaires de positionnement (aussi appelés *geometry managers* en anglais) permettant de disposer les objets dans une fenêtre :

- *pack* (que nous avons vu sans rentrer dans les détails jusqu'à présent) ;
- *grid* (une grille qui permet de définir des espacements réguliers entre les objets) ;
- *place* (qui permet de placer des objets à partir de coordonnées absolues) ;

Nous ne nous attarderons pas sur *place*, car cette méthode est en pratique assez peu utilisée. En revanche, les deux autres nous seront utiles pour coder le Mastermind. Il est à noter que lorsqu'on commence à utiliser l'un de ces gestionnaires de positionnement dans une fenêtre, on ne doit pas se mettre tout à coup à en utiliser un autre. Autrement dit, ils ne doivent pas être mélangés dans un même conteneur. Sinon, nous risquerions de passer un temps inimaginable à tenter de les réconcilier pour obtenir le rendu souhaité.

3.a Le gestionnaire pack

Nous avons jusqu'à maintenant utilisé cet outil sans jamais passer le moindre argument. Le résultat fut qu'il plaçait systématiquement les *widgets* les uns sous les autres. C'est un des deux comportements principaux, *pack* ne permettant que d'empiler des composants verticalement ou les uns à côté des autres horizontalement. Ceci est cependant parfois suffisant comme nous le verrons dans le code du Mastermind.

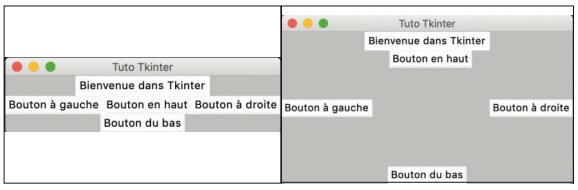
On peut cependant utiliser trois arguments pour gérer le positionnement des *widgets* :

- side
- fill
- expand

Le paramètre side permet de définir depuis quel bord de la fenêtre empiler les objets. Ces valeurs indiquant des bords sont des constantes définies par Tkinter et qu'il suffit d'importer : TOP, BOTTOM, LEFT et RIGHT. La valeur par défaut est TOP, ce qui explique pourquoi depuis le début pack empile les objets de haut en bas.

Illustration :

```
from tkinter import Button, Label, Tk
from tkinter import BOTTOM, LEFT, RIGHT, TOP

root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Bienvenue dans Tkinter').pack()
Button(root, text='Bouton du bas').pack(side=BOTTOM)
Button(root, text='Bouton à gauche').pack(side=LEFT)
Button(root, text='Bouton à droite').pack(side=RIGHT)
Button(root, text='Bouton en haut').pack(side=TOP)

root.mainloop()
```



Avant étirement (initialement) *Après étirement*

On constate que le Label du début, étant par défaut disposé contre le bord du haut, forme une pile avec le seul autre *widget* disposé via side=TOP.

Le paramètre fill, quant à lui, permet de demander de faire en sorte que le *widget* remplisse tout l'espace qui lui est alloué dans la direction horizontale (X), verticale (Y) ou les deux (BOTH).

```
from tkinter import Button, Label, Tk
from tkinter import BOTH, X, Y
```

```
root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Bienvenue dans Tkinter').pack(fill=X)
Button(root, text='Bouton 1').pack(fill=X)
Button(root, text='Bouton 2').pack(fill=Y)
Button(root, text='Bouton 3').pack(fill=BOTH)

root.mainloop()
```

Ce code produit une fenêtre qui se comporte ainsi lorsqu'on étire la fenêtre :

Sans étirement *Avec étirement*

On constate que fill=X permet d'occuper tout l'espace horizontal alloué aux boutons concernés (tous sauf le second, qui laisse apparaître la couleur de fond de la fenêtre sur ses côtés). De la même manière, fill=Y permet d'occuper la totalité de l'espace vertical alloué, ce qui n'est pas perceptible ici car cette allocation ne varie pas avec la taille de la fenêtre, comme illustré par la seconde image. La spécification fill=BOTH combine les deux effets. À noter que les constantes X, Y et BOTH sont des variables globales définies par Tkinter, qu'on peut remplacer respectivement par les chaînes de caractère 'x', 'y' et 'both'.
Le paramètre expand accepte comme valeur un booléen (ou un entier faisant office de booléen, 0 ou 1) et vaut par défaut False. S'il est mis à True, le *widget* voit alors son espace alloué varier lorsque la fenêtre est étirée.

```
from tkinter import Button, Label, Tk
from tkinter import BOTH, X, Y

root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Bienvenue dans Tkinter').pack(fill='x')
Button(root, text='Bouton 1').pack(fill='x', expand=True)
Button(root, text='Bouton 2').pack(fill='y', expand=True)
Button(root, text='Bouton 3').pack(fill='both', expand=True)

root.mainloop()
```

Le rendu obtenu par ce code au lancement est identique à celui du code précédent (avant étirement), et voici ce qui se passe si on étire la fenêtre :

Tous les boutons voient leur espace alloué augmenter, mais ne réagissent pas de la même manière. Le bouton 1 continue à remplir tout l'espace horizontal mais se déplace pour rester au centre de son espace verticalement. Pour le bouton 2 c'est l'opposé, il reste au centre horizontalement mais remplit tout son espace verticalement. Enfin, le bouton 3 occupe tout l'espace qui lui est alloué.

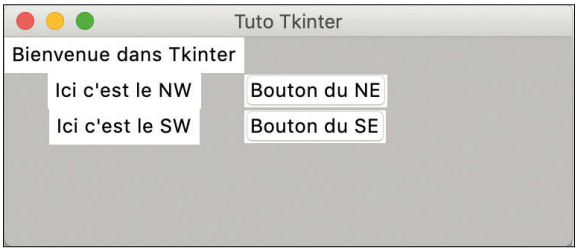
3.b Le gestionnaire grid

Ce gestionnaire est plus simple d'utilisation et plus flexible que *pack*. S'il ne fallait en apprendre qu'un pour commencer, ce serait sans conteste celui-ci. Il permet de définir une grille 2D régulière, dans laquelle on manipulera explicitement des indices de lignes et de colonnes, et où les cellules ont des tailles cohérentes les unes avec les autres. Dans chaque colonne la largeur est celle de la plus large cellule, et dans chaque ligne la hauteur est celle de la plus haute cellule (en fonction des *widgets* qui sont placés çà et là dans la grille).
L'utilisation de ce gestionnaire se fait simplement en appelant la méthode grid() sur chacun des *widgets* à placer. Démonstration :

```
from tkinter import Button, Label, Tk

root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Bienvenue dans Tkinter').grid(row=0, column=0)

Label(root, text='Ici c\'est le NW').grid(row=1, column=0)
Button(root, text='Bouton du NE').grid(row=1, column=1)
Label(root, text='Ici c\'est le SW').grid(row=2, column=0)
Button(root, text='Bouton du SE').grid(row=2, column=1)

root.mainloop()
```

Comme toujours en Python, les indices démarrent à 0. On passe explicitement les numéros de lignes et de colonnes via les argu-

ments nommés row et column. La grille ainsi obtenue n'est pas étirable par défaut :



On voit que la grille ne suit effectivement pas le mouvement. Il est possible de remédier à ça par des méthodes de configuration par ligne et par colonne, à savoir respectivement rowconfigure et columnconfigure appliquées directement à la fenêtre contenant les widgets et où chaque ligne et colonne est identifiée par son indice. Exemple :

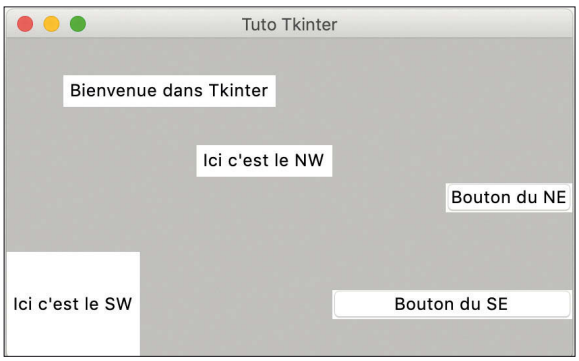
```
from tkinter import Button, Label, Tk

root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Bienvenue dans Tkinter').grid(row=0, column=0)

Label(root, text='Ici c\'est le NW').grid(row=1, column=0)
Button(root, text='Bouton du NE').grid(row=1, column=1)
Label(root, text='Ici c\'est le SW').grid(row=2, column=0)
Button(root, text='Bouton du SE').grid(row=2, column=1)

for i in range(3):
  root.rowconfigure(i, weight=1)
for j in range(2):
  root.columnconfigure(j, weight=1)

root.mainloop()
```

Il est nécessaire de préciser à grid comment répartir l'espace supplémentaire pour qu'il puisse redimensionner la grille en même temps que la fenêtre. Ici, nous constatons que la grille s'est bien étirée de manière homogène (c'est-à-dire en gardant les proportions initiales), car nous avons attribué des poids égaux à toutes les lignes et colonnes. Il est possible de définir des poids différents pour que certaines lignes et colonnes prennent moins ou davantage de place que d'autres.

Le redimensionnement rend visible le fait que les *widgets* sont placés par défaut aux centres des cellules. On peut modifier cette disposition à l'aide d'une option de grid appelée sticky, grâce à laquelle on peut préciser contre quel(s) bord(s) de cellule placer le *widget*.

Les bords sont repérés à l'aide de constantes (N, S, E, W, les points cardinaux, variables globales du module Tkinter), comme pour les valeurs du paramètre fill de pack. Voici un exemple :

```
from tkinter import Button, Label, Tk
from tkinter import E, N, S, W

root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Bienvenue dans Tkinter').grid(row=0, column=0)

Label(root, text='Ici c\'est le NW').grid(row=1, column=0, sticky=N+E)
Button(root, text='Bouton du NE').grid(row=1, column=1, sticky=E)
Label(root, text='Ici c\'est le SW').grid(row=2, column=0, sticky=S+W+N)
Button(root, text='Bouton du SE').grid(row=2, column=1, sticky=W+E)

for i in range(3):
  root.rowconfigure(i, weight=1)
for j in range(2):
  root.columnconfigure(j, weight=1)

root.mainloop()
```

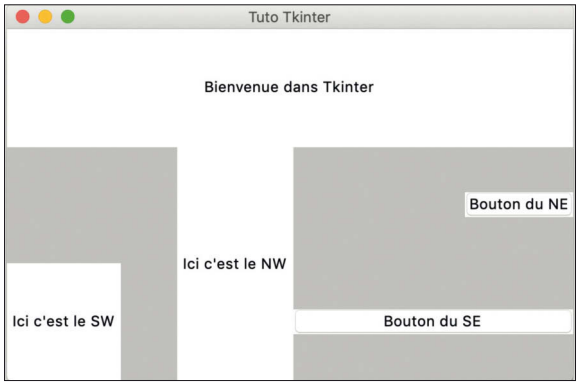


On constate bel et bien que non seulement les boutons sont déplacés vers les bords, mais aussi qu'on peut combiner les points cardinaux pour définir des coins de cellules, et que si deux points cardinaux opposés sont combinés, cela a pour effet d'étirer le *widget* dans la direction concernée. Pour information, ces points cardinaux peuvent aussi être assignés sous forme de chaînes de caractères : par exemple 'ne' équivaut à N+E, 'swe' à S+W+E, etc.

Notons également qu'il est possible de demander à un widget de recouvrir plusieurs lignes et/ou colonnes :

```
from tkinter import Button, Label, Tk
from tkinter import E, N, S, W

root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Bienvenue dans Tkinter').grid(row=0, column=0, columnspan=2, sticky='nsew')

Label(root, text='Ici c\'est le NW').grid(row=1, rowspan=2, column=0, sticky=N+E+S)
Button(root, text='Bouton du NE').grid(row=1, column=1, sticky=E)
```

```
Label(root, text='Ici c\'est le SW').grid(row=2, column=0, sticky=S+W+N)
Button(root, text='Bouton du SE').grid(row=2, column=1, sticky=W+E)

for i in range(3):
  root.rowconfigure(i, weight=1)
for j in range(2):
  root.columnconfigure(j, weight=1)

root.mainloop()
```



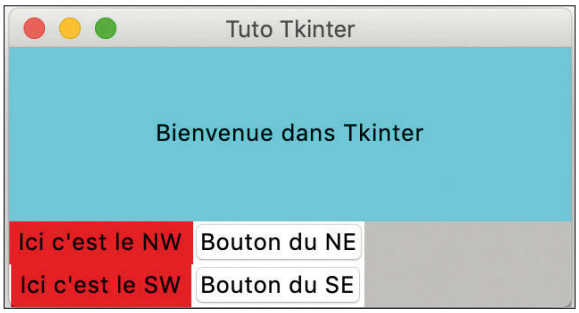
Voilà qui conclut notre tour d'horizon, bien sûr non-exhaustif, des gestionnaires de positionnement. Comme annoncé nous ne parlons pas de place, que nous n'utiliserons de toute façon pas par la suite, et il y a encore quelques options de configuration dont nous n'avons pas parlé, mais nous avons vu quasiment tout ce que nous utiliserons dans le Mastermind à venir et même plus.

4. Les conteneurs et nouvelles fenêtres

4.a Les conteneurs

Les conteneurs sont simplement des *widgets* invisibles mais dans lesquels nous pouvons agencer d'autres objets d'une manière totalement indépendante de la fenêtre dans laquelle ils s'insèrent. À tel point qu'on peut changer de gestionnaire de positionnement sans aucun problème. Illustration avec le plus simple d'entre eux, l'objet Frame :

```
from tkinter import Button, Frame, Label, Tk

root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Bienvenue dans Tkinter', bg='cyan').pack(fill='both', expand=True)

my_frame = Frame(root)
my_frame.pack(side='left')

Label(my_frame, text='Ici c\'est le NW', bg='red').grid(row=0, column=0)
Button(my_frame, text='Bouton du NE').grid(row=0, column=1)
Label(my_frame, text='Ici c\'est le SW', bg='red').grid(row=1, column=0)
Button(my_frame, text='Bouton du SE').grid(row=1, column=1)

root.mainloop()
```



Les quatre objets en bas à gauche de la fenêtre root appartiennent au Frame my_frame. Pour réaliser ceci, il suffit d'instancier la classe Frame et de passer l'instance en premier paramètre des *widgets* visés à la place de root. C'est l'isolation procurée par my_frame qui permet de passer de pack (utilisé pour le premier Label), à grid comme gestionnaire de positionnement.

Un exemple plus visible mais extrêmement proche est le *widget* LabelFrame qui permet de donner un titre à la zone occupée par le Frame :

```
from tkinter import Button, LabelFrame, Label, Tk

root = Tk()
root.title("Tuto Tkinter")
root.config(bg='silver')

Label(root, text='Bienvenue dans Tkinter', bg='cyan').pack(fill='both', expand=True)

my_frame = LabelFrame(root, text="Partie isolée")
my_frame.pack(side='left')

Label(my_frame, text='Ici c\'est le NW', bg='red').grid(row=0, column=0)
Button(my_frame, text='Bouton du NE').grid(row=0, column=1)
Label(my_frame, text='Ici c\'est le SW', bg='red').grid(row=1, column=0)
Button(my_frame, text='Bouton du SE').grid(row=1, column=1)

root.mainloop()
```

4.b Créer de nouvelles fenêtres

Une nouvelle fenêtre peut être générée à tout moment en créant une instance de la classe Toplevel. Cette fenêtre sera tout de même rattachée à la fenêtre principale. Si on ferme la fenêtre principale, toutes les autres se fermeront automatiquement. Exemple :

```
from tkinter import Label, Tk, Toplevel

root = Tk()
root.title("Tuto Tkinter")
```

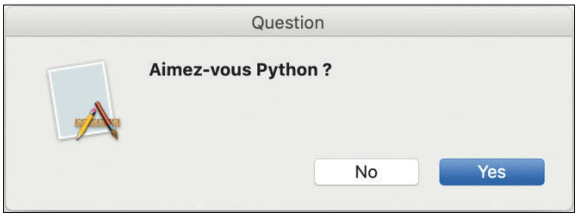
```
Label(root, text='Bienvenue dans Tkinter', bg='yellow').pack(fill='both', expand=True)

new_window = Toplevel(root)
new_window.title("Nouvelle fenêtre")
Label(new_window, text='Bienvenue dans la fenêtre secondaire', bg='green').pack(fill='both',
 expand=True)

root.mainloop()
```

Cette nouvelle fenêtre se comporte comme la fenêtre principale : elle pourra recevoir les mêmes *widgets*, avoir son propre gestionnaire de positionnement, définir des conteneurs pour isoler certaines parties, générer à son tour de nouvelles fenêtres, etc.

Il existe un certain nombre de fenêtres avec un style prédéfini, qu'on appellera volontiers des boîtes de dialogue, disponibles dans Tkinter sans passer par une quelconque extension. Elles sont situées dans des sous-modules de la bibliothèque. Montrons-en quelques-unes :

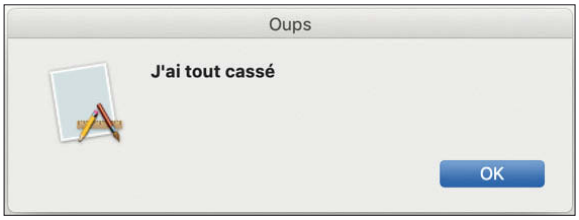
```
from tkinter import Label, Tk
from tkinter.messagebox import askquestion, askretrycancel, showerror, showinfo, showwarning
from tkinter.filedialog import askopenfile
from tkinter.colorchooser import askcolor

root = Tk()
root.title("Tuto Tkinter")
Label(root, text='Bienvenue dans Tkinter', bg='yellow').pack(fill='both', expand=True)

question = askquestion(title="Question", message="Aimez-vous Python ?")
retry_cancel = askretrycancel(title="RetryCancel", message="Une autre chance ?")

showerror(title="Oups", message="J'ai tout cassé")
showwarning(title="Attention", message="Ce message s'auto-détruira dans 2s")
showinfo(title="Scoop", message="La Terre aurait plutôt la forme d'une grosse patate.
 Coïncidence ?")

file = askopenfile()
print(file)
color = askcolor()
print(color)

root.mainloop()
```

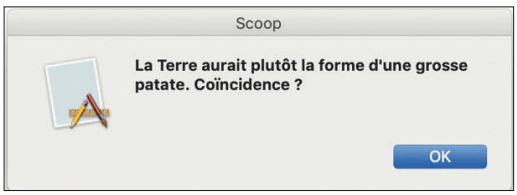
Ce code affiche successivement les différentes boîtes de dialogues, car elles ne rendent la main qu'une fois que nous les avons quittées. Ainsi, on verra successivement apparaître :



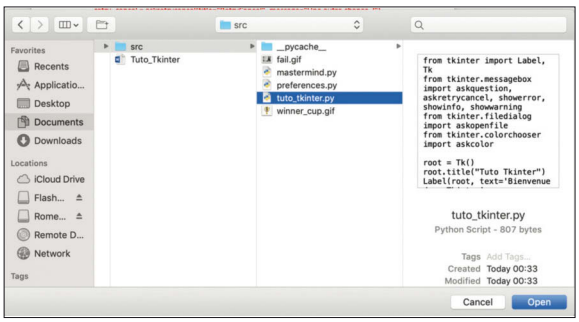
question vaut ensuite 'yes' ou 'no'.

retry_cancel vaut ensuite True ou False.





Ces 3 fenêtres ne retournent aucun résultat.



file est ensuite l'objet Python représentant le fichier ouvert, si on appuie sur « Open ».

color est ensuite un *tuple* (codes RGB, code couleur) si on appuie sur « OK ».

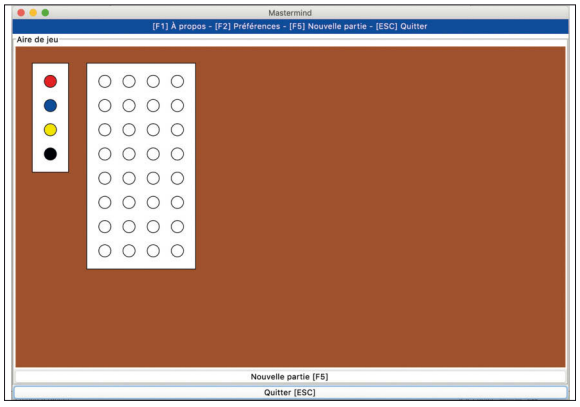
5. Codons le Mastermind !

Nous avons fait le tour des notions les plus essentielles pour commencer à programmer efficacement avec Tkinter. Il y a encore quelques notions ou *widgets* dont nous n'avons pas encore parlé et qui vont être utilisés, mais nous les découvrirons au fur et à mesure.

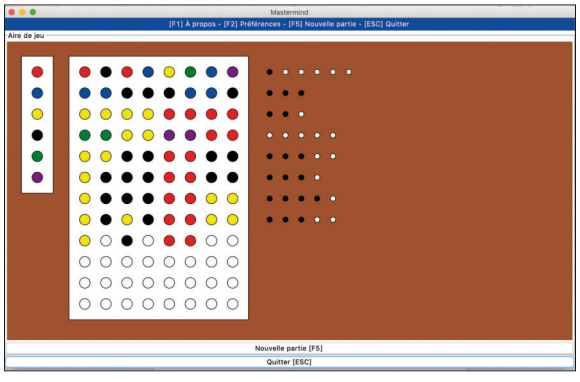
5.a Présentation du jeu

Pour introduire le sujet, rappelons que le Mastermind est un jeu qui consiste à trouver un code secret, ici composé de couleurs, en essayant successivement plusieurs combinaisons, chaque essai entraînant une évaluation qui montre le nombre de couleurs correctement placées (indiquées par des points noirs) et le nombre de couleurs présentes dans le code secret mais mal placées (indiquées par des points blancs).

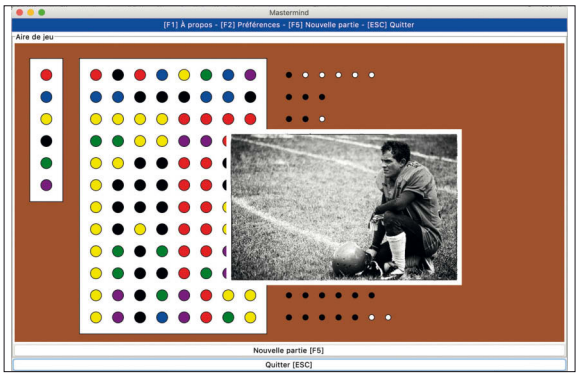
Voici ce à quoi ressemble le résultat du code que nous allons étudier :



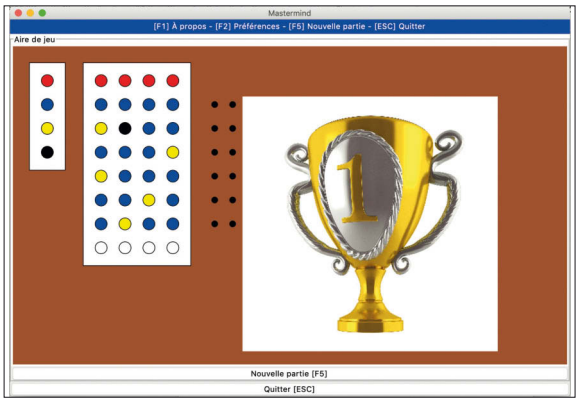
On pourra configurer le nombre d'essais maximum, le nombre de couleurs composant le code secret (avec potentiellement plusieurs fois la même couleur) et la taille de ce dernier. Voici un exemple avec une autre configuration et après quelques coups joués :



Fin de partie (perdue) :



Fin de partie (gagnée) :



Le *gameplay* est très simple : il faut commencer par sélectionner une couleur dans la barre de gauche en cliquant dessus. Une fois que c'est fait, la couleur est enregistrée et on peut colorer les cases de la zone de jeu principale, en cliquant dessus là aussi. On ne peut modifier que la dernière ligne incomplète, ni les suivantes ni les précédentes. L'évaluation d'une ligne se déclenche automatiquement dès que toutes ses cases sont remplies par une couleur. La ligne n'est plus modifiable par la suite.

5.b Étude du code principal

Le code se décompose en trois parties et autant de fichiers. Nous allons commencer par étudier le fichier principal, *mastermind.py*, assez court, qui sera l'occasion d'aborder brièvement deux notions que nous n'avons pas encore rencontrées : la création de menus et la définition de réactions à des signaux. Voici donc le contenu du fichier :

```
from tkinter import Tk
from tkinter import BOTH, N, S, X
from tkinter import Label, Menu
from tkinter.ttk import Button
from tkinter.messagebox import showinfo

from game_area import GameArea
from preferences import SettingsWindow


def about():
    """
    Display an informative window.
    """
```

```
    showinfo('À propos',
             message="Bienvenue dans cette demo de Mastermind avec Tkinter.\n\n"
                     "Ce jeu consiste à trouver un code secret composé de plusieurs couleurs, sachant que "
                     "chaque couleur peut apparaître plusieurs fois.\n\n"
                     "Vous pouvez configurer le nombre de couleurs différentes, la taille du code secret "
                     "et le nombre de tentatives que vous pouvez effectuer dans le menu Préférences.")


# Instantiate the global window:
root = Tk()
root.title('Mastermind')
root.resizable(True, True)


# Create canvas in which to draw the pickable pegs, the playing field and the guess results:
Label(root,
      text='[F1] À propos - [F2] Préférences - [F5] Nouvelle partie - [ESC] Quitter',
      foreground="white",
      background="blue").pack(anchor=N, fill=X)


# Create the game area:
game_area = GameArea(text="Aire de jeu")
game_area.pack(anchor=N, expand=True, fill=BOTH)


# Create and populate the main menu:
root_menu = Menu(root)
root['menu'] = root_menu
main_cascade = Menu(root_menu)
root_menu.add_cascade(label='Mastermind', menu=main_cascade)
main_cascade.add_command(label='Préférences', command=lambda: SettingsWindow(game_area))
main_cascade.add_separator()
main_cascade.add_command(label='À propos', command=about)


# Menu shortcuts:
root.bind("<F1>", lambda _event: about())
root.bind("<F2>", lambda _event: SettingsWindow(game_area))


# Add a last button for quitting the game:
Button(text='Quitter [ESC]', command=root.destroy).pack(anchor=S, fill=X)
root.bind('<Escape>', lambda _event: root.destroy())


# Launch the main loop that catches all user interactions:
root.mainloop()
```

Dans l'ordre, on trouve :

- Une fenêtre principale redimensionnable appelée root et intitulée *Mastermind*.
- Un Label bleu à texte blanc récapitulant les différents raccourcis créés (on y reviendra), étiré sur toute la largeur via pack pour donner l'aspect d'un bandeau d'en-tête. Il est ancré au bord Nord avec le paramètre anchor, une alternative à side dont nous n'avions pas parlé.
- L'aire de jeu principale (en marron) qu'on verra dans un fichier séparé, mais dont on peut déjà dire qu'il s'agit d'un gros LabelFrame étirable dans les deux directions.
- La création d'un menu composé de deux entrées (« Préférences » et « À propos », on y revient plus bas).
- La définition de raccourcis clavier pour certaines actions.
- L'ajout d'un bouton pour quitter le jeu en fermant la fenêtre.
- La boucle infinie mainloop(), comme toujours, qui permet à l'application de rester ouverte et de recevoir toutes les interactions de l'utilisateur.

Le bouton « Quitter » reçoit en argument la commande root.destroy. Cela signifie que dès que nous appuierons dessus, la fonction sera appelée sans paramètre (root.destroy()). Cette méthode destroy appliquée à tout *widget* permet de le supprimer définitivement. Appliquée à la fenêtre principale sans laquelle rien n'existe, elle a donc pour effet de fermer l'application.

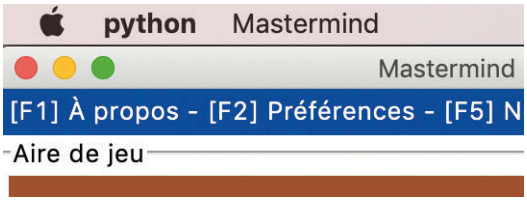
On peut également utiliser sur tous les *widgets* la méthode bind pour associer une action utilisateur à une fonction. Le premier argument doit être le code identifiant l'action, le second la fonction à appeler. Cette dernière sera automatiquement appelée avec comme argument unique un objet de type Event qui contiendra plusieurs informations sur l'évènement qui s'est produit. Par exemple, si my_event est cet objet de type Event, il contiendra entre autres :

- my_event.x et my_event.y qui sont les coordonnées du pointeur de la souris ;
- my_event.widget qui est une référence vers l'objet sur lequel bind a été appliqué (on peut donc l'utiliser pour lui appliquer de nouvelles méthodes) ;
- et encore d'autres informations…

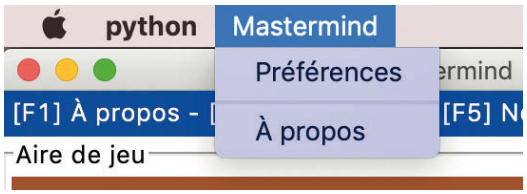
Dans le cas présent, on associe à la clé '<Escape>', qui représente la touche « esc » ou « Echap » du clavier, une fonction lambda qui ne fait pas usage des informations de l'évènement qu'elle reçoit mais appelle plutôt root.destroy(). On a ainsi défini un raccourci clavier qui permet d'obtenir via cette touche le même résultat que quand on appuie sur le bouton.

Nous avons réutilisé ce principe pour définir deux autres raccourcis clavier à partir des touches F1 et F2. Elles permettent d'accéder à des contenus qui sont normalement accessibles via le menu (que nous allons présenter ensuite). La touche F1 appelle la fonction about() qui fait apparaître une boîte de dialogue contenant du texte :

La touche F2 quant à elle, permet d'accéder à une fenêtre contenant toutes les options de configuration du jeu, que nous présenterons par la suite. Enfin, nous avons créé un menu, ce dont nous n'avions pas encore parlé dans ce tutoriel. Il existe plusieurs types de menus mais nous ne présenterons ici que celui que nous utilisons, à savoir une instance de la classe Menu. Cette instance root_menu est rattachée, comme tous les widgets, à un conteneur ou une fenêtre (ici la principale) qui est le premier argument passé lors de son instanciation. Cette première instance sert à créer une « barre » de menu. Nous créons ensuite un sous-menu, appelé main_cascade, intitulé Mastermind et créé à partir de la méthode add_cascade appliquée au menu principal. « cascade » est le nom du type de l'entrée créée dans le menu, ici un sous-menu (on peut rajouter plusieurs types d'entrée dans un menu). Dès ce moment, nous pouvons voir ceci :



Le rendu étant spécifique à un Macbook Pro bien sûr. C'est donc la première entrée du menu, et nous pourrions en rajouter d'autres horizontalement mais nous n'en aurons pas besoin. Le type « cascade » de ce sous-menu est synonyme de « menu déroulant ». Nous allons le voir tout de suite en rajoutant deux entrées de type « commande » via la méthode add_command appliquée à main_cascade. Ces deux entrées reçoivent un titre et une action. La première, « Préférences », fera apparaître la fenêtre de configuration du jeu (comme la touche F2), tandis que la seconde appellera la fonction about() (comme F1). Résultat :



Parlons maintenant de cette fenêtre de configuration accessible via ce menu ou via F2.

5.c Le code de la fenêtre de configuration

Voici d'abord à quoi elle ressemble :

Cet agencement est obtenu cette fois via le gestionnaire de positionnement grid. Voici le code qui est contenu dans le fichier *preferences.py* :

```
from tkinter import Button, IntVar, Label, Radiobutton, Spinbox, Toplevel
from tkinter.ttk import Combobox


ALL_COLORS = ['red', 'blue', 'yellow', 'black', 'green', 'purple', 'orange', 'cyan']


SETTINGS = {
  'n_colors': 4,
  'n_tries': 8,
  'code_size': 4
}


class SettingsWindow(Toplevel):

  def __init__(self, game_area):
    super().__init__()
    self.bg_color = 'wheat'
    self.config(bg=self.bg_color)
    self.title("Préférences")
    self.resizable(False, False)
    self.game_area = game_area

    paddings = {'padx': (12, 12), 'pady': (12, 12)}

    Label(self, text="Nombre de couleurs en jeu :", bg=self.bg_color) \
      .grid(row=0, column=0, columnspan=2, sticky='nws', **paddings)
    self.n_colors_box = Spinbox(self, from_=4, to=8)
    self.n_colors_box.grid(row=0, column=2, columnspan=2, sticky='nes', **paddings)

    Label(self, text="Nombre maximum d'essais :", bg=self.bg_color) \
      .grid(row=1, column=0, sticky='nws', **paddings)
    self.n_tries_box = Combobox(self, values=[8, 10, 12, 14])
    self.n_tries_box.grid(row=1, column=2, columnspan=2, sticky='nes', **paddings)
    self.n_tries_box.current(0)

    Label(self, text="Taille du code à trouver :", bg=self.bg_color) \
      .grid(row=2, column=0, columnspan=2, sticky='nws', **paddings)
    self.code_size_var = IntVar(value=SETTINGS['code_size'])
    Radiobutton(self, text="Facile (4)", variable=self.code_size_var, value=4, bg=self.bg_color) \
      .grid(row=2, column=1, **paddings)
    Radiobutton(self, text="Moyen (6)", variable=self.code_size_var, value=6, bg=self.bg_color) \
      .grid(row=2, column=2, **paddings)
    Radiobutton(self, text="Difficile (8)", variable=self.code_size_var, value=8, bg=self.bg_color) \
      .grid(row=2, column=3, **paddings)

    Button(self, text="Annuler", command=self.destroy, bg=self.bg_color) \
      .grid(row=3, column=0, columnspan=2, sticky='nesw', **paddings)
    Button(self, text="Appliquer", command=self.apply, bg=self.bg_color) \
      .grid(row=3, column=2, columnspan=2, sticky='nesw', **paddings)

    self.bind("<Escape>", lambda _event: self.destroy())

  def apply(self):
    global SETTINGS
    SETTINGS['n_colors'] = int(self.n_colors_box.get())
```

```
SETTINGS['n_tries'] = int(self.n_tries_box.get())
SETTINGS['code_size'] = int(self.code_size_var.get())
self.game_area.new_game()
self.destroy()
```

Ce code définit des variables globales qui seront importables lorsque nous coderons le cœur du jeu : ALL_COLORS et SETTINGS. Nous pourrons ainsi à tout moment manipuler les préférences et les couleurs en tenant compte des modifications qui auront été sélectionnées. Mais le plus intéressant est la fenêtre elle-même.
Cette fenêtre est une sous-classe de TopLevel dont nous avons déjà parlé. Autrement dit, nous définissions notre propre type de fenêtre autonome. Sa particularité sera de contenir déjà, à la création, tous les *widgets* dont nous avons besoin. Pour ce faire, le code qui aurait pu être écrit en-dehors de la classe, est écrit dans la méthode __init__ de celle-ci. Le premier argument de chaque widget n'est pas une instance de TopLevel mais self. C'est une façon de programmer en objet tout à fait classique avec Tkinter.
Le reste est aussi très classique par rapport à tout ce que nous avons déjà vu :

- La fenêtre (self) est déclarée non redimensionable (resizable), sa couleur de fond est 'wheat' (couleur « blé ») et son titre « Préférences » ;
- À la création, on reçoit une référence à l'objet contenant l'aire de jeu (game_area) et on la sauvegarde (on verra plus bas pourquoi) ;
- Le nombre de colonnes total est contraint par la ligne 2, avec le nombre de Radiobuttons (3) et le Label, ce qui fait que la grille a 4 colonnes ;
- Les deux premières lignes sont très classiques : un Label sur les 2 premières colonnes et un *widget* étendu sur les 2 suivantes. Une Spinbox et une Combobox respectivement ;
- La Spinbox permet de sélectionner les 4 à 8 premières couleurs de la liste ALL_COLORS avec lesquelles nous pourrons jouer, tandis que la Combobox permet de choisir 8, 10, 12 ou 14 tentatives maximum ;
- La troisième ligne suit le même principe mais le Label n'occupe qu'une colonne puisque les 3 suivantes sont réservées aux Radiobuttons, qui permettent de choisir le nombre de couleurs qui composeront le code secret à trouver (4, 6 ou 8) via une variable partagée de type IntVar, initialisée à la valeur courante de SETTINGS['code_size'] (celle utilisée pour la dernière partie jouée, et qui vaut 4 au démarrage de l'application) ;
- Au niveau esthétique, le paramètre sticky nous permet d'aligner tous les Labels à gauche de leurs cellules avec le 'w' de 'nws' et les autres widgets à droite via le 'e' de 'nes'. Les nouveaux paramètres de *padding* padx et pady n'apportent pas grand-chose visuellement (définitions de marges) mais améliorent un tout petit peu le rendu. Les Labels ont pour couleur de fond la même que la fenêtre elle-même, de telle façon qu'on ne distingue pas leurs bords ;
- La quatrième et dernière ligne contient deux boutons, « Annuler » et « Appliquer ». Le premier appelle self.destroy() et détruit donc la fenêtre Préférences, tandis que le second enregistre les choix effectués par l'utilisateur et modifie en conséquence les entrées du dictionnaire global SETTINGS (avant de détruire la fenêtre à son tour). Cette dernière action est codée dans une méthode spécifique de la fenêtre, apply. Néanmoins cette action ne doit pas influencer une éventuelle partie en cours, raison pour laquelle on considère que quiconque modifie les règles souhaite démarrer une nouvelle partie. Cette action est effectuée via une méthode spécifique au widget contrôlant l'aire de jeu, game_area. C'est pour cette unique raison que nous avions besoin de recevoir cette variable à la création de la fenêtre ;
- Enfin, mais c'est un détail, nous créons comme pour la fenêtre principale un raccourci clavier pour quitter la fenêtre (identique au bouton « Annuler ») quand on appuie sur la touche « esc ».

Passons à présent au plus complexe, l'aire de jeu.

5.d Le code de l'aire de jeu

Ce code est plus long et complexe, mais nous connaissons l'essentiel des concepts qui y sont manipulés hormis un nouveau *widget* central : le Canvas. C'est un conteneur qui nous permet principalement de dessiner des formes, un peu en mode « bac à sable » mais avec beaucoup de fonctionnalités très pratiques.
Nous allons procéder par morceaux plutôt que donner tout le code en un seul bloc. Voici le début du fichier *game_area.py* :

```
from random import randrange

from tkinter import BOTH, S, X
from tkinter import Canvas, Label, LabelFrame, PhotoImage
from tkinter.ttk import Button

from preferences import ALL_COLORS, SETTINGS


class GameArea(LabelFrame):

    EXTERNAL_OFFSET = 30
    OFFSET_X = 20
    OFFSET_Y = 20
    DIAMETER = 20
    SMALL_DIAMETER = 10

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.secret_code = None
        self.main_cv = None
        self.new_game_button = None
        self.active_row = 0
        self.selected_color = None
        self.victory_image = None
        self.failure_image = None
        self.new_game()

    def new_game(self):
        if self.main_cv is not None:
            self.main_cv.destroy()
        if self.new_game_button is not None:
            self.new_game_button.destroy()
        self.active_row = 0
        self.generate_fields()
```

```
    self.secret_code = self.make_secret()
    self.set_gameplay()
```

Plusieurs petites choses à dire sur ce début de code :

- L'aire de jeu est un conteneur de type LabelFrame inséré dans la fenêtre principale. Comme pour la fenêtre Préférences, cet objet est défini comme un nouveau type de conteneur à part entière héritant du conteneur LabelFrame. Tout le code qu'il contient est appelé en dernière analyse depuis sa méthode __init__ ;
- On définit comme constantes de classe des variables qui définissent les espacements entre les différents composants/dessins de notre aire de jeu ;
- Dans la méthode __init__, on ne fait qu'initialiser des attributs qui seront utilisés par la suite, et déléguer les actions à mener initialement à une autre méthode, self.new_game() ;
- Cette méthode new_game, qui est celle appelée depuis la fenêtre Préférences quand on applique des changements de règles (pour rappel), commence par détruire les *widgets* de la partie précédente s'il y en a eu une (c'est-à-dire si les variables qui les référencent n'ont pas leur valeur initiale, None). On remet à zéro le compteur qui indique quelle est la ligne courante que nous avons le droit de remplir. On génère ensuite les champs qui composent visuellement l'aire de jeu (generate_fields), on génère un code secret aléatoirement (make_secret) et on définit les actions à mener en réaction aux clics de l'utilisateur (set_gameplay).

Nous allons présenter ces différentes méthodes appelées par new_game dans leur ordre d'appel, qui est aussi l'ordre du fichier. generate_fields est la plus difficile à concevoir.

```
def generate_fields(self):
    n_colors = SETTINGS['n_colors']
    code_size = SETTINGS['code_size']
    n_tries = SETTINGS['n_tries']
    colors = ALL_COLORS[:n_colors]

    # Create the canvas in which the game takes place:
    self.main_cv = Canvas(self, bg="sienna", cursor="hand")
    self.main_cv.pack(expand=True, fill=BOTH)

    # Draw the field of choices, a white rectangle with the pegs of all colors that can be picked:
    band_width = self.OFFSET_X+self.DIAMETER
    self.main_cv.create_rectangle(self.EXTERNAL_OFFSET, self.EXTERNAL_OFFSET,
                                  self.EXTERNAL_OFFSET+self.OFFSET_X+band_width,
                                  self.EXTERNAL_OFFSET+self.OFFSET_Y+n_colors*(self.DIAMETER+self.OFFSET_Y),
                                  fill="white")
    offsets = (self.EXTERNAL_OFFSET+self.OFFSET_X, self.EXTERNAL_OFFSET+self.OFFSET_Y,
               self.EXTERNAL_OFFSET+self.OFFSET_X+self.DIAMETER, self.EXTERNAL_OFFSET+self.OFFSET_Y+self.DIAMETER)
    for color in colors:
        self.main_cv.create_oval(*offsets, fill=color, tags=color+'_choice')
        offsets = (offsets[0], offsets[1]+self.OFFSET_Y+self.DIAMETER,
                   offsets[2], offsets[3]+self.OFFSET_Y+self.DIAMETER)

    # Draw the field of guesses, a white rectangle with initially empty (white) slots:
    left_offset = 2*self.EXTERNAL_OFFSET + band_width + self.OFFSET_X
    self.main_cv.create_rectangle(left_offset, self.EXTERNAL_OFFSET,
                                  left_offset+code_size*band_width+self.OFFSET_X,
                                  self.EXTERNAL_OFFSET+self.OFFSET_Y+n_tries*(self.DIAMETER+self.OFFSET_Y),
                                  fill="white")
    for j in range(code_size):
        offsets = (left_offset + self.OFFSET_X + j*(self.DIAMETER+self.OFFSET_X),
                   self.EXTERNAL_OFFSET + self.OFFSET_Y,
                   left_offset + (j+1)*(self.DIAMETER+self.OFFSET_X),
                   self.EXTERNAL_OFFSET + self.OFFSET_Y + self.DIAMETER)
        for i in range(n_tries):
            self.main_cv.create_oval(*offsets, fill='white', tags='_'.join([str(i), str(j), 'guess']))
            offsets = (offsets[0], offsets[1] + self.OFFSET_Y + self.DIAMETER, offsets[2],
                       offsets[3] + self.OFFSET_Y + self.DIAMETER)

    # Restart:
    self.new_game_button = Button(self, text='Nouvelle partie [F5]', command=self.new_game)
    self.new_game_button.pack(anchor=S, fill=X)
    self.master.bind("<F5>", lambda _x: self.new_game())
```

Afin de simplifier les explications, nous ne nous attarderons pas sur les calculs qui permettent d'obtenir les bons espacements réguliers horizontalement et verticalement. C'est la partie la plus fastidieuse de la conception et le but de ce tutoriel est avant tout de comprendre le fonctionnement de Tkinter.

La méthode commence par instancier un objet de type Canvas sauvegardé dans l'attribut self.main_cv. Sa couleur de fond est une déclinaison du marron (« sienna »), le curseur de la souris prend la forme d'une main (cursor='hand') quand la souris passe dessus et il s'étend dans les deux directions pour occuper toute la place disponible.

Ensuite on dessine un premier rectangle via la méthode create_rectangle du Canvas. Ce qu'il faut bien comprendre, c'est le sens des 4 premiers arguments pour positionner le rectangle. Toutes les autres formes dessinées prendront les mêmes 4 premiers arguments et le sens sera le même. Ils forment ce qu'on appelle la « *boundary box* » (*bbox* dans la documentation de Tkinter). Les 2 premiers sont les coordonnées (x, y) du coin supérieur gauche de la *bbox*, tandis que les 2 suivants sont celles de son coin inférieur droit, de sorte à former une diagonale si on les relie. Même quand on dessine des formes circulaires ou quelconques, on raisonne à partir de la diagonale du plus petit rectangle qui puisse contenir le dessin. Le paramètre fill permet de préciser la couleur de fond du dessin effectué, ici blanc (fill='white'). Les calculs de coordonnées se basent nécessairement sur les entrées du dictionnaire SETTINGS définies dans *preferences.py* pour s'adapter automatiquement à toutes les configurations.

Une fois ce premier rectangle correctement positionné on crée, avec des coordonnées régulièrement espacées et incluses dans le rectangle, n_colors cercles de diamètre self.DIAMETER (largeur et hauteur de leurs propres *bbox*), via la méthode self.main_cv.create_oval. On remplit les nouveaux cercles avec chacune des n_colors premières couleurs de ALL_COLORS.

IMPORTANT : on assigne à chaque cercle un *tag* (fonctionnalité permise par le canvas) qui nous permettra plus tard de l'identifier. Les *tags* des cercles de cet espace sont de la forme : tag=color+'_choice'.

On répète l'opération pour le second rectangle, dont les dimen-

sions sont un peu plus complexes à gérer, et que l'on remplit de code_size x n_tries cercles initialement blancs (le blanc est interprété comme la couleur du vide), tous de la même taille (self.DIAMETER) que ceux de la palette de couleurs. Il nous sera nécessaire pour la suite d'avoir là encore des *tags* pour identifier chaque cercle. Ils sont ici de la forme : « i_j_guess » où i et j désignent les indices de ligne et de colonne de chaque cercle.
Tout en bas du *canvas*, on rajoute un bouton lié à l'aire de jeu (self) qui permet de recommencer une nouvelle partie en appelant la méthode new_game de self. On crée aussi un *binding* avec la touche F5 pour réaliser la même opération, qu'on lie non pas à l'aire de jeu cette fois, mais à la fenêtre parente (autrement dit la fenêtre principale ici). Il est d'ailleurs intéressant de noter qu'on peut toujours atteindre le conteneur parent d'un *widget* via son attribut master. Cette liaison à la fenêtre principale permet de commencer une nouvelle partie même lorsque le *focus* n'est pas sur l'aire de jeu. Maintenant que les zones de jeu sont dessinées, passons à la génération du code secret. Cette partie est extrêmement simple en Python :

```
@staticmethod
  def make_secret():
    return [ALL_COLORS[randrange(0, SETTINGS['n_colors'])] for _ in range(SETTINGS['code_size'])]
```

Pour chaque élément du code secret, on génère via la fonction randrange (importée du module random de Python) un nombre aléatoire compris entre 0 et SETTINGS['n_colors'] qui sert d'indice pour sélectionner une des couleurs de la liste ALL_COLORS.
Vient enfin la dernière méthode, set_gameplay, mais qui en appellera de nouvelles. Le tout forme la partie algorithmique du jeu.

```
def set_gameplay(self):

  def interpret_click(event):
    selected_item = self.main_cv.find_closest(event.x, event.y)
    try:
      selected_tag, _ = self.main_cv.gettags(selected_item)
    except ValueError:
      return

    # The tags of the pickable colors are of the form "<color>_choice":
    if 'choice' in selected_tag:
      self.selected_color = selected_tag.split('_')[0]

    # The tags of the settable slots are of the form "<row_index>_<column_index>_guess":
    elif 'guess' in selected_tag:
      selected_row = int(selected_tag.split('_')[0])
      if selected_row == self.active_row and self.selected_color is not None:
        self.main_cv.itemconfig(selected_item, fill=self.selected_color)
        # Detect if the row is fully filled:
        all_row_items = [self.main_cv.find_withtag('_'.join([str(selected_row), str(j), 'guess']))
                         for j in range(SETTINGS['code_size'])]
        all_row_colors = [self.main_cv.itemcget(item, 'fill') for item in all_row_items]
        if 'white' not in all_row_colors:
          all_scores = self.compute_scores(all_row_colors)
          self.draw_scores(*all_scores)
          self.active_row += 1
          if all_scores[0] == SETTINGS['code_size']:
            self.make_victory()
          elif self.active_row == SETTINGS['n_tries']:
            self.make_failure()

  self.main_cv.bind('<Button-1>', interpret_click)
```

Le *gameplay* pourrait se résumer à cette dernière ligne : appeler la fonction interpret_click pour chaque clic de l'utilisateur. Sauf que cette fonction est complexe et peut appeler elle-même d'autres méthodes. Au passage, la clé identifiant un clic de souris est '<Button-1>'.
interpret_click est une fonction liée à un évènement, donc elle reçoit automatiquement un objet event de type Event comme seul argument. Cet objet possède en attributs les coordonnées (x, y) où le clic a eu lieu. À partir de là l'objet Canvas permet, grâce à sa méthode find_closest, de trouver le dessin le plus proche (celui sur lequel on a donc cliqué ou voulu cliquer). La méthode renvoie un « item », une sorte de numérotation interne au *canvas*, à partir duquel on peut retrouver le *tag* associé via la méthode gettags (qui renvoie une paire dont le premier élément est le *tag* que nous avons assigné). On connaît alors selected_tag, le *tag* du cercle sur lequel l'utilisateur a cliqué.
Si ce *tag* contient la chaîne de caractères 'choice', on sait qu'il correspond à un cercle de choix de couleur (cercles du rectangle à la gauche du *canvas*). Or d'après notre convention de nommage des *tags*, on sait que la couleur est dans la première partie du *tag* (par exemple red_choice correspond au choix de la couleur rouge). On enregistre donc cette couleur dans l'attribut self.selected_color.
En revanche, si le *tag* contient la chaîne de caractères 'guess', cela signifie que l'utilisateur a cliqué sur un cercle de la zone de jeu et souhaite donc « remplir » ce cercle avec la couleur de son choix. Cette couleur de son choix doit être préalablement renseignée dans self.selected_color (autrement dit il a dû cliquer auparavant au moins une fois sur un des cercles de choix de couleur). On commence donc par détecter, sachant que le *tag* est de la forme « i_j_guess », l'indice i de la ligne courante. Si cet indice n'est pas égal à celui de la ligne que l'utilisateur a le droit de remplir (pour rappel, self.active_row), on ne fait rien. Autrement, on modifie la configuration du cercle pour que sa couleur de remplissage soit celle qui est enregistrée (fill=self.selected_color). Pour cela, le *canvas* nous met à disposition une méthode itemconfig qui prend en premier argument l' « item » à partir duquel nous avions récupéré le *tag* de l'objet cliqué. Après cela, nous souhaitons déterminer si cette dernière action termine le remplissage de toute la ligne self.active_row. Pour cela, on procède en trois temps :

- Grâce à la méthode find_withtag du *canvas*, on récupère tous les « items » des cercles qui ont un tag de la forme « i_j_guess » où i == self.active_row ;
- Grâce à la méthode itemcget du *canvas*, on récupère pour chacun de ces items la valeur du paramètre fill du cercle correspondant ;
- Si aucune de ces valeurs ne vaut 'white' (le blanc représentant le vide pour nous), cela signifie que la ligne a été complètement remplie.

Si tel est le cas, une autre phase s'enclenche : il faut calculer les scores, les représenter, et en fonction des résultats, décréter la victoire ou, si la ligne active était la dernière ligne autorisée, décréter la défaite. On n'oubliera pas non plus d'incrémenter l'indice de la ligne active (autrement dit d'activer la possibilité de remplir la ligne

suivante). Ces quatre actions (calcul, représentation, victoire, défaite) font l'objet de méthodes séparées. Commençons par le calcul des résultats avec la méthode compute_scores :

```
def compute_scores(self, row_colors):
    """
    Compute the scores.

    The "badly placed" score should take into account duplicates and reflect their occurences in the secrect code.
    """
    badly_placed_colors = {color: 0 for color in row_colors}
    for color in row_colors:
        n_occurences_in_secret = self.secret_code.count(color)
        if badly_placed_colors[color] < n_occurences_in_secret:
            badly_placed_colors[color] += 1

    exact_matches = {color: 0 for color in row_colors}
    for color, secret_color in zip(row_colors, self.secret_code):
        if color == secret_color:
            exact_matches[color] += 1
            badly_placed_colors[color] -= 1

    n_badly_placed = sum(badly_placed_colors.values())
    n_exact_matches = sum(exact_matches.values())

    return n_exact_matches, n_badly_placed
```

Il est nécessaire de commencer par évaluer indistinctement le nombre de couleurs bien ou mal placées mais en tout cas présentes dans le code secret. On comptera les résultats par couleur dans le compteur des couleurs mal placées. Pour chacune de celles de la réponse de l'utilisateur, on incrémente le compteur si la couleur est dans le code secret et si elle n'est pas déjà apparue autant de fois que dans le code secret.
Ensuite, pour chacun des scores par couleur, on retranche 1 point au compteur des couleurs mal placées et on augmente d'1 point le compteur des couleurs bien placées. Au final, on obtient bien des scores « exact_match » et « badly_placed » reflétant conjointement le nombre d'occurrences des couleurs dans le code secret. Il faut ensuite représenter ces scores. C'est le rôle de la méthode draw_scores :

```
def draw_scores(self, n_exact, n_badly_placed):
    code_size = SETTINGS['code_size']
    diameters_delta = self.DIAMETER - self.SMALL_DIAMETER
    left_offset = 3*self.EXTERNAL_OFFSET + 2*self.OFFSET_X + (code_size+1)*(self.OFFSET_X+self.DIAMETER)
    row_offset = self.EXTERNAL_OFFSET + self.OFFSET_Y + self.active_row*(self.OFFSET_Y+self.DIAMETER)
    offsets = (left_offset+0.5*diameters_delta, row_offset+0.5*diameters_delta,
               left_offset+0.5*diameters_delta+self.SMALL_DIAMETER,
               row_offset+0.5*diameters_delta+self.SMALL_DIAMETER)
    for _ in range(n_exact):
        self.main_cv.create_oval(offsets[0], offsets[1],
                                 offsets[2], offsets[3],
                                 fill='black')
        offsets = (offsets[0]+self.SMALL_DIAMETER+self.OFFSET_X, offsets[1],
                   offsets[2]+self.SMALL_DIAMETER+self.OFFSET_X, offsets[3])
    for _ in range(n_badly_placed):
        self.main_cv.create_oval(offsets[0], offsets[1],
                                 offsets[2], offsets[3],
                                 fill='white')
        offsets = (offsets[0]+self.SMALL_DIAMETER+self.OFFSET_X, offsets[1],
                   offsets[2]+self.SMALL_DIAMETER+self.OFFSET_X, offsets[3])
```

Comme pour generate_fields, toute la complexité réside dans le calcul des décalages entre les cercles, mais d'un point de vue technique il n'y a rien de nouveau. On crée des cercles à l'aide de la méthode create_oval du *canvas*, d'un diamètre plus petit contrôlé par la constante self.SMALL_DIAMETER. On commence par créer autant de cercles que de couleurs « bien placées », en les remplissant en noir, puis on crée autant de cercles remplis de blanc que de couleurs « trouvées mais mal placées ».
Il ne reste plus qu'à définir les actions en cas de victoire et de défaite, que nous allons présenter en même temps :

```
def make_victory(self):
    self.active_row = SETTINGS['n_tries']
    self.victory_image = PhotoImage(file='winner_cup.gif').subsample(3)
    left_offset = 20 * self.EXTERNAL_OFFSET
    row_offset = 10 * self.EXTERNAL_OFFSET
    self.main_cv.create_image(left_offset, row_offset, image=self.victory_image)

def make_failure(self):
    self.failure_image = PhotoImage(file='fail.gif').subsample(3)
    left_offset = 20 * self.EXTERNAL_OFFSET
    row_offset = 10 * self.EXTERNAL_OFFSET
    self.main_cv.create_image(left_offset, row_offset, image=self.failure_image)
```

Dans les deux cas, les actions menées sont similaires :

- En cas de victoire seulement, placer la ligne active juste après la dernière ligne dessinée (ce qui donne l'impression de figer la zone de jeu, puisqu'on ne peut plus remplir aucun cercle), cela étant déjà le cas lors d'une défaite ;
- Charger l'image appropriée grâce à la classe PhotoImage de Tkinter, qui prend en argument un chemin vers un fichier et n'accepte que les images au format GIF ou PPM/PGM ;
- Éventuellement, rétrécir l'image via la méthode subsample de la classe PhotoImage ou l'agrandir via sa méthode zoom ;
- L'afficher par-dessus la zone de jeu à une position (x, y) arbitraire grâce à la méthode du *canvas* create_image qui prend comme arguments x, y et l'image à afficher.

Et voilà qui termine notre implémentation du Mastermind !

6. Conclusion

Tkinter est une bibliothèque d'interfaces graphiques assez riche pour nous permettre de réaliser des applications bien plus évoluées que ce Mastermind. Elle est facile à prendre en main, on peut y faire de la programmation orientée objet comme on en a l'habitude en Python (son *design* est très « pythonique ») et elle est incluse par défaut dans l'installation standard de Python. Le rendu visuel n'est cependant pas à la hauteur de ce qui se fait de mieux ces dernières années (PyQt, wxPython, Kivy, …), mais le développement est souvent plus rapide. Son adoption dans un projet tiendra donc comme souvent en un arbitrage entre ces deux paramètres : haute qualité esthétique et rapidité de développement. •

Mehdi Rahim
Data Scientist
Margo Group

Une application mobile intelligente avec Azure Machine Learning Studio et Xamarin

Intelligence Artificielle : du prototype au déploiement

La contribution de l'IA dans notre quotidien ne cesse de croître à travers les diverses applications rendues accessibles, comme la reconnaissance vocale, la détection de visages sur des images, les chatbots, etc. Ces technologies utilisent des algorithmes de Machine Learning pour établir des modèles prédictifs spécifiques aux besoins.

niveau 200

Le déploiement de l'IA sous forme d'application web ou mobile est une phase clé afin de mieux tirer profit de son apport. Nous parlons d'une API (service web) qui sera interrogée à chaque nouvelle donnée (nouvelle commande vocale, nouvelle image à identifier). Dans cet article, nous vous présentons un cas concret de construction d'un modèle d'IA et de son déploiement via une application mobile, le tout en moins d'une heure chrono !

AzureML Studio pour le modèle, Xamarin pour le déploiement

Il existe de nombreuses librairies de Machine Learning, qu'elles soient open-source ou payantes, sous forme d'API ou de logiciels. Dans le cas présent, nous utiliserons Azure Machine Learning Studio : c'est un outil permettant de créer un modèle de Machine Learning depuis un navigateur web via une interface graphique, il n'est donc pas nécessaire d'écrire une ligne de code !

AzureML studio contient les étapes essentielles du Machine Learning comme la sélection des données, leur nettoyage, l'apprentissage et la validation.

Xamarin est une suite d'outils de développement en C# pour des applications mobiles natives (Android, iOS, Windows). Il permet de développer à la fois des interfaces graphiques communes ou spécifiques à chaque système d'exploitation.

Le lien entre AzureML et Xamarin se fait grâce à une fonctionnalité particulièrement intéressante de AzureML : il s'agit de la possibilité de déployer un modèle de Machine Learning sous la forme d'un service web. Ainsi, AzureML fournit les templates des classes en C# prêtes à être utilisées dans Xamarin.

Exemple concret : identifier des arbres !

Nous présentons un exemple de Machine Learning plutôt ludique : nous souhaitons prédire l'année de plantation d'un arbre à Paris depuis ses caractéristiques (taille, circonférence et diamètre). Pour cela, nous utilisons des données issues du portail open-data de la ville de Paris (https://opendata.paris.fr/), qui recense tous les arbres parisiens. Nous décrirons dans ce qui suit les étapes pour reproduire cet exemple. A

Construction du modèle avec Azure ML

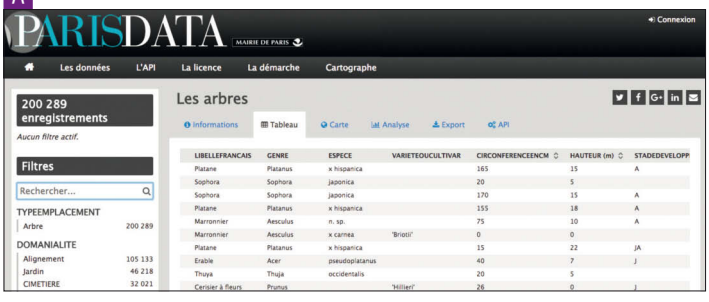
L'utilisation de AzureML se fait via l'adresse https://studio.azureml.net. Une fois connecté, l'outil propose de créer des projets contenant l'expérience de machine learning. Dans notre cas, nous souhaitons prédire l'année de plantation d'un arbre.

L'expérience se compose d'un ensemble de traitements représentés par des briques. Chaque brique applique une action de transformation sur les données, les connexions représentent l'ordonnancement des briques et donc des transformations.

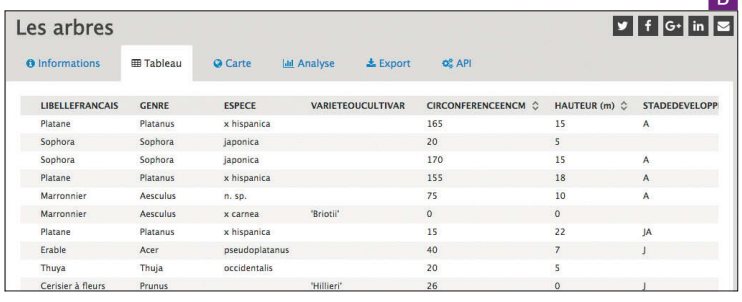
Nous ajoutons les différentes briques pour préparer les données et construire le modèle prédictif. Ainsi, l'expérience contient dans l'ordre les briques suivantes :

- Le jeu de données des arbres au format csv uploadé (1). Il est à noter que AzureML permet de lire plusieurs formats de fichier de données y compris des flux ou des bases de données.
- La sélection des colonnes utiles pour notre modèle (2). Nous avons retenu 4 colonnes décrivant l'espèce des arbres (exemple: chêne, platane, ...), leur circonférence (en centimètres), leur hauteur (en mètres) et l'année de leur plantation qui sera la variable à prédire par le modèle. B

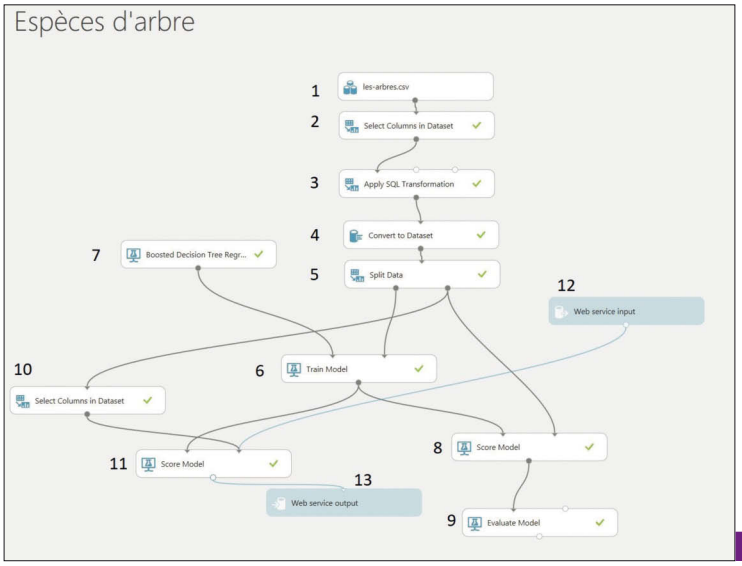
A



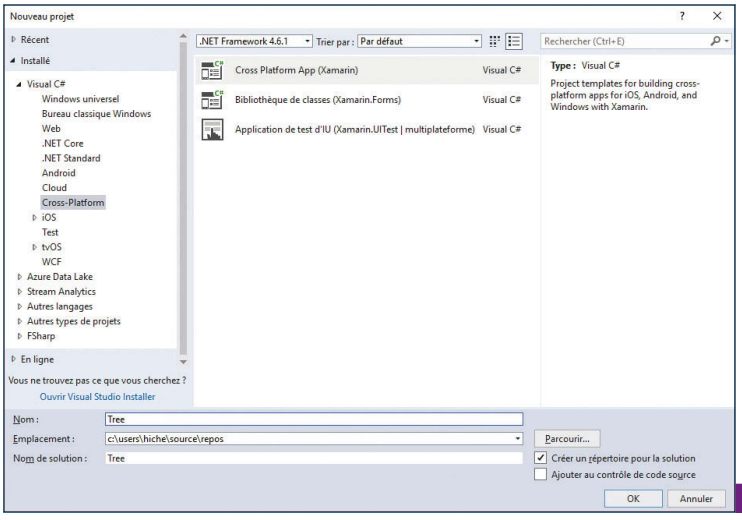
B



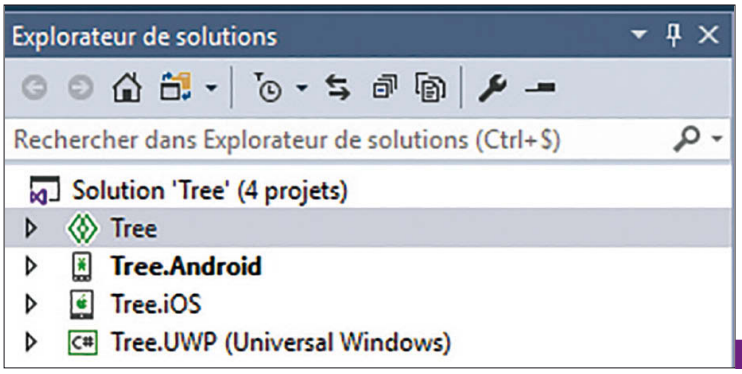
- Le nettoyage des données (3) à travers des transformations comme la suppression des lignes d'arbres avec des données manquantes ou encore l'extraction de l'année depuis la date de plantation. La sortie du nettoyage est ensuite convertie au format dataset (4) afin d'établir le modèle prédictif.
- Le découpage de l'ensemble des données en deux sous-ensembles (5) : un sous-ensemble d'apprentissage pour construire le modèle et un autre sous-ensemble de test pour évaluer le modèle. Ceci permet mesurer la performance de généralisation du modèle sur des données différentes à celles de l'apprentissage, et d'éviter le phénomène dit de "sur-apprentissage".
- L'entraînement du modèle sur le sous-ensemble d'apprentissage (6). Nous avons utilisé l'algorithme Boosted Decision Tree Regressor (7). Il présente l'avantage de combiner des informations hétérogènes ce qui correspond à la nature de nos variables (mesures, espèce, …) et de nécessiter un paramétrage minimal. L'algorithme Boosted Decision Tree Regressor produit un ensemble de règles regroupées sous forme d'arbre de décision. Dans notre cas, la décision consiste à estimer (ou régresser) l'année de plantation de l'arbre à partir d'un ensemble de conditions sur ses caractéristiques. On parle de régression puisque l'algorithme calcule une approximation de l'année (valeur quantitative).
- La brique scoring (8) permet de tester le modèle sur le sous-ensemble de test afin d'évaluer la qualité de notre modèle et d'estimer sa précision (9). Ainsi, le modèle est utilisé pour calculer les années de plantation. Ces estimations sont comparées aux années de plantation réelles afin de mesurer l'erreur du modèle. Cela nous a permis de constater que la marge d'erreur et de +/- six ans, ce qui est plutôt raisonnable.
- Afin d'utiliser le modèle sur de nouvelles données, le module de prédiction score model (11) prédit les années de plantation des arbres à partir de leur description. Le déploiement du modèle en tant que webservice s'effectue grâce à la brique webservice input (12) pour la collecte des données, et la brique webservice output (13) pour la restitution de l'année de plantation calculée. Le webservice consiste en une URL que l'on peut appeler depuis un client tel qu'un navigateur ou une application mobile. C



C



D



E

Création de l'application mobile avec Xamarin

L'objectif est maintenant de déployer notre modèle sur une application mobile native. Nous développerons avec Xamarin une interface permettant d'interroger le service web de AzureML studio. Concrètement, l'application collecte des informations saisies sur l'espèce de l'arbre, sa circonférence et sa hauteur. En réponse, elle prédit sa date de plantation. Là aussi, l'effort de codage est minimal car AzureML studio produit le code en C# pour soumettre les données au service web et récupérer la réponse calculée. Ce qui nous permet de nous concentrer sur la création de l'application Xamarin elle-même.

Pour commencer il faut créer un nouveau projet "Cross Plateform App (Xamarin)" dans Visual Studio. D

Une fois le projet créé, vous devez obtenir l'arborescence suivante dans l'explorateur de solution : E

Le premier projet est celui du code partagé, les autres projets sont spécifiques à chaque plateforme.

La puissance de Xamarin Forms nous permet d'écrire un seul code qui s'exécute sur toutes les plateformes mobiles. Il est toutefois possible d'écrire du code spécifique dans les projets dédiés lorsque cela s'avère nécessaire. Par exemple, la gestion de la caméra est complètement différente entre Android et iOS). Le code spécifique s'intègre ensuite facilement au code partagé par injection de dépendance.

Nous organisons notre code avec le design pattern recommandé pour les projets Xamarin : Model-View-ViewModel. Nous utilisons

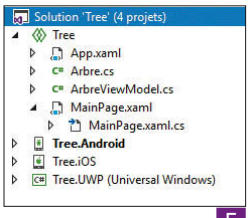
MainPage.xaml comme view, ajouter la classe Arbre comme Model et la classe ArbreViewModel comme ViewModel. F
Comme pour les projets WPF, la View est représentée par un fichier XAML. Au moment de la préparation de cet article, Xamarin ne dispose pas encore d'éditeur graphique d'interface. Il faut donc éditer manuellement le XAML. Cette opération est toutefois simplifiée par les suggestions et la complétion automatique qu'offre Visual Studio.

NB : *il est tout à fait possible d'écrire la View en tant que classe C#. Cependant, il est recommandé de le faire en XAML, pour plus de lisibilité et pour une meilleure séparation entre l'IHM et le code.*

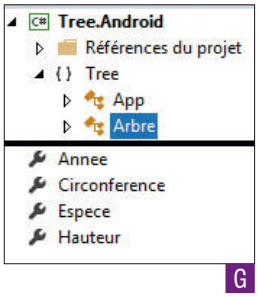
Le fichier MainPage.xaml est modifié comme suit :

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:local="clr-namespace:Tree"
        x:Class="Tree.MainPage">

  <StackLayout>
    <Entry Text="{Binding Circonference}" Placeholder ="Circonference (cm)"/>
    <Entry Text="{Binding Hauteur}" Placeholder ="Hauteur (m) :"/>
    <Entry Text="{Binding Espece}" Placeholder ="Espèce"/>
    <Button Text="Envoyer" Clicked="OnSendClicked"/>
    <ListView x:Name="ArbreList" SeparatorVisibility="Default" SeparatorColor="Black" Has
UnevenRows="True">
      <ListView.ItemTemplate>
        <DataTemplate>
          <ViewCell>
            <StackLayout>
              <Label Text="{Binding Circonference, Mode=OneWay, StringFormat='Circonference : {0}'}"/>
              <Label Text="{Binding Hauteur, Mode=OneWay, StringFormat='Hauteur : {0}'}"/>
             <Label Text="{Binding Espece, Mode=OneWay, StringFormat='Espèce : {0}'}"/>
              <Label Text="{Binding Annee, Mode=OneWay, StringFormat='Année de Plantation :
{0}'}" TextColor="Blue" FontAttributes="Bold"/>
            </StackLayout>
          </ViewCell>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </StackLayout>

</ContentPage>
```

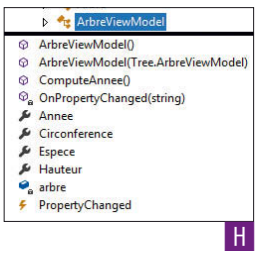
Dans cet exemple, les éléments de la vue sont entassés les uns sur les autres horizontalement grâce au StackLayout
La vue est composée de trois champs de saisie des propriétés de l'arbre, suivis d'un bouton d'envoi à l'API web d'AzureML et d'une liste pour afficher les requêtes envoyées avec leurs résultats (la prédiction de l'année de plantation).
Le lien entre les propriétés des éléments de la View et les propriétés de la ViewModel se fait par le mot clé Binding.
Dans l'exemple ci-après, la propriété "Text" de l'élément de saisie est liée à la propriété Circonference de ArbreViewModel.



F



G



H

```
<Entry Text="{Binding Circonference}"
```

Le lien au niveau des objets se fait en C# dans la classe MainPage associée au XAML, cela définit mainpage.xaml.cs dont voici le code :

```
using System;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace Tree
{
  public partial class MainPage : ContentPage
  {
    private ArbreViewModel ArbreViewModel { get; set; } = new ArbreViewModel();
   private ObservableCollection<ArbreViewModel> ArbresList { get; set; } = new Observable
Collection<ArbreViewModel>();
    public MainPage()
      {
      BindingContext = ArbreViewModel;
      InitializeComponent();
      ArbreList.ItemsSource = ArbresList;

    }

    private async void OnSendClicked(object sender, EventArgs e)
    {
      var arbreVM = new ArbreViewModel(ArbreViewModel);
      ArbresList.Add(arbreVM);
      await arbreVM.ComputeAnnee();
    }
  }
}
```

L'objet ArbreViewModel est défini comme BindingContext de la vue. Ce qui permet de lier ces propriétés dans le XAML.
Une Liste de ArbreViewModel est définie par la suite comme source de données à la ListView du XAML.

NB 1 : *l'utilisation de ObservableCollection est très importante. Cette collection notifie des changements qui se produisent, ce qui permet à la vue de se mettre à jour.*
Cette classe définit aussi la méthode appelée lors de l'appui sur le bouton : OnSendClicked. Elle ajoute un arbre à la liste et déclenche le calcul de l'année.

NB 2 : *le mot clé async permet l'exécution en mode asynchrone. Concrètement, après l'ajout de l'arbre à la liste et grâce au mot clé await, la méthode rend la main permettant le calcul de l'année. Ce traitement long (envoi de requête Http et attente de réponse du serveur) va se faire en arrière-plan. Ceci est très important car il permet de ne pas freezer l'interface graphique. Il permet aussi d'empiler les requêtes sans attendre le résultat des précédentes.*

Revenons au modèle. La classe Arbre, contient uniquement les propriétés de l'arbre : G
Le ViewModel sert d'intermédiaire entre la vue et le modèle. Il pré-

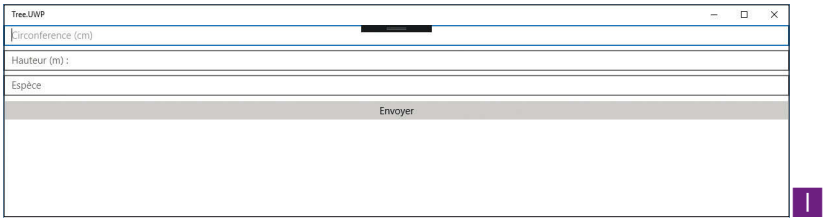
sente les propriétés de ce dernier à la première. H
Pour ce faire, il implémente la propriété INotifyPropertyChanged. À chaque fois qu'une propriété est modifiée, elle déclenche l'évènement PropertyChanged pour notifier la vue. Voici le code partiel de la classe pour illustrer ce mécanisme :

```
public class ArbreViewModel : INotifyPropertyChanged
  {
    public event PropertyChangedEventHandler PropertyChanged;
    private void OnPropertyChanged(string name)
    {
      PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }
    public string Circonference
    {
      get
      {
        return arbre.Circonference == 0 ? null : arbre.Circonference.ToString();
      }
      set
      {
        int intVal;
        int.TryParse(value, out intVal);
        if (arbre.Circonference != intVal)
        {
          arbre.Circonference = intVal;
          OnPropertyChanged(nameof(Circonference));
        }
      }
    }
}
```

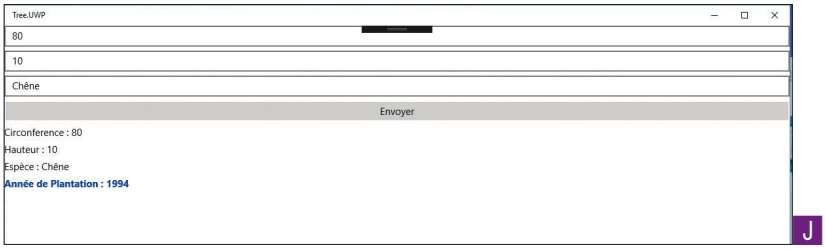
Le code de la méthode de calcul de l'année est déduit de l'exemple inclut dans la documentation du Web Service généré par AzureML. Notons qu'il faut écrire cette méthode de façon asynchrone pour les raisons expliquées précédemment :

```
public async Task ComputeAnnee()
{
 .
 .
 .

 response = await client.PostAsync("", new StringContent(jsonReq, Encoding.UTF8,
"application/json"));
 .
 .
 .
}
```

NB : ***l'utilisation du code de communication avec le service AzureML a nécessité l'installation d'un nuget défini dans la documentation générée. L'installation de nuget n'est pas possible sur le projet partagé lui-même. Il faut l'installer sur tous les projets des devices cible pour qu'il soit accessible à partir du projet partagé.***

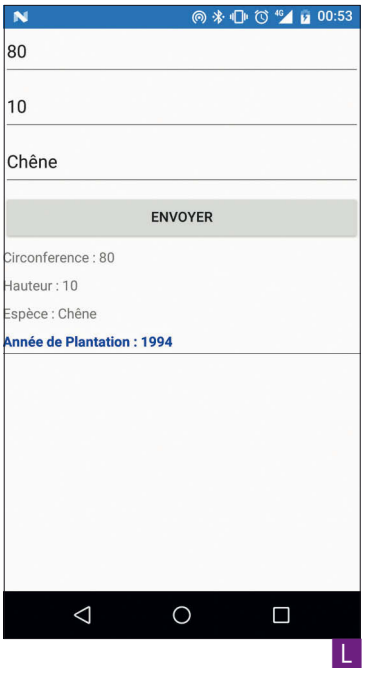


I



J

K



L

Pour exécuter l'application sur Windows, il faut définir le projet UWP comme projet de démarrage puis lancer l'exécution. Voici ce que cela donne au démarrage : I
Et après le lancement d'une requête : J
Pour exécuter sur Android, il faut définir le projet Android comme projet par défaut. Connecter un téléphone Android au PC puis lancer l'exécution. Voici ce que cela donne au démarrage : K
Et après le lancement d'une requête : L

Conclusion

Nous venons de créer - gratuitement - le contenu et l'interface d'une application IA mobile en quelques clics et quelques lignes de codes. Il est possible d'étendre l'usage à des sources de données plus évoluées comme l'utilisation de la caméra ou du microphone pour de la reconnaissance. L'exemple que nous avons présenté témoigne de la facilité et de l'accessibilité grandissante des outils de l'IA pour tous, sans exiger un prérequis fort en data science. •

Loïc Mathieu
Consultant à **Zenika Lille**

Arthas : l'outil de diagnostics d'Alibaba

Arthas est un outil de diagnostics d'applications Java écrit par Alibaba, le géant chinois du e-commerce. Il est disponible en open source (https://github.com/alibaba/arthas). Il a suscité beaucoup de curiosité dès sa sortie.

niveau 100

En effet, Alibaba est une très grosse société (plus grosse qu'Amazon), cependant on ne sait pas grand-chose sur la manière dont ses applications fonctionnent. On sait que les développements de l'entreprise sont en Java. On sait également qu'elle utilise ses propres composants (distributions Linux, Open-JDK, Tomcat, etc.), mais elle communique beaucoup moins que les autres grandes sociétés du web. Un outil mis en open source allait donc me permettre d'en apprendre un peu plus sur eux, et vous verrez que cet outil est assez atypique !

Arthas est un outil en ligne de commande qui permet de monitorer et débugger le comportement d'une application Java. Alibaba l'utilise en production. Au démarrage d'Arthas, on va sélectionner le process Java sur lequel on veut se connecter, puis on a ensuite une ligne de commande qui permet de lancer un ensemble d'outils (la liste est disponible en tapant la commande **help**) que nous allons parcourir dans cet article.

```
ID          NAME                                GROUP                  PRIORITY  STATE         %CPU    TIME      INTERRUPTED DAEMON
27          http-nio-8081-exec-5                main                   5         BLOCKED       28      5:21      false       true
28          http-nio-8081-exec-6                main                   5         BLOCKED       22      5:21      false       true
31          http-nio-8081-exec-9                main                   5         BLOCKED       22      5:22      false       true
30          http-nio-8081-exec-8                main                   5         BLOCKED       12      5:21      false       true
26          http-nio-8081-exec-4                main                   5         RUNNABLE      8       5:19      false       true
33          http-nio-8081-ClientPoller-0        main                   5         RUNNABLE      2       0:39      false       true
34          http-nio-8081-ClientPoller-1        main                   5         RUNNABLE      2       0:39      false       true
41          AsyncAppender-Worker-arthas-cache.res system               9         WAITING       0       0:0       false       true
39          Attach Listener                     system                 9         RUNNABLE      0       0:0       false       true
15          Catalina-utility-1                  main                   1         WAITING       0       0:0       false       false
16          Catalina-utility-2                  main                   1         TIMED_WAITIN  0       0:0       false       false
11          Common-Cleaner                      InnocuousThreadGroup   8         TIMED_WAITIN  0       0:0       false       true
37          DestroyJavaVM                       main                   5         RUNNABLE      0       0:6       false       false
3           Finalizer                           system                 8         WAITING       0       0:0       false       true
19          HikariPool-1 housekeeper            main                   5         TIMED_WAITIN  0       0:0       false       true
22          NioBlockingSelector.BlockPoller-1   main                   5         RUNNABLE      0       1:18      false       true
2           Reference Handler                   system                 10        RUNNABLE      0       0:0       false       true
Memory                     used     total    max      usage     GC
heap                       260M     514M     3922M    6,63%     gc.g1_young_generation.count         2796
g1_eden_space              117M     321M     -1       36,45%    gc.g1_young_generation.time(ms)      16334
g1_old_gen                 140M     190M     3922M    3,57%     gc.g1_old_generation.count           0
g1_survivor_space          3M       3M       -1       100,00%   gc.g1_old_generation.time(ms)        0
nonheap                    114M     151M     -1       75,93%
codeheap_'non-nmethods'    1M       2M       5M       22,16%
metaspace                  83M      85M      -1       97,28%
codeheap_'profiled_nmethods' 7M     28M      117M     6,60%
Runtime
os.name                                                         Linux
os.version                                                      5.0.0-15-generic
java.version                                                    11.0.2
java.home                                                       /usr/lib/jvm/jdk-11.0.2
systemload.average                                              2,03
processors                                                      12
uptime                                                          1571s
```

1

```
$ thread -b
"http-nio-8081-exec-9" Id=31 RUNNABLE
    at org.h2.value.Value.compareTo(Value.java:1160)
    at org.h2.engine.Database.areEqual(Database.java:371)
    at org.h2.expression.Comparison.compareNotNull(Comparison.java:304)
    at org.h2.expression.Comparison.getValue(Comparison.java:274)
    at org.h2.expression.Expression.getBooleanValue(Expression.java:178)
    at org.h2.command.dml.Select.isConditionMet(Select.java:312)
    at org.h2.command.dml.Select$LazyResultQueryFlat.fetchNextRow(Select.java:1455)
    at org.h2.result.LazyResult.hasNext(LazyResult.java:79)
    at org.h2.result.LazyResult.next(LazyResult.java:59)
    at org.h2.command.dml.Select.queryFlat(Select.java:527)
    at org.h2.command.dml.Select.queryWithoutCache(Select.java:633)
    at org.h2.command.dml.Query.queryWithoutCacheLazyCheck(Query.java:114)
    at org.h2.command.dml.Query.query(Query.java:371)
    at org.h2.command.dml.Query.query(Query.java:333)
    at org.h2.command.CommandContainer.query(CommandContainer.java:114)
    at org.h2.command.Command.executeQuery(Command.java:202)
    -  locked org.h2.engine.Database@1342a41e <---- but blocks 9 other threads!
    at org.h2.jdbc.JdbcPreparedStatement.executeQuery(JdbcPreparedStatement.java:114)
    -  locked org.h2.engine.Session@d82c5d
    at com.zaxxer.hikari.pool.ProxyPreparedStatement.executeQuery(ProxyPreparedStatement.java:52)
    at com.zaxxer.hikari.pool.HikariProxyPreparedStatement.executeQuery(HikariProxyPreparedStatement.java)
    at org.hibernate.engine.jdbc.internal.ResultSetReturnImpl.extract(ResultSetReturnImpl.java:60)
```

2

Le dashboard

Le premier outil à utiliser est le dashboard d'Arthas, il permet d'avoir une vision globale du comportement de la JVM : CPU, mémoire, Garbage Collection, thread…

Exemple d'utilisation de la commande dashboard : **1**

Ce qui est intéressant dans ce dashboard, c'est qu'il donne directement le pourcentage de CPU pris par chaque thread de l'application. Une information que les outils classiques (visualvm et autre) ne donnent pas directement.

Regarder ce que font mes threads

Grâce au dashboard, on sait que certains threads consomment beaucoup de CPU. On peut également consulter la stack de la thread grâce à la commande **thread <id>**. Malheureusement, nos applications ne sont pas aussi simples que cela. Il va donc falloir rechercher les threads les plus problématiques.

Arthas propose deux approches pour cela :

- Trouver le thread qui bloque le plus de thread via **thread -b,** il nous affiche alors la stack du thread avec une grosse flèche rouge sur la frame qui bloque !
- Lister les n threads qui consomment le plus de CPU via **thread -n <nb>**. On a alors les stacks des n threads qui s'affichent.

Exemple d'utilisation de la commande thread -b : **2**

Regarder le comportement d'une classe

Arthas fournit également beaucoup de commandes pour inspecter le comportement d'une classe :

watch : permet de regarder chaque appel de méthode, on doit lui spécifier ce qu'on souhaite regarder : les paramètres (sous commande **params**), le retour de la méthode (sous commande **returnObj**) ou les exceptions (sous commande **throwExp**). Le problème est que chaque appel est logué. Si la méthode en question est appelée très souvent ceci rend la commande inutilisable.

Utilisation : **watch <class> <method> <sous_commande>**

Exemple : **watch demo.MathGame primeFactors params**

tt : time tunnel : va permettre de faire la même chose que watch mais en enregistrant un nombre limité d'appels au lieu de le faire en temps réel. On pourra ensuite aller consulter un de ces appels enregistrés (via l'option -i) et même le rejouer (la méthode sera alors appelée de nouveau avec les mêmes paramètres d'entrées).

Exemple d'utilisation de la commande tt : **3**

monitor : permet une vue plus synthétique de l'exécution d'une méthode. La commande va nous donner période par période (configurable via l'option -c) le nombre d'appels en succès, en

erreur, le temps moyen d'exécution et le pourcentage d'erreur.
Exemple d'utilisation de la commande monitor : 4

trace : va afficher la trace de chaque appel de la méthode : sa stack d'appel avec le temps pris dans chaque méthode en mettant en rouge la méthode ayant pris le plus de temps. C'est une des commandes les plus efficaces pour trouver les problèmes de performance. En plus elle permet une sélection poussée des appels à tracer via une condition-expression sur les paramètres d'entrées ou le temps d'exécution.

trace <class> <method> isThrow : trace les appels de méthode en erreur ;

trace <class> <method> params[0]>1000 : trace les appels de méthode dont le premier paramètre est supérieur à 1000 ;

trace <cass> <methode> #cost>100 : trace les appels de méthode ayant pris plus de 100ms.

Exemple d'utilisation de la méthode trace : 5

Regarder le code qui tourne

Parfois, la personne qui investigue un problème de production n'est pas celle qui a développé l'application. D'autres fois, on ne sait tout simplement plus trop ce qui tourne…

Pour cela, Arthas est capable de réaliser des recherches dans les classloaders et d'afficher et modifier le code… oui vous avez bien lu : **modifier à chaud le code qui tourne** !!! Sans préjuger de l'aspect moral de la chose… pouvoir le faire c'est cool ;)

Pour rechercher une classe, il faut lancer la commande **sc <class-pattern>**. Le pattern passé en paramètre accepte les étoiles (wildcard). L'option --regex permet de passer une regex.

Exemple d'utilisation de la méthode sc : 6

Pour chercher une méthode, il faut lancer la commande **sm <class-patten> <method-pattern>**. Le method-pattern est optionnel. Les deux patterns fonctionnent comme pour la méthode sc.

Exemple d'utilisation de la méthode sc : 7

La commande **classloader** va permettre de lister les classloader de l'application et d'accéder à leurs informations, y compris la liste des classes qu'ils ont chargées.

Une fonctionnalité très puissante d'Arthas est de pouvoir modifier à chaud du code. Pour cela il inclut l'outil **jad** (Java Decompiler) qui permet de décompiler le code Java et donc d'accéder aux sources de l'application directement depuis Arthas.

On peut ensuite sauvegarder le code décompilé localement, le modifier et le recompiler. Pour le recompiler, l'outil **mc** (Memory Compiler) permet de le faire facilement depuis Arthas, sans soucis de classpath ! Une fois votre classe compilée, il faut la recharger via la commande **redefine** qui va charger votre code nouvellement compilé dans le classloader.

Et voilà comment débugger en production du code… à ne pas mettre entre toutes les mains.

Pour finir

Il y a d'autres commandes dont je ne vous ai pas parlé. De plus, nombre des commandes présentées ici ont beaucoup d'autres fonctionnalités. N'hésitez pas à parcourir la documentation officielle pour en savoir plus.

Pour conclure, parlons rapidement de comment marche Arthas. Au lancement d'Arthas, celui-ci va s'attacher à la JVM en cours de fonctionnement (comme un agent Java) puis utiliser ASM pour injecter du bytecode dans vos méthodes. Il va ensuite démarrer dans votre application un *serveur arthas*, qui va permettre au client ligne de commande de discuter avec votre application.

L'exécution de ce bytecode injecté a un coût, il faut donc arrêter le serveur arthas et l'instrumentation de vos classes quand vous avez fini de l'utiliser. Par défaut, quand vous quittez la ligne de commande (CTL-C ou la commande exit), le serveur arthas fonctionne toujours dans votre application, il faut l'arrêter via la commande **shutdown**.

Pour voir Arthas en action, voici le lien vers une présentation que j'ai fait à Devoxx France 2019 : https://youtu.be/e8CMlsMvIrc

•

```
$ tt fr.loicmathieu.perf.badapp.rest.CityRest searchCities -t -n 10
Press Q or Ctrl+C to abort.
Affect(class-cnt:1 , method-cnt:1) cost in 49 ms.
 1062     2019-05-23 15:51:49     42.585472   true      false     0x382c90c2        CityRest                        searchCities
 1063     2019-05-23 15:51:49     41.559583   true      false     0x382c90c2        CityRest                        searchCities

 1065     2019-05-23 15:51:49     42.070778   true      false     0x382c90c2        CityRest                        searchCities
 1064     2019-05-23 15:51:49     91.864055   true      false     0x382c90c2        CityRest                        searchCities
 INDEX    TIMESTAMP               COST(ms)    IS-RET    IS-EXP    OBJECT            CLASS                           METHOD
-----------------------------------------------------------------------------------------------------------------------------------
 1061     2019-05-23 15:51:49     40.434531   true      false     0x382c90c2        CityRest                        searchCities
 1066     2019-05-23 15:51:50     182.143194  true      false     0x382c90c2        CityRest                        searchCities
 1067     2019-05-23 15:51:50     57.748644   true      false     0x382c90c2        CityRest                        searchCities
 1070     2019-05-23 15:51:50     55.829196   true      false     0x382c90c2        CityRest                        searchCities
 1068     2019-05-23 15:51:50     56.053065   true      false     0x382c90c2        CityRest                        searchCities
 1069     2019-05-23 15:51:50     55.528122   true      false     0x382c90c2        CityRest                        searchCities

Command execution times exceed limit: 10, so command will exit. You can set it with -n option.
$ tt -i 1000
 INDEX          1000
 GMT-CREATE     2019-05-23 15:37:50
 COST(ms)       43.794573
 OBJECT         0x382c90c2
 CLASS          fr.loicmathieu.perf.badapp.rest.CityRest
 METHOD         searchCities
 IS-RETURN      true
 IS-EXCEPTION   false
 PARAMETERS[0]  @Integer[106449]
 PARAMETERS[1]  @Integer[8913025]
 RETURN-OBJ     @ArrayList[
                    @City[fr.loicmathieu.perf.badapp.domain.bo.City@20],
                    @City[fr.loicmathieu.perf.badapp.domain.bo.City@21],
```

3

```
$ monitor fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl listCities -c 5
Press Q or Ctrl+C to abort.
Affect(class-cnt:2 , method-cnt:2) cost in 80 ms.
timestamp            class                                                                    method      total  success  fail  avg-rt(ms)  fail-rate
-----------------------------------------------------------------------------------------------------------------------------------------------
2019-05-23 15:54:21  fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl                listCities  154    154      0     40,72       0,00%
2019-05-23 15:54:21  fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl$$EnhancerBySpringCGLIB$$43a03c9a  listCities  154  154  0  68,59  0,00%

timestamp            class                                                                    method      total  success  fail  avg-rt(ms)  fail-rate
-----------------------------------------------------------------------------------------------------------------------------------------------
2019-05-23 15:54:26  fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl                listCities  180    180      0     42,34       0,00%
2019-05-23 15:54:26  fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl$$EnhancerBySpringCGLIB$$43a03c9a  listCities  179  179  0  70,32  0,00%
```

4

```
$ trace fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl listCities
Press Q or Ctrl+C to abort.
Affect(class-cnt:2 , method-cnt:2) cost in 82 ms.
`---ts=2019-05-23 16:02:53;thread_name=http-nio-8081-exec-3;id=19;is_daemon=true;priority=5;TCCL=org.springframework.boot.web.emb
    `---[157.401487ms] fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl$$EnhancerBySpringCGLIB$$43a03c9a:listCities()
        `---[157.22206ms] org.springframework.cglib.proxy.MethodInterceptor:intercept() #0
            `---[69.512723ms] fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl:listCities()
                `---[69.345463ms] fr.loicmathieu.perf.badapp.domain.repository.CityRepository:findAll() #23

`---ts=2019-05-23 16:02:54;thread_name=http-nio-8081-exec-5;id=1b;is_daemon=true;priority=5;TCCL=org.springframework.boot.web.emb
    `---[512.328721ms] fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl$$EnhancerBySpringCGLIB$$43a03c9a:listCities()
        `---[512.265634ms] org.springframework.cglib.proxy.MethodInterceptor:intercept() #0
            `---[369.899551ms] fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl:listCities()
                `---[369.826088ms] fr.loicmathieu.perf.badapp.domain.repository.CityRepository:findAll() #23

`---ts=2019-05-23 16:02:54;thread_name=http-nio-8081-exec-5;id=1b;is_daemon=true;priority=5;TCCL=org.springframework.boot.web.emb
    `---[67.663063ms] fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl$$EnhancerBySpringCGLIB$$43a03c9a:listCities()
        `---[67.606352ms] org.springframework.cglib.proxy.MethodInterceptor:intercept() #0
            `---[9.801035ms] fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl:listCities()
                `---[9.731614ms] fr.loicmathieu.perf.badapp.domain.repository.CityRepository:findAll() #23
```

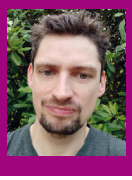
5

```
$ sc fr.loicmathieu.*
com.sun.proxy.$Proxy83
com.sun.proxy.$Proxy85
fr.loicmathieu.perf.badapp.BadAppApplication
fr.loicmathieu.perf.badapp.BadAppApplication$$EnhancerBySpringCGLIB$$a7e639a4
fr.loicmathieu.perf.badapp.config.GlobalConfiguration
```

6

```
$ sm fr.loicmathieu.perf.badapp.domain.svc.CityService
fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl <init>()V
fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl listCities()Ljava/util/List;
fr.loicmathieu.perf.badapp.domain.svc.impl.CityServiceImpl listNCities(I)Ljava/util/List;
```

7



Arnaud Thiefaine
Software Crafter
Arolla
Twitter:@ArnaudThiefaine

Man versus Legacy : Gilded Rose

Partie 2

Dans le premier épisode nous avions expliqué le contexte du kata et proposé une approche permettant de travailler la réécriture du code en toute sécurité. Pour cela nous avions écrit des tests de caractérisation, jusqu'à l'obtention d'un golden master.

Nous avons toute latitude pour refactorer le code afin d'y voir plus clair. La moindre erreur sera en effet mise en évidence par notre *golden master,* fournissant un vrai filet de sécurité. Cette seconde partie propose différentes techniques permettant de refactorer le code pour arriver à quelque chose de plus compréhensible. A partir de là, nous pourrons donner une dimension orientée plus métier au code, fournir des tests également centrés sur le métier, et au final nous affranchir du golden master.

La phase de refactoring

En entrant dans la phase de réécriture, il faut s'astreindre à nouveau à ces principes du TDD (Test-Driven Development):

- Avoir un feedback quasi-permanent (ce qu'on obtient en ayant le réflexe de relancer les tests à chaque modification du code, ou encore en s'aidant d'un lanceur de test automatique)
- Travailler par petits incréments (*baby steps*) et ne s'autoriser à poursuivre que tant qu'on est *dans le vert*

La conséquence principale est qu'en cas d'erreur, nous ne sommes qu'à un Ctrl+Z (*Undo)* de revenir à un état stable.

Lancer les tests c'est fastidieux …

Sérieusement ? C'est aussi pénible que ça ? On a presque l'impression d'entendre un ado se plaindre …

Relancer les tests, c'est juste l'affaire d'un raccourci clavier (Ctrl + F5 avec IntelliJ), et l'exécution est quasi-instantanée, à moins de développer avec une épave. Ce n'est donc pas la mer à boire.

Ceci étant dit, quand on est développeur, toutes techniques, astuces ou outils permettant de se faciliter la vie, même d'un *epsilon*, sont bons à prendre. La bonne nouvelle, c'est que ces outils existent : on parle de lanceurs automatiques de tests (comme NTests, Infinitests, ou directement intégrés à l'IDE). Ils permettent de rejouer automatiquement un ensemble sélectionné de tests à la moindre modification du code (sous réserve qu'il compile toujours). On peut alors se concentrer sur le refactoring pur et dur, on sait qu'on recevra (en principe) du feedback dès que la modification de code entraîne une modification de comportement. Même si ces lanceurs automatiques sont pratiques, ils peuvent se révéler capricieux : ils peuvent s'arrêter de fonctionner correctement du jour au lendemain, et la configuration initiale peut être une source de frustration. De plus, ils se prêtent assez mal à des volumes de code importants (pour des raisons de performance) et sont donc à réserver pour des interventions ciblées.

Le plus important est que les tests soient exécutés à chaque modification du code, que ce soit sous la forme d'un réflexe, ou d'une façon automatique. Notons que bénéficier d'un lanceur automatique, même si c'est bien pratique, ne nous aidera pas des masses à acquérir ce réflexe.

Par où commencer ?

Dans le cas de Gilded Rose, le démarrage du refactoring ne saute pas vraiment aux yeux.

Le premier réflexe est de réduire d'emblée la duplication. Pour cela, on recherche des morceaux de code récurrents, qui semblent être de bons candidats pour effectuer, selon le contexte des *Extract Constant, Extract Variable ou Extract Method*. En fournissant des noms judicieux, on a même l'impression de fournir un code mieux documenté.

Habituellement, les actions de refactoring suivantes sont observées :

- extraire des constantes pour les noms d'articles
- extraire des méthodes pour les conditions sur les noms : *isSulfuras()*, *isAgedBrie()* voire des choses comme *isAgedBrieOrBackstage()* : ce genre d'abstraction est dangereux parce qu'il est susceptible de masquer une duplication évidente sur le type d'article Aged Brie
- extraire des méthodes pour incrémenter ou décrémenter la qualité

Hé ! Pas si vite…

En procédant ainsi, on avance pendant un temps relativement bref, puis arrive le moment où on se retrouve coincé. On a beau scruter le code, on ne voit pas comment continuer. Toute tentative de refactoring supplémentaire réduit davantage le champ des possibles.

Voici quelques raisons qui font qu'on en arrive là :

- La complexité cyclomatique (i.e. le nombre de niveaux de branchement) reste importante
- On a du mal à identifier des niveaux d'abstraction, tout est mélangé; une même considération (par exemple les conditions sur le nom de l'article) se retrouve présente à différents niveaux de branchement
- Les attributs *quality* et *sellIn* sont utilisés en lecture (dans des conditions) et en écriture (incréments, décréments); modifier le code associé peut entraîner des effets de bord imprévus
- On trouve parfois de la fausse duplication (conditions paraissant semblables mais qui sont en fait inversées)
- Les lignes dupliquées ne se situent pas toujours au même niveau de branchement

Le leurre de l'optimum local

Si on est bloqué, c'est parce qu'on s'est retrouvé dans une situation d'optimum local. Pour s'en sortir, il faut accepter de dégrader du code de façon temporaire. En se retrouvant dans une situation (en apparence) moins favorable, on va pouvoir l'améliorer ensuite. C'est comme en alpinisme, pour atteindre un sommet plus haut, il est parfois nécessaire de redescendre un peu pour trouver une voie plus praticable. Ou comme lorsqu'on résout un Rubik's Cube, en acceptant de défaire temporairement une face complète.

En fait, on doit dès le départ faire quelque chose de totalement contre-intuitif : **introduire encore plus de duplication**. Même si le code initial s'étale sur de nombreuses lignes de code, il ne faut pas avoir peur de le rendre encore plus volumineux.
Vu sous un autre angle, on fait un peu comme en maths, où il est parfois nécessaire de développer certaines sous-expressions si on veut aboutir à une factorisation globale satisfaisante. Les approches de refactoring décrites reposent sur ce principe de départ.

Une première étape facile

Pour se donner du coeur à l'ouvrage et ne pas rester bloqué, nous allons commencer par un premier *Extract Method* au niveau global. Il s'agit d'extraire l'intérieur de la boucle dans une méthode dédiée. On pourrait le faire à la main, au risque de se louper, mais avec un IDE évolué (IntelliJ par exemple), il suffit de suivre ces étapes :

- on sélectionne tout le code à l'intérieur de la boucle
- on effectue l'action de refactoring *Extract Method*
- IntelliJ nous propose de créer une méthode dont la signature est *void updateQuality(Item item)*
- on valide et c'est fait (et on n'oublie pas de s'assurer que les tests sont toujours OK)

Maintenant, on peut prendre un instant pour admirer l'intelligence de l'IDE : la méthode créée aurait put avoir un paramètre de type *int*, correspondant au compteur de boucle. Seulement, IntelliJ a détecté que le seul usage du compteur de boucle *i* se situe dans l'expression suivante :

```
items[i]
```

dont le type est Item. D'où le fait de proposer directement un paramètre de ce type.

Remarque : avec un IDE moins futé, on procéderait en 2 étapes. Il suffirait de remplacer la boucle avec index par une boucle *for each* sur le type Item, et ensuite d'extraire la méthode.
On y gagne en élégance, en lisibilité mais aussi en qualité du design. La méthode *updateQuality(Item item)* n'a pas besoin de connaître l'état interne de la classe GildedRose (ce qui aurait été le cas en reprenant le compteur de boucle en paramètre), ce qui est préférable en termes d'encapsulation et de découplage. Il serait alors possible de déplacer cette méthode dans un composant dédié.

Approche classique

Nous nous basons sur le refactoring effectué par David Gageot dans la session suivante : https://www.youtube.com/watch?v=q11gydDAMSo. Le challenge est de taille puisque le speaker doit décrire le problème, écrire les tests, refactorer le code, construire une application Web simple et déployer sur le Cloud, tout cela en un temps très court.
Ces contraintes font que même si l'ensemble est brillamment exécuté, le performer présente un contenu très dense et énormément de techniques, de façon peut-être trop rapide. C'est pourquoi nous faisons le choix de nous attarder sur les différentes étapes.
Nous avons vu qu'il était inapproprié de chercher immédiatement à réduire la duplication, sous peine de se retrouver coincé. Dans le cas de Gilded Rose, même si c'est contre-intuitif, il ne faut pas avoir peur de faire grossir le code en augmentant ponctuellement la duplication.

Etapes du refactoring

Simplifier les conditions (c'est pas faux !)

On va chercher à transformer les conditions complexes en un ensemble de conditions plus simples. L'idée est de supprimer les NOT, AND ou OR dans l'optique d'obtenir uniquement des conditions à un seul terme.
Ce faisant, on va rendre le code en apparence plus complexe et volumineux, mais on obtient une meilleure vision de la logique de branchement. De plus, on va faire émerger de la duplication entre certaines conditions, offrant ainsi des opportunités de nettoyage du code. En fait, on cherche à augmenter la duplication pour pouvoir la tuer.
Voici les patterns de réécriture qui s'appliquent :

Type de condition	Avant réécriture	Après réécriture
NOT	`if (!cond) { //... bloc1 } else { //... bloc 2 }`	`if (cond) { //... bloc2 } else { //... bloc 1 }`
AND	`if (condA && condB) { //... bloc }`	`if (condB) { if (condA) { //... bloc } }`
OR	`if (condA || condB) { //... bloc }`	`if (condA) { //... bloc } else if (condB) { //... bloc }`

Cette étape demande assez peu d'efforts, l'IDE s'occupe de tout. Dans le cas d'expressions plus complexes mélangeant plusieurs opérateurs, on se basera sur la priorité des opérateurs. Dans le cas de Gilded Rose, on supprime d'abord les conditions NOT. Il ne reste alors qu'une condition OR à réécrire.
Cette étape produit les effets suivants :

- certaines sous-conditions deviennent mutuellement exclusives avec une condition de plus haut niveau, indiquant un code inatteignable (ou mort). On peut alors supprimer la sous-condition et le bloc d'instructions qu'elle contient
- d'autres sous-conditions deviennent redondantes (parce que identiques) avec une condition de plus haut niveau. On peut alors supprimer la sous-condition, mais pas le bloc de code qu'elle contient. Il faut néanmoins veiller à l'absence d'effet de bord entre les deux conditions pour faire la suppression sans risques

Un rappel important : en cas de doute, ne pas oublier que la présence du harnais de test, associé au fait de lancer les tests à chaque modification du code, offrent des garanties suffisantes.

Le cas Sulfuras

Nous commençons à avoir l'intuition que le nom des articles est au coeur de la logique métier :

- les valeurs possibles appartiennent à un espace discret très petit, ce qui offre une forme de catégorisation
- l'évolution demandée (ajouter une règle de gestion sur les Conju-

red Items) nous oriente naturellement sur ce choix
- un certain nombre de conditions basées sur le nom apparaissent au premier niveau de branchement

On va donc chercher progressivement à placer les conditions propres au nom au niveau le plus externe.
On constate que le type Sulfuras est particulier : le succès de la condition associée ouvre systématiquement sur un bloc de code vide.
Nous faisons alors le choix (payant) de remonter le cas Sulfuras dans une unique condition de garde en début de méthode :

```
if (item.name.equals("Sulfuras, Hand of Ragnaros")) {
  return;
}
```

Une symétrie commence à apparaître dans le code sur le nom des articles, ce qui permet d'entrevoir de façon subtile et prometteuse le métier.

Des conditions presque identiques

Le bloc de code suivant apparaît à de nombreuses reprises :

```
if (item.quality < 50) {
  item.quality = item.quality + 1;
}
```

C'est donc un candidat idéal pour un Extract Method. Seulement, il reste un endroit avec la configuration suivante :

```
if (item.quality < 50) {
  item.quality = item.quality + 1;
  // ... autres instructions
}
```

C'est presque le même code, mais les instructions à la suite nous embêtent. En examinant le code plus en détails, on a quand même l'intuition qu'on pourrait remonter l'accolade fermante avant les autres instructions sans que ça ne pose de problèmes.
Heureusement, notre filet de sécurité nous permet de valider cette hypothèse et de procéder à l'extraction d'une méthode *increaseQuality()*.
De façon symétrique nous extrayons également *decreaseQuality()* :

```
private void decreaseQuality() {
  if (item.quality > 0) {
    item.quality = item.quality - 1;
  }
}
```

Briser les règles pour mieux voir le code

On cherche maintenant à visualiser l'ensemble du code sur un seul écran. Pour cela, nous choisissons de renier les conventions de code et de supprimer les accolades pour les conditions sur le *sellIn*. On obtient alors un code plus compact et on voit apparaître des structures.

La Muraille de Chine

Cette portion du code nous est particulièrement pénible :

```
item.sellIn = item.sellIn - 1;
if (item.sellIn < 0) {
```

Elle agit comme une barrière au milieu du code, une sorte de Muraille de Chine qui nous empêche de fusionner et d'avoir au même niveau les blocs de code basés sur les noms d'articles.
Une idée pour s'en sortir ? Une fois encore, en augmentant temporairement la duplication. Mais pas n'importe comment :
- l'instruction d'incrément du *sellIn* va être dupliquée dans chacun des blocs du haut
  - attention à sa position dans le bloc sur le Backstage, il faut le laisser après les conditions sur le *sellIn* pour ne pas provoquer de régressions
- la condition sur le *sellIn* va être dupliquée dans chacun des blocs du bas

Nos conditions liées aux noms d'articles se retrouvent alors toutes au même niveau, le plus externe, sans séparation entre elle. On peut donc les fusionner facilement.

Décrément du sellIn

Malgré ça, le décrément du *sellIn* continue à nous ennuyer : cette ligne de code apparaît 3 fois. On aimerait bien la remonter avant la condition sur le Aged Brie. Seulement, le comportement du Backstage nous en empêche.
On part de ça :

```
if (item.sellIn < 11) increaseQuality(item);
if (item.sellIn < 6) increaseQuality(item);
item.sellIn = item.sellIn - 1;
```

En faisant ça, les tests échouent :

```
item.sellIn = item.sellIn - 1;
if (item.sellIn < 11) increaseQuality(item); // tests KO
if (item.sellIn < 6) increaseQuality(item);
```

On résout le problème en rétablissant l'équilibre dans les conditions :

```
item.sellIn = item.sellIn - 1;
if (item.sellIn + 1 < 11) increaseQuality(item);
if (item.sellIn + 1 < 6) increaseQuality(item);
```

Ce qui se simplifie en :

```
item.sellIn = item.sellIn - 1;
if (item.sellIn < 10) increaseQuality(item);
if (item.sellIn < 5) increaseQuality(item);
```

Ce qui ne pose plus aucun problème pour remplacer les 3 décréments en un seul avant la condition sur le Aged Brie. Cette duplication introduite temporairement est ainsi éliminée.

Les touches finales

Nous sommes maintenant libres d'effectuer au choix les améliorations suivantes :
- déléguer la mise à jour des articles à une classe ItemUpdater
- remplacer les *if … else* par un *switch*
- mieux encore, mettre en oeuvre un pattern Strategy basé sur le nom des articles

Mais c'est vraiment si on dispose d'encore un peu de temps. L'important est que nous soyons arrivé à un code beaucoup plus lisible, qui permet de bien comprendre le métier sous-jacent. Nous avons ainsi l'opportunité d'implémenter facilement et sans risques l'évolution demandée.
GitHub : https://github.com/athiefaine/gilded-rose-kata/tree/classic_way (chacune des étapes correspond à un commit spécifique)

Approche ludique

Woody Zuill propose une approche de refactoring beaucoup plus originale mais diablement efficace.

Encore plus de duplication !

Elle se base également sur l'intuition qu'une bonne partie de la logique métier se base sur le nom des articles, agissant comme une catégorisation. On va alors chercher à construire un chemin spécifique pour chaque catégorie d'article, isolant ainsi la logique métier qui lui est propre.

Dans le code initial, chacune de ces logiques métier est entremêlée avec les autres. L'idée est d'isoler chacune de ces logiques, l'une après l'autre. En choisissant une première catégorie d'article, nous créons deux chemins dans le code :

- le premier pour lequel l'article est dans la catégorie choisie
- le second pour lequel l'article appartient à n'importe quelle autre catégorie

Commençons par le Aged Brie :

```
if (item.name.equals("Aged Brie")) {
} else {
}
// ... legacy code
```

Le problème consiste maintenant à fournir une implémentation pour chacune des deux branches. Ca paraît difficile, étant donné qu'on ne comprend pas vraiment la logique métier ...

Le fait est que nous avons une implémentation qui fonctionne pour les deux branches, juste sous nos yeux : le code legacy en lui-même ! Réécrivons le code en introduisant temporairement davantage de duplication :

```
if (item.name.equals("Aged Brie")) {
    // ... legacy code
} else {
    // ... duplicated legacy code
}
```

Les tests passent toujours, c'est incroyable ! Mais nous ne sommes pas plus avancés ...

Pourtant, si nous avons introduit de la duplication (le volume de code a littéralement été multiplié par 2), c'est pour mieux la tuer par la suite. La question est d'identifier les portions de code à supprimer …

La couverture de code comme outil de refactoring

Dans la première partie de l'article, nous nous étions appuyés sur un outil de couverture de code pour déterminer la robustesse et la pertinence de notre harnais de tests. Cet objectif atteint avec une couverture optimale, on était en droit de supposer que la couverture de code ne nous était d'aucune utilité pour la phase de refactoring. En fait, c'est tout l'inverse, la couverture de code va nous fournir une aide précieuse.

Relançons les tests avec la couverture de code activée, et observons ce qu'il se passe pour la branche Aged Brie : 1

Les zones en rouge désignent le code qui n'a pas été exécuté par les tests. Cela signifie que ces lignes de code n'ont aucune utilité pour que notre nouvelle implémentation soit iso-fonctionnelle avec l'ancienne.

En d'autres termes, c'est du code mort, et en tant que tel il peut être supprimé sans danger.

En combinant la couverture de code et les indications de l'IDE, on peut appliquer les patterns suivants en vue de simplifier le code sans risques :

- ligne dont la couverture est rouge : c'est du code mort qui peut être supprimé
- condition signalée comme toujours fausse (par exemple elle est mutuellement exclusive avec une condition à un plus haut niveau) : on peut enlever la condition et le bloc qu'elle contient
- condition signalée comme toujours vraie (cela arrive si elle est redondante avec une condition de plus haut niveau) : on procède à un *unwrap* (on "déballe" le bloc de code, i.e. on le conserve mais on supprime la condition qui l'englobe)
- un *if* ou un *else* vide : on supprime

Une fois toutes ces opérations effectués, le chemin pour Aged Brie ressemble à ça : 2

Le fait qu'on retrouve une couverture optimale indique qu'on peut difficilement faire mieux à ce stade.

Le niveau de simplification est assez impressionnant. Mais ce qui l'est encore plus, c'est d'être arrivé à ce résultat sans se faire de noeuds au cerveau. En effet, nous nous sommes laissés guider par nos outils et les automatismes de l'IDE, en ne prenant aucun risque. De plus, c'est assez ludique, on cherche juste à tuer les lignes qui ne sont pas couvertes. Cette façon de faire me fait penser à certains jeux de réflexion qu'on pratique en mode semi-automatique,

```
private void updateQuality(Item item) {
    if (item.name.equals("Aged Brie")) {
        if (!item.name.equals("Aged Brie")
                && !item.name.equals("Backstage passes to a TAFKAL80ETC concert")) {
            if (item.quality > 0) {
                if (!item.name.equals("Sulfuras, Hand of Ragnaros")) {
                    item.quality = item.quality - 1;
                }
            }
        } else {
            if (item.quality < 50) {
                item.quality = item.quality + 1;

                if (item.name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                    if (item.sellIn < 11) {
                        if (item.quality < 50) {
                            item.quality = item.quality + 1;
                        }
                    }

                    if (item.sellIn < 6) {
                        if (item.quality < 50) {
                            item.quality = item.quality + 1;
                        }
                    }
                }
            }
        }

        if (!item.name.equals("Sulfuras, Hand of Ragnaros")) {
            item.sellIn = item.sellIn - 1;
        }

        if (item.sellIn < 0) {
            if (!item.name.equals("Aged Brie")) {
                if (!item.name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                    if (item.quality > 0) {
                        if (!item.name.equals("Sulfuras, Hand of Ragnaros")) {
                            item.quality = item.quality - 1;
                        }
                    }
                } else {
                    item.quality = item.quality - item.quality;
                }
            } else {
                if (item.quality < 50) {
                    item.quality = item.quality + 1;
                }
            }
        }
    } else {
```

1

```
private void updateQuality(Item item) {
    if (item.name.equals("Aged Brie")) {
        if (item.quality < 50) {
            item.quality = item.quality + 1;
        }

        item.sellIn = item.sellIn - 1;

        if (item.sellIn < 0) {
            if (item.quality < 50) {
                item.quality = item.quality + 1;
            }
        }
    } else {
```

2

comme les match-3, le 2048 ou encore Tetris. Je vous invite donc vivement à refaire ce refactoring à la maison pour en expérimenter le côté plaisant.

Rinse and repeat

L'idée est ensuite de passer aux catégories suivantes : Backstage, Sulfuras, avant de terminer par la catégorie d'article par défaut.
GitHub : https://github.com/athiefaine/gilded-rose-kata/tree/playful_way (chacune des étapes correspond à un commit spécifique)

Variante

Dans cette approche, nous n'avons supprimé le code mort que pour la branche "vraie" de la condition.
Une variante consiste à éliminer le code mort pour les deux branches de la condition à chaque tour. La branche concernant les autres cas se retrouve alors simplifiée de façon plus progressive.
Adopter l'une ou l'autre des variantes est une affaire de goût, on aboutit au même résultat final.
GitHub : https://github.com/athiefaine/gilded-rose-kata/tree/playful_way_alt (chacune des étapes correspond à un commit spécifique)

La couverture de code, un couteau suisse ?

Ou "pourquoi ça fonctionne" ?
La couverture de code sert au départ à identifier les lignes de code qui ne sont pas testées. Lorsqu'on atteint une couverture optimale (proche de 100%), on peut être rassuré sur le fait que le code est suffisamment testé.
C'est le cas pour notre code legacy, aucun doute là-dessus. Le fait qu'un code alternatif fournissant exactement les mêmes fonctionnalités comporte des lignes non couvertes leur confère une nature de code mort.
Bien sûr, cette propriété provient de la nature de nos tests, en *golden master*, où on compare le comportement de deux implémentations différentes. Ce contexte précis entraîne que la couverture de code peut être employée comme détecteur de code mort, ce qui constitue le point-clé de cette approche.

Point de convergence

A ce stade, nous atteignons un code relativement lisible, mais pas aussi épuré qu'avec l'approche classique. Nous pouvons appliquer certaines de ses étapes pour améliorer le code :
- extraire *increaseQuality()* et *decreaseQuality()*
- remonter le décrément sur le *sellIn* pour en réduire la duplication

On s'épargne les difficultés suivantes de l'approche classique :
- ne pas s'embêter à simplifier les conditions
- ne pas être confronté à la Muraille de Chine sur le *sellIn* et obtenir des conditions sur les noms fusionnées dès le départ

L'inconvénient majeur est que l'approche ludique ainsi que son hypothèse de départ (catégoriser sur les noms d'articles) ne sont pas nécessairement intuitives.

Passons aux choses sérieuses

Une fois le refactoring effectué, on obtient un code plus lisible, plus compréhensible. On commence à avoir une vision plus claire du métier. On pourrait s'estimer satisfait et être tenté de s'arrêter là, d'autant que le golden master fournit la couverture de test souhaitée. Le problème réside dans le fait que cela nous obligerait à conserver à la fois:
- Un code de test qui n'apporte aucune documentation sur la fonctionnalité, il se contente juste de comparer le comportement du code legacy et du code refactoré
- Un code legacy bien pourri (raison d'être de ce refactoring) qui va polluer la base de code à vie

De plus, toute évolution devrait être rétro-implémentée dans le code legacy (ce qui n'est pas une mince affaire) pour garantir le bon fonctionnement des tests. Et nous avons une règle supplémentaire à produire, sur les articles de type Conjured. Conserver le *golden master* nous empêche de faire évoluer le code.
Il serait plus raisonnable de commencer à isoler des règles métiers, ces dernières pouvant être explicitées dans des tests dédiés. On peut par exemple dédier des tests à chaque type d'article, vérifiant la qualité et le nombre de jours avant la vente. L'objectif est d'obtenir également une couverture de 100% avec ces tests, en excluant bien sûr le golden master des mesures. Une fois ceci atteint, le golden master peut définitivement être mis à la poubelle.
Un effet secondaire (et positif) de l'écriture de ces tests orientés métier réside dans le fait qu'on achète de la connaissance supplémentaire sur le fonctionnel. Dès lors, on peut apporter des refactoring supplémentaires guidés par cette compréhension accrue du métier.

Testons le métier

Avant de commencer, déplaçons l'implémentation legacy dans les sources de test. Ceci évitera qu'elle soit utilisée par erreur dans le code de production. Notons qu'au passage, on perd la couverture sur ce code legacy, mais ce n'est pas très grave, l'idée étant de se débarrasser du *golden master*.
Nous allons maintenant écrire des tests basés sur notre compréhension métier. L'objectif est d'obtenir un ensemble de tests qui explicitent les règles de gestion, tout en atteignant une couverture de code optimale. L'idée est de commencer par les règles paraissant les plus simples, puis de monter progressivement en complexité.

Sulfuras l'imputrescible

L'article Sulfuras est un bon candidat : quoiqu'il arrive, ses propriétés ne changent pas. Nous créons une classe de test pour le composant ItemUpdater, puis une méthode de test dédiée à Sulfuras. Dans celle ci, nous appelons updateQuality() un certain nombre de fois (16 fois, comme pour le *golden master*), et vérifions que les valeurs initiales de sellIn et de quality ne bougent jamais. Au passage, lorsqu'on lance la couverture de tests, on constate que seule la condition de garde sur Sulfuras est couverte, ce qui était prévisible.

Le Vieux Brie : plus ça pue et meilleur c'est

Ce type d'article possède des propriétés odorantes et gustatives qui au final rappellent notre code legacy. Les tests émergent facilement de la lecture du code :
- Un premier test vérifiant que la qualité s'améliore à chaque jour qui passe
- Un second test pour s'assurer que la qualité s'améliore encore plus lorsque le *sellIn* est expiré

Place de concert : la hype monte !

La qualité du Backstage s'améliore de plus en plus vite au fur et à mesure que les jours passent, avec 2 paliers : à 10 jours, puis à 5 jours du terme. Attention toutefois, une fois la date du concert pas-

sée (*sellIn* expiré), l'article ne vaut plus un clou. Nous avons donc besoin des tests suivants :

- La qualité augmente chaque jour à plus de 10 jours du terme
- Elle augmente deux fois plus vite entre 5 et 10 jours du terme
- Elle augmente trois fois plus vite entre 5 jours et le terme
- Passé le terme, la qualité est nulle

Les articles communs

Ajoutons deux autres tests :

- Par défaut, la qualité diminue quotidiennement
- Par défaut, elle diminue deux fois plus vite une fois le *sellIn* expiré

Ajoutons également des tests vérifiant que la qualité est toujours comprise entre 0 et 50, et nous obtenons une couverture de 100%.
GitHub : https://github.com/athiefaine/gilded-rose-kata/tree/domain_tests

Mutation testing

Toutes les lignes de notre code sont couvertes, mais cela ne veut pas dire que chaque ligne de code est pertinente ni que les valeurs importantes sont couvertes. Pour s'en assurer, nous utilisons PIT, un outil de mutation testing. La mise en oeuvre est assez simple, nous lançons la tâche dédiée du build et allons consulter le rapport généré.
On se rend compte que certaines mutations ont survécu. Elles sont toutes du même type et liées aux bornes de conditions. L'outil a produit des mutations (en rouge ci-dessous) en remplaçant l'opérateur < par <=, et lance une alerte si les tests passent quand même. 3
En fournissant des valeurs de *sellIn* plus proches des bornes dans les tests, on arrive à tuer les mutations.
Avec une couverture de test à 100% accompagné d'un mutation testing pour lequel aucune mutation ne survit, on peut estimer que nos tests orientés métier sont très solides.
GitHub : https://github.com/athiefaine/gilded-rose-kata/tree/mutation_testing

ET L'ÉVOLUTION DEMANDÉE, ELLE VIENT ?

En retrouvant à nouveau à une couverture de tests optimale, de nouvelles possibilités s'offrent à nous.
Il est temps de mettre le *golden master* (classe de test + implémentation legacy) à la poubelle, l'ensemble de tests orientés domaine étant suffisamment couvrant.
Et nous pouvons enfin réaliser ce pourquoi nous avons été sollicités, c'est-à-dire permettre de gérer une nouvelle classe d'articles, les Conjured. Leur qualité diminue deux fois plus vite que pour les articles classiques.
La meilleure façon de procéder est de suivre une approche en TDD (test-driven development), à savoir écrire les tests décrivant la nouvelle fonctionnalité, puis de faire fonctionner l'implémentation.
On commence par un premier test qui vérifie que la qualité diminue deux fois plus vite. Il échoue, on gère le cas pour faire fonctionner.
Un second test permettra de vérifier le cas où l'article est périmé (sellIn < 0), avec une qualité qui diminue alors de 4 points par jour.
Comme on travaille sur du code réécrit et relativement lisible, supporter cette nouvelle fonctionnalité est très facile. Voici une implémentation très naïve mais qui fonctionne :

```
if (item.name.equals("Conjured")) {
  decreaseQuality();
  if (item.sellIn < 0) {
    decreaseQuality();
    decreaseQuality();
  }
}
```

```
void updateQuality() {
    if (item.name.equals("Sulfuras, Hand of Ragnaros")) {
        return;
    }

    item.sellIn = item.sellIn - 1;

    switch (item.name) {
        case "Aged Brie":
            increaseQuality();
            if (item.sellIn < 0) increaseQuality();
            break;
        case "Backstage passes to a TAFKAL80ETC concert":
            increaseQuality();

            if (item.sellIn < 10) increaseQuality();
            if (item.sellIn < 5) increaseQuality();
            if (item.sellIn < 0) item.quality = item.quality - item.quality;
            break;
        case "Conjured":
            decreaseQuality();
            decreaseQuality();
            if (item.sellIn < 0) {
                decreaseQuality();
                decreaseQuality();
            }
            break;
        default:
            decreaseQuality();
            if (item.sellIn < 0) decreaseQuality();
            break;
    }
}
```

3

Il reste encore plusieurs possibilités d'amélioration du code, mais nous allons nous arrêter là, l'objectif d'implémenter de façon fiable l'évolution demandée avec un code lisible étant atteint.

Rétrospective

On retrouve sur cette seconde partie les enseignements suivants :

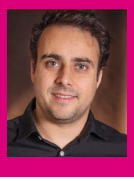
- Il est important de prendre du recul et d'expérimenter des approches contre-intuitives, notamment en augmentant de façon temporaire la duplication
- Il est crucial de s'appuyer sur les outils (IDE, couverture de code) pour éviter de se faire trop de noeuds au cerveau
- La couverture de code peut fournir un faux sentiment de sécurité, il est important d'en comprendre les limites et le cas échéant de la compléter avec des approches complémentaires comme le mutation testing

Nous avons également découvert les aspects suivants :

- Il n'est pas nécessaire de comprendre tout de suite ce que fait le code, et qu'en se basant sur des outils et automatismes la compréhension va émerger progressivement
- Il faut lancer les tests à la moindre modification pour bénéficier d'un feedback permanent
- On peut refactorer de plusieurs façons différentes, certaines permettant de vraiment s'amuser
- Le *golden master* doit rester éphémère, c'est un échafaudage nécessaire au départ mais qui doit disparaître une fois le code rendu plus clair
- Il est important de savoir s'arrêter, sans rechercher la perfection du code : le fait que le code soit compréhensible, permettant d'écrire des tests proches du domaine et de réaliser les évolutions demandées constitue un bon signal

Remerciements

Je tiens à remercier encore une fois les personnes suivantes pour le temps passé à relire mon texte, et les différents conseils, idées et encouragements qu'ils ont pu me prodiguer : Dorra Bartaguiz, Alexia Hoang, Benjamin Hugot, Arnauld Loyer, Cyrille Martraire, Laury Maurice, Hadrien Mens-Pellen, Nicolas Morin et Yvan Phélizot..



Steve Houël
Architecte Cloud & DevOps chez Ippon Technologies, évangéliste Serverless et contributeur sur des projets Open Source comme JHipster et daSWAG. Il est fier d'être un #GeekEnthusiast.

IPPON
Discovery to Delivery

Architecture Serverless : de la Théorie à la pratique 3/3

Aujourd'hui, de nouveaux services font parler d'eux dans le monde de l'IT. Lorsque l'on parle de fournisseur de services Cloud, c'est principalement le Serverless. Dans cet article, nous nous intéresserons aux différentes étapes qui constituent la réalisation d'un projet basé sur les services Serverless et aux impacts que ce type de technologie peuvent avoir sur notre conception.

niveau 200

La sécurité

Qui dit nouvelle technologie dit aussi nouvelles perspectives d'attaques et nouvelles vulnérabilités, ainsi que nouveau partage des responsabilités entre l'utilisateur et le fournisseur de services. Considérons le modèle suivant : 5

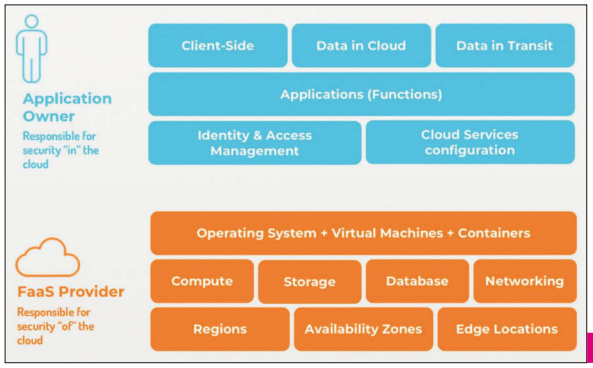
Les points de vigilance de ce type d'architecture peuvent se catégoriser en 4 parties :

- **Augmentation de la surface d'attaque** : comme les fonctions Serverless consomment des données provenant de multiples sources d'évènements telles que les API HTTP, les files d'attente de messages, le stockage et même de périphériques IoT, la surface d'attaque va en conséquence croître. Cela induit aussi l'utilisation de protocoles et de nouvelles structures de données complexes, qui sont difficiles à inspecter par un pare-feu applicatif Web.
- **Complexité d'utilisation** : d'emblée, la surface d'attaque de l'architecture est relativement nouvelle, d'où la difficulté d'adaptation et d'évolution pour les développeurs ; la probabilité d'une mauvaise configuration est très élevée.
- **Complexité globale du système** : il est très difficile de visualiser et de surveiller les applications développées avec des architectures Serverless car ce n'est pas un environnement logiciel typique et dans les ¾ des cas de figures, des solutions de supervision seront déjà présentes et ne fourniront pas les briques nécessaires à l'exploitation de ce type de services (intégration spécifique, granularité par fonction, ...). Par conséquent, l'enregistrement des évènements et des métriques sont essentiels pour le dépannage et l'intervention lors d'évènements de sécurité.
- **Insuffisance des tests de sécurité** : les tests de sécurité sur les applications construites sur des architectures Serverless sont beaucoup plus complexes que les applications standard. Ceci est principalement dû au manque d'information sur l'environnement d'exécution. Nous rentrons alors dans le partage des responsabilités entre vous et votre fournisseur de services. Le périmètre des développeurs se restreint à la gestion des données et du code source, et nous permettra de tendre à des pratiques comme le DevSecOps.

Etudions plus précisément les 10 risques les plus importants lors de l'utilisation de ce type de technologie. Nous nous baserons pour cela sur l'étude faite par l'OWASP et le livre blanc réalisé par PureSec (https://www.puresec.io/hubfs/The-12-Most-Critical-Risks-for-Serverless-Applications.pdf).

(SAS-1) - Injection de données événementielles

Il n'est pas étonnant que l'injection d'évènements arrive en tête de la liste du Top 10 de l'OWASP comme étant le défaut le plus critique qui soit sur ce type d'architecture. Cette faille se produit lorsqu'un évènement non fiable est transmis directement à un interpréteur et est exécuté ou évalué par une fonction.



(source : https://www.puresec.io) 5

La plupart des architectures Serverless fournissent et acceptent une multitude d'évènements comme source de leur invocation (fichiers, base de données, requête HTTP, file de messages, notifications, ...). Cet ensemble de sources d'évènements augmente la surface d'attaque potentielle et introduit des complexités lorsqu'on tente de protéger nos fonctions contre ces injections de données, d'autant plus que ces nouveaux services ne sont pas aussi bien compris que les environnements Web actuels où les développeurs connaissent les éléments sensibles des messages (exemple des paramètres GET/POST, les entêtes HTTP, etc.).

(SAS-2) - Authentification (permission d'invocation)

Contrairement aux architectures traditionnelles, les fonctions Serverless s'exécutent dans des conteneurs Stateless. Cela signifie qu'il n'y a pas un seul grand flux géré par un serveur unique, mais plutôt des centaines de fonctions différentes qui fonctionnent séparément. Il est obligatoire d'appliquer des schémas d'authentification robustes, qui assurent un contrôle d'accès et une protection appropriée à chacune des fonctions (type d'événement déclencheur, permissions).

Dans notre cas, les attaquants tenteront de rechercher une ressource oubliée, comme un stockage rendu public ou des API ouvertes. Attention cependant, les ressources externes ne devraient pas être la seule préoccupation. Par exemple, si une fonction est déclenchée par les e-mails d'un domaine précis, les attaquants pourront envoyer des courriels usurpés et ainsi déclencher une fonction sans fournir de jeton d'authentification.

(SAS-3) - Configuration de déploiement non sécurisée

Les services cloud en général et les architectures Serverless en particulier offrent de nombreux paramètres de configuration afin de les adapter à chacun de vos besoins ou environnements. Certains de ces paramètres doivent faire l'objet d'une attention particulière vu l'impact possible sur votre sécurité, nous prendrons ici l'exemple des services de stockage trop permissifs pouvant exposer des données sensibles.

(SAS-4) - Permissions et rôles d'exécution des fonctions

Il est toujours sage de suivre le principe du "Least Privilege". Techniquement, cela signifie que les fonctions ne doivent recevoir que les privilèges nécessaires à l'exécution de la logique voulue. Si pour une raison ou une autre votre fonction dispose de droits plus permissifs, il vous sera alors possible d'exécuter des actions non voulues et ainsi de récupérer des informations sensibles. Ce cas est d'autant plus vrai lors de l'usage d'un pattern de type monolithe composé d'une seule fonction pour l'ensemble de votre application.

(SAS-5) - Insuffisance de supervision et de monitoring

Un élément clé lors d'une attaque est votre capacité à la détecter et à répondre à cet incident en temps réel. De nombreuses attaques pourraient être évitées si ces briques de supervision étaient implémentées correctement. L'un des aspects clé d'une architecture Serverless est qu'elle réside dans un environnement Cloud en dehors du périmètre de sécurité d'une organisation. Nos anciens processus et outils de sécurité ne fonctionnent plus sur ce type de technologies et deviennent obsolètes. Même si de nombreux fournisseurs de services proposent des solutions clé en main pour la supervision et le monitoring de vos fonctions, elles n'ont malheureusement pas la granularité requise pour garantir l'audit des évènements de sécurité. Il faudra fournir un effort supplémentaire par le biais de vos développeurs pour ajouter les différentes briques de supervision nécessaires.

(SAS-6) - Dépendances applicatives compromises ou obsolètes

Techniquement, une fonction Serverless est un petit morceau de code qui exécute une seule tâche. Parfois, pour effectuer cette tâche, la fonction aura besoin de dépendances tierces ou même de consommer des services Web distants par le biais d'appels API. Vous serez alors sujet aux problématiques de tout développeur d'applications lors de la gestion de dépendances et de leurs possibles obsolescences ou vulnérabilités.

(SAS-7) - Stockage non sécurisé des éléments de configuration

Au fur et à mesure que la taille et la complexité des applications augmentent, il est nécessaire de stocker et maintenir des paramètres tels que : Clés API, URL de connexion à une base de données, informations d'authentification, etc. L'une des erreurs les plus courantes est l'exposition de ces paramètres en clair sans sécurisation. Ainsi, n'importe quel utilisateur possédant des droits en lecture sur une fonction pourra récupérer ces informations possiblement sensibles directement dans le code, ou via l'utilisation de variables d'environnements.

(SAS-8) - Déni de service (DDoS) et épuisement des ressources financières

Les attaques par Déni de Service ont fait parler d'elles dans l'actualité. L'objectif des attaquants est de surcharger l'infrastructure d'une application afin de la rendre indisponible ou de créer des effets de bords, par exemple un impact conséquent sur la facture du fournisseur de services. Malheureusement, les architectures Serverless ne vont pas échapper à cette règle, même si elles vont nativement garantir une certaine élasticité et absorber cette montée en charge.

(SAS-9) - Manipulation de votre logique business

La manipulation de la logique métier de votre application n'est pas un problème spécifique au Serverless mais commun à de nombreuses applications. Dans notre cas, il sera amplifié par le fait que nos fonctions sont généralement construites dans un contexte micro-services avec possiblement des enchaînements logiques dans nos invocations. Une fois exploitée, cette faiblesse va permettre aux attaquants d'exécuter des fonctions en dehors de leurs scopes initialement prévus et pourraient ainsi contourner les phases d'authentification.

Afin d'illustrer au mieux ce point nous prendrons l'exemple suivant :

- Une fonction permet l'upload et la validation d'un fichier sur un bucket S3,
- Suite à l'insertion sur le bucket S3, cela invoquera une autre fonction qui procédera au traitement de ce fichier.

Si par mégarde la configuration de votre stockage est trop permissif, il vous sera possible d'envoyer directement un fichier et de lancer la phase de traitement de celui-ci sans passer par l'étape de validation.

(SAS-10) - Traitement incorrect des exceptions et verbosité des messages d'erreur trop grande

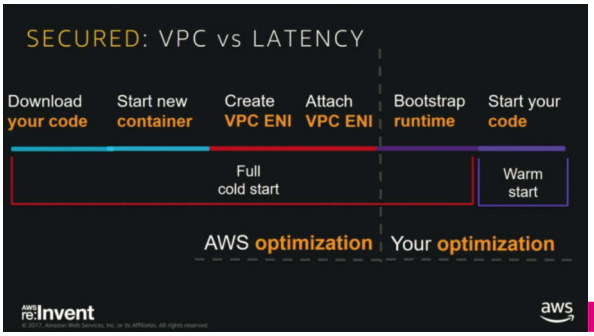
Considérant la maturité actuelle de l'écosystème Serverless, les développeurs manquent d'outils, principalement durant les phases de debugging et de tests. L'une des erreurs que l'on rencontre souvent est l'utilisation de logs avec une importante verbosité, laissés à l'abandon par les développeurs. Ceux-ci sont généralement accessibles directement via un service dédié du cloud provider et fourniront des informations sensibles sur votre logique business ou vos paramétrages aux attaquants.

Quelques astuces

Optimiser son cold start

Faisons un petit rappel sur les différentes phases qui surviennent lors de l'exécution d'une fonction : 6

Si on a pour objectif d'optimiser ce temps, notre périmètre d'action pourra se faire sur plusieurs niveaux :



6

- Diminuer la taille de l'archive pour minimiser la phase de téléchargement du code. Ici, il faut penser à la gestion des dépendances vu qu'elles représentent généralement le plus gros volume de votre archive.
- Minimiser la phase d'initialisation du runtime. Il ne faut charger et initialiser que ce dont vous avez besoin pour l'exécution de votre code.
- Trouver le bon ratio coût / performance pour choisir le meilleur paramètre mémoire de votre fonction (certains projets existent pour vous aider dans ce choix : https://github.com/alexcasalboni/aws-lambda-power-tuning)
- Eviter au maximum de déployer vos fonctions dans un environnement réseau (VPC). Cela supprimera ainsi les phases de création et d'attachement de l'interface réseau (ENI).

Prenons un exemple simple d'étude de l'impact du chargement des dépendances AWS et X-Ray sur votre démarrage. Pour ce faire nous nous baserons sur une étude menée par Yan Cui (AWS Hero Serverless).

Nous observons que :

- Sans aucune dépendance, le temps d'initialisation est en moyenne de 1,72 ms.
- L'ajout du SDK AWS comme seule dépendance ajoute une moyenne de 243 ms. C'est assez significatif.
- En important uniquement le client DynamoDB, cela permet d'économiser jusqu'à 124 ms !
- Le coût de l'utilisation du SDK X-Ray est à peu près le même que celui du SDK AWS.
- Il n'y a pas de différence statistiquement significative entre l'utilisation du X-Ray SDK complet et du X-Ray SDK Core.
- Certaines briques technologiques comme Webpack (pour le langage NodeJS) via le plugin serverless-webpack (https://github.com/serverless-heaven/serverless-webpack) peuvent impacter positivement vos chargements.

Vous l'aurez compris, l'import de vos dépendances nécessitera une attention particulière.

```
const DynamoDB = require('aws-sdk/clients/dynamodb')
const documentClient = new DynamoDB.DocumentClient()
```

Et de ne plus importer l'intégralité d'un framework comme joint.

```
const AWS = require('aws-sdk')
module.exports.handler = async () => {
}
```

Configuration de votre fonction

Lorsque vous utilisez du FaaS, vous serez vite amené à devoir passer des paramètres à votre code. Ceux-ci pourront être de simples valeurs non sensibles, vous pourrez alors aisément les passer au travers de variables d'environnements définies directement dans le paramétrage de votre fonction.

```
CostExplorerReport:
 Type: 'AWS::Serverless::Function'
 Properties:
  FunctionName: platform_cost_explorer_report
  Description: "Function for generating / sending monthly cost report"
  MemorySize: 256
  Timeout: 60
  Handler: src.lambda.main_handler
  Runtime: python3.6
  CodeUri: ./bundle.zip
  Role: !GetAtt CostExplorerReportLambdaIAMRole.Arn
  Environment:
   Variables:
    S3_BUCKET: !Ref S3ReportBucket
    SES_SEND: !Ref SESSendTo
    SES_FROM: !Ref SESSendFrom
    SES_REGION: !Ref SESRegion
    COST_TAGS: !Ref ListOfCostTags
    ACCOUNT_LABEL: !Ref AccountLabel
    CURRENT_MONTH: !Ref CurrentMonth
    INC_SUPPORT: 'false'
```

Le problème se posera lors du passage d'informations sensibles comme des identifiants ou une URL de connecxion à votre base de données. Vous aurez alors les contraintes suivantes :

- Vos données devront être chiffrées at rest ;
- Vos données devront être chiffrées in-transit ;
- Appliquer le principe du "Least Privilege" à votre fonction et vos données.

L'intégration de plugins spécifiques pourra résoudre ces points (exemple du serverless-secrets-plugin), vous pourrez aussi faire appel à des solutions tierces comme *Consul* ou *etcd* pour ces problématiques, en revanche elles vous feront revenir dans la gestion opérationnelle de vos serveurs, ce qui n'est pas trop Serverless. La chance pour nous c'est que certains fournisseurs de services proposent des solutions embarquées comme **SSM Parameter Store** chez AWS qui adressera l'ensemble de vos besoins.

Avoir un endroit centralisé pour stocker les paramètres n'est qu'un côté de la médaille. Vous devrez tout de même investir des efforts dans la création d'une bibliothèque client robuste, facile à utiliser et qui prend en charge :

- mise en cache et son expiration ;
- changement à chaud au sein de votre fonction lorsque la valeur change.

Bien sûr, cela dépendra de vos cas d'utilisations, rien ne vous oblige à implémenter cette capacité de rafraîchissement de vos valeurs. Vous pourrez alors lier directement ces paramètres à vos variables d'environnements (nativement via le Serverless framework).

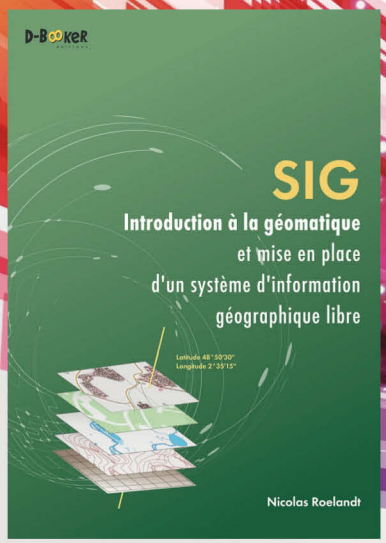
Conclusion

Nous avons pu voir au travers de ces différentes phases de conception d'un projet Serverless que la complexité n'est pas des moindres et que même si nous retrouvons des problématiques liées au monde du développement applicatif en général, certains aspects restent spécifiques au Serverless et peuvent être structurants pour votre projet et votre entreprise. Le choix du langage de programmation est un parfait exemple, au travers du Serverless, des langages comme le Python, le NodeJs ou le Go sont mis en avant au détriment du Java et du .Net. Cela aura un impact certain sur votre gestion des compétences et pourra même être un facteur bloquant pour certaines entreprises vu la tension actuelle du marché et la difficulté à recruter ce type de profils très techniques (compétences dans le développement et dans les services Cloud). •

DERNIÈRES PARUTIONS

D-BOOKER
éditions

e-books
chapitres à l'unité
livres imprimés



www.d-booker.fr

Bilan de projet

Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : S. Saurel, la rédaction de ZDNet, Commitstrip
Nos experts techniques : J-N Gerard, W. Chegham, C. Pichaud, A. Giretti, M. Bacci, R. Huart, M. Rahim, L. Mathieu, A. Thiefaine, S. Houel
Couverture : Angulaire, Python, Oracle, D.R. - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@programmez.com.
Imprimeur : Moderna Printing, Paal-Beringen, 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, septembre 2019

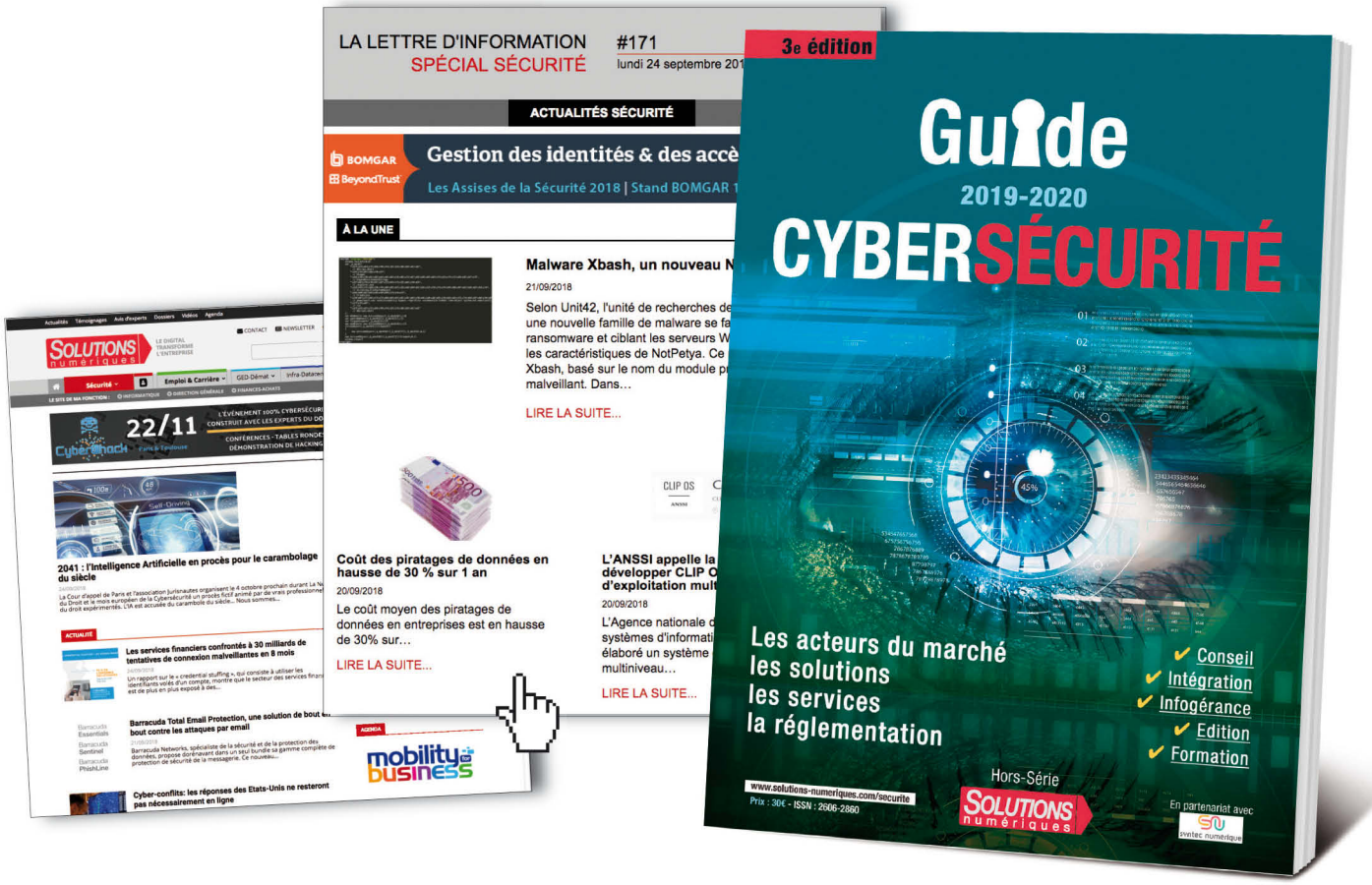
Abonnement :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com*
Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30.

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

Cybersécurité : RESTEZ INFORMÉ

Nouvelle édition Octobre 2019



- **L'actualité quotidienne**
  News, avis d'experts, témoignages, livres blancs, etc.
  https://www.solutions-numeriques.com/securite/

- **La newsletter**
  Chaque lundi, comme 40 000 professionnels et décideurs, recevez la synthèse des informations.
  C'est gratuit, inscrivez vous :
  https://www.solutions-numeriques.com/securite/inscription/

- **L'annuaire en ligne**
  Trouvez l'éditeur de solution, le prestataire de services qu'il vous faut.
  https://www.solutions-numeriques.com/securite/annuaire-cybersecurite/

barnsten

Crée une application avec l'IDE Delphi.

Compétition spéciale rentrée 2019

Élection du meilleur jeune développeur Delphi.

Tu es étudiant et tu veux apprendre à développer avec un des langages les plus utilisés dans le monde ? Cette compétition est faite pour toi !

Montre-nous ce que tu sais faire en développant une application utilisant l'IDE Delphi (version 10.3 minimum) et tente ta chance pour remporter un iPad et/ou une licence RAD Studio !

Inscris-toi sur www.barnsten.com/election et **rejoins une communauté de plus de 3 millions de développeurs à travers le monde** (et reçois une licence d'essai gratuite de 30 jours) !

Prix des gagnants :

1er Prix
Un iPad Pro 12,9-inch 64 GB (valeur de €1,019) et une licence RAD Studio Architect (valeur de €7,275).

2ème Prix
Une licence RAD Studio Enterprise (valeur de €4,850).

3ème Prix
Une licence RAD Studio Professional (valeur de €2,922).

Présentation et remise du 1er prix à Paris, lors de la prochaine conférence Delphi le 19 novembre 2019.

Les frais de transports vers l'événement pour le gagnant seront pris en charge par Barnsten (limité à la France métropolitaine).

Membres du jury :
Maxime Capellot, Paul Toth et Patrick Premartin.

Conditions d'inscription :

- Être en cursus scolaire ou formation en but d'obtention d'un diplôme.
- Proposer une application utilisant l'IDE Delphi (version minimum 10.3).
- Être français.
- Livrer le projet avant le 7 novembre 2019 à minuit.

Les inscriptions sont limitées à 100.

Rendez-vous sur www.barnsten.com/election pour t'inscrire !

Des questions ? N'hésite pas à nous contacter au 09 72 19 28 87 ou par email equipe@barnsten.com.