JAVA 11 | JHIPSTER | ROOTKIT | PYTHON | OPENFAAS | TENSORFLOW

programmez.com

[Programmez!]

Le magazine des développeurs

225 JANVIER 2019

TOP OUTILS 2019

JHIPSTER

Générer rapidement vos codes & applications

FUTUROLOGIE 2019
TOPS & FLOPS

DÉVELOPPEMENT MOBILE
Ionic, React, Xamarin, Cordova, Natif vs hybride… Comment choisir ?

M 04319 - 225 - F: 6,50 € - RD

JHipster

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

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

Rejoignez
notre cordée !

elcimaï LE GROUPE

INFORMATIQUE

LA PERFORMANCE
NE DOIT RIEN AU HASARD

Melun

Paris

Elcimaï, éditeur informatique en plein développement
recrute de forts potentiels sur Melun et Paris

Vous êtes :

- **Ingénieur études et développement .Net C#** H/F
- **Ingénieur support applicatif** H/F
- **Ingénieur d'études C / C++ Unix** H/F
- **Ingénieur études et développement Java** H/F
- **Ingénieur études et développement Sharepoint** H/F
- **Consultant AMOA banque** H/F
- **Consultant AMOA assurance** H/F

Retrouvez toutes nos opportunités de carrières sur
www.elcimai.com - E-mail : **candidature@elcimai.com**

Vous interviendrez sur :
Expertise Technique (MOE) C / C++ ;
C#, .Net, JAVA, J2EE, sharepoint…
Expertise et assistance Métier/ Fonctionnelle
(MOA et AMOA) : banque et assurance

EDITO

Low tech : un retour en arrière ou une réelle solution ?

L'informatique est une fuite en avant depuis les années 70 sur les matériels et les logiciels. Et avec la dématérialisation des logiciels, les apps mobiles, le cloud computing, toute une partie de la technologie ne se voit plus, même si elle est toujours là. Parfois, on s'obstine à sortir les grosses plateformes pour générer des projets légers, nécessitant à peine quelques semaines de développement. Est-ce une bonne chose ? Une approche low code ne serait-elle pas plus adaptée ?

Un constat résume l'inflation technique : en 2010, une page web pesait en moyenne moins de 500 Ko, en 2018, nous sommes à x3 ou x6 ! Et ce n'est pas terminé. Cette inflation suit les performances des terminaux et des réseaux et la multiplication des couches techniques. Une app web se compose d'une multitude de technologies, sur le front et sur le back. Nous ne pouvons que constater la complexité croissante de la plomberie !

Nous avons récemment mis à jour notre suite bureautique, résultat ? 2 Go de mise à jour pour 3 logiciels ! Vous me direz que ce n'est absolument pas nouveau. Mais nous sommes en train d'atteindre des sommets dans le poids des apps. Il y a de quoi se poser des questions. Qui est responsable de ce délire ? Les éditeurs, les développeurs, les utilisateurs ? Nous sommes tous responsables. Malheureusement, nous ne voyons pas d'accalmie. Or, des millions d'utilisateurs téléchargent ces mises à jour. Les conséquences sont multiples : surcharge des serveurs, forte utilisation des réseaux, consommation électrique côté utilisateur. Et s'il n'y a pas assez place sur son terminal, que fait-on ? On supprime d'autres logiciels ? On change de matériel ? On ne prend pas en compte la MàJ ? Or, ce n'est pas toujours possible…

Bienvenue en 2019 !

François Tonic
ftonic@programmez.com

SOMMAIRE

Dans le prochain numéro !
Programmez! #226, dès le 1er février 2019

Angular 6 : *les nouveautés*
WebAssembly :
un nouveau langage avec les performances du C++ sur son navigateur web

Visual Basic.net fait de la résistance

Le langage de programmation de Microsoft se hisse à la cinquième place des langages les plus populaires selon l'index Tiobe. La nouvelle pourrait surprendre, mais il convient de détailler ici le fonctionnement de l'index Tiobe : celui-ci se base sur le nombre de résultats Google pour évaluer la popularité d'un langage. Un grand nombre de créations de pages web au sujet de Visual Basic .Net pourrait donc avoir fait remonter sa cote au sein de l'index : celui-ci affiche en effet une croissance solide depuis bientôt un an et demi.

Sur npm, une porte dérobée se glisse dans un module populaire

Une porte dérobée a été découverte au sein d'un module JavaScript mis à disposition sur la plateforme npm. Ce module « event-stream » était utilisé dans de nombreux projets et un nouveau mainteneur du module avait ajouté une librairie contenant du code malveillant. Celui-ci visait à siphonner les cryptomonnaies contenues dans les porte-monnaies développés par la société Copay et ne s'activait que lorsqu'il reconnaissait l'environnement de travail des employés de Copay. Le code malveillant a depuis été repéré et des versions corrigées du module en question ont été republiées. Plus de peur que de mal au final, mais les utilisateurs de l'application en question sont passés à deux doigts d'un cryptobraquage de haute volée.

Navigateurs : Edge se prépare à faire le grand saut

Après trois ans de bons et loyaux services, Microsoft s'apprête à abandonner le moteur de rendu de son navigateur Edge au profit d'un nouveau moteur open source. Celui-ci sera basé sur Chromium et abandonnera ainsi son moteur de rendu maison « EdgeHTML ». Cette évolution de l'architecture du navigateur lui permettra notamment de bénéficier du support de webextensions, la technologie d'extensions de navigateurs déjà utilisée par Firefox et Chrome. Edge rejoindra ainsi Brave, Vivaldi et Opera qui utilisent également le moteur de rendu de Chromium. Les alternatives restent Safari, qui a recours au moteur de rendu WebKit, et Firefox qui fonctionne grâce à son moteur de rendu Gecko.

Kubernetes : plus c'est gros, plus ça craint

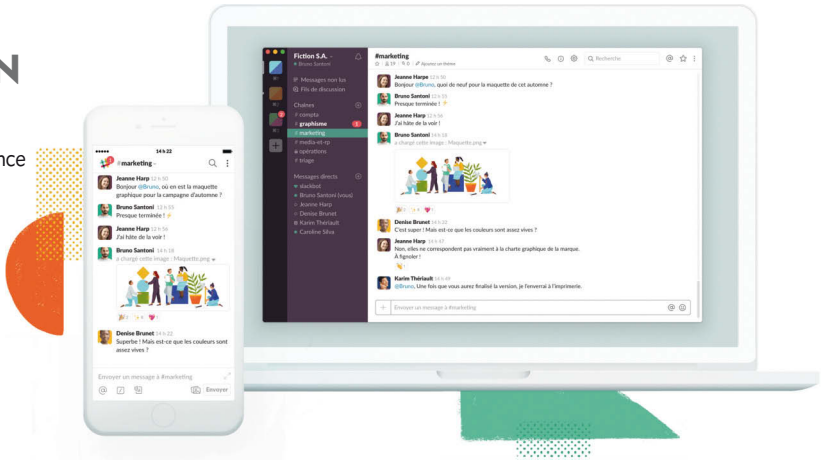
Kubernetes est aujourd'hui devenu l'un des principaux acteurs sur le marché des solutions de gestion des conteneurs open source. Mais cette popularité en fait également une cible pour les cybercriminels. Dans ce contexte, la nouvelle faille de sécurité découverte au début du mois de décembre avait toutes les raisons de susciter l'inquiétude : la faille CVE-2018-1002105 permettait ainsi une élévation de privilèges au sein d'OpenShift Container Platform. Cette vulnérabilité est présente dans les versions 3.6 et supérieure de la plateforme. Des correctifs ont été publiés dans les versions 1.10.11, 1.11.15, 1.12.3 et 1.13.0 de Kubernetes. Si la découverte de cette faille ne suffisait pas à vous motiver à patcher, on ajoutera que depuis le début du mois de décembre, plusieurs preuves de concept exploitant la vulnérabilité en question sont disponibles sur Github et peuvent donc être facilement adaptées à des fins malveillantes.

Fermeture de GOOGLE + : un nouveau bug d'API accélère le calendrier

Un premier bug dans une API utilisée par Google + découvert au mois d'octobre avait poussé Google à annoncer la fin du service pour août 2019. Mais dans le cadre de son audit en profondeur, de la sécurité du réseau, Google a trouvé une seconde vulnérabilité dans les API utilisées par Google +, qui permettait cette fois d'accéder aux données personnelles d'environ 52 millions d'utilisateurs, contre 500 000 pour la première faille de sécurité. En conséquence, Google annonce que son réseau social fermera ses portes dès janvier 2019. Au rythme où ça va, on peut comprendre que Google préfère remettre son réseau social au placard avant d'avoir à annoncer une nouvelle faille de sécurité affectant cette fois 500 millions d'utilisateurs.

SLACK VISE UNE ENTRÉE EN BOURSE EN 2019

Le service de messagerie entend passer le cap en 2019. Selon l'agence de presse Reuters, la société américaine aurait engagé la banque Goldman Sachs pour préparer son entrée en bourse. Slack rejoint ainsi la longue liste des sociétés de la Silicon Valley qui entendent faire de même en 2019. Slack viserait une valorisation de 10 milliards de dollars, un sacré bout de chemin parcouru pour une entreprise qui ambitionnait à ses débuts de lancer un jeu vidéo.



WEBDEV 24: ATELIER DE GÉNIE LOGICIEL PROFESSIONNEL

CRÉEZ DES SITES WEB RELIÉS À VOS BASES DE DONNÉES (LOCAL, RÉSEAU, CLOUD)
LA SÉCURITÉ DE VOS DONNÉES EST ASSURÉE

LE COIN MAKER

Tech Inn'Vitré : du 1er au 3 mars à Vitré.
TFEEA : le 5 mars à Paris (ministère de l'Économie). Makers et travailleurs en situation de handicap.
Metromix : 28 & 29 mars (hackathon sur la mobilité urbaine).
Makeme Fest Nantes : du 12 au 14 avril sur la Foire de Nantes.
Makeme Fest Angers : du 25 au 29 avril sur Foire d'Angers.
Maker Faire France organisera d'ores et déjà plusieurs évènements :
26 & 27 janvier : mini-Maker Faire à Grenoble
1er & 3 mars : Maker Faire Lille

CONCOURS PROLOGIN

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

Girls Can Code !

Les stages Girls Can Code! c'est une semaine d'initiation à la programmation pour les collégiennes et lycéennes. On y apprend les bases de la programmation avec le langage Python, et de quoi se lancer dans des petits projets de son choix ! Les stages sont gratuits, entièrement organisés par des bénévoles de l'association Prologin.

Au programme : apprentissage du code, ateliers pratiques, réseaux. Une occasion de découvrir le métier de développeur et le monde de l'informatique. Une excellente initiative que nous ne pouvons que soutenir !

Pour en savoir plus : https://gcc.prologin.org/

Baromètre Hired de la recherche d'emploi

Le Ruby et le DevOps ont cristallisé les demandes des recruteurs en novembre

Le Ruby et le DevOps sont à l'honneur ce mois-ci dans le baromètre Hired de la recherche d'emploi des développeurs. Si le premier est habitué aux places d'honneur, le second fait un retour fracassant sur le devant de la scène dont il avait été éclipsé depuis cet été.

Si les développeurs DevOps se sont toujours fait aussi rares sur la plateforme Hired (1,44 % des candidats), la demande a complètement explosé en novembre avec une moyenne de 7,9 demandes d'entretiens reçues en moyenne par candidats. Seul Ruby fait mieux avec 8,8 sollicitations par candidat. Mais si cette technologie est abonnée aux places d'honneur avec React (7,8), Node (6,6) et Go (5,7) qui complètent le top 5, le DevOps, lui, voit sa popularité fluctuer grandement suivant les périodes de l'année et les besoins des entreprises. Cette technologie n'en reste pas moins une valeur sûre pour les développeurs puisque sur les 6 derniers mois, les candidats l'ayant mise en avant sur leur CV se sont vus proposer en moyenne 6,4 demandes d'entretiens par mois. De l'autre côté du classement, .NET, iOS et Android figurent parmi les technologies les moins demandées par les recruteurs en novembre, mais restent toutefois attractives avec respectivement 4,4, 4,1 et 3,2 demandes d'entretiens suscitées pour les candidats les ayant mises en avant.

Du point de vue des candidatures, le classement évolue peu par rapport aux mois précédents avec toujours Java, Node et PHP parmi les technologies les plus mises en avant qui voient ce mois-ci le Python s'intercaler à la deuxième place. Bien que très représentées sur la plateforme Hired, ces technologies jouissent toujours d'un certain attrait pour les recruteurs avec un minimum de 4,7 sollicitations par mois (pour le Java) pour les développeurs les ayant mises en avant.

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

novembre 2018

Technologies demandées	Pourcentage de candidatures développeurs	Technologies demandées	Nombre moyen de demandes d'entretiens
Java	20.16%	Ruby	8.8
Python	14.61%	DevOps	7.9
Node	14.20%	React	7.8
PHP	11.11%	Node	6.6
React	10.29%	Go	5.7
Angular	9.05%	Vue	5.9
.NET	5.97%	Python	5.1
Go	4.73%	PHP	5,0
Android	3.70%	Angular	4.8
Ruby	2.47%	Java	4.7
Vue	2.26%	.NET	4.4
DevOps	1.44%	iOS	4.1
iOS	1.44%	Android	3.2

De juin à novembre 2018

Technologies demandées	Pourcentage de candidatures développeurs	Technologies demandées	Nombre moyen de demandes d'entretiens
Java	20.08%	React	7.3
Node	14.61%	Vue	6.8
Python	12.68%	Go	6.5
PHP	11.94%	Ruby	6.5
React	10.54%	DevOps	6.4
Angular	10.41%	Node	6.3
Android	5.54%	Python	6.0
.NET	4.27%	Angular	5.5
Go	3.40%	Java	4.9
Ruby	3.27%	PHP	4.8
iOS	2.80%	.NET	4.5
Vue	2.13%	Android	3.7
DevOps	1.13%	iOS	3.5

COMMUNAUTÉS

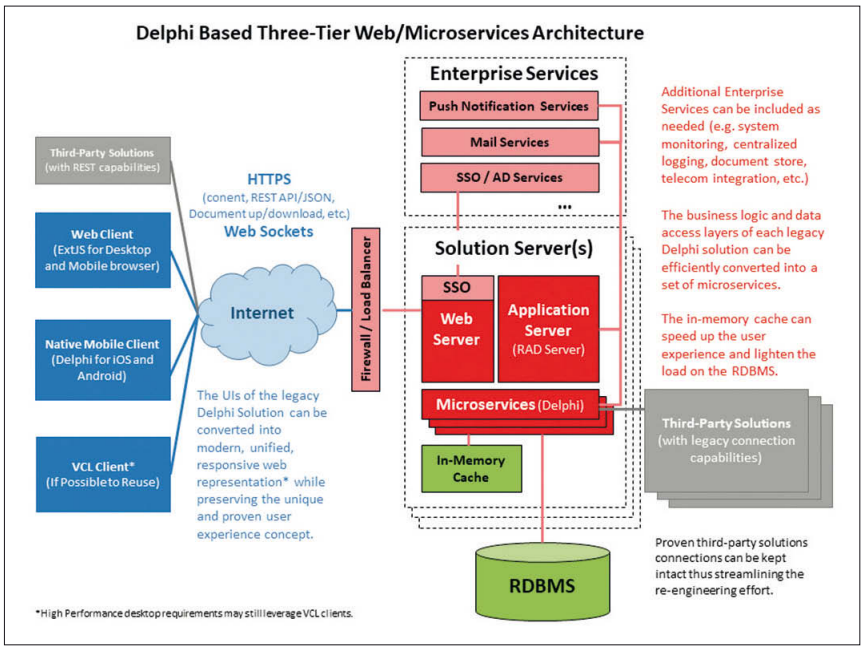
9/01 : meetup Drupal Montpellier
https://www.meetup.com/fr-FR/drupal-france-francophonie/events/rchtdpyzcbmb/
15/02 – 17/02 : Drupalcamp Paris 2019. Le grand évènement communautaire Drupal revient à Paris !
23/01 : C++ Enthousiasts ! Meetup C++ à Sophia Antipolis
https://www.meetup.com/fr-FR/CPP-User-Group-Sophia-Antipolis/events/257097244/
9/01 : meetup ReactXP à Valbonne
24/01 : visualization immersive des données au CERV – Best
https://www.meetup.com/fr-FR/BrestJS/
14/01 : Android Nantes #14, Nantes.
https://www.meetup.com/fr-FR/gdg_nantes_android/
8/01 : young blood VI / JUG Paris
15/01 : Minecraft, JUG Nantes - https://nantesjug.org
22/01 : product owner. Par Paris Agile Community
7/01 : mentors on rails, Paris.rb - https://www.meetup.com/fr-FR/parisrb/
8/01 : Paris Typescript #17 - https://www.meetup.com/fr-FR/Paris-Typescript/
15/01 : Progressive Web Apps, Paris.
https://www.meetup.com/fr-FR/Smile-With-Us/events/257104914/
23/01 : Magic Leap, par Glass Camp - https://www.meetup.com/fr-FR/glasscamp/

14 juin : DevFest Lille !

La grande conférence DevFest revient à Lille pour la 3e année ! Une occasion pour rencontrer les communautés et parler technologies ! L'appel aux sessions est valable jusqu'à fin mars ! https://devfest.gdglille.org

Modernisation de l'architecture des applications Delphi avec 10.3

Delphi est utilisé pour créer certaines des applications Serveur Client les plus puissantes au monde. De la défense à la santé, Delphi est un choix fiable, en particulier lorsque les clients recherchent des clients Windows de hautes performances. Le début des années 2000 a été entièrement concentré sur les architectures homogènes, Java et .Net, et ont gagné en popularité auprès des développeurs. Beaucoup de nos clients hésitaient à mettre à jour leurs applications Delphi, incertains quant à l'avenir. Certains ont essayé de changer de plate-forme à de grands frais et avec un niveau de risque élevé.



Aujourd'hui, le développement d'applications est plus hétérogène que jamais. Java et .NET sont à la croisée des chemins avec un niveau d'incertitude élevé quant à l'avenir et aux coûts. Les grandes entreprises ont reconnu que les technologies ont des objectifs différents. Même le légendaire JavaScript a de nombreuses variétés et structures qui ont tous des cas d'utilisation différents. La pénurie de développeurs qualifiés a conduit l'industrie à redécouvrir le RAD.

Delphi est particulièrement bien placé pour reprendre son élan. Il reste le langage le plus facile à apprendre pour les entreprises. La conception du langage et des outils permet des performances égales et parfois meilleures que Java. Il existe des outils et des structures de haute qualité pour rendre le développement extraordinaire, rapide et efficace. Le meilleur moment pour commencer à moderniser votre application Delphi n'a jamais été aussi parfait avec la sortie de RAD Studio 10.3.

QUE SIGNIFIE LA MODERNISATION DE L'APPLICATION DELPHI ?

La modernisation de Delphi nécessiterait probablement de passer d'un serveur client à une architecture à plusieurs niveaux. Le développement Delphi habituel permettait de réduire la séparation entre la logique applicative et le client Windows. Une application à plusieurs niveaux bien conçue nécessitera une séparation plus rigide des niveaux. Le processus de séparation de la logique applicative et du client s'adapte très bien à l'adoption de micro-services, devenus l'essentiel dans les architectures à plusieurs niveaux. Il s'agit d'un processus et, même s'il existe des analyseurs syntaxiques qui peuvent aider, des efforts et des ressources sont évidemment nécessaires. Bien sûr, il ne s'agit que d'une infime partie de ce qu'une nouvelle plate-forme sérieuse exigerait.

Les développeurs devraient se familiariser avec les nouvelles technologies associées, tant du côté serveur que du client. Les technologies du serveur Delphi ont récemment fait un bond en avant avec le RAD Server et plusieurs systèmes d'open source. RAD Server permet la publication et la gestion automatisées des REST/JSON API avec plusieurs services supplémentaires. L'hébergement de serveur peut nécessiter une compréhension des clouds et éventuellement de nouvelles plateformes. Linux est disponible pour RAD Server et est très économique, mais également nouveau pour de nombreux développeurs Delphi. Quant aux clients, ils doivent considérer notre technologie mobile native FMX et le client Web comme un effort de modernisation. FMX est très similaire à la VCL avec des nuances sur la configuration de l'environnement IDE. La plupart des développeurs Delphi apprennent très vite, mais il est évident que la maîtrise prendra un certain temps. Selon les cas d'utilisation déconnectés et les exigences de sécurité, les clients doivent envisager InterBase, qui comprend aujourd'hui de nombreuses fonctionnalités innovantes, telles que Change Views. Sur le site Web, les développeurs Delphi disposent de plusieurs options, mais pour des applications et des attelages plus grandes, il est probablement préférable d'envisager une structure JS Web. Nous proposons Ext JS, qui est la structure la plus robuste au niveau des entreprises. Il est très performant et comprend de nombreux composants prêts à l'emploi. Il adopte une approche familière aux développeurs Delphi. Un développeur JS générique doit maîtriser une courte courbe d'apprentissage.

EXEMPLE D'ARCHITECTURE

L'architecture ci-dessus a été mise en œuvre afin de moderniser une application de plus de 4 millions de lignes de code (par notre partenaire KER-Soft Kft). Le coût du projet était de 15% par rapport à la nouvelle plate-forme et la durée était de 6 mois contre 2 ans. L'application Delphi résultante présente certes des performances similaires à l'ancienne, mais dispose de plus de fonctionnalités et d'une interface utilisateur moderne. L'architecture fournit également un chemin flexible pour continuer à ajouter de nouvelles fonctionnalités, de manière plus rapide et avec moins de dépendances.

BARNSTEN EN PLEINE CROISSANCE

Barnsten se développe et depuis le mois de Novembre , nous sommes le distributeur officiel et le Centre de Support de tous les pays francophones (sauf Canada) dans lesquels Embarcadero est présent.

Pour plus d'informations visitez notre site:
https://www.barnsten.com

François Tonic

Air602 : une alternative aux ESP… en théorie

Encore une nouvelle carte sur le marché ! On peut la comparer aux ESP par le prix et le WiFi par défaut. Cette nano carte fonctionne comme une ATTiny avec le connecteur USB pour la connecter à son PC, dans son modèle Development Board.

Les +
- Taille
- Prix
- Alternative aux ESP
- WiFi

Les –
- Très faible documentation
- Manque d'outils
- GPIO très limité
- Retrait de LUA
- Communauté

Cette board, avec ou sans connecteur USB, utilise un SoC ARM, le Winner Micro W600. Ce module se classe dans la catégorie des MCU, donc des microcontrôleurs. L'ambition est grande sur les tarifs : 1,90 $ pour le module seul, 2,9 $ pour la carte complète ! Soit le prix des excellents ESP8266 (modèles premiers prix). Franchement, c'est impressionnant au regard des specs de la board.
Bien entendu, vu la taille de la carte, les I/O sont limitées mais suffisantes pour les usages courants.

Côté specs, nous trouvons :

Réseau	WiFi 802.11b/g/n à 2,4x Ghz Antenne intégrée
Interfaces / protocoles	UART SPI, GPIO, I2C, I2S, RTC
Voltage	3,3V (module) 5V (carte développement)
Compatibilité sécurité (accélération matérielle)	WEP, WPA-PSK, SHA1, AES, etc.
SoC	Winner Micro W600 basé sur um Cortex-M3, 128K + 160k de RAM, ROM intégrée

Côté pins disponibles sur la carte, le choix est limité :

GND
TX1
RX1
RTS
CTS
IO

En réalité, ces broches ont de multiples supports : UART, SPI principalement. Cela signifie que l'on connecte un nombre limité de capteurs mais vu l'usage prévu pour l'Air602 ce n'est pas forcément gênant. Elle se destine aux petits objets, bornes embarquées.
Par définition, avec peu de besoins et avec le WiFi, on peut faire du mesh en utilisant x modules et un point central comme une box domotique.

Comment développer avec ?

Air602 possède par défaut un firmware Luat. Celui-ci se compose de 4 fichiers mais le wiki officiel préconise d'utiliser le firmware AirM2M_Luat_V0011_W600T_USER. Pour la partie développement et interaction, on va utiliser AT. Vous l'aurez compris, cette board n'est pas forcément la plus conviviale. Et malheureusement, une grosse partie de la documentation officielle est en Chinois.
Tout n'est pas disponible en Anglais. Et c'est honnêtement un problème pour démarrer rapidement dessus. Heureusement, quelques tutos sont disponibles sur Seeed Studio.
Connectez la board directement à un port USB du PC. Nous avons eu quelques soucis de reconnaissance via un hub.
Ensuite, téléchargez deux archives : Firmware Tool et Air602 SDK. Le premier contient les outils nécessaires pour le développement. Celui qui nous intéresse : LuaTools.exe.
Seul problème : LuaTools est en Chinois et le chargement du firmware ne fonctionne pas toujours… Impossible d'avoir une connexion Putty. Par contre, on accède facilement aux commandes AT. Ouf ! Mais c'est un peu l'âge de pierre.

Conclusion incomplète

La frustration arrive très vite avec cette carte. Dommage que Seeed Studio n'ait pas sorti la documentation complète ni les outils nécessaires en Anglais. Autre grosse déception, l'exclusion du langage LUA alors qu'il avait été annoncé. Le manque de tutoriels est particulièrement frustrant.
Bref, pour le moment, l'AIR602 n'a que peu d'intérêt en l'absence d'outils et de langages de programmation.
Espérons que Seeed sorte rapidement le nécessaire. Ce n'est pas la première fois que Seeed sort une carte prometteuse mais sans support derrière. •

UPDUINO V2

Dans la catégorie minimaliste, nous avons l'UPDuino v2, de type Arduino mais équipé d'un FPGA (Lattice Ultraplus) et d'un connecteur USB. Elle propose 1 Mo de SPRAM, 34 GPIO, d'un voltage de 3,3V, le tout open source. Pour la partie programmation, on peut utiliser les outils Lattice. Cela signifie que le modèle de développement change, et n'est pas le classique Arduino IDE.

Il faut bien suivre les tutos notamment sur la partie configuration et de déploiement. Nous sommes dans des outils industriels qui n'ont rien à voir avec les outils Arduino classiques. Dans les exemples générés par le constructeur, on utilise le langage Verilog HDL. UPDuino peut vous aider à comprendre le FPGA sans investir dans des cartes chères. La carte proprement dite est vendue à 13,99 $.
Pour en savoir plus : http://www.gnarlygrey.com

A DÉCOUVRIR D'URGENCE

NOUVEAU !

LE CADEAU GEEK IDÉAL

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

9,99 €
(+ 3 € de frais postaux)

116 pages - Format magazine A4

☐ **Découvrez l'âge d'or des micro-ordinateurs de 1973 à 2007** : 9,90 € (+3 € de frais postaux) = 12,90 €

PROG 225

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

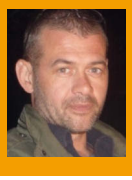
Adresse :

Code postal : Ville :

E-mail : @

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

commandez sur www.programmez.com



Dr Patrice Guichard
Expert en sécurité et en criminalistique
Patrice.guichard@celteam.com
www.celteam.com

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

Rootkit – key Logger

Partie 1

Avertissement : ceci n'est pas un tutoriel pour créer un rootkit, mais l'explication d'un cas pratique à des fins éducatives uniquement.

Ce mois-ci, nous abordons la sécurité et la programmation système au cœur du système d'exploitation via le développement de drivers (pilotes) ou d'outils de collecte de données. Les derniers outils volés à la CIA, la NSA ou à la société Italienne HackingTeam sont de ceux-là.

niveau 300

Introduction aux Rootkits

Avant de commencer à rentrer dans le vif du sujet concernant les rootkits, nous allons essayer de définir le plus simplement possible ce qu'est véritablement un rootkit et comment ils font pour se cacher dans les tréfonds de nos systèmes.

La première vocation d'un rootkit est d'être une « cape d'invisibilité », pour malwares et pour cela il peuvent se loger à différents endroits de votre ordinateur.

On distingue 3 types de rootkits :

- les Kernel-Rookits ou Rootkit (Ring 0) qui agissent là où tournent l'OS et les différents drivers ;
- sous divisions des Kernel-Rootkits, les rootkits de boot communément appelés Bootkits qui agissent au niveau du disque et plus particulièrement au niveau du MBR (Master Boot Record), de la VBR (Volume Boot Record) ou du secteur de "boot sector', ce type de rootkit ayant été spécialement conçu pour outrepasser la sécurité des disques chiffrés ;
- les Userland-Rootkits ou Rootkits Ring 3 qui agissent là ou tournent les applications.

Un rootkit se doit donc d'être indétectable. Le code et les données doivent être cachés comme les fichiers et les répertoires. Les autres caractéristiques des rootkits sont leurs fonctionnalités de récupérations de données via l'écoute des cartes réseaux ou du clavier ou du gestionnaire de fichiers.

Le rootkit doit être considéré comme de la 'pure' technologie même si on peut penser que c'est toujours utilisé dans des cas de compromission par nos « ennemis » ou par des cybercriminels. Cette technologie est également utilisée par des organisations 'étatiques' ou sociétés d'armement pour dissimuler des programmes empêchant par exemple qu'un missile construit dans un pays mais vendu à un tiers ne puisse le retourner contre le pays du fabriquant, en masquant par exemple des backdoors et autres codes espions. Logiciels et technologies mis en évidence dans les publications d'Edward Snowden.

Le code des rootkits utilise par exemple des techniques sophistiquées comme :

- le cryptage interne du code pour tromper les antivirus ;
- la signature de drivers avec des certificats piratés (donc reconnus valides par l'OS) ;
- l'utilisation d'un module command & control distant qui répond à des commandes ;
- des low level filter-drivers pour mieux récolter de la donnée ;
- utilisent des bugs pour faire des stack-overflow et exécuter du code malveillant.

Un rootkit n'est pas un virus. Il ne se reproduit pas, son but est simplement de s'installer dans le système et de s'y terrer le plus longtemps possible.

Pour cela, il va s'attacher à :

- cacher des process (hidden processes) ;
- cacher des threads (hidden threads) ;
- cacher des services (hidden services) ;
- cacher des fichiers (hidden files) ;
- cacher des secteurs de disque (hidden disk sectors (MBR)/VBR)) ;
- cacher des Alternates Data Streams (hidden Alternate Data Streams) ;
- cacher des clés de registre (hidden registry keys) ;
- crocheter la SSDT, par l'intermédiaire d'un pilote (drivers hooking SSDT) ;
- crocheter l'IDT , par l'intermédiaire d'un pilote (drivers hooking IDT) ;
- crocheter les appels IRP, par l'intermédiaire d'un pilote (drivers hooking IRP calls) ;
- crochetage en ligne (inline hooks).

Comme nous pouvons le constater cela ressemble beaucoup à un ensemble de logiciels dédiés à des fonctionnalités système non ? Et c'est également un moyen d'accéder au Kernel par des moyens qui peuvent être non documentés.

Le monde de la guerre numérique

Dans le monde du cyberwarfare, la maîtrise des systèmes d'exploitation et des failles est un "asset", un pouvoir intelligent. De nombreux pays sont passés maîtres dans ces technologies système comme la Russie, la Chine, les Etats-Unis, Israël, etc. Les exemples de piratage comme Stuxnet sont des exploits technologiques extraordinaires et il existe des papiers faits par des chercheurs qui expliquent ces diverses étapes de hacking. Symantec et Kaspersky font régulièrement des études sur ces exploits. Renseignez-vous et téléchargez ces documents, vous y apprendrez beaucoup de choses.

Outils CIA & NSA à vendre

Il y a peu, des outils de la CIA et de la NSA étaient à vendre pour 15K$ sur le darknet. Ces outils révèlent des mines d'or de secret technologiques. Ces outils sont faits en C/C++ avec des bouts

d'assembleur. Ces outils keyloggers, sniffers, 0 days exploits PDF, IE, Windows, outils command & control, drivers fake, outils attaques DOS, etc., sont des outils qui permettent de faire des actions ciblées dans la cyberguerre que mènent les Etats-Unis contre la Chine et la Russie. Attention, l'utilisation de ces outils est illégale donc si vous mettez la main dessus, agissez en chercheur…

Exemples de techniques standard

Voici quelques techniques de dissimulation. La liste n'est pas exhaustive :

- Il est possible de cacher des fichiers et des répertoires ;
- Il est possible d'écrire des données dans les images JPG, PNG ;
- Il est possible de crypter des fichiers ;
- Il est possible d'injecter un thread dans un processus existant et ainsi d'exécuter du code en //.

Le système est complice

Dans le monde du système d'exploitation, il existe deux modes :

- le mode User : là ou tournent les applications ;
- le mode kernel : là ou tournent l'OS et les drivers.

Les drivers permettent de gérer le matériel et il est possible d'interagir à 3 niveaux :

- avant la commande ;
- pendant la commande ;
- après la commande.

Dans le monde des drivers, il existe ce que l'on nomme des low level filter-drivers qui sont des éléments à l'écoute. Il est possible d'en faire pour différentes classes de périphériques : un gestionnaire de fichiers, un clavier, une carte réseau, etc.

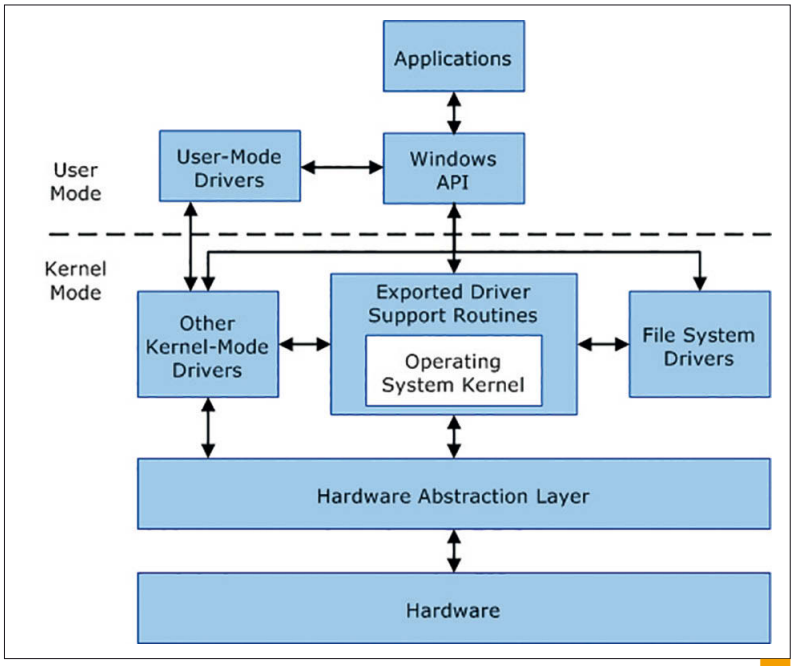
Outils de développement SDK et DDK

Pour développer sous Windows, il faut obtenir Visual Studio et le Visual C++. Il faut installer le SDK (Software Development Kit) Windows pour avoir accès au User mode et le DDK (Driver Development Kit) pour le mode Kernel. En User mode on ne développe pas en Java ou en .NET car sinon, il faut redistribuer un framework et un runtime : c'est stupide ! On développe de manière efficace en C/C++. Le DDK doit être téléchargé séparément. Pour développer sur Windows, on fait du C/C++. Pourquoi ? Parce qu'est ce langage naturel du système d'exploitation. Il n'y a pas de runtime, pas de framework : on build *on the metal !* Les drivers sont faits en C car c'est le plus rapide.

Maintenant que nous connaissons les outils pour développer, rentrons dans le vif du sujet. Pour rendre un programme invisible, il faut que celui-ci ne soit pas visible :

- au niveau de son processus ;
- au niveau de son répertoire d'installation ;
- au niveau de la base de registre ;
- au niveau des services.

Comment cela fonctionne ? Petit rappel à l'attention des codeurs Windows. Il existe essentiellement 2 espaces d'adressage et les applications ne peuvent faire partie que d'un seul d'entre eux. Cela signifie qu'une application est conçue pour s'exécuter soit en mode utilisateur RING 3 (application classique, application avec interface utilisateur, services,…), soit en mode noyau 'kernel' RING 0 (pilotes en mode noyau).



Source Microsoft. 1

Les programmes de haut niveau s'exécutent donc en mode utilisateur, alors que les programmes de bas niveau s'exécutent en mode noyau. Avec une différence fondamentale dans leur fonctionnement. En effet l'espace qui adresse des processus en mode utilisateur est privé (et virtuel). Ce qui signifie qu'à l'intérieur d'un contexte de processus ceux-ci voient la même plage d'adresses. Lorsque quelque chose d'inattendu se produit en mode utilisateur, seul le processus impliqué se bloque, n'impactant pas ses voisins.

Contrairement au mode utilisateur que nous venons de voir précédemment, l'espace d'adressage en mode noyau est, lui, partagé. Ce qui signifie que nous pouvons lire / écrire dans la mémoire de tout autre processus. De cette spécificité en découle malheureusement un risque pouvant être critique ; si quelque chose d'inattendu se produit et n'est pas géré correctement, cela engendre un beau BlueScreen of death (BSOD sur vos écrans) et un redémarrage ou pas de votre PC. 1

Mais revenons à nos moutons, comment font les rootkits pour se cacher avec leur cape d'invisibilité ?

Ici nous aborderons les rootkits « Kernel-Mode » qui fonctionnent au niveau du noyau et non pas au niveau utilisateur.

Mais avant de commencer, quelques petits rappels.

1) Les API Windows : ce sont des fonctions utilisées en programmation (Application Programming Interface). Il en existe plusieurs catégories. Certaines catégories permettent d'interagir avec le système d'exploitation, pour, par exemple, obtenir l'accès aux systèmes de fichiers, aux processus, au registre Windows, au réseau, etc.

Concrètement dans notre cas, lorsqu'un programme désire lister des processus, fichiers, etc., plutôt que le développeur écrive un programme spécifique pour chaque accès système, Microsoft a mis à leur disposition ces API qui sont ni plus ni moins que des passerelles d'accès au système.

2) Appel Système - Syscall : c'est lorsqu'un programme fait simplement un appel à une API via l'instruction CALL.
3) Table SSDT - System Service Descriptor Table : c'est la table qui contient l'adresse des API.
Lorsqu'un appel système 'Syscall' est fait. Windows regarde dans la table SSDT l'adresse de l'API afin de diriger l'appel système vers l'API afin de pouvoir l'exécuter.
Pour faire simple, lorsqu'un programme, comme par exemple un gestionnaire de tâches, désire obtenir la liste des processus en cours, le programme fait un appel système (SysCall). Windows regarde alors dans la table SSDT l'adresse de cette API puis l'exécute et vous obtenez la liste des processus en cours.
En revanche, dans le cas d'un rootkit actif, celui-ci va tout simplement altérer cette table SSDT avec un hook (crochetage), afin de rediriger les appels systeme SYSCALL non plus vers les API natives de Windows mais vers leurs propres API afin de fausser le(s) résultat(s). Dès lors si vous souhaitez lister les processus en cours, Windows va regarder la table SSDT, mais l'adresse de l'API ayant été modifiée par le rootkit, celui renverra la liste de tous les processus à l'exception du sien (CQFD). Le rootkit procédera de la même manière pour masquer les fichiers dans les répertoires, le service qu'il utilise et ses clés de registre.

Composition d'un root Kernel Mode

Un rootkit 'Kernel-mode' est toujours composé d'au moins un driver (.sys en général), celui-ci est chargé par un service qui doit-être invisible (non visible par service.msc), le fichier driver doit lui aussi être caché au niveau de la base de registre pour ne pas apparaître au niveau de la clé : HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services, mais également au niveau de sa localisation sur le disque ...

Si l'on souhaite faire disparaître une clé de registre, voici un petit code qui vous montre comment faire. Ce code ne montre pas comment «hooker » la SSDT ; il se contente d'écrire la fonction de 'hook' qui permettra de masquer notre clé de registre en ciblant. l'API NtEnumerateValueKey. 2

Le code 2 montre que si le nom de la valeur contient la chaîne «_root_», nous appelons l'API une deuxième fois pour masquer les résultats du premier appel. Cela signifie que nous ne recevons aucune information sur la valeur cachée et que cette clé devient 'invisible'.

Exemple : Ici nous pouvons distinguer la valeur '_root_' avant le hook 3

Après installation du rootkit et mise en place du hook, cette valeur est invisible. 4

Dans le cas où vous souhaiteriez creuser comment hooker la SSDT, nous ne pouvons que vous conseiller un article particulièrement bien écrit du site : https://resources.infosecinstitute.com/hooking-system-service-dispatch-table-ssdt/

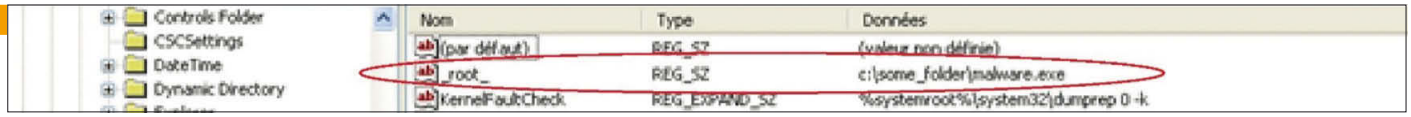
Programmation en mode Kernel : interruptions et IRQLs

Les OS ont un mécanisme pour répondre aux évènements du matériel : les interruptions. Chaque évènement matériel est associé à une interruption. Une interruption demande à l'horloge et à l'ordonnanceur de tourner, une autre au driver de clavier de tourner, etc. Quand un évènement matériel arrive, Windows délivre l'interruption associée au processeur. Quand une interruption est délivrée

2

```
NTSTATUS NewZwEnumerateValueKey(IN HANDLE KeyHandle,IN ULONG Index,IN KEY_VALUE_INFORMATION_CLASS KeyValueInformationClass,
                                OUT PVOID KeyValueInformation OPTIONAL, IN  ULONG Length, OUT PULONG ResultLength)
{
    NTSTATUS ntStatus;
    PWSTR ValueName;

    // Call the original API (NtEnumerateValueKey)
    ntStatus = ((ZWENUMERATEVALUEKEY)(OldZwEnumerateValueKey)) (KeyHandle, Index, KeyValueInformationClass, KeyValueInformation, Length, ResultLength);

    // Get value name
    if (KeyValueInformationClass == KeyValueFullInformation)
    {
        ValueName = ((PKEY_VALUE_FULL_INFORMATION)KeyValueInformation)->Name;

        // If the registry value name contains _root_ we increment the index and call the API a second time, hiding the first call results.
        if (ValueName != NULL && wcsstr(ValueName, L"_root_") != NULL)
        {
            DbgPrint("[ENUMVALUE] %d [%d] -- %ws\n", Index, KeyValueInformationClass, ValueName);

            // Skip index
            Index++;
            ValueName = NULL;
            return ((ZWENUMERATEVALUEKEY)(OldZwEnumerateValueKey)) (KeyHandle, Index, KeyValueInformationClass, KeyValueInformation, Length, ResultLength);
        }
    }
    return ntStatus;
}
```

3



4

Tous les numéros de

Programmez! Le magazine des développeurs

sur une clé USB (depuis le n°100)

+

1 carte de prototypage Attiny85, compatible Arduino, offerte !

34,99 €*

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

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

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

Complétez votre collection

Prix unitaire : ***6,50€***

Lot complet

1 CADEAU À OFFRIR !

48,90 €

(au lieu 58,90 €)

☐ 200	: __ ex	☐ 218	: __ ex
☐ 211	: __ ex	☐ 220	: __ ex
☐ 212	: __ ex	☐ 221	: __ ex
☐ 217	: __ ex	☐ 222	: __ ex
		☐ 223	: __ ex

☐ Lot complet :
200 - 211 - 212 - 217 - 218
220 - 221 - 222 - 223
48,90 €

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

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

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

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

Prénom : ____ Nom : ____

Adresse : ____

Code postal : ____ Ville : ____

E-mail : ____ @ ____

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

Offres pouvant s'arrêter à tout moment, sans préavis **PROG 225**

au processeur, le système ne crée pas un nouveau thread pour servir l'interruption. A la place, Windows interrompt un thread existant pour une courte durée qui est nécessaire pour que la routine de service gère l'interruption. Quand le traitement de celle-ci est terminé, le système redonne la main au thread. Les interruptions ne sont pas forcément déclenchées directement par des évènements matériels. Windows supporte aussi les interruptions logicielles sous la forme d'appels de procédures différées (deferred procedure call alias DPCs). Windows ordonnance une DPC en délivrant une interruption au processeur approprié. Les drivers en mode Kernel utilisent les DPCs pour des besoins comme le traitement coûteux en temps de la gestion d'une interruption matérielle. Chaque interruption est associée à un niveau appelé niveau IRQL qui gouverne le mode de programmation Kernel. Le système utilise différentes valeurs d'IRQL pour permettre que la part plus importante et consommatrice en temps des interruptions soit gérée en premier. Une routine de service tourne au niveau IRQL qui est associée à l'interruption. Quand une interruption intervient, le système localise la routine de service correcte et l'assigne à un processeur. Le niveau IRQL auquel le processeur tourne détermine quand une routine de service tourne et quand elle peut interrompre l'exécution d'un thread qui tourne sur le processeur. Le principe, c'est que le niveau IRQL soit le plus haut à la priorité. Quand un processeur reçoit une interruption, voici ce qui se passe :

- si le niveau IRQL de l'interruption est supérieur à celui du processeur, le système déclenche l'IRQL du processeur au niveau de l'interruption. Le code qui était exécuté sur le processeur est mis en pause et ne reprend pas tant que la routine de service est terminée et que l'IRQL du processeur ne revient pas à sa valeur originale. La routine de service peut être interrompue à son tour par un autre routine de service avec un IRQL supérieur ;
- si l'IRQL de l'interruption est égale à celle du processeur, la routine de service doit attendre jusqu'à ce que toutes les autres routines avec le même IRQL soit terminées. La routine tourne alors jusqu'à achèvement à moins qu'une interruption n'arrive avec un niveau d'IRQL supérieur.
- si le niveau d'IRQL est inférieur à celui du processeur, la routine de service doit attendre que toutes les interruptions avec une IRQL supérieure soient terminées.

Cette liste décrit comment les routines des drivers avec des IRQL différentes tournent sur un processeur particulier. Cependant, les systèmes modernes ont 2 ou plusieurs processeurs. Il est possible pour un driver d'avoir ses routines avec différentes IRQL qui tournent au même moment sur différents processeurs. Cette situation peut amener un deadlock si les routines ne sont pas bien synchronisées.

Les IRQLs sont complètement différentes des priorités des threads. Le système utilise les priorités de threads pour gérer le dispatching des threads durant un traitement normal. Une interruption, par définition, est quelque chose qui tombe de l'extérieur hors d'un traitement normal et doit être gérée le plus vite possible. Un processeur ne résume le traitement normal de gestion des threads seulement après que les interruptions soient gérées. Chaque IRQL est associée à une valeur numérique. Cependant, les valeurs peuvent varier pour différentes architectures de CPU, donc les IRQLs sont nommées par des nom de macros via #define. Seulement quelques IRQLs sont utilisées par les drivers ; la plupart des IRQLs sont réservées par le système. Les IRQLs sont les plus utilisées par les drivers. La liste commence par la plus faible valeur :

- PASSIVE_LEVEL : c'est l'IRQL la plus faible et la valeur par défaut assignée au traitement général de thread. C'est la seule IRQL qui n'est pas associée à une interruption. Toutes les applications user-mode tournent en PASSIVE_LEVEL, comme les routines de drivers à faible priorité. Les routines qui tournent en PASSIVE_LEVEL ont accès à tous les principaux services Windows ;
- DISPATCH_LEVEL : c'est le niveau IRQL le plus élevé associé à une interruption logicielle. Les DPCs et les routines de drivers à haute priorité tournent en DISPATCH_LEVEL. Les routines qui tournent en DISPATCH_LEVEL n'ont accès qu'à un sous-ensemble limité des principaux services Windows ;
- DIRQL : cette IRQL est plus grande que DISPATCH_LEVEL et c'est la valeur la plus haute qu'un driver puisse utiliser. C'est un ensemble d'IRQLs associées à des interruptions matérielles appelées aussi « périphériques IRQLs ». La gestionnaire PnP assigne un DIRQL à chaque périphérique et le passage au driver approprié au démarrage. Ce niveau n'est pas important à connaître, il faut juste savoir que la routine d'un tel service est bloquante et empêche toute autre routine de tourner sur un processeur. Les routines qui tournent en DIRQL n'ont accès qu'à un sous-ensemble limité des principaux services Windows.

Le non-respect des règles ci-dessus provoque le crash du driver et du système d'exploitation : le fameux blue-screen of death ou BSOD.

Exemple d'outil : un keylogger

Un keylogger est un filtre qui permet de connaître les touches qui sont pressées sur le clavier. Pour faire un keylogger, il faut faire un driver et s'enregistrer en tant que filtre. On y inclut une routine qui permet d'être notifié d'un évènement (une callback). Pour permettre une fluidité dans le traitement de la requête, on stocke les éléments dans un buffer, et un thread dédié peut enregistrer les évènements sur disque. C'est un moyen pour capter un mot de passe Windows, des identifiants de connexion FTP, des mots de passe bash sudo, etc.

Un driver est un service Windows de type kernel. On le démarre en automatique au boot généralement. S'il déraille, c'est le blue-screen assuré !

Le code sur keylogger est ici :

https://github.com/bowlofstew/rootkit.com/tree/master/Clandestiny/Klog%201.0/Src

On y trouve un code Basic qui fait le job.

Interception et logging des data du clavier

Lorsque le filter driver est installé, le code est divisé en deux : l'interception et le logging de la data.

L'interception, c'est de capter l'IRP « read » du périphérique clavier. Voici la séquence :

- l'I/O Manager de l'OS envoie un paquet IRP vide dans la pile du périphérique clavier ;
- les IRP « read » sont interceptées par la routine de Dispatch du

filter-driver pour IRP_MJ_READ.

L'IRP est taguée avec une routine de réalisation. C'est un callback. L'IRP est captée et le driver fabrique la pile IRP pour le suivant via IoGetCurrentIrpStackLocation et IoGetNextIrpStackLocation.

- le driver fixe la routine de réalisation sur l'IRP courante en appelant IoSetCompletionRoutine ;
- le driver passe l'IRP au prochain driver dans la pile avec IoCallDriver ;
- quand la touche est pressée sur le clavier, le driver complète l'IRP. Elle est complétée avec le scan code de la touche pressée et renvoyée sur la pile de périphérique ;
- les routines de réalisation sont appelées et l'IRP est passée. Cela permet au filter-driver de récupérer l'information de scan code stockée dans le paquet et de l'ajouter dans une queue pour un traitement spécifique. La routine d'achèvement est appelée en live au niveau DISPATCH_LEVEL et il faut donc faire attention aux restrictions d'API et d'allocations mémoire à ce niveau pour récupérer le code et le stocker... Il faut réaliser l'opération au niveau File I/O. Les APIs du système de fichier ne peuvent être appelées qu'au niveau IRQ_PASSIVE_LEVEL donc nous devons créer un thread séparé qui tourne en PASSIVE_LEVEL pour le gérer. Avoir un thread séparé permet un mécanisme de queue et un accès synchronisé à la queue. Un sémaphore peut être utilisé pour notifier le thread qui fait l'opération de file I/O qu'un scan code est disponible à logger. Utilisez KeWaitForSingleObject sur le sémaphore. La mémoire pour la liste des codes doit être allouée depuis la mémoire « non paged pool » car la routine de completion tourne en DISPATCH_LEVEL.
- Avant d'écrire le code dans un fichier, il est converti en ASCII et cela dépend du paramétrage du clavier…

Code du thread du keylogger

```
VOID ThreadKeyLogger(IN PVOID pContext)
{

    PDEVICE_EXTENSION pKeyboardDeviceExtension = (PDEVICE_EXTENSION)pContext;
    PDEVICE_OBJECT pKeyboardDeviceOjbect = pKeyboardDeviceExtension->pKeyboard
Device;

    PLIST_ENTRY pListEntry;
//custom data structure used to hold scancodes in the linked list
    KEY_DATA* kData;
    //Enter the main processing loop...
//This is where we will process the scancodes sent
    //to us by the completion routine.
    while (true)
    {
        // Wait for data to become available in the queue
        KeWaitForSingleObject(&pKeyboardDeviceExtension->semQueue,
Executive, KernelMode, FALSE, NULL);

        pListEntry = ExInterlockedRemoveHeadList(
&pKeyboardDeviceExtension->QueueListHead,
&pKeyboardDeviceExtension->lockQueue);

        ////////////////////////////////////////////////////////////////////
        // NOTE: Kernel system threads must terminate themselves. They cannot
        // be terminated from outside the thread. If the driver is being
        // unloaded, therefore the thread must terminate itself. To do this
        // we use a global variable stored in the Device Extension.
        // When the unload routine wishes to terminate, it will set this
        // flag equal to true and then block on the thread object. When
        // the thread checks this variable and terminates itself, the
        // Unload routine will be unblocked and able to continue its
        // operations.
        ////////////////////////////////////////////////////////////////////
        if (pKeyboardDeviceExtension->bThreadTerminate == true)
        {
            PsTerminateSystemThread(STATUS_SUCCESS);
        }

        ////////////////////////////////////////////////////////////////////
        // NOTE: the structure contained in the list cannot be accessed directly.
        // CONTAINING_RECORD returns a pointer to the beginning of the data
        // structure that was inserted into the list.
        ////////////////////////////////////////////////////////////////////
        kData = CONTAINING_RECORD(pListEntry, KEY_DATA, ListEntry);

        //Convert the scan code to a key code
        char keys[3] = { 0 };
        ConvertScanCodeToKeyCode(pKeyboardDeviceExtension, kData, keys);

        //make sure the key has returned a valid code before writing it to the file
        if (keys != 0)
        {
            //write the data out to a file
            if (pKeyboardDeviceExtension->hLogFile != NULL)
            {
                IO_STATUS_BLOCK io_status;
                DbgPrint("Writing scan code to file...\n");

                NTSTATUS status = ZwWriteFile(
pKeyboardDeviceExtension->hLogFile, NULL, NULL, NULL,
                    &io_status, &keys, strlen(keys), NULL, NULL);

                if (status != STATUS_SUCCESS)
                    DbgPrint("Writing scan code to file...\n");
                else
                    DbgPrint("Scan code '%s' successfully written to file.\n", keys);
            }//end if
        }//end if
    }//end while
    return;
}
```

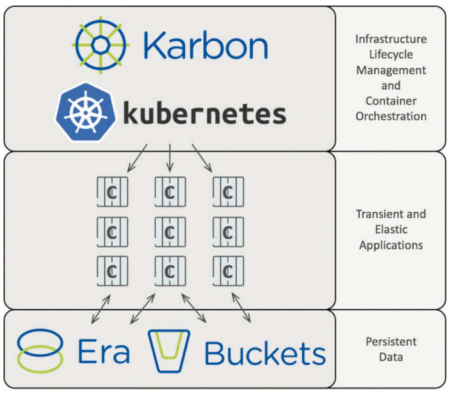
Conclusion

Un keylogger est un composant périphérique de type file-driver. Il capte la data et peut la traiter. Pour faire tourner ce driver, il faut l'installer via un fichier INF et le signer avec un certificat. Sous Windows 10, tous les drivers doivent être signés. Cet article montre du code en mode Kernel et utilise le DDK. •

Nutanix : les APIs au coeur du développement et de la collaboration

Que ce soit pour le développement de ses solutions ou pour la collaboration avec des intervenants extérieurs comme les partenaires, les clients ou les développeurs, Nutanix a décidé de s'appuyer entièrement sur des API RESTFul.

Les API sont aujourd'hui indispensables, aussi bien au niveau des couches basses de l'infrastructure qu'à celui des applications. Elles permettent de rendre plus simples les interactions entre les différents systèmes et leur consommation des services. « Les API sont fondamentales chez **Nutanix** aussi bien pour le développement que pour intégrer nos solutions au sein des environnements de nos clients », déclare James Karuttykaran, Directeur SE pour la région Southern EMEA de **Nutanix**. Dès le début, la firme spécialisée dans l'hyperconvergence basée à San José, en Californie, a fait le choix d'être « API First ». Chaque produit ou fonctionnalité est ainsi doté à sa création d'une interface GUI ou CLI liée à une API unique pour faciliter l'interaction entre les différentes équipes de développement. Chacune d'entre elles peut ainsi accéder aux ressources développées par une autre équipe de façon structurée pour ainsi garantir une offre complètement intégrée, dès sa conception. Les APIs sont par ailleurs au coeur du fonctionnement de la console d'administration unique de **Nutanix**, Prism, qui au passage est développée en HTML5. Elle peut ainsi facilement récupérer les informations issues des différentes solutions **Nutanix** mises en place par un client, mais également s'intégrer à ses propres solutions de management. Il est ainsi possible de consolider en un seul point les informations des serveurs physiques, du stockage, du réseau, des machines virtuelles, de l'hyperviseur, de corréler ces informations pour identifier un problème plus rapidement, et surtout de pouvoir interagir via un seul set d'API à tous ces différents services.



« Au-delà de nos politiques de développement internes, les APIs jouent également un rôle énorme dans le cadre de nos collaborations externes, que ce soit avec des partenaires technologiques, nos clients finaux ou les développeurs », ajoute James Karuttykaran. Toutes ces populations ont toutefois des cas d'utilisation qui leur sont propres. Pour les partenaires technologiques, l'offre **Nutanix** se compose d'un ensemble de services tels que le stockage, la virtualisation, ou la conteneurisation pour ne citer que ceux-là. Suivant son profil, le partenaire va vouloir s'appuyer sur un ou plusieurs de ces services pour enrichir son offre. « Par exemple,

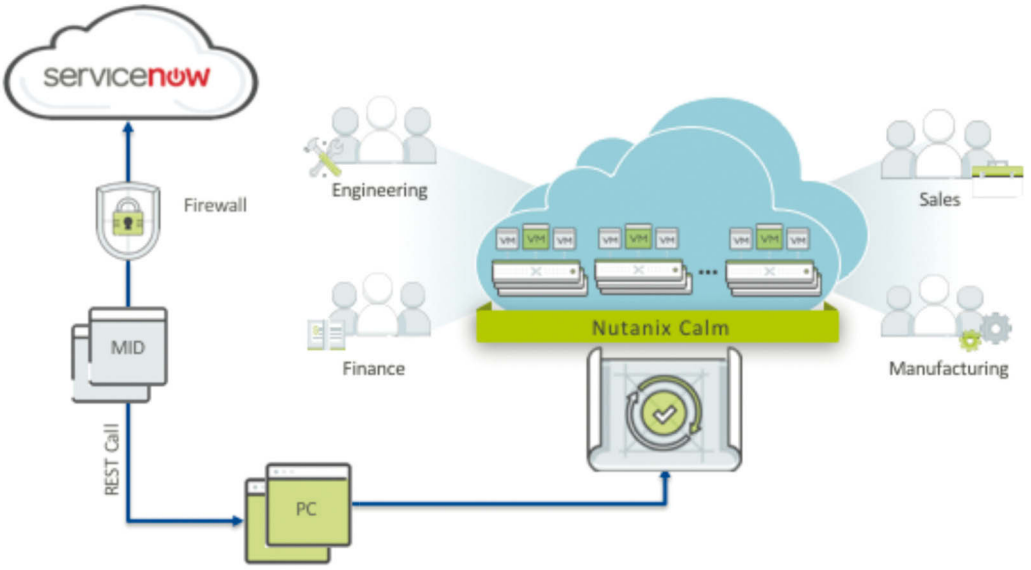
Conférence .Next

Multicloud, Edge, IoT : Nutanix étend son offre du cœur de l'infrastructure à ses extrémités

Nutanix rassemble les pièces du puzzle. Acteur historique de l'hyperconvergence, la firme souhaite aujourd'hui proposer une plateforme multiclouds hybride capable de s'étendre d'un bout à l'autre de l'infrastructure pour en automatiser et en simplifier tous les aspects. Si les briques de base étaient déjà présentes avec le système AOS, l'administrateur Prism et l'hyperviseur AHV, la firme a profité de la conférence .Next Europe 2018 qui se tenait à Londres les 28 et 29 décembre, pour apporter de nombreuses nouvelles pierres à cet édifice.

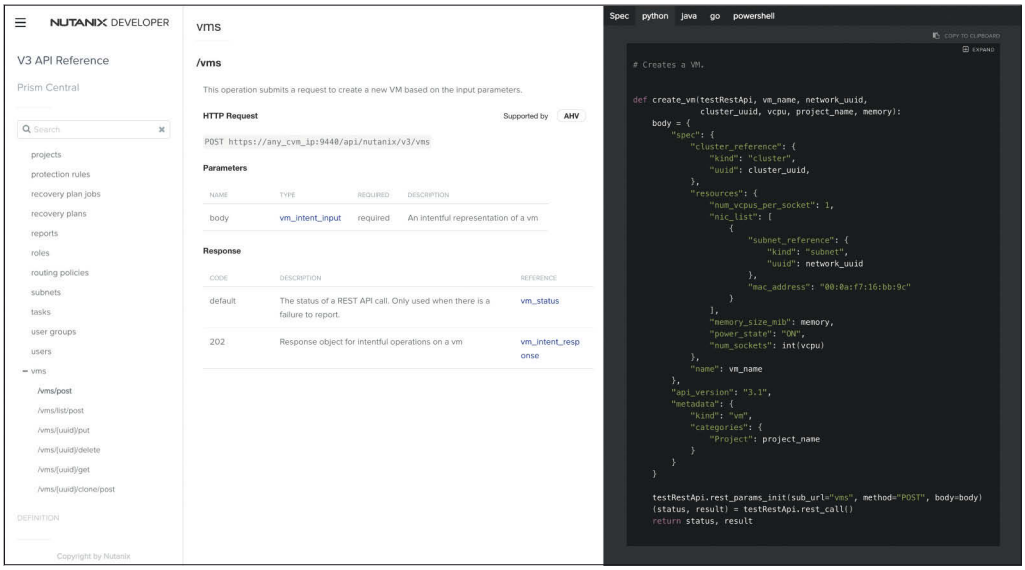
Et c'est par le lancement de Xi Cloud Services, esquissé en mai dernier lors de la précédente conférence .Next, que Nutanix a d'abord décidé de s'illustrer. Cette offre regroupant 5 services différents vise à apporter une vision globale de l'infrastructure hybride étendue, et à faciliter l'administration et la migration des applications à travers différents clouds, publics et privés. Xi Cloud Services s'appuie ainsi sur une solution de Disaster Recovery as a Service (DRaaS), Xi Leap. Il y a aussi un service d'analyse et d'observation des performances basées sur les micro-services et les APIs. Xi Epoch, est quant à lui un outil de gouvernance multicloud permettant d'optimiser les dépenses. Xi Beam, provient d'un service de Desktop as a Service (DaaS) issu de l'acquisition de Frame réalisé cet été : Xi Frame. Enfin la pierre angulaire de l'offre naissante de **Nutanix** en matière de Edge computing et d'IoT s'appelle Xi IoT. Cette dernière englobe un système d'exécution de conteneurs, notamment avec Kubernetes, un moteur d'exécution de fonctions évènementielles en mode serverless, un moteur d'inférence TensorFlow, ainsi qu'une gestion avancée et automatisée des « data pipelines » acceptant des sources et des destinations de données multicloud.

En parallèle de ces annonces très orientées sur le cloud, **Nutanix** a également fait la part belle aux infrastructures internes que la firme aimerait muter en cloud privé, du moins pour ce qui est de l'administration et de l'agilité. Pour la partie stockage, **Nutanix** a ainsi présenté Files, qui permet de consolider le stockage de fichiers sous forme de NAS distribué. Volume, dans la même offre, facilite l'intégration des solutions de la firme aux infrastructures existantes en permettant la reconnaissance de ses solutions de stockage par les serveurs historiques sous la forme de baie SAN. Dans la logique d'une approche qui se veut de plus en plus DevOps, **Nutanix** a également présenté Karbon, une implémentation de Kubernetes à sa solution AOS, directement gérée par Prism, et Buckets une solution dédiée au stockage objet. Dans la même veine, la société a annoncé le lancement des solutions déjà présentées, Calm, pour l'automatisation et l'orchestration de l'infrastructure, et Flow, qui assure la micro-segmentation et la sécurisation des applications. Il faut ajouter à ceux-ci Era, qui facilite les déploiements et les mises à jour sur les bases de données Oracle, SQL Server, Postgres SQL et Maria DB sur site ou dans le cloud.



5 chiffres du Nutanix Enterprise Cloud Index

- **91%** des entreprises estiment que le cloud hybride est la meilleure solution pour répondre à leurs besoins
- Seuls **18%** des entreprises ont déjà déployé une infrastructure cloud hybride
- **70%** des entreprises françaises sont déçues de l'utilisation du cloud public
- **41%** des entreprises françaises ayant déployé des solutions de cloud public ont dû faire face à des dépassements de budget
- **71%** des entreprises déclarent que la sécurité est un facteur déterminant pour savoir ou faire tourner leurs charges de travail.



un éditeur de solutions de sécurité peut vouloir mettre en place des web hooks pour surveiller une machine virtuelle pour ensuite actionner des tâches pour la configuration d'un élément réseau tierce, par exemple configurer automatiquement un load balancer lorsqu'on clone des serveurs web pour que ces serveurs web soient rajoutés dans le pool du load balancer. Tout cela est faisable de manière extrêmement simple avec nos APIs », détaille James Karuttykaran.

Dans le cas des clients finaux, deux situations se démarquent quant à l'utilisation des APIs selon **Nutanix**. Il y a d'une part ceux qui souhaitent consolider les informations dans une optique de reporting et, d'autre part, ceux qui souhaitent automatiser les actions. Pour les premiers, une solution de monitoring tierce peut s'appuyer sur les APIs **Nutanix** pour récupérer des informations telles que l'utilisation des ressources, descendre jusqu'à certains détails comme la latence du stockage ou le nombre d'IOPS, ou même combiner certaines actions avec du Powershell, VBScript, ou d'autres moyens supportés par la solution, et proposer toutes ces informations dans une seule console pour un environnement qui ne serait pas uniquement **Nutanix**.

Quant à ceux qui souhaitent automatiser certaines tâches, de nombreuses entreprises ont déjà en place des CMP (Cloud Management Platform) ou un portail de Self-Service. Plutôt que d'imposer son portail, **Nutanix** propose grâce aux APIs de s'intégrer dans un environnement existant. Par exemple, il est possible de piloter AHV (Acropolis Hypervisor), l'hyperviseur de **Nutanix**, Calm (une solution de gestion du cycle de vie applicatif) ou toutes autres fonctionnalités depuis l'ensemble des solutions du marché telles que Service Now, VMware vRealize Automation ou OpenStack.

Vient enfin le cas des développeurs qui ne savent pas toujours comment fonctionne une infrastructure ou n'ont simplement pas envie ou le temps de s'y intéresser. « C'est ce qu'ont parfaitement compris des acteurs tels qu'Amazon Web Services, Google Cloud Platform ou Azure. Ce que **Nutanix** veut proposer c'est la même expérience que ces acteurs du web, mais on premise, dans le datacenter du client, avec des applications traditionnelles non éligibles au cloud », explique James Karuttykaran. **Nutanix**, veut ainsi rendre l'infrastructure et les clouds invisibles, pour que les utilisateurs puissent se focaliser sur les applications. Le développeur va pouvoir consommer les ressources fournies par **Nutanix**, cpu, ram, stockage, vlan, vm, container … en utilisant les API Rest ou les différents SDK fournis par la firme. Les usages sont multiples. Si on regarde les dernières offres **Nutanix**, le développeur peut aussi automatiser le déploiement des bases de données, des blueprints applicatifs, définir des règles de micro segmentation, ou déployer des serveurs de fichiers. Chez **Nutanix** les API sont par ailleurs RESTful, et utilisent des requêtes HTTP pour lire/écrire des informations depuis/vers le système, et le résultat des commandes est au format JSON. Les API sont maintenues par **Nutanix** et sont disponibles à tous les utilisateurs des solutions de la firme, sans coût supplémentaire. Elles évoluent rapidement en fonction des besoins tout en restant compatibles avec l'existant. **Nutanix** vient ainsi de lancer la version 3 de ses APIs qui prend en charge les dernières fonctionnalités de notre plateforme. Elles sont par ailleurs accessibles librement à tous les clients **Nutanix**, et également dans la version gratuite appelée **Nutanix** Community Edition.

Ressources supplémentaires avec exemple d'implémentation :

Nutanix a un site qui recense les ressources pour les développeurs : https://developer.nutanix.com

Le lien vers le Github **Nutanix** https://github.com/nutanix/ des tutorials avec quelques exemples concrets https://developer.nutanix.com/reference/documentation/tutorial/html/ et un forum pour échanger https://next.nutanix.com/api-31

François Tonic

2019 : notre futurologie garantie 100 % vraie (1)

C'est la période de l'année où tout le monde sort la boule de cristal et tente de décrypter les messages d'au-delà des nuages… La rédaction de Programmez ! vous propose un petit décryptage des (possibles) tendances, confirmations et des flops attendus.

© dédegonde

LES TECHNOS À SURVEILLER

Kotlin & Swift

Désormais ces deux langages sont bien implantés dans le paysage développeur. Ils sont encore jeunes et en constante évolution, mais ils ont su trouver une place auprès des développeurs mobiles. Ils remplacent peu à peu les langages historiques d'Android et d'iOS (Java et Objective-C).
Ils sont ouverts vers les communautés. Une fondation a été créée pour aider à diffuser Kotlin et côté Swift, des éditeurs soutiennent le langage. L'un des plus actifs est IBM. Swift 5 arrivera début de l'année. Côté Kotlin, le langage continue son développement. La dernière version majeure est la 1.3, sortie en automne dernier.

DevSecOps + Secure by design

Oui, oui et re-oui. Il faut absolument regarder de près le DevSecOps. La sécurité doit être au cœur de vos préoccupations. Et avec la génération des bonnes pratiques et des approches de type DevOps, il est essentiel d'être « sécurité compatible » dans les gènes des équipes. Il faut en finir avec le mode pompier. Le développeur doit avoir sa part dans la sécurité. Il n'aurait jamais dû perdre son rôle. Le développeur a une responsabilité dans la sécurité. Le nier c'est élargir la surface d'attaque. Oui la sécurité a un surcoût, mais peut-on encore prendre prétexte de cela pour ne pas en faire dès la conception ?

IoT

On en parle tout le temps ou presque et partout. Les objets connectés, les fameux IoT, sont là. À force de le dire, ils vont arriver. Dans l'industrie, le supply chain (pardon, la logistique), le retail (oups, le commerce en tout genre), la smart city (re-pardon, la ville intelligente), les IoT, et tous les capteurs qui vont avec, sont là et se déploient par milliers. Plusieurs composants IT vont avec : le matériel (plateforme MCU typiquement, capteurs, etc.), le logiciel embarqué, les librairies/API/SDK, les serveurs & plateformes d'interconnexions (typiquement les hubs IoT, la gestion des évènements, les flux de données batch et stream, etc.). Le développeur joue évidemment un rôle central, car sans code, un IoT n'a aucune intelligence.

Infrastructure as Code, serverless

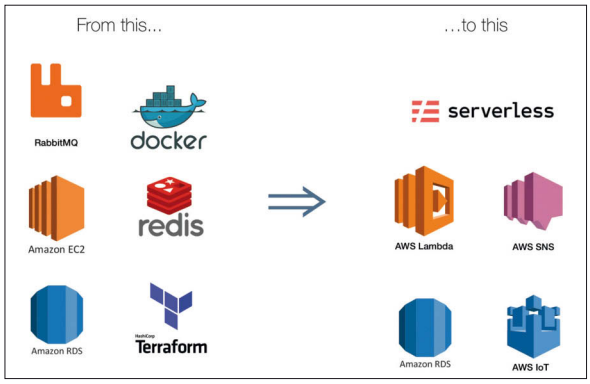
L'infrastructure est en pleine mutation. Le IaaS avait déjà fortement secoué le secteur. Et ce n'est pas fini. Fini la simple virtualisation, nous sommes largement au-delà et la tendance de l'infrastructure vue par le logiciel s'impose ainsi que la possibilité de se soustraire à la partie matérielle via le serverless. Mais attention, oui, nous manipulons des assets dématérialisés mais le matériel reste bel et bien présent. Suivez attentivement l'Infrastructure as Code et tout ce qui se passe autour du serverless. Là encore, le développeur doit absolument s'y intéresser, car il sera impliqué tôt ou tard !

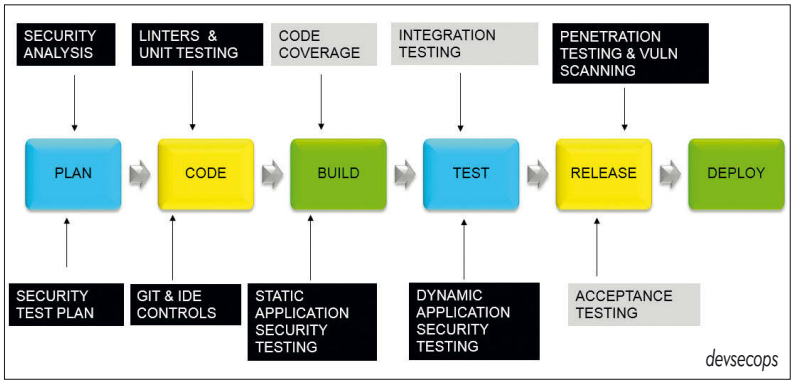
Quantique

Ce fut l'un des buzz de 2018. Le quantique est encore éloigné de l'utilisateur et du développeur, notamment à cause du matériel. Mais aujourd'hui, on peut tester et comprendre les concepts. Les éditeurs s'y

(1) Avec une légère marge d'erreur.

serverless





mettent activement. A regarder un peu toute l'année.

IA/ML + DL

Là encore, deux gros buzz de l'année 2018 ! Malheureusement, l'IA est mise à toutes les sauces, et parfois de très mauvaise qualité. Si une IA se limite à un chatbot, parfois très mal codé et à l'intérêt 0, peut-on parler d'IA ? Les véritables IA ne sont pas aussi fréquentes que ça.

Le machine learning et le deep learning sont deux éléments clés pour l'IA. Comment faire apprendre quelque chose à la machine (si elle est codée pour ça). Rien n'est plus difficile que l'apprentissage et l'entraînement. Les technologies bougent vite dans ces deux domaines. Tensorflow est une des vedettes du machine learning.

LES TECHNOS (DÉSORMAIS) MATURES

Les tops OWASP

Malheureusement, nous ne pouvons que constater la maturité des classements OWASP ! Chaque année, les meilleures failles, notamment en développement Web, nous laissent songeurs. Il est inacceptable et incompréhensible qu'en 2018 (maintenant 2019), on puisse encore et toujours rencontrer des failles XSS, de l'injection de code !

JavaScript et tous les frameworks : oui, mais…

Incontestablement, JavaScript est LE langage de développement Web, et mobile, du moment. Son succès, malgré une syntaxe décriée, est dû aux nombreux frameworks et librairies du marché. C'est sa force ! Et le langage devient mature. Mais un problème provient aussi de cette profusion de frameworks, car le développeur ne sait plus vers

quoi se tourner. Et la question de la pérennité de tous ces composants est ouverte. Un choix se fera forcément.

Vous pouvez tester les librairies qui sortent, mais évitez de les mettre en production trop rapidement surtout si elles ne proviennent pas d'un éditeur ou d'une forte communauté. Et finalement, seule une poignée émerge réellement.

Bref, privilégiez les classiques : Angular, Vue, Vanilla, React…

Cloud computing

La question ne se pose plus. Le cloud computing s'est imposé et il ne peut que progresser. Tous les grands éditeurs migrent leurs logiciels dessus. IBM, Oracle, Microsoft, pour ne citer qu'eux, migrent leurs plateformes en mode IaaS, PaaS ou SaaS. La guerre entre les différents fournisseurs est vive. Le cloud va encore s'accélérer avec les grandes entreprises qui migrent progressivement leur IT dans le nuage. C'est une autre manière de penser et de consommer les ressources IT. Pour le développeur, cette migration n'est pas neutre, surtout en contexte PaaS et SaaS. L'architecture logicielle est modifiée pour adopter les bonnes pratiques. D'autre part, de plus en plus de services cloud sont utilisés en backend applicatif. Si vous n'êtes pas encore habitué au cloud, un conseil : il est temps de s'y mettre sérieusement !

Développement mobile

Cela fait 10 ans que le développement mobile explose. Désormais le marché est largement mature. Les ventes de smartphones commencent même à baisser et nous sommes maintenant en phase saturation. Et les ruptures technologiques sont moins nombreuses. Pour les développeurs, la demande en développement mobile est toujours aussi forte, car il faut alimenter les plateformes. Le mobile se résume à Google et à Apple.

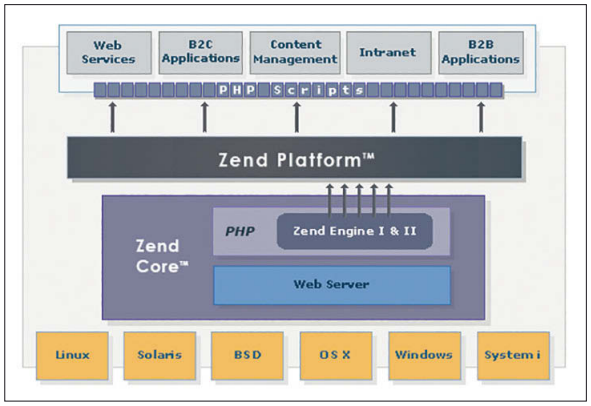
Par contre, la fragmentation des plateformes n'a jamais été aussi forte pour les développeurs et elle commence à s'accélérer sur iOS même si Android est bien plus fragmenté. Il y a toujours un débat app native et app hybride comme vous le verrez dans le dossier du mois.

Bref, une valeur sûre si vous voulez changer de poste ou devenir développeur.

Open Source

Sans surprise, l'open source reste une valeur sûre ! Le débat n'existe plus entre propriétaire et ouvert. Même sans le savoir,

réalité augmentée



vous utilisez forcément des couches open source. Microsoft est l'un des plus grands contributeurs et chaque mois, de nouvelles annonces sont faites. Récemment, c'est même l'abandon du cœur du navigateur Edge au profit du projet Chromium, servant de fondation à Chrome ! C'est un sacré revirement qui ne sera pas simple à réaliser. Autre annonce spectaculaire : le rachat de Red Hat par IBM pour 34 milliards ! Reste à voir comment l'opération sera gérée. Mais cela amorce bel et bien un tournant définitif vers l'open source.

C++

On ne présente plus C++. On a voulu faire croire à sa mort. Mais non ! Ce langage référence est plus que jamais d'actualité. C++ reste la référence dans l'embarqué et dans toutes les couches basses système. Les évolutions C++ 17 ont permis de rebooster l'intérêt pour le langage et la future évolution promet de belles choses. Dire que le C++ est fini et qu'il n'a aucun intérêt, c'est méconnaître son rôle fondamental dans l'informatique. Windows, Linux, macOS possèdent des millions de lignes de codes en C++.

LES TECHNOS SUR LESQUELLES NOUS AVONS DES DOUTES OU DES ?

L'univers des technologies n'est jamais une science exacte. Depuis 20 ans, nous le vérifions. Certains technos qui paraissaient en pleine ascension explosent parfois en vol, pour des raisons x, y, z.

Réalité augmentée/ virtuelle/mixte

Nous en avons parlé très régulièrement en 2017, mais moins en 2018. Il faut dire que la VR/VA se comporte étrangement sur le marché. Depuis fin 2017, nous voyons moins d'annonces matérielles. Même Facebook avec Oculus se montre assez discret malgré un prometteur Oculus Go. Magic Leap, après des attentes très élevées, a fini par sortir un matériel… décevant. Apple et Google qui avaient fait la course à l'implémentation en 2017 ont clairement calmé le jeu en 2018.

Microsoft avait beaucoup communiqué autour de Windows Mixed Reality et des partenaires avaient sorti des casques compatibles. Mais force est de constater que l'effet n'a pas duré ! La plateforme Microsoft demeure très très loin derrière Vive et Oculus. Certains constructeurs ont sorti des casques clairement cheap à la qualité un peu douteuse avec une stabilité logicielle parfois aléatoire.

Un des problèmes en VR est le coût du casque et le filaire qui demeure la règle. Cependant, la tendance est au casque autonome. Oculus Go, malgré les lacunes qu'il peut avoir, est remarquable sur ce point, avec un tarif très agressif, tranchant avec l'inertie du marché Windows Mixed Reality. Le problème ne vient pas forcément du matériel, mais plutôt du contenu et des usages. Et là, les choses ne bougent pas beaucoup. Malheureusement.

En VA et réalité mixte, la réalité est un peu différente. L'offre est assez nombreuse, mais là encore, se pose le problème des usages et du contenu. Personne n'a encore trouvé le déclic. Magic Leap a beaucoup déçu en dévoilant la première version de ses lunettes. Il faut dire que le constructeur avait énormément promis pour, au final, proposer quelque chose de volumineux, de peu pratique et techniquement en bêta. Hololens est clairement orienté entreprise, et les nombreuses lunettes à réalité augmentée sont le plus souvent dédiées aux professionnels. Le marché existe avec des usages maintenant connus : formation, aide active sur le terrain. Mais là encore, le coût peut rester un frein.

En France, des salles VA/VR ouvrent mais il est trop tôt pour en connaître le réel impact sur le marché. Et nous ne pensons pas que cela va booster les ventes. Car encore une fois, le problème est « simple » : les usages, les contenus (en dehors du jeu).

Agilité/ DevOps

Étonnant de retrouver l'agilité et le DevOps ici. Honnêtement nous ne savions pas comment les traiter. Oui, ce sont des approches désormais acquises, mais pas toujours pour les bonnes raisons. Le marketing a dénaturé l'objectif et la définition même de ces méthodes/bonnes pratiques. Et c'est bien là le problème. Pour faire bien, tout le monde se dit agile et DevOps même si ce n'est pas le cas ou très mal. Au bout du compte, on embrouille les équipes et les responsables, et, trop souvent, les éditeurs et prestataires donnent leur propre définition.

Quand on me dit : oui j'ai un chef de projet agile dans mes équipes Scrum, cela laisse rêveur, non ?

Zend : le début de la fin ?

Zend Server et Zend Studio étaient les composants phares du monde PHP. Mais trois événements ont contribué à leur chute : Symfony, Symfony dans Drupal et le rachat de Zend par l'éditeur Rogue Wave en 2015. Depuis cette date, l'activité de l'éditeur a été irrégulière. L'avenir même des outils Zend est toujours dans le doute.

Quel avenir ? Le focus a été mis sur Zend Server. Et Zend Engine, Zend Studio et Zend Framework ? Ce n'est clairement plus une priorité pour l'éditeur.

Dart

Dart est l'autre langage de Google. Il est cependant bien moins visible que Kotlin et Go. Ces deux langages sont mis en avant par Google. Même si nous mettons Dart en ? pour 2019, n'y voyez pas une défection de notre part. Simplement un certain flou sur la place de Dart chez Google et comment l'éditeur veut l'utiliser. Le langage continue à évoluer. La version 2.1 a été livrée il y a quelques semaines avec beaucoup d'améliorations. La v2.2 sortira courant 2019 avec des changements sur certaines syntaxes, de l'optimisation.
Peut-être qu'une vie en dehors de Google serait une bonne idée.

OpenJDK selon Amazon Web Services

Comme évoqué dans ce numéro, Amazon Web Services travaille à sortir sa propre JDK open source et gratuite avec support long terme. L'annonce a surpris. La version finale arrive dans les prochaines semaines. La question est de savoir comment Corretto trouvera sa place alors que les développeurs et les entreprises ont déjà un large choix. Nous ne pourrons pas tirer le moindre bilan avant fin 2019.

Java

Nous avons beaucoup hésité sur la place de Java. Le langage demeure la référence du marché et dans les offres d'emploi, mais il a perdu de son intérêt par rapport à d'autres langages plus récents. Il faut dire que Java a échoué dans le mobile. Sur le Web, le retrait des applets avait marqué un début de reflux. Les multiples retards dans les versions majeures, le rachat de Sun par Oracle n'ont pas facilité les choses. Oracle a voulu redonner une nouvelle dynamique en se séparant de plusieurs projets (Java EE, NetBeans) pour se recentrer sur Java, tout en redéfinissant une nouvelle politique de licencing.
La modification en profondeur de la roadmap, et instaurer une itération tous les 6 mois, doivent redynamiser l'écosystème et redonner confiance aux communautés. A suivre attentivement en 2019.

PHP

Nous avons eu du mal à placer PHP pour 2019. Cela ne vaut pas dire que PHP est sans avenir ou qu'il chute dans son usage. Il reste une des valeurs sûres du marché Web et du développement. Mais reconnaissons tout de même que les concurrents sont là et bien là. La communauté reprend de la vitalité avec les versions 7.x. La question est de savoir comment PHP va pouvoir évoluer et répondre à une concurrence de plus en plus vive. Sur les CMS, il reste le leader incontesté.

ARM

Nous avons beaucoup hésité sur la place des processeurs ARM. Ils se sont imposés dans les mobiles, une partie des IoT et des cartes maker, désormais, nous attendons avec impatience la déferlante sur les serveurs et nos ordinateurs. Mais là, rien n'est simple. Les atouts des ARM (consommation, chaleur) ne masquent pas un problème récurrent : des performances qui ne peuvent rivaliser avec les plus puissants processeurs desktop/serveur. Or, pour prétendre prendre la place des x86, ARM devra faire un saut important dans les performances.
Les PC ARM sont en vente, notamment grâce à Windows 10 ARM. On se souvient encore de l'échec cinglant de Windows RT. Mais pour le moment, ces machines peinent à trouver leur public notamment par manque de performances/puissance pour utiliser des logiciels plus exigeants. Comme les apps ARM sont peu nombreuses, le système utilise une émulation pour exécuter les apps x86.
PC ARM = Surface RT + Windows RT ? Trop tôt pour le dire, car les constructeurs PC tentent l'aventure, mais ce n'est pas un gage de réussite pour autant. Il faudra rapidement adapter les apps et la recompilation ne sera pas suffisante. Les processeurs devront aussi sérieusement monter en puissance. Et certains constructeurs regardent avec envie la réussite d'Apple dans ses processeurs Ax qui flirtent de plus en plus avec les performances de certains PC. Depuis des mois, la rumeur veut qu'Apple prépare, pour 2020 ?, une nouvelle migration massive de la plateforme Mac : de x86 à ARM.
Contrairement à d'autres constructeurs, Apple possède plusieurs atouts pour faire cette opération : l'expérience de la transition (680x0 vers PowerPC vers x86), des équipes CPU et GPU reconnues, l'implication des développeurs via des outils et des couches techniques dédiées.

Tableau 1 : les langages

Langages	Indicateur de suivi (sur 5 *)
C#	***
Java 11 & suivant	***
Rust	***
Python	****
JavaScript et les frameworks JS	*****
C++	****
PHP	** à ***
Ruby	** à ***
Swift	****
Go	****
Kotlin	****

Tableau 2 : architectures

Architecture	Indicateur de suivi (sur 5 *)
Kubernetes, conteneurs	****
Micro-services	*****
Edge computing	****
Serverless	****

Tableau 3 : technologies/plateformes

Technologies/ plateformes	Indicateur de suivi (sur 5 *)
Machine/ Deep Learning	****
Services cognitifs	****
.Net Core 3	***
IoT	****
Mobile	****
Réalité virtuelle/augmentée	**
CMS	***
Cloud computing	*****
Blockchain	****
Machine/ Deep Learning	*****
Quantique	***

Tableau 4 : Méthodes/bonnes pratiques

Pratiques	Indicateur de suivi (sur 5 *)
Scrum	***
DevOps	****
Kanban	*****
Craftmanship	*****

mobioos

Mobioos Forge : une autre manière de concevoir les applications

Générer des applications à partir d'un modèle sans se soucier du langage utilisé ? N'est-ce pas la quête de nombreux développeurs, utilisateurs et entreprises ?

Processus de Mobioos FORGE

Nombreux sont ceux à s'y être essayé. La génération d'applications étant au coeur des approches low code / no code, nous avions donné sa chance au MDA (Model Driven Architecture). Ce dernier était tellement complexe et peu souple, qu'il était quasiment impossible de l'utiliser au quotidien.

Fruit d'une collaboration étroite entre des développeurs ayant plus de 20 ans d'expérience et des laboratoires de recherche en software factory, la suite logicielle **Mobioos,** et plus particulièrement **Mobioos Forge,** se proposent de répondre à ce défi.

Méta-modélisation de l'application

La plateforme innove et se démarque de la concurrence en s'appuyant sur un outil de modélisation qui ne rentre volontairement pas dans les détails. En effet la modélisation s'effectue via un manifest (de type DSL) propre à **Mobioos,** et couvre toutes les couches d'une application (UI, API, Data, etc.), indépendamment des technologies.

Le méta-modèle permet de définir, entre autres :

- Les fonctionnalités globales de l'application (langues, plateformes supportées, etc.) ;
- Les différents écrans de l'application et les actions associées ;
- Les API disponibles (format REST) ;
- Les bases de données (modèle objet).

Des fonctionnalités de compilation et de validation du méta-modèle permettent de modéliser rapidement, et avec efficacité, l'application tout en garantissant sa cohérence.

Spécialisation du méta-modèle : l'interview

Les méta-modèles n'ont jamais suffi à répondre aux besoins des développeurs du fait d'une trop grande « variabilité » des scénarios techniques et fonctionnels. D'où l'intérêt de coupler aux méta-modèles d'autres formes de collecte d'information plus spécialisées. C'est pourquoi la conversion du méta-modèle en une application spécifique est réalisée à l'aide d'un assistant de type chatbot. Il gère, de façon élégante, l'épineux problème délaissé par la méta-modélisation : la spécialisation du modèle vers des complications d'implémentations et la gestion des interactions et contraintes entre les différentes couches d'application.

Le chatbot assistant

Après analyse du métamodèle, le chatbot interroge le développeur pour comprendre plus finement les besoins. Faut-il générer un frontend ? Un backend ? Une base de données ? Chaque réponse peut apporter d'autres questions plus spécifiques, mais il va ignorer les questions qui ne seraient pas en phase avec l'application en devenir et l'architecture ciblée. Cette approche résout de nombreux problèmes qu'il aurait été difficile de gérer par le biais d'une modélisation classique.

Cette "interview" permet de dépasser le périmètre fonctionnel et technique du méta-modèle pour aller au fond des vraies problématiques auxquelles ont affaire les développeurs : style d'architecture, framework, nomenclature du code, tests unitaires, etc. C'est ce qui différencie en profondeur **Mobioos Forge** des outils MDA complexes et des produits existants de low code / no code.

Les générateurs de code : une extension du métamodèle

En plus d'être en charge de générer le code dans les technologies souhaitées, les générateurs de code jouent un rôle prépondérant durant l'interview. Ils interagissent directement avec le développeur pour réclamer des réponses techniques propres à la technologie qu'ils génèrent.

Après l'interview, la génération du code est déléguée aux générateurs. Chaque générateur prend en compte les réponses fournies par l'interview et s'adapte aux contraintes édictées.

Ces générateurs sont disponibles dans un store dédié. Il contient plus de 11 langages de programmation différents (1) au lancement de la bêta.

Pour que la communauté des développeurs puisse intégrer de nouveaux concepts de programmation, les générateurs sont mis en open source sur GitHub ! A vous de le forker, les adapter, les améliorer. La documentation technique de l'implémentation d'un générateur est disponible.

Les générateurs, avant d'être déposés sur le store **Mobioos Forge,** sont testés via des jeux de tests dont ceux de SonarQube. Ces tests assurent la qualité et le fonctionnement du code généré.

EndoCode : code safe

A la fin d'une interview, **Mobioos Forge** génère 60 à 70% du code. Le code ainsi généré se nomme l'EndoCode, c'est à dire endogène aux générateurs. De par la fiabilité fournie par les générateurs, 100% du code généré est dit « zéro distorsion ». C'est à dire testé, fiable, robuste, sans bugs et sans dette technique.

ExoCode : et le développeur dans tout ça ?

Le code développé à la suite d'une génération de code se nomme l'ExoCode, car exogène aux générateurs. La distinction entre EndoCode et ExoCode permet de bien délimiter le périmètre de responsabilité entre l'apport réalisé par Mobioos Forge et celui que doit apporter le développeur.

Ce code est particulièrement relatif aux besoins métiers et aux ajustements. L'ExoCode reflète toute la richesse, la puissance et la créativité des développeurs, là où la machine n'est pas capable d'apporter de solutions.

Ceci, sans avoir besoin de définir de cadre d'architecture comme l'imposent certains outils low code /

no code qui au final apporte à la fois des contraintes et des interfaces qui se ressemblent toutes. C'est là toute la puissance de Mobioos Forge.

Open Source

Le modèle économique de Mobioos Forge est basé sur l'Open Source pour les générateurs et sur un modèle gratuit pour le produit. La solution est monétisée au travers d'un support et de formations pour ceux qui en ont besoin.

L'écosystème Mobioos : vers l'infini et au-delà

Mobioos Forge est au cœur d'une suite logicielle, nommée **Mobioos**, visant à améliorer la productivité, la gestion et le déploiement d'applications. Cet écosystème est composé de deux autres produits : **Mobioos Fusion (Store)** et **Mobioos Magnet (déploiement).**

CES 2019 : DÉCOLLAGE IMMÉDIAT !

Actuellement en **BETA fermée**, la plateforme sera en **BETA ouverte** lors de son annonce officielle au CES de Las Vegas 2019, le 8 janvier 2019.

Les évolutions du produit seront très soutenues jusqu'à sa version finale prévue le 8 Janvier 2020. A cette date, deux autres outils seront annoncés : **Mobioos Fusion** et **Mobioos Magnet**. Entre-temps les produits seront améliorés avec la communauté et les compagnies qui utilisent déjà la plateforme.

COMMENT TESTER LA PLATEFORME ?

Rendez-vous sur *Mobioos.ai* ou sur le *marketplace Visual Studio* en précisant dans vos recherches « Mobioos Forge ». Un seul prérequis : Visual Studio 2017 en Community Edition ou toute autre version supérieure (version Windows).

Depuis GitHub, téléchargez notre projet d'exemple, et lancez une première génération de code en suivant la Documentation Forge.

A suivre...

Avec en ligne de mire une compatibilité avec d'autres IDE comme **Visual Studio Code**, l'ajout de nouveaux générateurs technologiques, et la création d'une plateforme dédiée pour gérer et déployer les applications générées**,** il y a fort à parier que nous entendrons souvent parler de **Mobioos** dans les années à venir.

PREMIÈRE GÉNÉRATION DE CODE : L'AVENTURE COMMENCE ICI

Installation

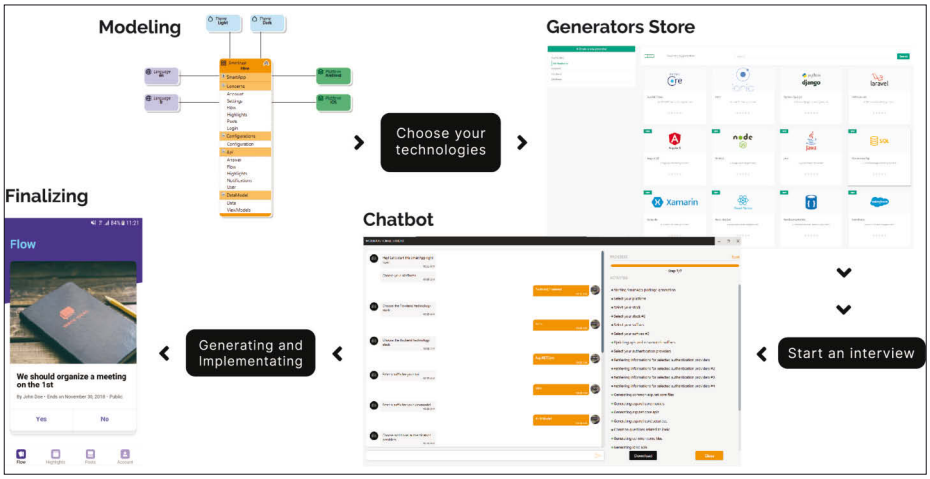
L'installation de Mobioos Forge se fait par le biais d'une extension VisualStudio (VSIX) disponible dans le MarketPlace de l'IDE.

Modélisation

La modélisation se fait à partir d'un DSL (2) propre à l'approche et au concept poussé par la suite **Mobioos**. Afin de faciliter la création d'une modélisation, quelques templates sont mis à disposition. Pour cet exemple, le projet vide par défaut suffit. La modélisation s'effectue ensuite autour d'un objet central, la **SmartApp.** Cet objet contient des informations importantes comme les plateformes (Android, IOS , web, etc.) et les thèmes disponibles ainsi que les langues supportées.

Une **SmartApp** adresse de nombreux domaines d'une application :

- Les vues et les interfaces utilisateur sont modélisées sous forme de **Concerns**.
- Les données et entités à modéliser sont créées dans le domaine **DataModels.**
- Les interactions entre les services et couches applicatives s'effectuent en modélisant des objets **API**.



De la modélisation à la génération de code

Validation

Le développeur pourra ensuite "valider" sa modélisation (par un ensemble de règles définies dans l'outil) afin de pouvoir accéder à ***la génération du code.*** avec un clic droit -> Generate code.

Génération du code

Le développeur utilisera ses identifiants Mobioos (compte créé gratuitement et permettant notablement d'accéder au store des générateurs) pour s'authentifier et commencer l'interview de génération. Par la suite, un ensemble de questions fonctionnelles et techniques lui seront posées, permettant aux générateurs de code de générer du code au plus proche des besoins du développeur, comme la sélection des plateformes cibles (frontend, backend, database …), les technologies à utiliser pour ces plateformes et d'autres questions posées directement par les générateurs concernés.

Et voilà !

A ce stade 60% du code de votre application a été généré automatiquement. Une fois le code généré et téléchargé, le développeur est libre de s'approprier ledit code à sa guise, et d'y implémenter la logique métier nécessaire au bon fonctionnement de l'application.

Mobioos Forge est en permanente évolution, et si aujourd'hui, la création des designs de l'application est par exemple sous la responsabilité du développeur, il est prévu à l'avenir de lui donner la possibilité par exemple de s'interfacer à des outils spécialisés UI/UX comme Sketch, Framer, etc… Et encore bien d'autres fonctions et services qui seront précisés dans la feuille de route du produit.

(1) A l'heure où nous imprimons, Forge propose les générateurs suivants : ASP.Net Core MVC, Xamarin, Apex (Salesforce), React-Native, SQL, DocumentDB, Laravel, Python, Java, Node.JS, Ionic. D'autres générations sont à prévoir dans le futur.
(2) Domain Specific Language

Développement mobile

Choisir les bons outils

Encore un dossier pour parler de développement mobile ? Si ce sujet vous paraît inutile et multi-abordé, vous auriez un peu raison. Mais il n'est jamais inutile de rappeler quelques fondamentaux et surtout les tendances actuelles.

Car finalement, le développement mobile évolue régulièrement : nouvelles versions des environnements, nouveaux frameworks, nouvelles approches applicatives. Dans ce dossier, le premier de l'année 2019, nous allons parler de différents aspects pour comprendre les enjeux et les bonnes questions à se poser : hybride vs natif, JavaScript, Xamarin, les problèmes du multi-plateforme, Ionic.

Une des questions essentielles est l'éternelle : quelle technologie / plateforme pour mon projet ? N'oubliez jamais que ce choix n'est pas neutre et conditionne vos développements.

La rédaction

Kevin VAVELIN,
Directeur Général de Geek-Inc.

Applications mobiles : hybride ou natif, faut-il encore choisir ?

En 2007, lorsqu'Apple présenta l'iPhone, la téléphonie s'est vue révolutionnée. Apple n'était pas la première compagnie à inventer le concept des applications sur un objet aussi commun qu'un téléphone, Handspring l'avait fait en 2002 avec son smartphone Treo avec la même interface que le PDA Palm, Nokia l'avait fait dès 1999 avec un système plus rudimentaire. Mais Apple a été la première à définir des interfaces simples à utiliser avec nos doigts, les applications nouvelles et la couleur et à proposer une plateforme complète.

Depuis, les choses se sont enchainées rapidement. Les premières applications sur iPhone étaient des applications web. L'année suivante, Apple présentait son App Store. Et c'est à ce moment-là que les choses ont réellement changé ; on avait droit à toute la maturité de leur système OS X dans un os mobile.

Les premières applications sur l'App Store étaient écrites en Objective-C en utilisant la bibliothèque créée par Apple et basée sur Cocoa : UIKit. Dans le même temps, Google mettait à disposition son OS Android, et les applications étaient ici créées avec Java. S'ensuit une décennie de prouesses technologiques. Les sites web avaient maintenant des applications pour augmenter leurs possibilités sur les différentes plateformes.

Voici quelques exemples d'une des premières applications natives ayant connu le succès sur l'App Store d'Apple (et qui a permis notamment la démocratisation de Twitter, à l'époque connu sous le nom Twttr), Twitterrific : 1

Vous pouvez suivre la genèse de cette application vieille de 10 ans déjà sur le blog d'IconFactory :

https://blog.iconfactory.com/2018/03/a-lot-can-happen-in-a-decade/

1

Première version de Twitterrific sur iOS Avant apparition du SDK avec iOS 2.0

Twitterrific sur iOS 2.0 A la sortie de l'App Store

La révolution mobile

De 2008 à 2015, le domaine des applications mobiles et des applications web a grandi rapidement. Les navigateurs mobiles permettant d'accéder à des composants mobiles (position gps, gyroscope, etc.), de nouveaux formats d'affichage (on commençait donc à parler de responsive design), un tout nouveau monde. Toutes ces nouvelles technologies ont donc permis à JavaScript de gagner de plus en plus de présence et d'impact. Les navigateurs ouvrant de plus en plus d'API.

La liste des API disponibles est accessible ici-même :

https://developer.mozilla.org/en-US/docs/Web/API

Parmi ces APIs figurent par exemple le gyroscope, apparu dès la sortie d'iOS 2.0 sur les iPhone 3G et la sortie des premiers appareils Android :

```
let gyroscope = new Gyroscope({frequency: 60});
```

```
gyroscope.addEventListener('reading', e => {
  console.log("Angular velocety along the X-axis " + gyroscope.x);
  console.log("Angular velocety along the Y-axis " + gyroscope.y);
  console.log("Angular velocety along the Z-axis " + gyroscope.z);
});
gyroscope.start();
```

Ou encore l'interface TouchEvent et Touch qui permettait de détecter une interaction multi-touch comme si l'on cliquait avec la souris sur notre bureau.

Dans ce contexte de grande compétitivité, les éditeurs de sites web ou de solutions web avaient deux solutions qui s'ouvraient à eux :

- Investir du temps et donc de l'argent pour créer/continuer leurs applications natives ;
- Optimiser leurs sites web et en faire des applications web.

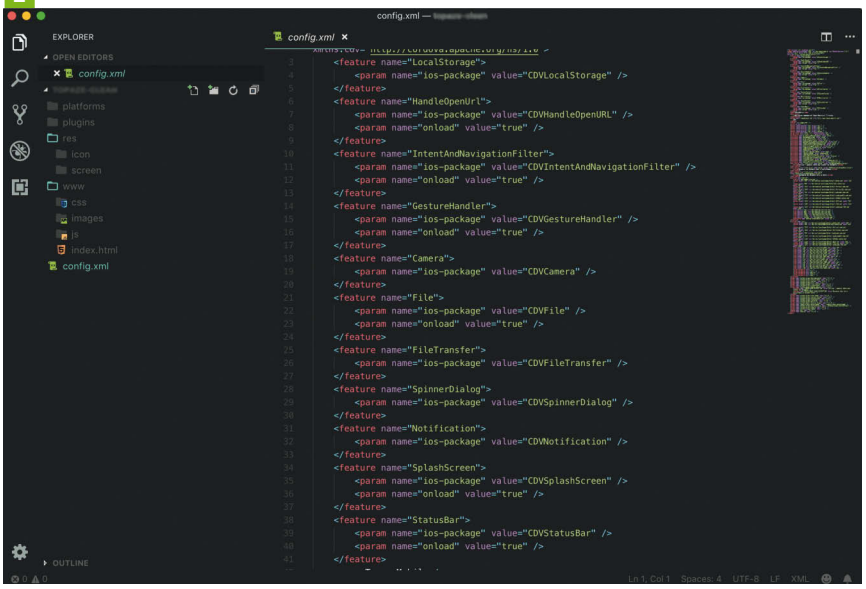
Des solutions comme Cordova/PhoneGap, réduisaient fortement cet investissement sur les applications natives en permettant aux applications web de s'interfacer simplement avec les possibilités des plateformes natives grâce à des plugins. Les éditeurs avaient donc enfin la possibilité d'appeler du code natif directement depuis leurs scripts JavaScript, leur permettant de prendre des photos, accéder au lecteur d'empreintes, etc.

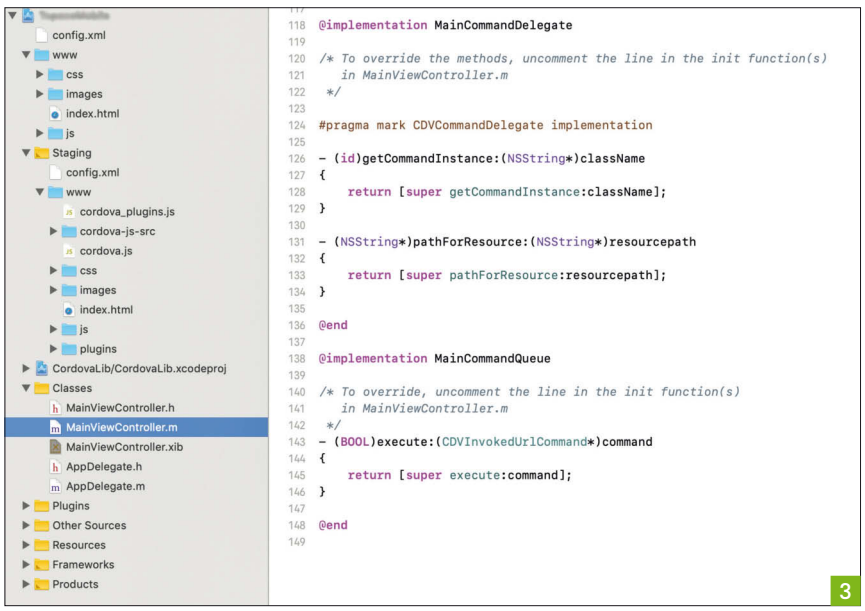
Ce fut pendant quelques années le meilleur moyen d'avoir une application native avec une base web, le seul investissement était uniquement sur la partie JavaScript et CSS et leur site. Puis arrivèrent deux frameworks qui révolutionneront considérablement l'industrie web : NodeJS et AngularJS.

La montée en puissance des frameworks Web

De 2008 à 2009, le seul moyen d'avoir des applications hybrides était de passer par ce que l'on appelle un "wrapper" afin de profiter des plateformes de distributions mobiles des différents éditeurs (App Store, Google Play Store, Microsoft Store), la technique consistait à embarquer nos fichiers HTML, CSS et JavaScript dans notre application et de charger leur contenu dans un composant appelé WebView.

2





3

En 2009, lors de l'iPhoneDevCamp, quelques ingénieurs de la société Nitobi Software commencèrent à développer un framework qu'ils appelèrent PhoneGap. Ce framework fut une grande réussite et remporta même un prix lors de la O'Reilly Media Web Conference. Apple donna son accord pour que les applications créées avec PhoneGap puissent être publiées sur le store.

La technologie de PhoneGap était quelque peu révolutionnaire à l'époque car elle permettait en quelques commandes de configurer une application native en générant automatiquement le code nécessaire pour utiliser le dit "wrapper". Ils sont même allés beaucoup plus loin en inventant une API autorisant les développeurs à concevoir leurs propres plugins et de les intégrer à PhoneGap (cela ouvrit notamment la voie à des plugins très célèbres comme le scan de QRCode, l'utilisation de la bibliothèque de photo, etc.).

A cette période, les applications étaient toujours écrites en pur HTML/CSS/JavaScript et contenaient uniquement un fichier config.xml indiquant à PhoneGap toutes les informations nécessaires à intégrer (nom de l'auteur, nom du package, plugins utilisés, autorisations spéciales, dossier contenant les images pour le site web, etc.).

A partir de 2010, la librairie jQuery Mobile sortait enfin, permettant d'adapter nos sites web avec les nouvelles interactions rendues possibles grâce aux smartphones. Elle fut très largement utilisée conjointement avec PhoneGap. (NB: Encore aujourd'hui, une application leader dans les solutions Télévitale est encore en circulation et écrite avec PhoneGap et jQuery Mobile, Topaze Télévitale). Voici à quoi ressemble une architecture d'une application Cordova : 2

Comme vous le voyez en 2, écrire une application Cordova nécessite juste d'écrire votre code HTML/CSS/JavaScript dans le dossier www, et une petite commande "cordova build ios" pour générer une application iOS ou "cordova build android" pour générer une application Android. 3

Dans le même temps, à quelques mois d'écart, sortait également AngularJS. L'idéologie initiale d'AngularJS était de compléter les possibilités de Cordova (initialement PhoneGap mais renommé suite au rachat de Nitobi Software par Adobe, puis donné à la fondation Apache). Ce framework connut le succès, à la fois chez les développeurs web que chez les développeurs mobiles. Les entreprises adoptèrent ce framework assez rapidement (aujourd'hui le framework est toujours maintenu et utilisé par Apple sur son site web, Twitter, Google Maps, Air France, etc.).

Cependant, il manquait encore une chose pour que les applications hybrides soient pleinement considérées par les industriels. Il manquait le "look and feel" natif pour créer l'illusion parfaite d'une application native avec ses transitions, ses tailles, ses couleurs, etc. Et c'est dans ce contexte qu'arrive Ionic en 2013, basé sur Cordova et AngularJS.

Ionic permit d'avoir un environnement complet de développement d'applications hybrides. Avec une prévisualisation de l'apparence de son application directement depuis son navigateur web, un outil de packaging clé en main pour la publication sur les stores, et

même une application compagnon permettant d'envoyer son application en aperçu chez l'utilisateur qui n'avait plus qu'à installer leur application Ionic View pour la tester (ou suivre l'avancement du développement).

Ionic fonctionne exactement comme Cordova, voici par exemple une application générée par Ionic (la v1 d'Ionic reposait sur AngularJS comme notre exemple) : 4

On y retrouve la même architecture, le dossier www, notre config.xml, et les différents plugins de base Cordova. L'avantage d'Ionic était de permettre aux développeurs de prévisualiser leurs applications directement depuis leur navigateur internet, grâce à ce que les concepteurs d'Ionic appellent, l'Ionic-Lab. Voici à quoi cela ressemble : 5 6

Ionic fournit une excellente documentation pour son produit, avec des exemples de codes et des prévisualisations de ce que cela fera sur votre OS. Voici par exemple le code pour créer une barre d'items au bas de votre page :

```
<div class="tabs">
 <a class="tab-item">
  Home
 </a>
 <a class="tab-item">
  Favorites
 </a>
 <a class="tab-item">
  Settings
 </a>
</div>
```

Et voici le résultat : 7

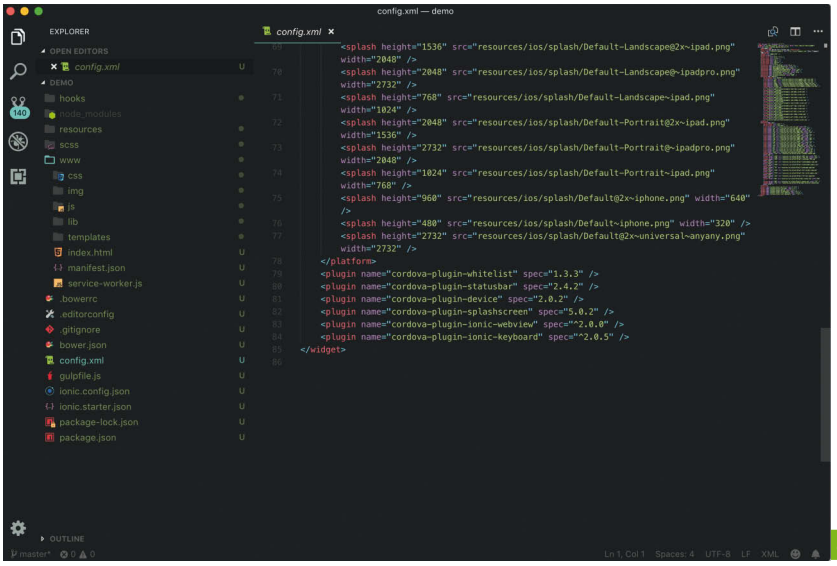
7

Pour apprendre Ionic, rien de plus simple que de se rendre sur https://ionicframework.com/ avec vos connaissances en AngularJS (pour la v1) ou Angular (pour la v3, v4 actuellement en beta).

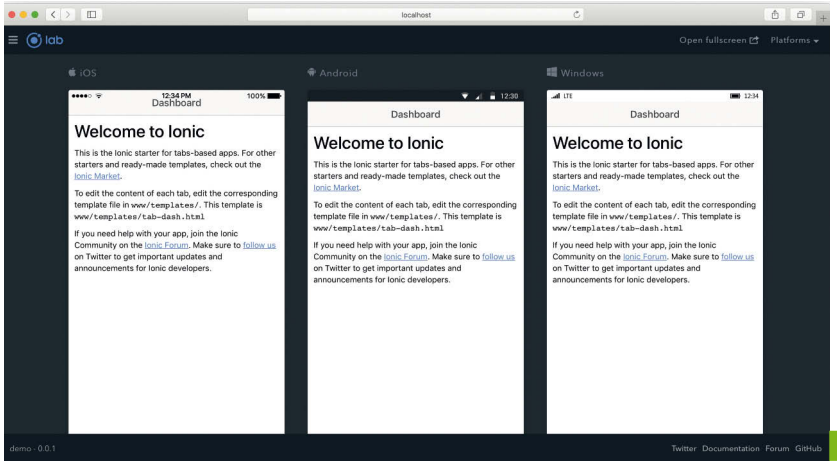
Vers la disparition de Cordova

Cordova a servi pendant de très longues années à notre industrie, mais les évolutions des SDK natifs se faisant de plus en plus rapidement (mise à jour majeur annuelle, puis trimestrielle pour les mises à jour mineures), les développeurs de Cordova ont de plus en plus de mal à maintenir le projet compatible avec d'anciens SDK et à adopter les nouvelles fonctions.

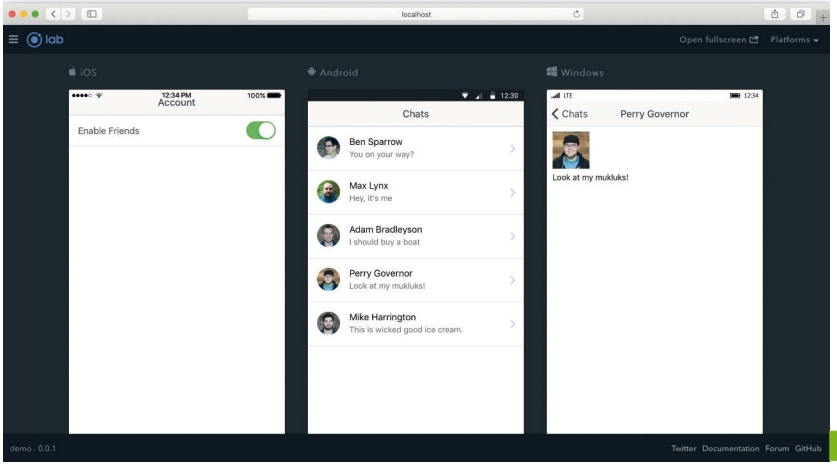
Facebook, face à cette problématique, décida de créer React Native. Une librairie similaire à Cordova mais qui permet cette fois de générer chaque composant nativement sans passer par un générateur de code. React Native est encore jeune (commencé en 2012 et toujours au stade de beta à la publication de cet article), mais cela n'empêche pas les entreprises de miser sur ce framework qui montre déjà des possibilités extraordinaires et des performances proches du natif. On peut citer notamment Netflix, Facebook, Whats App, Facebook Messenger, Airbnb, Uber, comme utilisateurs de React Native afin de créer leurs applications natives.



4



5



6

Voici à quoi ressemble une application fraîchement initialisée avec React Native : 8

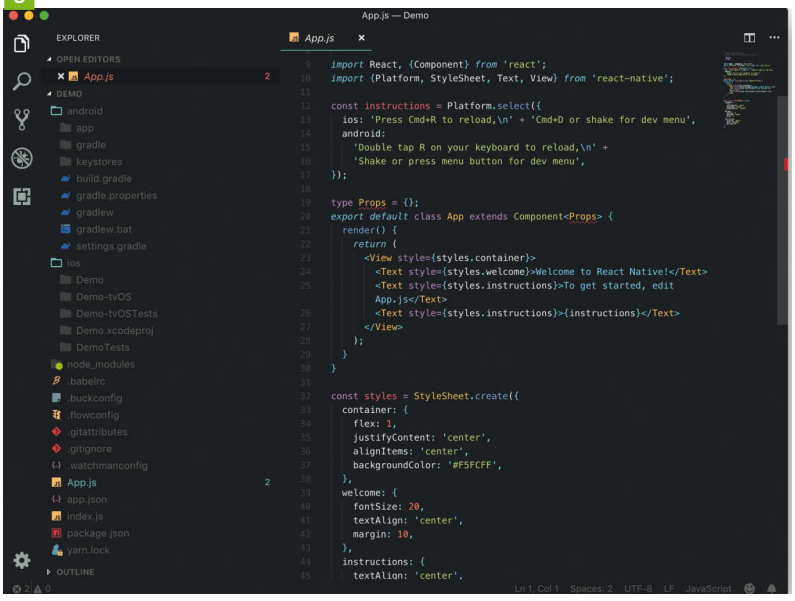
Comme vous pouvez le constater, il suffit d'écrire votre application avec React (et les composants de react-native), et vous avez une application native générééé sans Cordova).

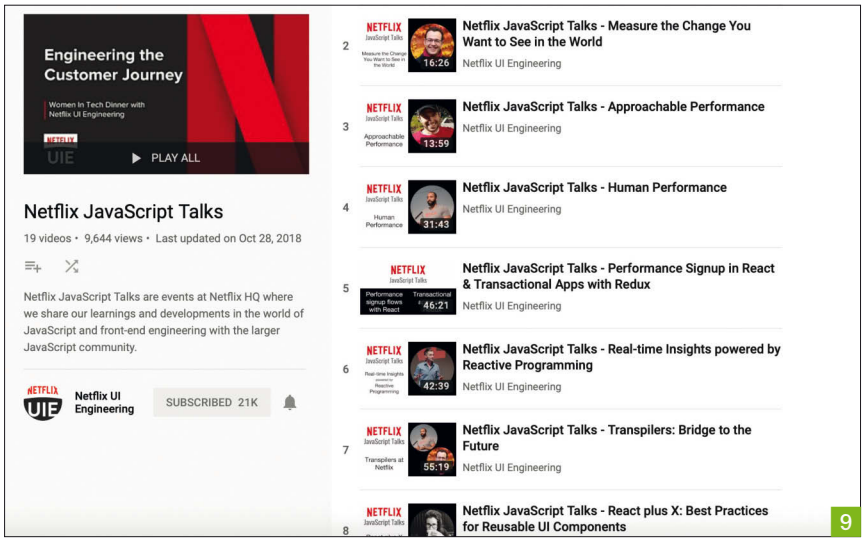
React Native est un framework encore jeune mais très prometteur. Vous pouvez constater qu'il ne génère pas autant de dépendances que Cordova, et se concentre entièrement sur ses fichiers JS. Vous pouvez expérimenter ce framework et en apprendre davantage sur lui en vous rendant sur sa documentation officielle https://facebook.github.io/react-native/docs/getting-started .

Les composants sont encore très peu disponibles (encore une fois, react-native n'est qu'en phase de beta) mais les éditeurs comme Netflix n'hésitent pas à concevoir leurs interfaces en pur React et les transposer ensuite sur React Native (c'est pour cela que vous avez la même experience sur toutes les plateformes que supporte Netflix).

Si vous souhaitez en savoir plus sur comment Netflix gère toutes ses plateformes et surtout son code, sachez qu'ils ont une chaîne dédié sur YouTube ou ils diffusent leurs conférences internes : 9

8





9

Ionic est toujours présent et se repose toujours sur Cordova pour l'essentiel mais est également en train de s'en séparer progressivement, préférant adopter des concepts de React Native afin de coller au mieux au besoin de l'industrie. Aujourd'hui Ionic permet par exemple de créer ce que l'on appelle des Progressive Web Apps grâce à Angular (une nouvelle version d'AngularJS basée sur TypeScript cette fois-ci, reprenant les concepts de React et de Polymer), qui permettent d'obtenir des applications natives sans passer par le store des éditeurs.

Google n'est pas en reste : ils ont également créé leur propre framework concurrent à React Native. Pour répondre au besoin de créer des applications hybrides tout aussi performantes que les applications natives, Google sortit le framework Flutter. 10

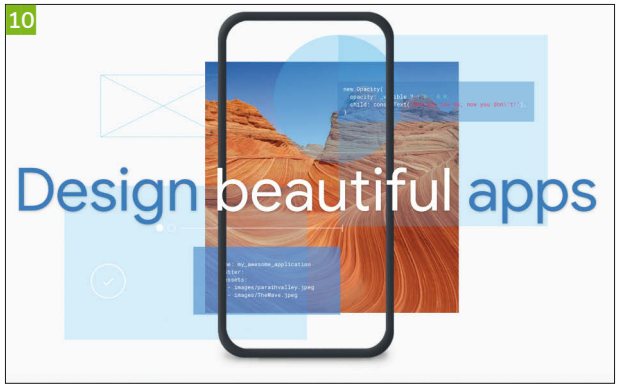
Aujourd'hui, pour écrire une application avec Flutter, il faut maîtriser le langage de programmation Dart. Mais Google n'exclut pas la possibilité, dans le futur, d'exploiter Angular.

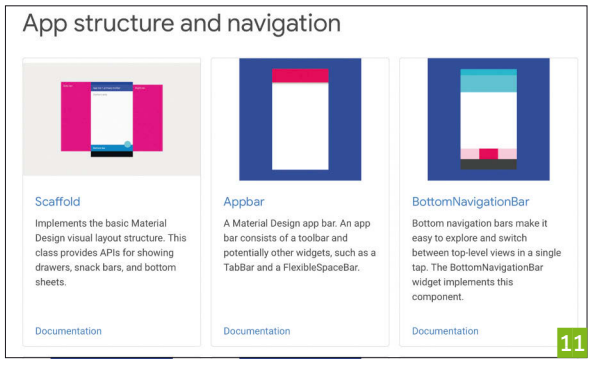
Flutter s'appuie énormément sur Material Design, et une bonne liste de composants est déjà disponible afin de créer des applications très simplement avec Flutter. 11

Voici un petit exemple de code avec un layout tout préparé qui reprend le design de Gmail sur appareils mobiles :

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {

 @override
 Widget build(BuildContext context) {
  return MaterialApp(
```

10





11

```
    title: 'Material Design test',
    theme: ThemeData(
      primarySwatch: Colors.blue,
    ),
    home: MyStatefulWidget(),
  );
 }
}

class MyStatefulWidget extends StatefulWidget {
 MyStatefulWidget({Key key}) : super(key: key);

 @override
 _MyStatefulWidgetState createState() => _MyStatefulWidgetState();
}

class _MyStatefulWidgetState extends State<MyStatefulWidget> {
 int _count = 0;

 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Programmez.com Code'),
    ),
    body: Center(
      child: Text('Vous avez appuyer sur le bouton $_count times de fois.'),
    ),
    bottomNavigationBar: BottomAppBar(
      child: Container(
        height: 50.0,
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () => setState(() {
          _count++;
        }),
      tooltip: 'Incrementer',
      child: Icon(Icons.add),
    ),
    floatingActionButtonLocation: FloatingActionButtonLocation.centerDocked,
  );
 }
}
```

Hybride ou natif : comment choisir ?

Le choix entre l'hybride ou le natif est une question qui revient souvent dans les entreprises. Alors dans un contexte aussi difficile, comment faire son choix ?

Le choix entre développer une application hybride ou une application native se situe désormais essentiellement sur le rythme de publication. Est-ce que vous allez souvent apporter des modifications sur votre produit ? Est-ce que vous devez appliquer des évolutions très rapidement ? Est-ce que vous devez avoir un socle commun pour les différentes plateformes ?

Si la réponse à toutes ces questions est oui, alors n'hésitez plus et foncez vers une application hybride. En revanche, si vous souhaitez fournir une expérience spécifique par plateforme, exploiter les appareils de cette plateforme dans l'intégralité, que vous pouvez investir du temps dans le développement, alors foncez vers le natif.

Le développement natif a un avantage indéniable sur l'hybride, c'est la mise à disponibilité des nouvelles interfaces, form-factor, dès la sortie des appareils. Sur ce point, le développement natif sera toujours en avance, même si depuis 2 ans, les frameworks hybrides réussissent à exploiter ces API en quelques mois.

Les éditeurs de plateformes natives en revanche, investissent de plus en plus de temps à la mise à disposition de leurs API natives sous forme web (exemple : MapKitJS, ApplePayJS, MusicKitJS). La différence entre hybride et natif devient de moins en moins flagrante, notamment sur Android qui exploite de plus en plus de framework JavaScript en son sein, mais ce gouffre se refermera d'ici quelques années car l'industrie a besoin d'investir dans l'hybride afin de limiter le coût et gagner du temps sur le développement d'applications très simples.

Par exemple pour MusicKitJS avec Apple, il vous suffit d'intégrer leur script JavaScript :

```
<script src="https://js-cdn.music.apple.com/musickit/v1/musickit.js"></script>
```

Configurer vos balises meta (avec Apple, il vous faudra un compte développeur afin d'y générer ce que l'on appelle un token pour pouvoir intégrer le contenu d'Apple Music sur votre page) :

```
<meta name="apple-music-developer-token" content="DEVELOPER-TOKEN">
<meta name="apple-music-app-name" content="My Cool Web App">
<meta name="apple-music-app-build" content="1978.4.1">
```

Et enfin, utiliser MusicKit en JavaScript comme suit :

```
let music = MusicKit.getInstance();

// This is called with or without authorization:
music.player.play();

// This ensures user authorization before calling play():
music.authorize().then(function() {
 music.player.play();
});

// You can wrap any method to ensure authorization before calling:
music.authorize().then(function() {
 music.api.library.albums.then(function(cloudAlbums) {
  // user's cloudAlbums
 });
});
```

Aujourd'hui, les écoles d'ingénierie enseignent à leurs étudiants essentiellement le développement d'applications natives, en insistant peu sur le développement d'applications hybrides (juste quelques heures sont assignées au développement hybride car ils préfèrent enseigner les bases du web avant tout). Mais certaines écoles comme l'Up ESI à Montpellier mise sur un enseignement calqué sur le marché informatique actuel, c'est à dire d'enseigner à la fois le développement natif, mais également le développement hybride, en insistant bien sur les raisons pour lesquelles nous devrions faire ce choix. •

David Poulin
Leader technique, architecte cloud & mobilité
Expert du développement logiciel et mobile, je suis passionné de technologies & d'innovation. Formateur certifié, .NET, Xamarin, depuis ces 2 dernières années, je me suis orienté principalement vers les technologies Javascript front que j'affectionne particulièrement (Angular, VueJs, React, etc..)

Ionic vs React Native : la battle des frameworks

Passionné du développement mobile natif ou cross plateforme, j'ai utilisé de très nombreuses solutions pour répondre aux besoins de mes clients. Si je ne devais m'attarder spécialement que sur les solutions cross plateforme hybrides ou celles où l'on peut utiliser JavaScript, je resterais sur Ionic et React Native qui sont de très bonnes technologies. Il est vrai que Ionic 1 avait été pour moi une expérience intéressante mais quelque peu douloureuse. Récemment, j'ai dû développer en utilisant Ionic 2+. Et avec un peu de recul, elle peut présenter un atout sérieux pour pas mal d'entreprises. Mais React Native n'en reste pas moins une excellente alternative. Lequel choisir ?

Approches de développement mobile

Il existe plusieurs approches de développement mobile parmi lesquelles nous retrouvons le développement natif, hybride (*PhoneGap, Cordova, Ionic*) ou cross plateforme (Xamarin, React Native, Flutter, Native Script, etc.). Chaque approche a ses avantages et ses inconvénients qui ont déjà été abordés dans de nombreux dossiers. Ce que l'on peut retenir, c'est que les arguments en défaveur du développement hybride ont en très grande majorité disparu (le « problème des performances et du rendu ») à cause de l'amélioration constante des solutions elles-mêmes, mais aussi des appareils et des navigateurs. Si on compare les solutions hybrides aux applications natives, on pourrait récapituler ainsi :

Application Native	Application Hybride
Développée à l'aide d'un langage spécifique tel que Java/Kotlin pour Android, Objective-C, Swift pour iOS, C#/XAML pour Windows (ou C# / XAML ou UI specifique pour Windows, iOS ou Android)	Développée en utilisant HTML, CSS et Javascript
Code séparé pour chacune des plateformes (sauf si utilisation d'un framework cross plateforme natif, Xamarin.Forms par exemple)	« Omnichannel », c'est-à-dire un code commun fonctionnant partout (tant sur le web que sur les appareils mobiles), un seul codebase
Cycle et temps de développement plus ou moins coûteux (selon l'utilisation ou pas d'une solution cross plateforme).	Cycle & Temps de développement rapides
Performances & expérience utilisateur excellentes	Performances & expérience utilisateur très intéressantes
Coût de développement & des ressources élevés	Temps & coûts de développement rapide

App Hybride versus Native

Qu'est-ce que Ionic ?

Ionic est un framework open source, optimisé pour la mobilité mais compatible pour le web. Il est possible de développer des applications cross plateformes avec HTML, JS et CSS. Ionic fait partie des frameworks dits hybrides, c'est-à-dire que l'on a un socle applicatif commun type web application qui sera intégré dans un composant type WebView natif (*Android, iOS, …*). L'avantage de cette nouvelle version d'Ionic (*contrairement à la version d'Ionic 1*) est qu'elle s'est beaucoup rapprochée des standards web.

Ionic est aussi une surcouche de Cordova.
Pour vulgariser, Cordova est un framework de développement mobile qui :

- Encapsule des API natives avec JavaScript ;
- Permet de créer des applications mobiles avec HTML, JS, CSS ;
- Dispose d'un bel écosystème de plugins ;
- A des millions de téléchargements.

Pourquoi donc Ionic 2+ (aujourd'hui en version 3 stable et 4 beta) ?

Pour plusieurs raisons, il faut l'avouer, mais si je ne devais citer que les plus importantes :

- Il est gratuit & open-source ;
- Il est construit sur Angular2+/TypeScript ;
- La gestion de la navigation a été revue (et grandement améliorée) ;
- La gestion des thèmes spécifiques plateformes est relativement aisée (*enfin…*) ;
- Il dispose d'un grand nombre d'outils pour faciliter la création des projets et l'industrialisation (Ionic cli, node, etc…) ;
- Dispose d'une excellente documentation ;
- Facile à apprendre ;
- A de bonnes performances globales ;
- Il permet de créer des apps à destination de iOS, Android, Windows, du Web et même PWA (contrairement à React Native qui ciblera uniquement iOS & Android).

Et React Native, qu'est-ce que c'est ?

Même si on utilisera du Javascript *et des composants React* avec React Native, une application mobile créée en utilisant la technologie n'est pas hybride, mais bel & bien une application native réelle. Créé par Facebook, React Native est une extension de ReactJs, utilisant les mêmes principes, c'est-à-dire qu'un DOM virtuel est utilisé pour créer une interface utilisateur. Son fonctionnement est particulier. Disons qu'en gros, un moteur JS va exécuter le code Javascript, être à l'écoute des changements UI et les appliquer. Il écoute, calcule, prépare en arrière-plan les modifications nécessaires à appliquer, et, une fois que tout est prêt, il les applique en petits batchs. Il s'exécute dans un thread différent du thread UI mais communique avec lui grâce à un pont capable d'aider le moteur JS et le thread principal à échanger de façon asynchrone des informations. Avant d'aller plus loin dans le sujet, si on devait

résumer et comparer grossièrement les 2 technologies, Ionic et React Native, nous retrouverions ce tableau.

	Ionic	React Native
Type d'application	Hybride	Cross-plateforme
Langages	Technologies Web (HTML, CSS, Javascript, Angular, TypeScript)	React (JSX), Javascript / TypeScript
Plateformes supportées	iOS, Android, UWP, PWA, Web	iOS, Android, UWP
Performances	Performances intéressantes (rendues via la WebView native)	Excellentes performances
Documentation	Excellente documentation, claire et consistante	Bonne documentation, mais assez légère sur certains points
Réutilisation du code	Optimal, c'est le même code base pour toutes les plateformes (y compris le web)	Même code base pour toutes les plateformes (**sauf** le web). Notons que le code spécifique d'une plateforme a besoin d'être adapté
Courbes d'apprentissage	Excellente car utilise des technologies Web connues et plus ou moins maitrisées par le plus grand nombre	Bonne, mais demande pas mal d'efforts au début, ce qui peut prendre plus ou moins de temps.
Tests de son code	Émulateur, vrai appareil ou n'importe quel navigateur	Émulateur ou vrai appareil nécessaire

Ionic vs React Native

Création des applications, ça donne quoi ?

Quand on débute sur le développement mobile, démarrer avec des CLIs est vraiment pratique. React Native & Ionic en proposent tous les 2 mais pas vraiment avec les mêmes capacités. Du coup, pour les exemples ci-dessous, il est impératif d'avoir NodeJs installé sur votre machine, et nous utiliserons sa console.

De l'installation à la création de votre premier projet React Native

Il faudra pour démarrer installer Expo. C'est un outil extra pour pouvoir démarrer son application React sans trop d'efforts.

```
npm install -g expo-cli
```

Une fois installé, nous pouvons créer notre projet avec la ligne de commande

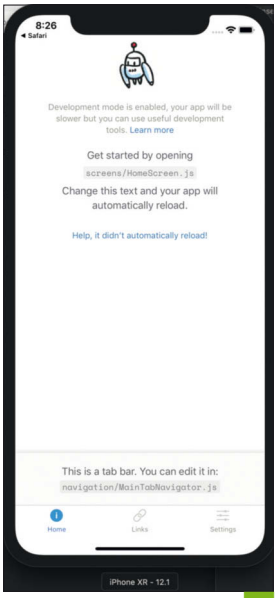
```
expo init ma-superbe-app-react
```

On a la possibilité de choisir un template (entre _blank ou tabs). En choisissant ce dernier
Une fois le projet créé, il suffit juste de le démarrer.

```
cd ma-superbe-app-react
npm start
```

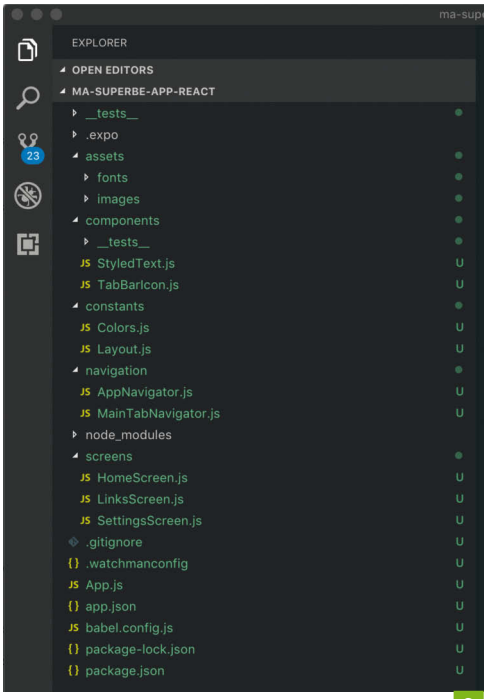
Alors… c'est là où cela peut être ensuite un peu compliqué. Pour tester votre application sur votre mobile, il est nécessaire d'avoir créé un compte expo (désolé pour l'ascenseur émotionnel), ou alors avoir installé au préalable les émulateurs iOS / Android. 1

1



2

Et en l'installant sur mon émulateur iPhone, j'ai enfin mon rendu ! 2
En ouvrant les sources de votre application, vous réaliserez que la hiérarchie par défaut, même si elle ne vous est pas familière, constitue un excellent point de démarrage pour une première prise en main (un dossier screens pour … des écrans peut être ? un dossier components pour des composants … ? etc., etc.). 3



3

De l'installation à la création de votre premier projet Ionic

C'est également assez rapide pour un premier démarrage.
1 – L'installation se fait avec une simple ligne de commande :

```
npm install -g ionic
```

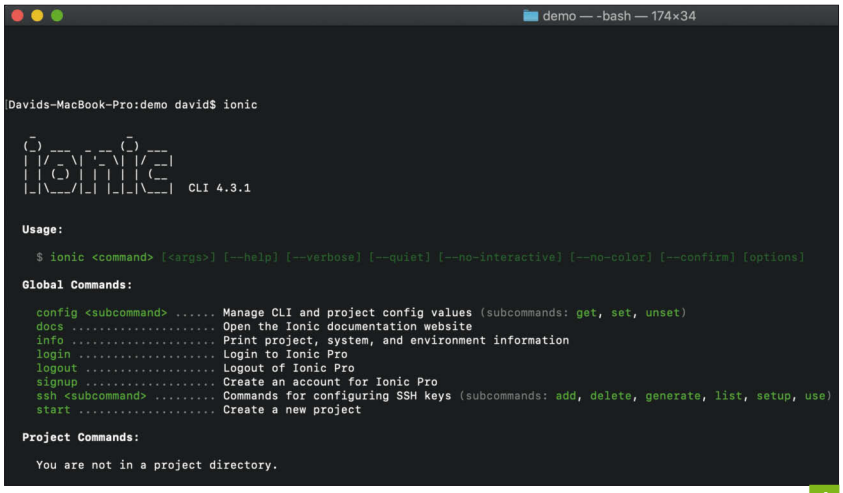
2 – la CLI Ionic est assez sympathique et jolie. Si vous êtes perdus ou ne savez pas quoi faire, il y a juste à taper la commande : ionic
Vous retrouverez un récapitulatif de toutes les commandes possibles à utiliser avec Ionic. 4
3 – Si vous souhaitez créer un type de projet spécifique Ionic, pour découvrir les possibilités vous pouvez vous aider de la commande :

```
ionic --list
```
5

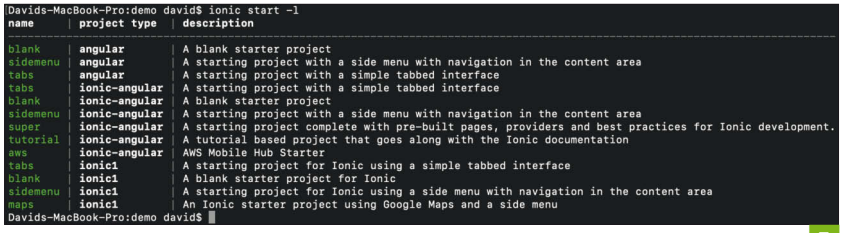
4 – Ainsi donc, si je souhaite créer une application avec des onglets, je n'ai juste qu'à exécuter la commande :

```
ionic start super-projet-ionic tabs
```

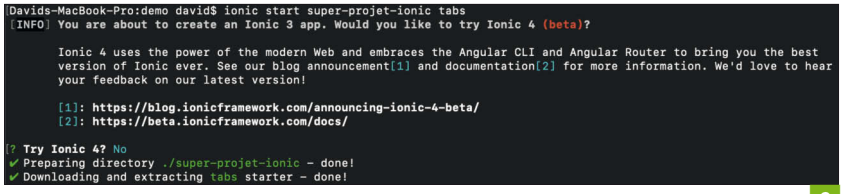
(*Notez que le type de projet se trouve à la fin de la commande*). 6
Par défaut, la CLI me demande si je souhaite essayer Ionic 4 (*la ver-*



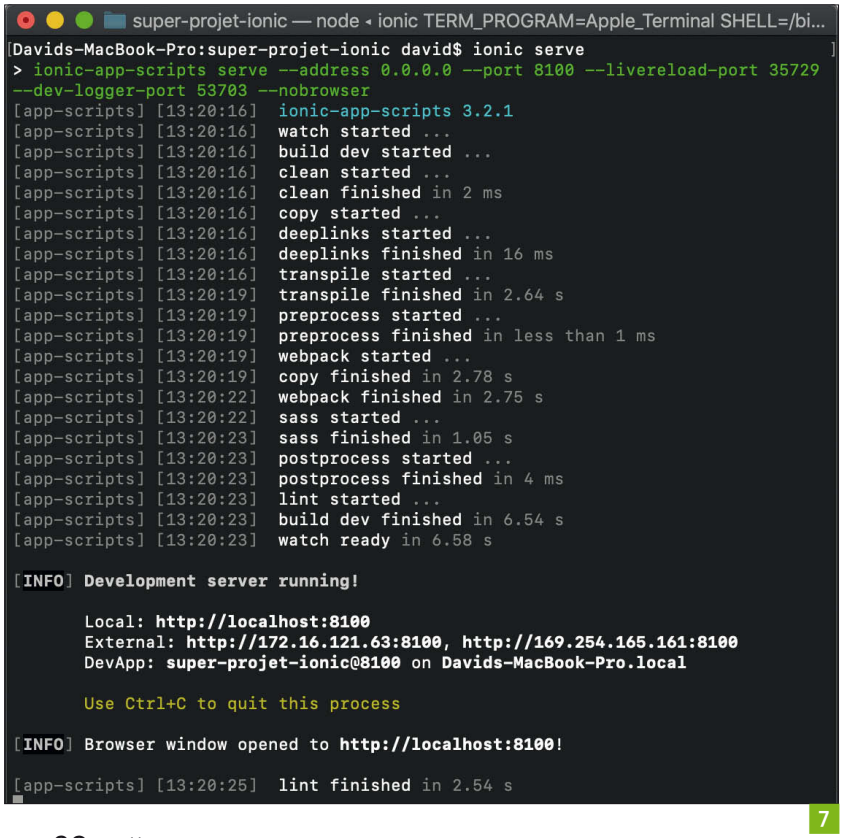
4



5



6



7

sion 3 à ce jour étant la dernière version stable). Je choisis non, et je laisse tout rouler. Et c'est tout. C'est un point fort de Ionic là où avec React Native, par défaut, on ne peut pas choisir un type de projet que l'on voudrait créer.

5 – last but not least, si on souhaite tester notre application créée, il sera possible de le faire via le navigateur web, un vrai appareil, un émulateur ou avec Ionic View. Dans cet article, uniquement le navigateur web nous intéresse.

Nous utilisons donc la commande :

```
ionic serve
```

7

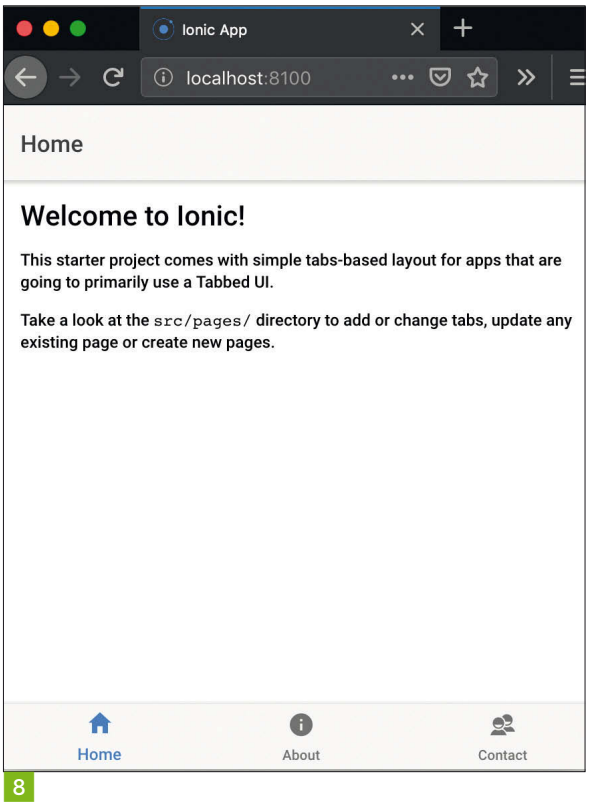
Une nouvelle fenêtre (ou un nouvel onglet) du navigateur s'ouvre et on y découvre notre jolie application : 8

Un des avantages de Ionic réside aussi dans sa capacité à restituer les modifications instantanément (le « hot reload », tout comme pour React Native d'ailleurs). Par exemple, je peux modifier un label, sa couleur, juste enregistrer, et le projet se remet à jour dans notre navigateur. Si vous affichez la fenêtre de développement Web, en activant le design mode responsive, il est possible de pré-visualiser le rendu sur un appareil spécifique : 9

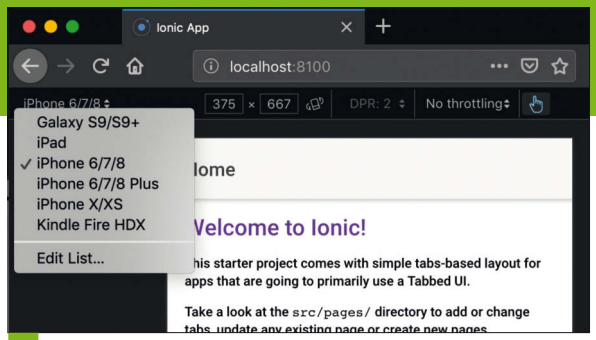
Structure d'une nouvelle application

Si j'utilise la commande tree sur mon mac (installée via brew install tree), on y découvre cette arborescence. 10

Il n'y a rien de spécifique à une plateforme en particulier (idem pour React Native au début). En effet, le développement avec ces approches nous impose de nous concentrer sur le développement de notre application sans avoir à se soucier de la plateforme de destination pour le moment. Si vous faites du développement web (*en particulier Angular*), il sera assez facile pour vous de prendre en main un projet Ionic ou alors de monter en compétence sur le sujet. Quand bien même la plupart des composants Ionic vous seraient



8



9

inconnus. Il n'est donc pas particulièrement « difficile » de créer un projet Ionic de bout en bout. La documentation est très bien fournie sur tous les sujets principaux (les composants, la navigation, le cycle de vie, les composants spécifiques, ou encore Ionic native).

Et donc ? React Native ou Ionic ?

J'aime beaucoup les 2 approches. Chaque framework dispose d'une énorme communauté, d'un bon support, de forums de discussions, ce qui rend la possibilité d'être aidé assez aisément si on est coincé. J'avoue que j'ai une affinité certes pour React Native (pour les performances et le rendu), mais surtout pour Ionic pour la simplicité de prise en main et du démarrage.

Qu'est-ce que je ne dis pas concernant React Native ?

Il faut vraiment de solides compétences. Le JSX n'est pas facile à prendre en main si vous n'avez pas un background React. Le JSX n'est pas du HTML, donc, il ne suffit pas de connaître ce dernier pour prendre en main React Native. Et puis, on va créer beaucoup de composants. Vraiment beaucoup pour avoir une UI intéressante. Par ailleurs, il faut bien souvent des connaissances dans le développement natif, car … il peut manquer des modules (composants) personnalisés que vous aurez à créer vous-mêmes. J'ai pu rencontrer des difficultés pour utiliser des composants Android (mon impression avait été que React Native avait été créée principalement pour iOS…). Aujourd'hui, il faut avouer que 90% du code toutefois peut être partagé.

Qu'est-ce que je ne dis pas concernant Ionic ?

Alors, oui, pour être complètement transparent, la technologie est très intéressante, et permet de créer des applications très rapidement. Si on a trop de callbacks, ou alors des interfaces avec des graphiques complexes (avec animations), il est possible d'avoir des problèmes de performances comme sur les tests que j'avais pu mener (*et nous pourrons en parler*). Là, où pour React Native, les performances étaient sans égales. Après, si la communauté Ionic est assez active, le développement va dépendre de plugins créés pour Cordova/PhoneGap (via Ionic native qui propose des wrappers par-dessus ces plugins), et parfois, ils ne sont plus vraiment maintenus. Par ailleurs, il va être rapidement nécessaire d'utiliser beaucoup de plugins pour utiliser des fonctionnalités natives.

Mais la révolution arrive-t-elle peut-être enfin pour Ionic ?

Effectivement, récemment, l'équipe Ionic a annoncé Capacitor https://capacitor.ionicframework.com/ .

Pour l'expliquer simplement, Ionic repose principalement sur Cordova. En reposant sur lui, on a besoin de ses dépendances (plugins notamment). Demain, cette surcouche disparaîtra au profit de Capacitor qui prendrait la relève (même s'il y a un support pour la rétro compatibilité avec les plugins Cordova).

10

Capacitor va s'installer via une simple ligne de commande :

```
npm install @capacitor/cor@capacitor/cli
```

Pour que Capacitor puisse fonctionner correctement, nous avons besoin que les sdk natifs (iOS & Android) soient installés sur notre machine de développement.

Si nos applications n'avaient pas besoin d'être compilées, c'est maintenant différent avec Capacitor. Son fonctionnement se rapproche beaucoup de ce qui se fait avec React Native, ce qui fait que Ionic pourrait ainsi représenter un concurrent sérieux dans le cadre du développement mobile cross plateforme parce qu'il réussit à résoudre les problématiques liées à ses points faibles (dépendances externes et soucis de performances notamment) avec ce runtime quand bien même en beta.

Comment donc faire le bon choix ?

Il n'y a pas vraiment une bonne réponse adaptée, car cela dépend des deadlines du projet, de la complexité, des compétences de l'équipe, du budget, des performances attendues (bien que sur ce dernier aspect, Ionic réussit à combler ses faiblesses).

En considérant tous ces facteurs alors, il est possible de prendre une décision facile. Si aujourd'hui, votre équipe maîtrise le développement React, alors, React Native peut être un excellent choix. C'est pareil si votre équipe utilise aujourd'hui principalement TypeScript, Angular, Ionic est le choix légitime.

Quoi de mieux de tester les 2 approches sur des projets simples pour vous donner une meilleure idée ? Bon développement ;) •

John Thiriet
@johnthiriet
Microsoft MVP
Technical Manager Mobilité
@Cellenza

Cellenza

Les bonnes pratiques mobiles

« CELUI QUI NE PROGRESSE PAS CHAQUE JOUR RECULE CHAQUE JOUR » CONFUCIUS

Il n'existe pas de recette enseignant comment créer à coup sûr une bonne application mobile. Elle est surtout le fruit d'un long travail d'amélioration continue permettant de répondre de manière satisfaisante aux besoins de ses utilisateurs. Dix ans de développement mobile nous ont cependant montré ce qui menait à coup sûr à l'échec d'une application.

Je vais me concentrer ici sur quelques éléments à suivre pour éviter de gros soucis sur vos applications. Ces éléments sont généraux et ne dépendent pas d'une technologie mobile particulière. Cependant, les quelques illustrations de cet article utiliseront Xamarin et C#, les technologies que je maîtrise le mieux.

Réseau

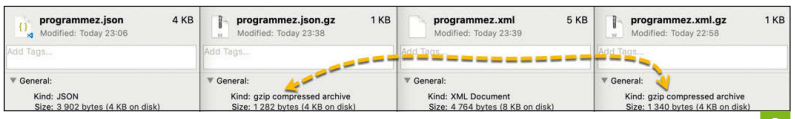
Au risque de démontrer une évidence, la plupart des applications mobiles dépendent d'un accès réseau pour charger les données qui leur sont indispensables. Un téléphone n'est cependant pas branché à un bon vieux câble réseau permettant une connexion stable. Cela assurerait aux programmeurs un risque d'échec des appels très faible. Une connexion mobile alterne entre Wifi, 4G, 3G et toutes leurs versions intermédiaires et peut échouer à n'importe quel moment. De même, un téléphone peut être connecté en 4G et pourtant ne pas avoir de connexion à internet. Cela dépend du nombre de personnes dans la même zone, ou bien du téléphone lui-même.

Une application mobile dispose donc par essence d'une connexion non fiable, à vitesse parfaitement variable et pouvant échouer à n'importe quel moment.

L'objectif étant de fournir aux utilisateurs une bonne expérience, et ce dans le maximum de situations possibles il existe quelques techniques simples pour limiter l'impact des problèmes de connexions, que l'on pourrait résumer en un seul fait, être économe.

1 - Poids des requêtes

Moins on passe de temps à charger une donnée, moins on a de risque de coupure, et plus les données s'affichent à l'écran rapidement. Puisque l'on ne contrôle pas la vitesse de la connexion, il ne reste plus qu'une seule chose à faire : limiter la taille de la requête.



1.1 - JSON / XML

XML est un format bien adapté aux documents et JSON aux données.

JSON a de plus l'avantage d'être plus concis permettant un gain appréciable de place sur le réseau. C'est maintenant un standard de fait en mobile. 1

Pour les plus joueurs d'entre nous Protobuf est extrêmement performant :-)

1.2 Ne retourner que l'indispensable

Si une application n'a besoin d'afficher que les noms et prénoms d'un utilisateur, il ne sert à rien de renvoyer leurs adresses et numéros de téléphone. Tout ce qui n'est pas utile à l'application ne devrait jamais lui être envoyé.

De la même manière, les champs dont les valeurs sont nulles ou les valeurs par défaut ne doivent pas être remontées.

1.3 - Compression

Les formats JSON et XML étant des formats texte, ils se compressent très bien. Il ne faut donc pas hésiter à activer la compression gzip sur les retours d'API destinés à l'application. 2

1.4 - Nommage des propriétés

Il est également possible de simplifier le nom de vos propriétés. Si vous retournez une liste de nombreux éléments, le nom des propriétés est répété. Ainsi le réseau transporte énormément de choses qui ne sont pas utiles pour l'utilisateur. L'utilisateur ne lira pas le JSON directement, alors il ne faut pas hésiter non plus. 3

1.5 - Efficacité des chargements

Une bonne pratique lorsque l'on fait du développement mobile est de s'imposer qu'une seule requête réseau maximum doit

être nécessaire à l'affichage d'un écran. Si deux appels sont nécessaires, soit il faut une nouvelle API qui fusionne les deux, soit il faut deux écrans.
En procédant ainsi on accélère le chargement des écrans, mais surtout, on simplifie grandement la gestion des erreurs de chargements partiels de certains écrans quand une des multiples requêtes échoue.

1.6 - Diagnostics et tests réseau

Des outils tels que CharlesProxy ou Fiddler permettent de simuler de nombreuses situations réseau instables, avec des vitesses variables et en substituant les retours serveur par d'autres. Ils sont les partenaires indispensables des développeurs souhaitant rendre leurs applications très robustes.

2 - Back-end

2.1 - Localisation des APIs

Afin d'améliorer la performance d'une application, une solution consiste à positionner les serveurs hébergeant les APIs au plus près des utilisateurs. Plus les APIs sont proches, plus les temps de réponse seront rapides. Il est possible de faire cela soi-même, mais lorsque le nombre de serveurs se multiplie, la gestion de la répartition de charge devient assez complexe. A ce moment-là, c'est un signe qu'une solution Cloud telle que Microsoft Azure est probablement intéressante à étudier.

2.2 - SSL

Les plateformes mobiles sont de plus en plus strictes et nécessitent par défaut l'utilisation de SSL. SSL n'est pas une protection en soit, mais devient le strict minimum acceptable aujourd'hui.
Attention cependant iOS et Android ne réagissent pas de la même manière, surtout en ce qui concerne les certificats auto-signés. Un conseil, éviter autant que possible ce genre de certificats et optez pour des certificats signés par des autorités de certification reconnues.

2.3 - OAuth/OpenID

Tout comme JSON est le standard de fait du monde mobile, OAuth et OpenID sont les standards de fait de l'authentification et de l'autorisation dans le monde mobile. Chaque plateforme fournit maintenant des mécanismes intégrés pour gérer l'authentification. Ce sont les Chrome Custom Tabs sur Android et ASWebAuthentication Session sur iOS. Ils offrent l'avantage d'être des composants sécurisés tirant parti des gestionnaires de mots de passe présents sur les téléphones.

```
{
  "id": "5c0d8f2498f676117ed67552",
  "index": 0,
  "guid": "5b079ee9-add7-4729-964e-ff75e5dc2ee6",
  "isActive": true,
  "balance": "$1,678.34",
  "picture": "http://placehold.it/32x32",
  "age": 35,
  "name": {
    "first": "Ollie",
    "last": "Baxter"
  },
  "company": "SUPPORTAL",
  "email": "ollie.baxter@supportal.biz",
  "phone": "+1 (871) 506-3330",
  "address": "636 Sharon Street, Cowiche, Montana, 6560",
  "latitude": "-80.8394",
  "longitude": "-99.955458"
},
```

```
{
  "id": "5c0d8f2498f676117ed67552",
  "ind": 0,
  "guid": "5b079ee9-add7-4729-964e-ff75e5dc2ee6",
  "ia": true,
  "b": "$1,678.34",
  "pi": "http://placehold.it/32x32",
  "a": 35,
  "n": {
    "f": "Ollie",
    "l": "Baxter"
  },
  "c": "SUPPORTAL",
  "e": "ollie.baxter@supportal.biz",
  "p": "+1 (871) 506-3330",
  "ad": "636 Sharon Street, Cowiche, Montana, 6560",
  "la": "-80.8394",
  "lo": "-99.955458"
},
```

3

2.4 - Logique métier

L'application mobile devrait, dans la mesure du possible, embarquer le moins de logique possible et en déléguer le plus possible à son back-end. Se faisant, toute modification de la logique pourrait se faire beaucoup plus rapidement puisqu'on s'affranchit des contraintes des magasins d'applications.

3 - Front-end

3.1 - Création des formulaires

Plutôt que d'avoir de grandes pages de formulaires complexes et difficiles à saisir en situation de mobilité, il vaut mieux privilégier une succession d'écrans de petits formulaires.
Ainsi un formulaire d'inscription classique, nécessitant le nom, le prénom, la date de naissance, l'email, le numéro de téléphone, un mot de passe ne sera potentiellement pas fait en un seul écran mais en quatre :

- Nom et prénom ;
- Date de naissance ;
- Email et confirmation email ;
- Mot de passe et confirmation de mot de passe.

3.2 - Libérer le thread principal

Il est de très mauvais goût que d'effectuer de longs traitements sur le thread principal sous peine de voir l'application ramer et ne pas répondre correctement aux interactions de l'utilisateur. Le thread principal est celui qu'utilise le système pour mettre à jour l'interface graphique ou encore réagir aux évènements de l'utilisateur.
Pour avoir une interface fluide il faut 60 frames par seconde ce qui donne une mise à jour de l'interface toutes les 16,67 ms environ. Cette dernière prenant elle même un peu de temps il ne reste que peu de disponibilité pour effectuer des traitements. Il est par ailleurs possible de mesurer cette fluidité sur iOS avec Instruments. Android quant à lui affiche des messages dans la console de sortie lorsque l'application rame.
L'idée est donc d'avoir recours dès que possible à l'asynchronisme et au parallélisme.

3.3 - Asynchronisme et parallélisme

Asynchronisme et parallélisme constituent les deux faces d'une même pièce. Si on simplifie légèrement on peut les décrire ainsi :

- L'asynchronisme est le fait de lancer une opération et d'en obtenir le résultat plus tard sans bloquer le contexte courant.
- Le parallélisme est le fait de lancer plusieurs opérations en même temps en utilisant les capacités des multiples cœurs des processeurs actuels.

Les langages de programmation mobiles modernes tels que C#, Swift, Java, Kotlin, Dart ou Javascript proposent tous le support natif de la programmation asynchrone. Il en va de même pour le parallélisme à l'exception de Javascript qui n'utilise pas de multiples threads mais plutôt des web-workers.
L'exemple typique est l'appel aux APIs dont voici quelques cas d'usage en C# :

- Mauvais asynchronisme [Figure 4] ;
- Bon asynchronisme [Figure 5] ;
- Mauvais parallélisme [Figure 6] ;
- Bon parallélisme [Figure 7] ;

```4
public static List<Person> GetPersons()
{
    using (var client = new HttpClient())
    {
        var json = client.GetStringAsync(MyApi).Result;
        return JsonConvert.DeserializeObject<List<Person>>(json);
    }
}
```

```5
public static async Task<List<Person>> GetPersonsAsync()
{
    using (var client = new HttpClient())
    {
        var json = await client.GetStringAsync(MyApi);
        return JsonConvert.DeserializeObject<List<Person>>(json);
    }
}
```

```6
public static async Task InitializeAsync()
{
    var personDetail1 = await GetPersonDetailAsync(1);
    var personDetail2 = await GetPersonDetailAsync(2);
}
```

```7
public static async Task InitializeParallelAsync()
{
    var details = await Task.WhenAll(new[] {
        GetPersonDetailAsync(1),
        GetPersonDetailAsync(2)
    });
}
```

3.4 - Informer l'utilisateur

Le plus important dans une application mobile n'est pas la performance pure mais plutôt la performance ressentie par les utilisateurs. Ainsi, il faut toujours informer l'utilisateur lorsqu'un traitement est en cours par un loader ou bien une petite animation. C'est aussi un bon moyen pour l'utilisateur de savoir que l'application est toujours réactive et n'est pas plantée.

3.5 - Hors-ligne

La présence d'un mode hors-ligne est bien souvent requis dans une application. Il ne faut cependant pas confondre données en mode hors-ligne et synchronisation bidirectionnelle des données.

Bien souvent au lieu d'une synchronisation, on parle de descente ou de remontée de données différentes. Par exemple on met à jour un référentiel de produits, mais on n'envoie au serveur que des commandes.

S'il y a un vrai besoin de synchronisation bidirectionnelle avec gestion des conflits entre client mobile et serveur, il peut être intéressant de s'orienter vers des solutions commerciales spécialisées.

3.6 - Choix du stockage

Le premier réflexe des développeurs consiste à utiliser une base de données Sqlite. Pourtant Sqlite c'est comme les antibiotiques, ce n'est pas automatique.

Sqlite est une base de données qui pose les mêmes soucis que toutes les bases en ce qui concerne les mises à jour de schéma de données. Ainsi à chaque mise à jour du modèle de donnée de l'application, il faut mettre à jour le schéma de la base et s'assurer que tout se passe bien. Sqlite est particulièrement adapté aux données relationnelles ou aux cas où il y a beaucoup d'opérations de recherche ou d'écriture d'éléments précis au milieu d'un ensemble et où le schéma des données évolue peu.

Dans le cas de données non relationnelles, il est judicieux de se poser la question de la pertinence de Sqlite et d'envisager de juste stocker les données au format Json dans le système de fichier.

Le stockage des données dépend essentiellement du type de la donnée, du type d'opérations à effectuer et de leurs fréquence respective (lecture, écriture, recherche).

Voici quelques exemples de situations et le stockage le plus probablement adapté :

- Pour des articles de presse à consulter en mode hors-ligne il est possible de juste stocker le Json de retour de l'appel réseau sur le disque et de le lire quand une connexion est absente.
- Les paramètres des utilisateurs devraient plutôt être stockés dans les systèmes de paramètres de chaque plateforme plutôt que dans une table Sqlite.
- Pour des listes d'images, on les stockera dans des fichiers et non dans des colonnes d'une base de données. On peut tirer parti de bibliothèques comme Glide, Picasso ou FFImageLoading pour la gestion du stockage des images.
- On stockera aussi probablement un fichier de configuration sur le système de fichier.
- Pour une liste de films dans laquelle on va devoir faire des recherches par nom, date ou autres critères, Sqlite est le plus adapté.
- Pour un panier simple, dans lequel on ajoute des éléments, un simple fichier plat peut faire l'affaire.
- A contrario, s'il n'existe pas de relation dans la donnée, il est peut-être plus judicieux d'envisager un stockage sur disque.

Bien entendu il faut également rester cohérent dans les choix et ne pas multiplier à outrance les systèmes de stockage. Autrement dit, si Sqlite a commencé à être utilisé, autant en profiter.

4 - Sécurité

4.1 – Jetons d'authentification

Les jetons d'authentification ne doivent jamais être stockés en clair dans les paramètres des utilisateurs. Ils doivent être chiffrés et écrits dans les stockages prévus à cet effet :

- KeyChain sur iOS ;
- KeyStore + Cipher + Preferences sur Android ;
- Pour les développeurs Xamarin il est possible d'utiliser le package Xamarin. Essentials.

4.2 - Chiffrement

Chaque système d'exploitation fournit des capacités de chiffrements. La règle d'or est que toutes données permettant d'identifier un utilisateur (noms, prénoms et numéros de téléphone…) et devant être stockées sur le téléphone, doivent être chiffrées.

4.3 - Composants tiers

Il est dommage d'essuyer une attaque à cause d'une faille d'un composant tiers. Cela ruine la confiance des utilisateurs alors qu'elle est si difficile à obtenir.

Ainsi, il faut toujours vérifier le code des projets Open Source intégrés à vos applications ou au moins s'assurer qu'ils l'ont été par des personnes en charge de la sécurité.

Pour les composants commerciaux dont le code source n'est pas disponible, il est intéressant de s'assurer auprès du fabriquant de ses engagements et de décider s'il est digne de confiance ou pas.

4.4 - Man in the middle

En utilisant le point 1.6, on effectue en fait une attaque de type « man in the middle ». D'autres peuvent donc le faire également. Mettre en place un « certificate pinning » dans vos applications lors de la mise en production permet d'éviter cela.

4.5- Légal

Pensez assez tôt à intégrer les conditions générales d'utilisation et la politique de protection des données. Il arrive de rater d'importants jalons de livraison pour ces oublis.

5 - DevOps

L'adoption des pratiques de développement DevOps sont indispensables au succès d'un projet. On note deux grandes phases dans un projet mobile : le développement et la production.

5.1 - Développement

Avant même de coder le premier écran de l'application, il est important de mettre en place une usine d'intégrations et de déploiements continus tels qu'Azure DevOps et AppCenter. L'idée derrière tout cela est de :

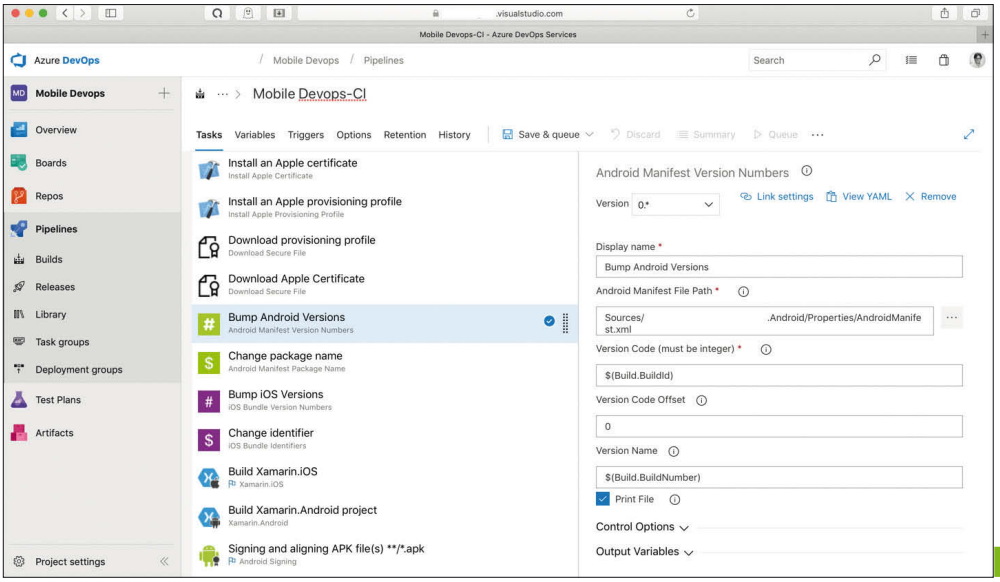
- Evaluer la qualité et la couverture du code automatiquement avec SonarQube par exemple ;
- Evaluer la sécurité du code avec CheckMarx par exemple ;
- Détecter des régressions très en amont ;
- Compiler automatiquement iOS et Android ;
- Modifier les configurations des applications ;
- Déployer en mode bêta sans passer par les stores grâce à AppCenter.

J'utilise Azure DevOps et AppCenter tous les jours depuis plusieurs années sur des projets mobiles et c'est vraiment une source d'amélioration certaine de productivité et de fiabilité des applications. 8

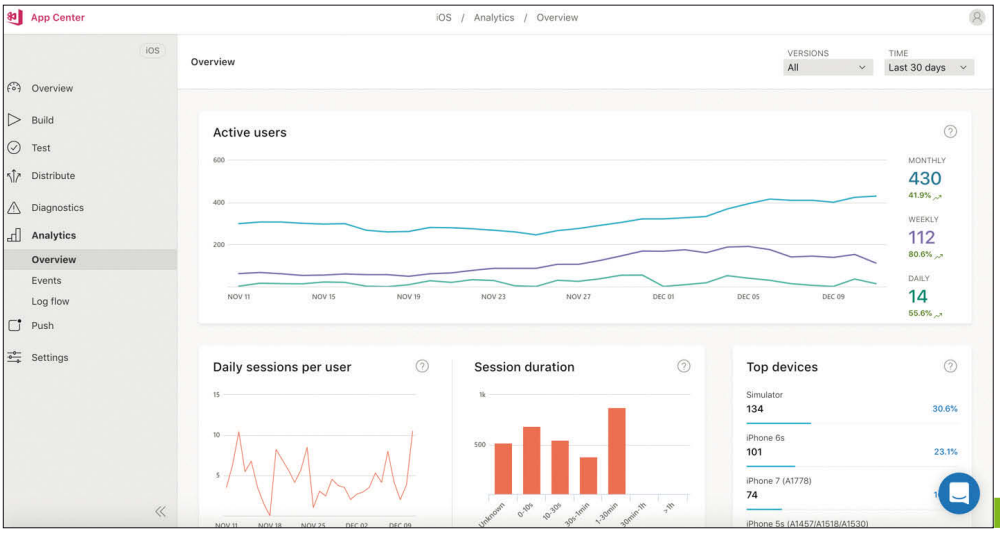
5.2 - Production

Une fois l'application déployée sur les téléphones, il faut pouvoir savoir si tout se passe bien, s'il n'y a pas de crashs et si les utilisateurs se servent de l'application comme on l'espérait.

Il existe divers outils pour cela dont AppCenter qui répond à ce besoin de manière très poussée et ce, quelle que soit la technologie utilisée. Grâce à ses briques d'analytics et de crashs reporting, il est très facile d'avoir de la matière pour corriger et améliorer l'application.



8



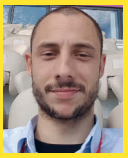
9

Il permet également de lancer des tests UI de l'application fonctionnant sur de vrais téléphones et tablettes. C'est un allié précieux pour aller diagnostiquer le bug qui ne se produit que sur un téléphone bien particulier qu'on ne peut ou qu'on ne souhaite pas se procurer exprès.

C'est également lui qui gérera les groupes de distribution bêta de l'application précédemment évoquée. 9

Conclusion

Développer une bonne application mobile est quelque chose de complexe. Les recettes du succès ne sont pas toujours bien identifiées, en revanche, les erreurs menant à l'échec oui. Bien qu'il existe des écueils spécifiques à chaque plateforme et chaque technologie, j'ai voulu, au travers de cet article, revenir sur les points auxquels nous devons tous faire attention afin de proposer aux utilisateurs la meilleure expérience possible. •



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

Amazon Corretto : une distribution gratuite du JDK avec support long terme

La sortie de Java 11 aura marqué un changement de licence majeur pour le JDK distribué par Oracle. Ce changement rend le Java d'Oracle payant pour des usages en production sans que cela ne soit apparu de manière claire au premier abord. Fort heureusement, les développeurs pourront continuer à bénéficier gratuitement de la distribution OpenJDK d'Oracle. Néanmoins, des doutes sur le futur gratuit de Java ont pu naître chez certains développeurs. C'est dans ce contexte qu'Amazon a choisi de dévoiler Corretto sa distribution gratuite d'OpenJDK sur laquelle nous allons nous attarder dans cet article.

Sorti en Septembre 2018, Java 11 aura fait naître de sérieuses craintes sur la gratuité de Java dans le futur pour des usages en production. Fort heureusement, la distribution OpenJDK continue d'être maintenue par Oracle. Dans ce contexte, Amazon a décidé de présenter sa distribution d'OpenJDK nommée Corretto qui est d'ores et déjà utilisée en interne par le géant du commerce en ligne. Il est bon également de préciser que Corretto est une implémentation certifiée compatible du standard Java SE.

Support long terme pour Corretto

Pour paraphraser l'annonce faite par Amazon, Corretto est donc une distribution gratuite d'OpenJDK, multiplateforme et prête pour un usage en production dès à présent. Amazon propose un support long terme pour Corretto incluant des améliorations liées aux performances mais également des patchs de sécurité. Dans la FAQ accompagnant Corretto, Amazon précise que la version actuellement disponible sera supportée au moins jusqu'à Juin 2023. Tout ceci est un élément essentiel puisqu'Amazon proposera un support de 4 à 5 ans minimum là où Oracle se contentera désormais d'un support gratuit de 6 mois au niveau de performances et des patchs de sécurité au sein de l'OpenJDK.

Corretto 11 pour 2019

Actuellement, la version proposée est Corretto 8. Elle est naturellement à rapprocher de l'OpenJDK 8 tout en précisant qu'Amazon a réalisé un travail de backports des versions 9, 10 et 11 vers Corretto 8 pour tout ce qui est patchs de sécurité ou améliorations liées aux performances du JDK. Amazon annonce une sortie de Corretto 11 sur la première partie d'année 2019 avec un support long terme via des mises à jour trimestrielles allant au moins jusqu'à Août 2024 !

Installation sous macOS

Corretto est multiplateforme et supporte les plateformes Amazon Linux 2, Windows à partir de la version 7 ou encore macOS à partir de la version 10.10. Dans ce qui suit, nous allons donc tenter d'installer Corretto 8 sur macOS. La première étape consiste à récupérer le binaire de Corretto 8 à l'adresse suivante :

https://d3pxv6yz143wms.cloudfront.net/corretto-jdk-8u192-macosx-x64.pkg

Il faut ensuite double-cliquer sur le fichier packaging récupéré afin de lancer le wizard d'installation. Une fois les différentes étapes du wizard complétées, Corretto 8 est installé au sein du répertoire /Library/Java/JavaVirtualMachines/. Afin d'obtenir le chemin d'installation complet, il est également possible de lancer la commande suivante dans un terminal :

```
/usr/libexec/java_home --verbose
```

Si vous souhaitez faire d'Amazon Corretto 8 votre version de Java par défaut sur votre appareil tournant sous macOS, vous pouvez ensuite configurer la variable JAVA_HOME de la sorte au sein d'un terminal :

```
export JAVA_HOME=/Library/Java/JavaVirtualMachines/amazon-corretto-8.jdk/Contents/Home
```

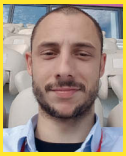
Ces opérations réalisées, vous pouvez utiliser Corretto 8 et commencer à bénéficier de la toute nouvelle distribution OpenJDK proposée par Amazon. Pour terminer, il est bon de préciser que l'ensemble des sources de Corretto 8 est mis à disposition sur un dépôt GitHub accessible à l'adresse suivante :

https://github.com/corretto/corretto-8

Au sein du dépôt parent Corretto, le développeur Java pourra également trouver une image Docker directement prête à l'emploi. Tout ceci étant la preuve de la volonté d'Amazon de rendre accessible sa distribution d'OpenJDK au plus grande nombre.

Conclusion

Les semaines qui viennent de s'écouler depuis la sortie de Java 11 ont été faites d'incompréhensions concernant la gratuité de Java dans le futur. Amazon Corretto est un exemple supplémentaire de distribution gratuite du JDK fournissant de surcroît un support long terme. En outre, la licence choisie par Amazon pour Corretto est la GNU Public License version 2 with Class Path Exception (GPLv2 with CPE), ce qui garantit aux développeurs qu'Amazon ne pourra pas faire payer les utilisateurs pour l'usage et la distribution de Corretto dans le futur. Voilà de quoi rassurer les développeurs Java qui ont désormais entre leurs mains une nouvelle alternative de qualité en matière de JDK. •



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

ZGC : un Garbage Collector scalable à faible latence pour Java 11

niveau 200

Conformément à la nouvelle cadence de mise à jour du JDK annoncée par Oracle en 2017, Java 11 est sorti en Septembre 2018, tout juste 6 mois après la sortie de Java 10. Cette nouvelle politique de mise à jour plus rapprochée implique que les nouveautés intégrées au JDK sont moins spectaculaires que ce qui pouvait se faire autrefois lors de la sortie d'une nouvelle version majeure. Néanmoins, certaines valent le détour et dans cet article nous allons nous intéresser plus particulièrement à ZGC, le nouveau Garbage Collector de la JVM.

Le nouveau cycle de mise à jour du JDK décidé par Oracle nous aura permis d'avoir à disposition 3 nouveaux JDK en tout juste 1 an. C'est finalement assez incroyable lorsqu'on se rappelle qu'il avait fallu attendre près de 5 ans entre Java 6 et Java 7, puis 3 ans entre chaque version suivante du JDK jusqu'à Java 9. Pour mettre fin à la sensation que la plateforme Java était en perte de vitesse et n'évoluait plus, Oracle a donc pris la décision qui s'imposait. Celle-ci implique naturellement des fonctionnalités nouvelles, un peu moins ambitieuses que ce qu'on connaissait auparavant.

En creusant un petit peu dans la liste des nouveautés de Java 11, on découvre tout de même la présence du Z Garbage Collector. Plus connu sous son acronyme de ZGC, ce nouveau Garbage Collector n'est pour l'instant disponible qu'à titre expérimental et supporté uniquement sur les plateformes Linux 64 bits. On comprend ainsi aisément qu'il est destiné au monde des serveurs. Le fait qu'il soit en phase expérimentale n'empêche cependant pas qu'il soit très prometteur et qu'il semble nécessaire de s'y pencher plus en détails.

Pourquoi un nouveau Garbage Collector ?

La première question qu'il convient de se poser est : pourquoi un nouveau Garbage Collector a-t-il été introduit par Oracle avec Java 11 ? Après tout, c'est une question sensée puisqu'avec Java 10, le JDK propose déjà pas moins de 4 Garbage Collectors différents. De plus, ces Garbage Collectors sont tous configurables à l'extrême, ce qui permet de les adapter à un grand nombre de besoins spécifiques. Le dernier né était, jusqu'alors, G1 qui fut ajouté au JDK en 2006 avec Java 6.

Pour bien comprendre ce qu'était le monde des serveurs à cette époque, il suffit de se rappeler que la plus grosse instance proposée par AWS (Amazon Web Services) comprenait alors 1 CPU virtuel avec 1.7GB de RAM. Aujourd'hui, AWS propose une instance avec 128 CPUs virtuels pour un total assez phénoménal de 3904GB de RAM !

On comprend ainsi aisément la nécessité de concevoir un Garbage Collector adapté à ces nouveaux besoins pour coller au mieux à l'évolution des serveurs. C'est en partant de ce constat qu'est née la volonté de créer ZGC afin de proposer un Garbage Collector répondant aux objectifs suivants :

- des temps de pause ne dépassant pas 10ms ;
- des temps de pause n'augmentant pas avec la taille du tas ou l'ensemble des objets vivants ;
- la Gestion d'un tas pouvant aller de quelques centaines de Mégabytes à plusieurs Térabytes !

En outre, le potentiel de ZGC est assez phénoménal puisque son utilisation pourrait être étendue dans le futur à des tas répartis en multi-tiers, c'est-à-dire avec envoi sur la DRAM des objets chauds, et envoi vers de la mémoire NVMe flash pour des objets accédés moins fréquemment. Dans ce qui suit, nous allons nous intéresser de plus près aux mécanismes qui permettent à ZGC de tenir ses promesses.

Phases d'un Garbage Collector

Afin de pouvoir profiter au mieux des explications sur le fonctionnement de ZGC, il paraît important de commencer par rappeler quelques détails sur les mécanismes de fonctionnement d'un Garbage Collector. Rappelons qu'un Garbage Collector a pour fonction basique d'identifier de la mémoire qui n'est plus utilisée, et de la rendre accessible pour faciliter sa réutilisation. Les Garbage Collectors modernes vont réaliser ce processus via différentes phases que l'on peut décrire de la façon suivante :

- **Parallèle** - Lors de l'exécution de la JVM, plusieurs threads d'application de même que plusieurs threads liés au Garbage Collector sont créés. La phase Parallèle est menée à bien par ces différents threads de garbage collection qui séparent leur travail afin de mieux l'optimiser. Néanmoins, on ne sait rien sur la manière dont le travail des threads de garbage collection pourrait venir s'entrechoquer avec les threads d'application.
- **Série** - La phase Série est réalisée au sein d'un thread de garbage collection unique. Comme pour la phase Parallèle, on ne sait rien sur la manière dont ce thread pourrait venir s'entrechoquer avec les threads d'application en cours d'exécution.
- **Arrêt du monde** - Connu en Anglais sous le nom de "Stop the World", cette phase suspend les threads d'application pour permettre au Garbage Collector de mener son travail à bien. Si vous avez déjà rencontré des pauses du Garbage Collector, elles sont dues à cette phase d'Arrêt du Monde.

```
// Address Space & Pointer Layout
// ------------------------------
//
//  +--------------------------------+ 0x00007FFFFFFFFFFF (127TB)
//  .                                .
//  .                                .
//  .                                .
//  +--------------------------------+ 0x0000140000000000 (20TB)
//  |         Remapped View          |
//  +--------------------------------+ 0x0000100000000000 (16TB)
//  |     (Reserved, but unused)     |
//  +--------------------------------+ 0x00000c0000000000 (12TB)
//  |         Marked1 View           |
//  +--------------------------------+ 0x0000080000000000 (8TB)
//  |         Marked0 View           |
//  +--------------------------------+ 0x0000040000000000 (4TB)
//  .                                .
//  +--------------------------------+ 0x0000000000000000
//
//
//   6                 4 4 4  4 4                                            0
//   3                 7 6 5  2 1                                            0
//  +-------------------+-+----+-------------------------------------------------+
//  |00000000 00000000 0|0|1111|11 11111111 11111111 11111111 11111111 11111111|
//  +-------------------+-+----+-------------------------------------------------+
//  |                   | |    |
//  |                   | |    * 41-0 Object Offset (42-bits, 4TB address space)
//  |                   | |
//  |                   | * 45-42 Metadata Bits (4-bits)  0001 = Marked0      (Address view 4-8TB)
//  |                   |                                 0010 = Marked1      (Address view 8-12TB)
//  |                   |                                 0100 = Remapped     (Address view 16-20TB)
//  |                   |                                 1000 = Finalizable  (Address view N/A)
//  |                   |
//  |                   * 46-46 Unused (1-bit, always zero)
//  |
//  * 63-47 Fixed (17-bits, always zero)
//
```

1 *Multi-mapping dans ZGC*

- **Concurrente** - Lors d'une phase Concurrente, le Garbage Collector réalise son travail en même temps que les threads d'application effectuent le leur. Ces phases sont complexes parce qu'elles doivent être capables de prendre en compte des threads d'application pouvant potentiellement invalider leur travail avant la fin complète de la phase.
- **Incrémentale** - Lors d'une phase incrémentale, le Garbage Collector se lance pour une certaine période, et peut se terminer prématurément sous certaines conditions telles qu'un dépassement du temps de travail prévu, ou le besoin de lancer une phase du Garbage Collector de priorité plus haute. Cette phase doit s'exécuter complètement pour pouvoir être productive.

Il est évident que ces phases induisent toutes certaines contraintes. Ainsi, une phase Parallèle utilisera de multiples threads de garbage collection pour réaliser son travail, ce qui peut provoquer un overhead dans la coordination des threads. De même, une phase Concurrente ne mettra pas en pause les threads d'application, mais pourrait mener à une augmentation significative des overheads et de la complexité pour gérer les threads d'application qui viendraient à invalider son travail de manière concurrente.

ZGC introduit de nouvelles techniques

Maintenant que vous avez découvert les différents types de phases d'un Garbage Collector sur la JVM, nous pouvons explorer ZGC et son fonctionnement en détails. Pour atteindre les objectifs fixés par ses créateurs, ZGC utilise 2 nouvelles techniques qui n'étaient pas utilisées jusqu'à présent par les différents Garbage Collectors de Hotspot, la machine virtuelle Java : le pointer colouring et les load barriers.

Pointer colouring

Le pointer colouring est une technique consistant à stocker l'information au sein même des pointeurs. Appliqués au monde Java, ces pointeurs deviennent des références. Tout ceci est rendu possible par le fait que sur les plateformes 64 bits, un pointeur peut adresser largement plus de mémoire qu'un système ne peut en avoir de manière réaliste. De fait, il devient possible d'utiliser les bits restants sur le pointeur pour stocker un état. Ceci permet également de comprendre pourquoi ZGC est pour le moment supporté uniquement sur les systèmes 64 bits.

ZGC se restreint lui-même à 4 Térabytes de tas, ce qui nécessite 42 bits de représentation. Il reste donc 22 bits possibles pour stocker un état alors que ZGC n'en utilise actuellement que 4 pour les états suivants : finalizable, remap, mark0, mark1. Cette technique de pointer colouring n'est pas sans contraintes, cependant puisqu'elle va nécessiter un travail supplémentaire au moment du déréférencement d'un pointeur. En effet, il faudra alors remettre à zéro les informations contenues dans ces bits. Sur certaines plateformes, telles que SPARC, cela n'est pas un problème car le système a déjà un mécanisme interne au niveau du hardware visant à vider les informations d'un pointeur. En revanche, sur des plateformes x86, cela constitue un problème que l'équipe en charge de l'implémentation de ZGC a résolu avec un tour de passe-passe : le multi-mapping.

Multi-mapping

Afin de comprendre comment le multi-mapping fonctionne, revenons un instant sur ce qui différencie la mémoire virtuelle de la mémoire physique. La mémoire physique est la RAM actuellement disponible sur le système. Habituellement, elle correspond à la capacité des barrettes de DRAM installées. La mémoire virtuelle est une abstraction impliquant que les applications ont leur propre vue, souvent isolée, au sein de la mémoire physique. Le système d'exploitation est responsable du maintien du mapping entre les zones de mémoire virtuelle et la mémoire physique. Pour ce faire, l'OS utilise une page des tables, le processeur de Memory Management Unit (MMU) et le Translation Lookaside Buffer (TLB), lequel se charge de transposer les adresses demandées par les applications.

La technique de multi-mapping nécessite d'associer différentes zones de mémoire virtuelle à la même partie de la mémoire physique. Par design, seulement un des états remap, mark0 et mark1 peut être positionné à la valeur 1 à un instant donné. Il est donc possible de faire cela avec 3 mappings comme le montre la figure 1, issue du

code source de ZGC disponible au sein d'OpenJDK 11. 1

Load Barriers

L'autre technique nouvelle introduite par ZGC est celle des Load Barriers. Il s'agit de morceaux de code s'exécutant chaque fois qu'un thread d'application charge une référence depuis le tas. Considérons le code suivant :

```
void displayName(Person person) {
  String name = person.name;
  System.out.println(name);
}
```

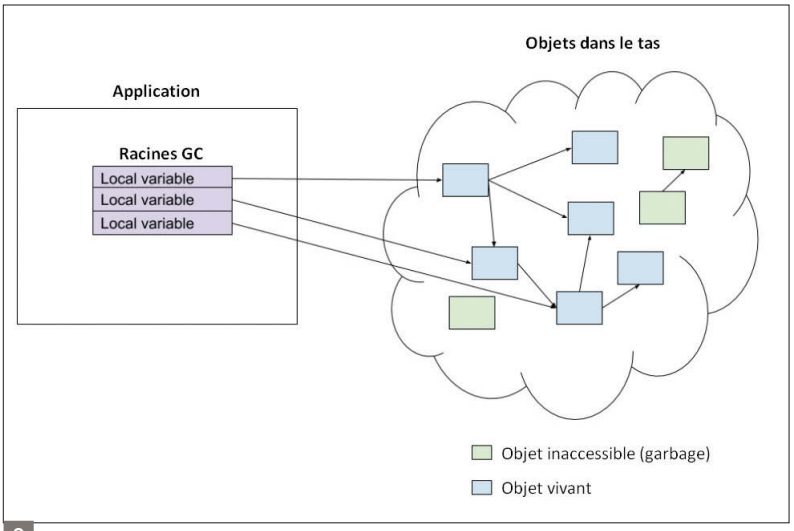
Sur la première ligne, nous assignons une valeur à la variable locale *name* en suivant la référence associée à la donnée d'un objet présent sur le tas, puis en accédant à la référence contenue au sein de *name*. L'exécution de cette ligne déclenche une Load Barrier. Sur la seconde ligne, on affiche le contenu de la variable locale *name* à l'écran. Il n'y a donc pas de déclenchement d'une Load Barrier puisque *name* est une variable locale et qu'il n'y a pas besoin de charger une référence depuis le tas. Il pourrait cependant y avoir le déclenchement d'une Load Barrier au niveau du référencement des objets *System* et *out*, ou bien au sein de la méthode *println*.

Ces Load Barriers contrastent avec les write-barriers utilisées par d'autres Garbage Collectors tels que G1. Le job d'une Load Barrier est d'examiner l'état de la référence et, potentiellement, de réaliser un travail sur la référence (ou peut-être même sur une référence différente) avant de la retourner à l'application.

Au sein de ZGC, cette tâche est réalisée en testant la référence chargée afin de voir si certains bits sont valorisés, tout en tenant compte de la phase en cours d'exécution. Si la référence passe le test, alors aucun travail supplémentaire n'est réalisé. En cas d'échec, des tâches spécifiques à la phase en cours sont réalisées avant que la référence soit ensuite retournée à l'application.

Cycle de ZGC

Maintenant que nous avons vu les nouvelles techniques introduites par ZGC, nous pouvons passer à l'étude d'un cycle de garbage collection de ZGC. La première partie du cycle est le Marking alors que la seconde partie est la Relocation.



2 *Marquage des racines dans ZGC*

Marking

La partie Marking nécessite de trouver et de libeller d'une certaine manière tous les objets du tas qui pourraient être accessibles par l'application en cours d'exécution. En d'autres termes, on cherche à trouver les objets qui ne sont pas à recycler.

Le marquage de ZGC est découpé en 3 phases. La première phase est celle d'Arrêt du Monde évoquée précédemment au cours de laquelle les racines du Garbage Collector sont marquées comme étant en vie. Les racines du Garbage Collector sont des choses telles que les variables locales que l'application peut utiliser pour accéder au reste des objets du tas. Si un objet n'est pas atteignable par un parcours du graphe d'objets démarrant depuis les racines du Garbage Collector, alors il n'y a aucun moyen pour qu'une application puisse y accéder. Cet objet est donc à recycler. L'ensemble des objets accessibles depuis les racines est connu sous le nom d'ensemble des objets vivants. Cette étape de marquage est très courte puisque le nombre total de racines est souvent relativement faible 2.

Une fois le marquage des racines complété, l'application reprend son travail, et ZGC peut commencer la phase suivante, laquelle va parcourir, de manière concurrente, le graphe des objets en marquant tous les objets comme accessibles. Durant cette phase, la Load Barrier teste toutes les références chargées en utilisant un masque déterminant, si elles ont été marquées, ou pas encore, pour cette phase. Si une référence n'a pas été marquée, elle est ajoutée dans une queue pour le marquage. Enfin, lorsque ce parcours est terminé, il y a un bref passage final par une phase d'Arrêt du Monde permettant de gérer certains cas limites, et la phase Marking peut alors se terminer.

Relocation

La grande partie qui suit dans le cycle de garbage collection de ZGC est la relocation. La Relocation va consister à déplacer les objets vivants dans le but de libérer des sections du tas. Pourquoi déplacer des objets plutôt que de simplement compléter les espaces ? Certains Garbage Collectors se contentent de ce travail, mais cela a pour conséquence malheureuse que l'allocation devient beaucoup plus coûteuse ensuite. En effet, lorsqu'une allocation est nécessaire, celui qui alloue doit trouver un espace libre pour placer l'objet. ZGC réalise en quelque sorte un travail similaire à une défragmentation comme celle réalisée sur vos disques durs sous Windows par exemple. Ainsi, si des larges zones de mémoire peuvent être libérées, alors l'allocation revient simplement à incrémenter le pointeur du total de mémoire nécessaire à l'objet.

ZGC divise le tas en pages et, au début de cette phase, il va sélectionner, de manière concurrente, un ensemble de pages dont les objets en vie peuvent être réimplantés. Quand l'ensemble à réimplanter est sélectionné, il y a une pause Arrêt du Monde qui est faite pendant laquelle ZGC réimplante tous les objets faisant partie de l'ensemble qui sont référencés comme étant des

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

- Symfony 3*
  Développer des applications web robustes
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

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

- Symfony 3*
  Développer des applications web robustes
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

* Offre limitée à la France métropolitaine

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

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

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

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

☐ Vidéo : Symfony 3
☐ Vidéo : Raspberry Pi

PROG 225
Offre limitée, valable jusqu'au 1 février 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**

Offres 20e anniversaire !*

1 an (11 numéros)
79,99 €
(au lieu de 147,99 €)

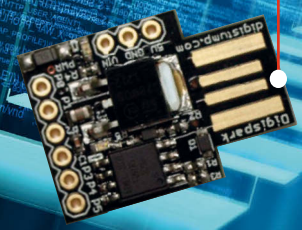
2 ans (22 numéros)
99,99 €
(au lieu de 177,99 €)

+

1 clé USB contenant
tous les numéros depuis le n°100

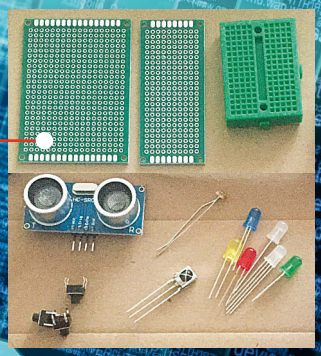
+

1 carte maker Digispark ATtiny84
compatible Arduino



+

1 lot de composants / capteurs
(selon arrivage du jour)

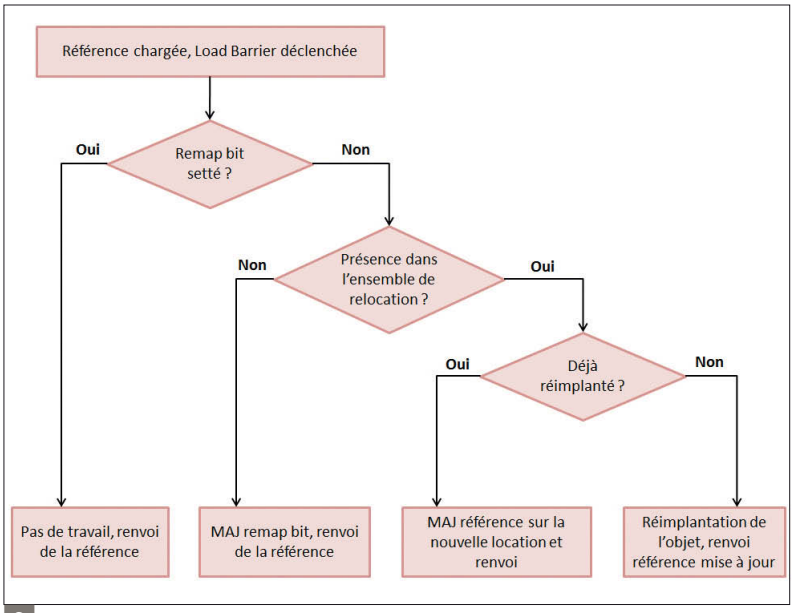


+

1 an de Pharaon Magazine soit 4 numéros :
pour découvrir l'Egypte des Pharaons !

* Offre réservée à la France métropolitaine

Sur notre site web uniquement : https://www.programmez.com/catalog/20eme-anniversaire



3 *Mécanisme de relocation via les Load Barriers*

racines (variables locales, …); il mappe leurs références sur leurs nouveaux emplacements. Comme pour l'Arrêt du Monde précédent, le temps de pause va dépendre ici du nombre d'objets racines, du ratio des tailles des ensembles à réimplanter, et du total de l'ensemble des objets vivants. Ce dernier étant normalement relativement petit car il ne s'adapte pas à la taille globale du tas comme le font de nombreux autres Garbage Collectors.

Après le déplacement des racines qui n'étaient plus nécessaires, la phase suivante est la relocation concurrente. Durant cette phase, les threads du Garbage Collector vont parcourir l'ensemble de relocation, et réimplanter tous les objets dans les pages qu'il contient. Les threads d'application peuvent également réimplanter des objets dans l'ensemble à réimplanter, s'ils essaient de les charger avant que le Garbage Collector ait pu les réimplanter. Ceci est rendu possible via les Load Barriers, lesquelles sont déclenchées lors du chargement des références depuis le tas. Ce mécanisme étant détaillé à la figure 3.

Tout ceci garantit que toutes les références que l'application voit, ont été mises à jour et qu'il n'y a pas de possibilité pour l'application d'agir sur un objet qui est en cours de réimplantation.

Les threads du Garbage Collector ont finalement réimplanté tous les objets présents dans l'ensemble de relocation. Cependant, il pourrait encore y avoir des références pointant sur les anciens emplacements de ces objets. Le Garbage Collector pourrait parcourir le graphe d'objets pour remapper toutes les références vers leurs nouveaux emplacements, mais il s'agirait ici d'une opération très coûteuse. Au lieu de cela, le travail est combiné dans la prochaine phase Marking. Durant le parcours, si une référence est trouvée comme n'étant pas remappée, elle est remappée, et ensuite marquée comme étant en vie. Le cycle d'une garbage collection de ZGC est alors terminé.

Des perspectives prometteuses pour ZGC

Comme expliqué précédemment, les techniques de Pointer Colouring et de Load Barriers introduites par ZGC laissent entrevoir de belles perspectives pour le futur. Avec les mémoires flash et non-volatiles qui sont de plus en plus répandues, une possibilité serait d'autoriser des tas multi-tiers sur la JVM, avec des objets techniquement vivants mais rarement utilisés qui pourraient être stockés sur un tiers mémoire plus lent.

Ceci pourrait être rendu possible en étendant les métadonnées des pointeurs via l'utilisation des bits restants à disposition pour décider où déplacer un objet qui requiert une réimplantation. La Load Barrier pourrait alors retrouver l'objet plus tard depuis le stockage au moment opportun. Alternativement, au lieu de réimplanter les objets dans un tiers mémoire plus lent, les objets pourraient être gardés au sein de la mémoire principale mais sous une forme compressée. Ils n'auraient qu'à être décompressés et réimplantés dans le tas par la Load Barrier quand ils seraient requêtés.

Conclusion

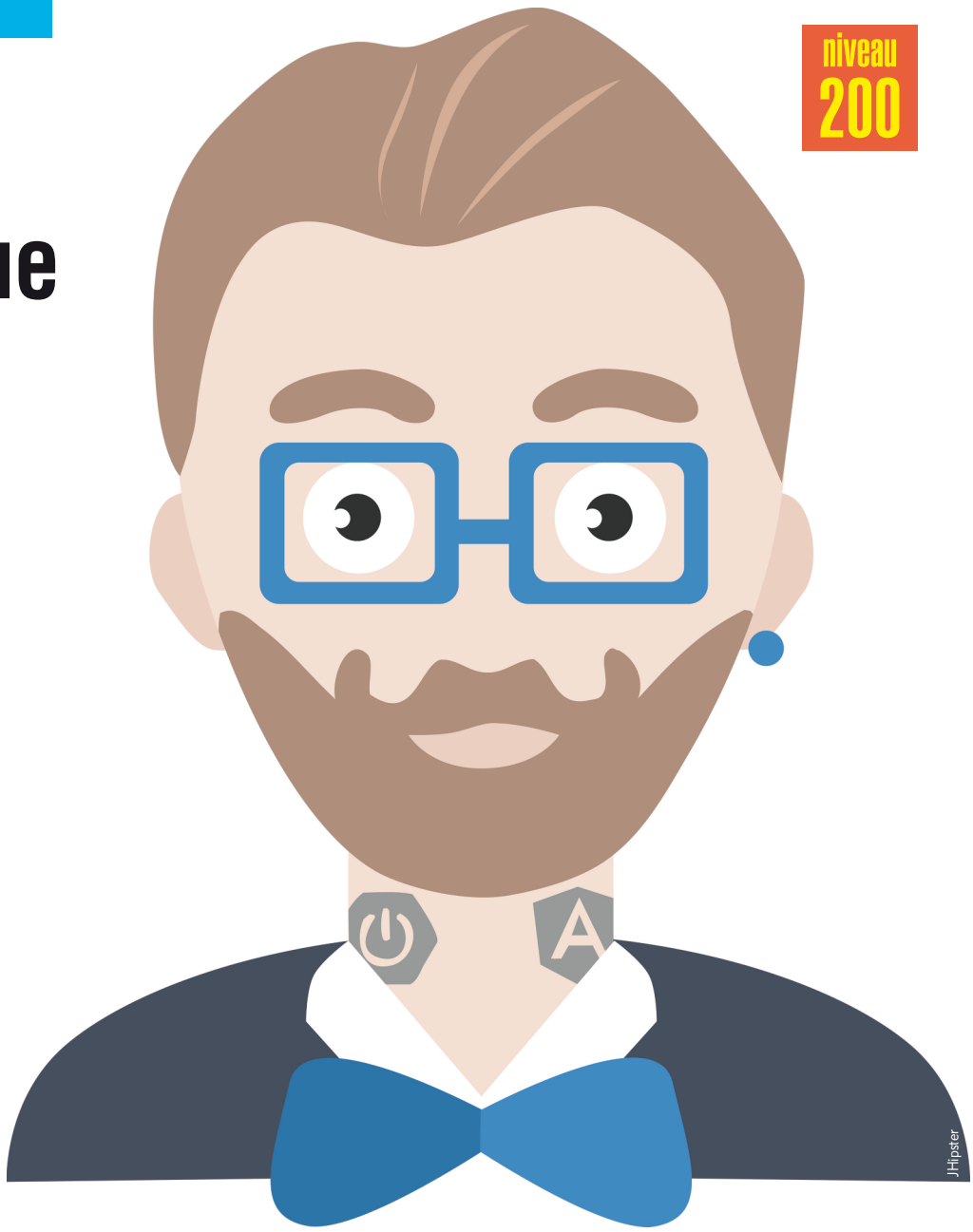
Cet article nous aura permis de détailler le fonctionnement de ZGC. Expliquer de manière isolée en quoi ce fonctionnement améliore drastiquement les performances des applications utilisant énormément de RAM n'est pas chose aisée. Néanmoins, vous avez pu constater que toutes les pauses réalisées par ZGC concernent un travail dépendant de l'ensemble des racines du Garbage Collector plutôt que de l'ensemble des objets vivants, la taille du tas ou encore des objets à recycler. La pause finale réalisée durant la phase de Marking, qui gère le marquage de terminaison, est une exception à cela, mais elle est incrémentale et le Garbage Collector peut annuler le marquage concurrent si le temps alloué est excédé tout en le reportant au passage suivant.

Les tests de performances préliminaires réalisés sur des benchmarks dédiés sont en tous cas très prometteurs avec un ZGC qui réalise un nombre de passages comparable à Parallel GC, qui est pourtant optimisé pour cela, tout en ayant des temps de pause compris entre 1ms et 4ms. A titre de comparaison, G1 et Parallel GC ont des temps de pause dépassant les 200ms sur ces benchmarks ! Il faudra donc compter sur ZGC dans le futur et suivre son évolution de près. •

Julien Dubois
Créateur et le principal développeur de JHipster. Il est également directeur de l'innovation chez Ippon Technologies, société qui emploie de nombreux contributeurs à JHipster.

niveau 200

Construire une boutique en ligne avec JHipster



Qu'est-ce que JHipster ?

JHipster est une plateforme de développement d'applications "full stack", disponible sur https://www.jhipster.tech. Le projet est entièrement Open Source (licence Apache), et son code est hébergé sur GitHub. Une communauté d'environ 500 personnes contribue à son développement, dont 24 font partie de l'équipe principale qui gère le projet. Il est très largement utilisé en entreprise, avec 260 sociétés l'utilisant officiellement, et plus de 1,5 million d'installations. Techniquement, JHipster génère une application full stack à l'état de l'art, ce qui inclut :

- Du code côté serveur développé en Java, et reposant sur Spring Boot ;
- Du code côté client développé en TypeScript, et utilisant soit Angular, soit React (et bientôt Vue.js !) ;
- Un ensemble de configurations permettant de construire, tester et déployer en production l'application finale ainsi générée.

Cette application peut être très largement personnalisée via un système de questions/réponses ou par des modules tiers. Ainsi, on peut par exemple choisir son type de base de données (base de données relationnelle ou NoSQL), et même l'architecture technique voulue (monolithe ou microservice).

Plutôt que d'intégrer une stack technique complète à la main, JHipster propose donc de la générer. Cette approche permet non seulement de gagner un temps précieux au démarrage du projet, mais elle permet également de garantir un haut niveau de qualité dans cette intégration. Ceci est évidemment dû au fait qu'elle a été réalisée par une large communauté et de nombreux experts de chaque domaine.

Quelles utilisations pour JHipster ?

L'utilisation première de JHipster est la génération d'applications avec des formulaires ou des listes de données. À ce titre, on retrouve beaucoup d'applications de gestion, typiquement dans les grandes banques, sociétés d'assurance ou compagnies aériennes. Le code généré étant particulièrement adaptable et performant, JHipster se voit également utilisé pour des sites Web grand public, par exemple dans les services de l'État français ou des sites de recrutement en ligne.

JHipster propose aussi bien de générer des applications dites "monolithiques" simples, avec quelques écrans, que des architectures micro-services complexes. C'est d'ailleurs l'une des grandes forces du projet,

car il peut ainsi se retrouver utilisé par une simple startup d'une personne (un exemple concret : un site de vente en ligne de saucisson) que dans des grandes entreprises avec des équipes de plus de 50 personnes. Les développeurs peuvent ainsi réutiliser leurs compétences et leurs pratiques de développement, quelle que soit la taille du projet.

Fonctionnement du projet

JHipster est un projet entièrement communautaire : nous vous encourageons à participer au projet, que ce soit par du code, de la documentation ou des questions. Il n'y a pas de version "entreprise" ou de modules payants : tout ce que fait l'équipe du projet est public et libre d'accès.

Le projet possède actuellement une équipe d'environ 24 personnes qui constitue sa "core team" : ces personnes ont les droits en écriture sur le projet et les sous-projets associés, et font généralement les revues de code. Si vous souhaitez contribuer au projet, vous les rencontrerez certainement ! Cette équipe est élue via un processus proche de celui de la fondation Apache. À la tête du projet se trouvent les deux principaux contributeurs du projet : Julien Dubois (auteur de cet article) et Deepu K Sasidharan.

JHipster est un projet communautaire, avec des membres divers qui viennent des quatre coins du monde. À ce titre, le projet a mis très tôt en place un code de conduite, disponible sur https://github.com/jhipster/generator-jhipster/blob/master/CODE_OF_CONDUCT.md. Il garantit le bon fonctionnement du projet et le respect mutuel de ses membres. Nous vous encourageons donc à le lire !

Création d'une première application avec JHipster Online

JHipster est un module NPM, qui permet de créer une application full stack. À ce titre, pour correctement l'utiliser, vous devez avoir installé sur votre poste :

- Une version de Java 8 (ou Java 11 si vous avez une version de JHipster utilisant Spring Boot 2.1 ou supérieur)
- Une version "long term support" (LTS) de Node.js
- Git
- Docker

Ces outils étant relativement classiques, nous ne détaillons pas leur installation, mais si vous avez des problèmes particuliers, par exemple si vous utilisez un serveur proxy, nous vous conseillons d'aller sur la documentation officielle de JHipster. Vous pouvez également utiliser la machine virtuelle "clef en main" fournie par le projet, ce qui vous fera gagner beaucoup de temps car elle est entièrement configurée avec ces outils : https://github.com/jhipster/jhipster-devbox.

Vous avez ensuite deux solutions pour créer votre première application JHipster :

- Installer JHipster sur votre poste de travail en lançant la commande `npm install -g generator-jhipster` puis en exécutant JHipster avec la commande `jhipster`.
- Utiliser JHipster Online, qui est une application Web disponible sur https://start.jhipster.tech et qui propose une version légèrement simplifiée de JHipster, et qui s'exécute dans le cloud.

Ces deux manières de faire sont relativement proches, et pour cet article vous pouvez utiliser l'une ou l'autre de manière indifférente. Nous allons nous concentrer sur JHipster Online car il est plus simple d'accès, et donc mieux adapté à un tutoriel, mais cela ne changera rien à l'application générée au final.

La première chose à faire est donc de créer un compte sur JHipster Online : allez sur https://start.jhipster.tech et inscrivez-vous via le formulaire. Un e-mail de confirmation vous sera envoyé pour valider votre adresse e-mail et éviter les abus sur le système.

Une fois authentifié sur le système, allez dans l'onglet "Configure" et reliez JHipster Online à votre compte GitHub ou GitLab : cela permettra à JHipster Online de créer des Pull Requests sur vos projets. Dans cet article, nous utiliserons GitHub car il est plus répandu, mais tout fonctionne de manière identique sur GitLab.

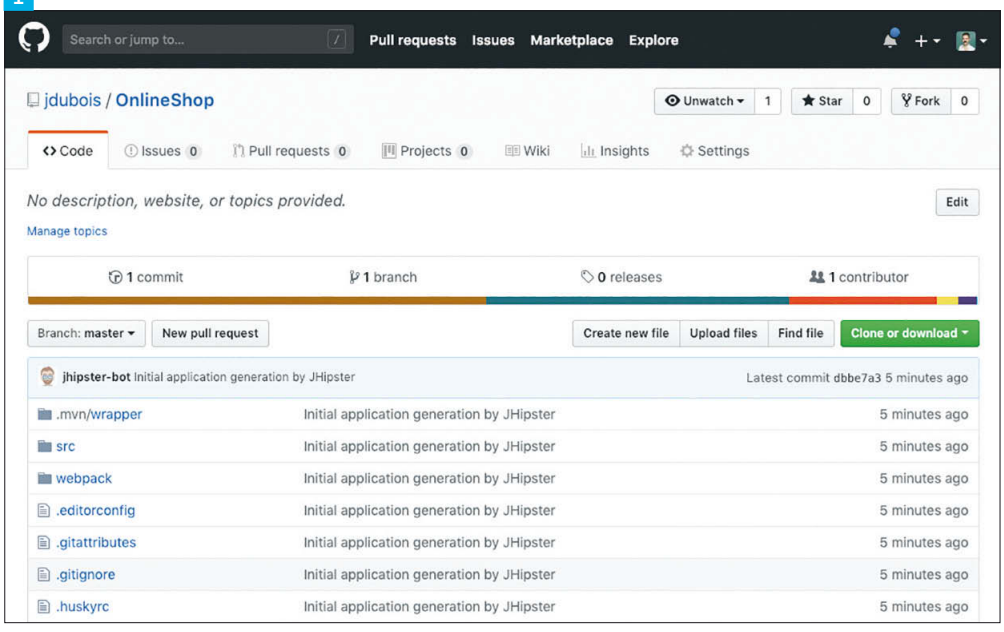
Allez maintenant dans l'onglet "Create application" et créez une application sur votre compte GitHub, que vous appellerez "OnlineShop". Notez toutes les configurations proposées par JHipster, mais laissez les options par défaut. En bas du formulaire, sélectionnez le bouton "Generate on GitHub" : votre application va être automatiquement créée sur votre compte GitHub. 1

Cette application est d'ores et déjà prête à l'emploi : sur la page GitHub de l'application, cliquez sur le bouton "Clone or download" et copiez l'URL du repository Git. Nous allons maintenant cloner cette application sur le poste de travail local. Un projet JHipster fonctionne avec les principaux IDEs du marché comme Eclipse, IntelliJ IDEA ou Netbeans. Nous allons utiliser ici Visual Studio Code car il est entièrement Open Source et rapide à utiliser, mais il n'y a aucune obligation à l'utiliser.

Ouvrez Visual Studio Code. Nous vous conseillons également d'ajouter les plugins suivants, qui vont faciliter l'expérience de développement :

- Java Extension Pack, afin d'avoir un support de Java ;

1



- Spring Boot Tools et Spring Boot Dashboard, car nous allons développer une application basée sur Spring Boot ;
- JHipster JDL.

Notons la présence d'un plugin spécifique JHipster, également disponible pour Eclipse, et qui facilitera le travail sur les fichiers JDL, que nous verrons plus tard dans cet article. Lancez la "Command Palette..." (menu "view") dans Visual Studio Code, et tapez Git: Clone puis collez l'adresse du repository Git du projet que vous venez de générer. 2

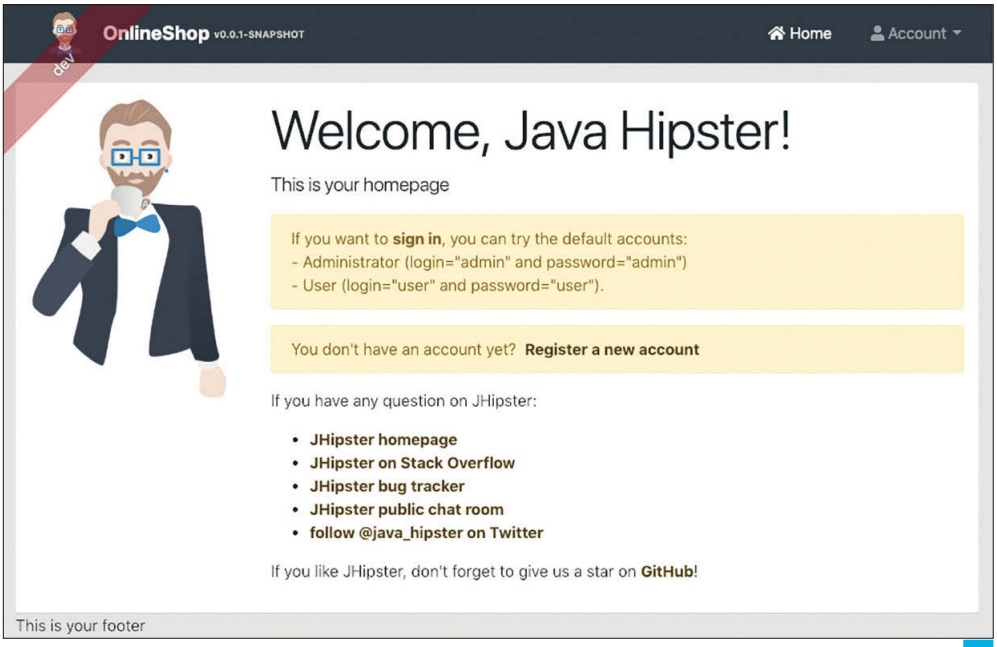
Ouvrez le projet qui vient d'être cloné. Il va être automatiquement reconnu par Visual Studio Code, en ouvrant l'onglet "Spring Boot Dashboard" vous pourrez même démarrer le projet simplement en cliquant. Vous allez alors lancer la partie "serveur" de l'application, sur le port 8080. Cette application n'a pas encore de partie "cliente", donc si vous ouvrez votre navigateur sur http://localhost:8080/ vous verrez que l'application est lancée, mais vous aurez un message d'erreur.

Comme l'indique ce message d'erreur, il vous faut maintenant compiler et lancer l'application cliente, qui est basée sur Angular. Il faut donc pour cela utiliser NPM pour télécharger les dépendances, construire et lancer le projet avec Webpack. Dans un terminal séparé ou dans l'onglet "console" de Visual Studio Code, tapez les commandes suivantes :

- npm install pour installer les dépendances. Cela créera un nouveau fichier package-lock.json avec les versions exactes des dépendances utilisées ;
- npm start pour lancer un serveur Webpack (sur le port 9060) et un serveur Browsersync (sur le serveur 9000) et avoir ainsi accès à l'application générée. Une fenêtre doit automatiquement s'ouvrir sur http://localhost:9000 . 3

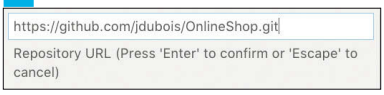
Vous pouvez vous connecter à cette application en utilisant le compte administrateur par défaut (login : "admin" / mot de passe : "admin") et parcourir les écrans ainsi générés. Ces écrans sont de trois types :

- Des écrans nécessaires à un squelette d'application "basique" : page d'erreur, home page…
- Des écrans liés à la sécurité : page d'authentification, de création de compte, d'oubli de mot de passe…



3

2



- Des écrans liés à la santé et à la documentation de l'application, dans le menu "Administration". Nous vous conseillons en particulier l'écran "API", qui fournit automatiquement une documentation Swagger de l'application.

Le projet généré est avant tout un "squelette", qui va servir de base pour développer une application métier plus complexe.

Étude du projet généré

Le projet généré contient du code Java pour la partie serveur, du code TypeScript pour la partie cliente, ainsi que de nombreux fichiers de configuration. Nous allons étudier ce code plus en détail dans cette section.

Code côté serveur

Dans la partie Java, on peut reconnaître un projet Java avec une structure Maven "classique" :

- Un fichier pom.xml qui importe ses versions de dépendances depuis un "Bill of Materials" fourni par JHipster (qui lui-même hérite du "Bill of Materials" de Spring Boot) ;
- Un répertoire src/main/java qui contient le code source Java ;
- Un répertoire src/main/resources qui contient la configuration du projet, en particulier la configuration Spring Boot ;
- Un répertoire src/test/java qui contient les tests du projet.

Le code généré suit une structure très classique dans les projets Spring, où nous avons une séparation par couche technique : des fichiers repository pour accéder à la base de données, service pour la logique métier, et des contrôleurs Spring MVC REST dans le répertoire web/rest.

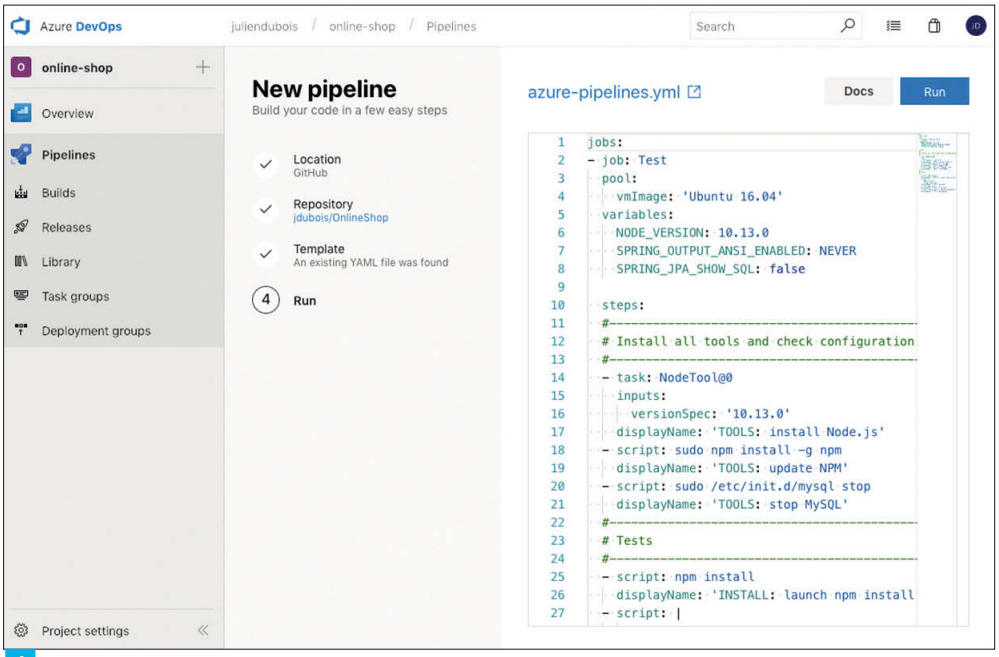
On retrouve ici la première règle du projet JHipster, décrite sur https://www.jhipster.tech/policies/ : pour chaque technologie, la configuration par défaut est utilisée autant que possible, car cela évite les surprises pour les développeurs.

Pour que tous les développeurs utilisent la même version de Maven, et ainsi éviter des risques de construction non reproductible, une version embarquée de Maven est disponible dans le répertoire .mvn, utilisez-la pour lancer les tests Java du projet : ./mvnw test pour Linux ou MacOS X, et mvnw test pour Windows.

Code côté client

Côté client, on retrouve une application Angular dont les fichiers sont côte à côte du code serveur étudié dans la section précédente. Deux choses sont importantes à noter ici :

- Nous avons créé ici un monolithe par défaut, il est tout à fait possible de faire des configurations plus avancées (et plus complexes) si l'on veut découper le code serveur et le code client, mais cela limite



4

la simplicité du code et des workflows de test et de déploiement ;

- L'application cliente développée par défaut utilise Angular, mais une option React est également disponible, et une option Vue.js devrait aussi l'être d'ici peu.

Ce code se sépare donc en deux parties :

- Dans le répertoire src/main/webapp se trouve l'application Web.
- Dans le répertoire src/test/javascript se trouvent les tests. Par défaut, seuls des tests unitaires utilisant Jest sont disponibles, mais il y a également une option pour créer des tests Protractor, qui permettent de tester l'application intégralement de bout en bout.

L'application Angular ainsi générée est nettement plus avancée qu'un simple "Hello, world" comme on peut en créer en utilisant Angular CLI (disponible sur https://cli.angular.io/), avec lequel l'application reste néanmoins en partie compatible. L'idée ici est de proposer une application métier complète, sans pour autant qu'elle soit trop lourde, elle propose donc :

- Une utilisation de Twitter Bootstrap comme framework CSS, car c'est l'outil le plus populaire aujourd'hui.
- L'intégration des images Fort Awesome, également en raison de leur popularité.
- Un certain nombre de composants, en particulier venant du projet ng-bootstrap, afin de gérer des cas classiques comme la pagination ou la gestion des dates.

De même que pour le côté serveur, testez l'ensemble de ce code avec la commande npm test.

Qualité du code généré

Le projet JHipster utilise Sonar pour tester la qualité du code généré. Comme nous venons de le voir dans les deux sections précédentes, un grand nombre de tests sont proposés par défaut, aussi bien pour le code côté serveur que pour le code côté client. L'objectif est d'avoir toujours la note "AAA" chez Sonar, et donc avec 0 bug et 0 vulnérabilité connue.

Ce code est d'ailleurs testé de manière continue à chaque nouveau commit sur la branche de développement de JHipster, et le rapport est disponible en ligne sur https://sonarcloud.io/dashboard?id=io.github.jhipster.sample%3Ajhipster-sample-application.

En conséquence, chaque projet généré a une excellente qualité de code, avec en particulier une couverture de tests que nous vous encourageons à garder au fur et à mesure des développements futurs. Pour cela, vous pouvez utiliser Sonar sur votre projet de la manière suivante :

- Afin de lancer une instance Sonar facilement sur votre poste local, JHipster fournit une configuration Docker Compose clef en main. A la racine de votre projet, il suffit donc de lancer : docker-compose -f src/main/docker/sonar.yml up -d
- Connectez-vous ensuite sur Sonar à l'adresse http://localhost:9001
- A la racine du projet, lancez les tests et les rapports Sonar en exécutant la commande ./mvnw sonar:sonar sous Linux ou MacOS X, ou mvnw sonar:sonar sous Windows.

Veuillez noter que sur cette instance locale, étant donné que nous utilisons la version Open Source de Sonar, seuls les tests Java sont pris en compte.

Mise en place de l'intégration continue

Maintenant que le projet a été généré, et que nous avons une vision sur la qualité du code généré, nous allons mettre en place un système d'intégration continue. Ceci va nous permettre de garantir le bon maintien de la qualité du projet.

Retournez sur JHipster Online et sélectionnez l'onglet "Continuous Integration" :

- Cliquez sur le bouton "Refresh" afin que votre nouveau projet soit synchronisé depuis GitHub, et soit ainsi accessible, puis sélectionnez-le.
- JHipster propose une intégration avec les principaux systèmes d'intégration continue du marché. En utilisant la version en ligne de commande vous aurez plus d'options, mais sur la version en ligne vous avez déjà accès à Travis, Jenkins, GitLab et Azure Pipelines.
- Nous allons sélectionner Azure Pipelines pour cet article : les autres systèmes sont similaires à celui-ci, et nous le privilégions uniquement car à l'heure actuelle il propose d'avoir gratuitement 10 nœuds pour lancer des tests. Dans la liste déroulante en bas de l'écran, sélectionnez "Azure Pipelines" puis cliquez sur le bouton "Yes, add Continuous Integration".
- JHipster Online va alors générer une Pull Request sur votre projet GitHub : il vous suffit de suivre le lien alors proposé pour pouvoir la merger dans le projet principal. Félicitations, vous venez d'ajouter automatiquement votre configuration !
- Il faut maintenant configurer Azure Pipelines. Pour cela il vous faut un ajouter Azure Pipelines à votre compte GitHub en passant par la marketplace GitHub : https://github.com/marketplace/azure-pipelines.
- Une fois Azure Pipelines installé, retournez sur votre projet GitHub et dans l'onglet "Settings", configurez-le en lui donnant accès à votre projet. Suivez ensuite l'installation afin de configurer votre projet dans Azure Pipelines : une

fois le projet créé, il va automatiquement reconnaître la configuration que nous avons générée plus haut. 4

- À l'étape finale de la configuration d'Azure Pipelines, cliquez sur le bouton "Run" en haut à droite pour vérifier que tout est bien configuré, et que les tests s'exécutent. Cette étape doit prendre un peu plus de 5 minutes, et vous recevrez un e-mail de confirmation à la fin. Vous pouvez donc continuer cet article sans attendre !
- À présent, chaque nouveau commit et chaque nouvelle Pull Request sur le projet seront automatiquement testés, nous allons donc pouvoir commencer à développer en toute confiance !

Premières lignes de code

L'ensemble de notre application est prêt, nous allons faire quelques modifications pour étudier de quelle manière JHipster fonctionne. Votre application doit toujours tourner dans Visual Studio Code, le service Java sur le port 8080 et l'application Angular sur le port 9000.

Modification de code côté serveur

Nous allons modifier du code côté serveur afin de voir de quelle manière on développe habituellement avec JHipster, et en particulier pour observer comment le rechargement à chaud du code Java fonctionne.

Recherchez le fichier UserDTO.java : il s'agit d'un Data Transfer Object (DTO) c'est-à-dire d'un objet qui fait le lien entre une entité managée par JPA et un contrôleur Spring MVC REST. C'est cet objet qui est utilisé lorsque l'on fait une requête sur l'URL /api/account, qui est la requête que réalise l'application à l'authentification d'un utilisateur.

- Modifiez la méthode getLogin() pour qu'elle retourne le login de l'utilisateur en majuscule, et donc en ajoutant la méthode "toUpperCaser()" à la variable "login" : login.toUpperCase()
- Sauvegardez juste le fichier : Visual Studio Code va automatiquement le compiler à la sauvegarde, et Spring Boot va ensuite relancer le conteneur Spring avec ce nouveau code. Sur un PC normal, cette mise à jour de Spring Boot doit prendre environ 2 secondes.
- Dans l'application cliente, sur http://127.0.0.1:9000, vous pouvez maintenant vous déconnecter et vous réauthentifier : le login est maintenant en majuscule !

Nous venons de voir ici que l'application générée par JHipster permet de mettre à jour le code Java à chaud, sans redémarrage du serveur d'application, ce qui fait gagner un temps précieux.

Lancez les tests, soit avec Maven (./mvnw test sour Linux ou MacOS X, mvn test sous Windows) soit avec votre IDE : cette modification casse 4 tests unitaires. Comme nous l'avons vu dans les sections précédentes, la couverture de code de JHipster nous permet de nous protéger contre les régressions, et la mise en place de l'intégration continue va nous empêcher de travailler avec ce code s'il n'est pas corrigé (ou si les tests ne sont pas modifiés). Afin de ne pas être bloqué par la suite, annulez les modifications que nous venons de faire (ou corrigez les tests si vous en avez envie, mais cela n'apporte aucune plus-value métier ici).

Pour les utilisateurs les plus avancés, veuillez également noter que nous utilisons ici une base de données H2 embarquée, dont une interface graphique est disponible via l'application Web, dans le menu "Administration > Database". Cette base est également mise à jour automatiquement, en utilisant Liquibase : à chaque redémarrage du conteneur Spring, les scripts Liquibase s'exécutent, ce qui a pour effet de mettre à jour la base de données automatiquement.

Modification de code côté client

Nous allons maintenant modifier du code dans l'application Angular.

- Recherchez le fichier home.component.html - il s'agit de la page de bienvenue qui est affichée lorsque l'on ouvre l'application.
- Remplacez le texte "Welcome, Java Hipster!" par "Welcome to the hipster shop!"
- Quand le fichier est sauvegardé, l'application disponible sur http://127.0.0.1:9000 doit également se mettre à jour automatiquement.
- Vous pouvez également ouvrir un deuxième navigateur pointant vers cette URL : une fois authentifié, les deux navigateurs vont se synchroniser automatiquement. En effet, les actions (scrolling, entrer un texte, cliquer sur un bouton) sont répliquées entre les navigateurs grâce à Browsersync (https://www.browsersync.io/) - ce qui est particulièrement pratique pour tester son code sur plusieurs navigateurs différents, ou avec des résolutions différentes (desktop, tablet, mobile).

Ces modifications sont mises à jour automatiquement grâce à un serveur Webpack intégré, qui a été lancé par la commande npm start. Le code client, comme le code serveur, peut donc être mis à jour automatiquement, sans aucune autre action que de sauvegarder un fichier.

Envoi sur GitHub

Sauvegardons ce code et envoyons-le sur GitHub : dans Visual Studio Code, dans l'onglet "Source Control", commitez les 2 fichiers modifiés (package-lock.json et home.component.html) et pushez-les sur GitHub.

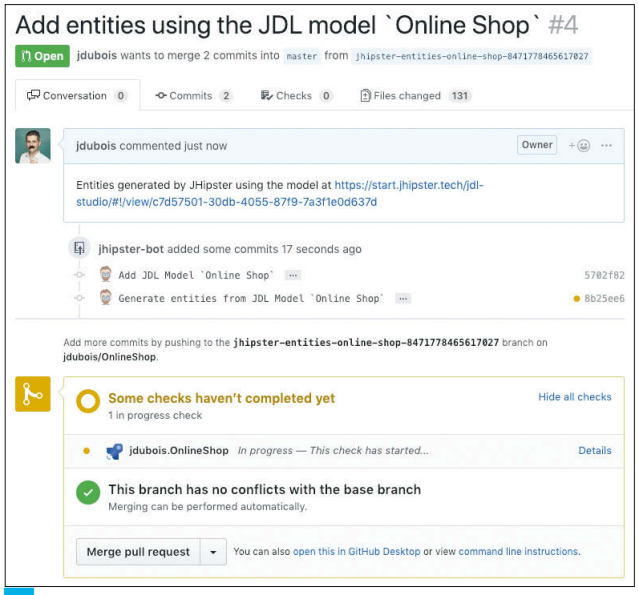
Génération des entités avec le JDL

JHipster peut être vu comme un simple générateur de squelette d'application, ce que nous avons fait jusqu'ici. Cependant, il possède un système de "sous-générateurs" qui permettent d'enrichir une application déjà générée, et qui lui apportent une grande puissance supplémentaire.

Nous allons ici utiliser le sous-générateur le plus populaire, qui est le générateur d'entités. Il permet de générer des CRUD (Create Read Update Delete), et plus précisément :

- Des tables en base de données (grâce à l'utilisation de Liquibase).
- Des entités JPA.
- Une infrastructure Spring gérant des entités et les exposant sous forme de endpoints REST : des repositories, des services (optionnellement), des DTOs (optionnellement), des contrôleurs Spring MVC REST.
- Des écrans Angular gérant l'affichage, la création, l'édition et l'effacement de ces entités.
- L'ensemble des tests associés à ces codes, aussi bien côté serveur que côté client.

Ce sous-générateur peut être utilisé en ligne de commande, ce qui est son fonctionnement basique, mais il va alors rapidement devenir très complexe à configurer, dès que plusieurs de ces entités ont

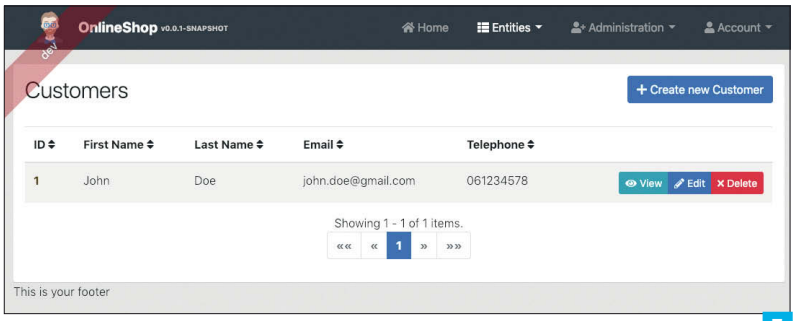


5

6

des relations entre elles. Pour réaliser des applications métier, avec plusieurs entités liées entre elles, JHipster propose pour cela son propre "domain-specific language", c'est-à-dire son propre langage dédié, avec un IDE Web spécifique. Ce langage se nomme le "JHipster Domain Language", ou JDL. Les utilisateurs d'Eclipse et de Visual Studio Code ont également un plugin spécifique permettant d'utiliser ce langage sans quitter leur IDE : c'est le plugin JHipster JDL que nous avons installé dans Visual Studio Code en début d'article. Utilisons ce JDL pour générer notre boutique en ligne :

- Récupérez le JDL représentant une boutique en ligne sur https://github.com/jhipster/jdl-samples/blob/master/online-shop.jh.
- Téléchargez ce fichier (ou copiez/collez son contenu).
- Retournez sur JHipster Online et sélectionnez l'onglet "Design Entities". Cliquez en haut à droite sur "Create a new JDL model".
- Supprimez le code fourni par défaut pour le remplacer par celui de la boutique en ligne, et sauvegardez votre JDL en ligne.
- Retournez sur la page "Design Entities" de JHipster Online, et à côté de votre JDL nouvellement créé, sélectionnez le bouton "Apply".



7

- Sélectionnez votre projet "OnlineShop" dans l'écran suivant, et cliquez sur "Yes, apply this JDL model".
- Une Pull Request va être automatiquement créée sur votre projet, avec le code de l'ensemble des entités créées. 5
- Etant donné que nous avons préalablement intégré Azure Pipelines au projet GitHub, cliquez sur les détails de la Pull Request pour voir si les tests passent correctement, avant de faire le merge.

Maintenant que ce code a été mergé dans la branche principale, il est temps de mettre à jour notre projet en local : dans Visual Studio Code, allez dans l'onglet "Source Control" et cliquez sur l'icône "..." pour faire un pull du projet : 6

Si vos serveurs Java et Webpack fonctionnent toujours, il n'y a rien à faire : ces nouveaux fichiers étant automatiquement sauvegardés et compilés, vos applications serveur et client vont être mises à jour à chaud automatiquement. Attendez juste quelques secondes, et vous aurez accès aux entités de la boutique en ligne dans l'application Angular : 7

D'autre part, si vous voulez étudier plus en détail ce qui est exposé par l'application Spring Boot, restez dans l'application Angular et allez dans le menu "Administration > API" afin d'avoir une vue Swagger de l'ensemble des endpoints REST exposés. Cela vous permettra de les tester facilement, et de mieux comprendre ce qui est proposé par l'application côté serveur.

Passage en production

Maintenant que notre application a été générée et que les tests passent, nous allons la mettre en production. Stoppez les serveurs Java et Webpack qui permettaient de faire du développement, nous allons maintenant construire et déployer notre application pour de vrai !

JHipster supporte de nombreuses plate-formes pour déployer ses applications : Google App Engine, Amazon Web Services, Azure, Heroku, Cloud Foundry, Kubernetes... La grande majorité de ces intégrations sont d'ailleurs faites conjointement entre le projet JHipster et les fournisseurs de cloud eux-mêmes. Nous allons ici lancer notre application dans un container Docker, car quelle que soit votre plateforme vous devriez aisément pouvoir utiliser ce système.

Cette étape consiste à packager l'application sous forme d'image Docker et à l'exécuter :

- Lancez votre daemon Docker (si ce n'est pas déjà fait).
- À la racine du projet, lancez la commande Maven pour construire l'image : `./mvnw package -Pprod verify jib:dockerBuild` avec Linux ou Mac OS X, ou `mvnw package -Pprod verify jib:dockerBuild` sous Windows. Étant donné que cette étape va lancer l'ensemble des tests, elle peut durer quelques minutes.
- Une fois cette image construite, vous pouvez la lancer en utilisant le fichier Docker Compose fourni par JHipster, et qui inclut la base de données de production : `docker-compose -f src/main/docker/app.yml up -d`

Votre application est désormais disponible sur http://localhost:8080 ! Elle a été spécifiquement optimisée pour la production : en particulier, elle utilise un build Webpack de production, qui rend sa partie cliente nettement plus performante que lors de la phase de développement. •

Suivez Julien sur Twitter: @juliendubois
Suivez JHipster sur Twitter: @java_hipster
Retrouvez toute la documentation de JHipster sur https://www.jhipster.tech/

Je tiens particulièrement à remercier les nombreux collègues qui ont relu cet article !

Dorra DHOUIB
Consultante sénior Python

Pandas : le data management à la portée de toutes les mains

Pandas est un package Python permettant de traiter des jeux de données de manière efficace et compréhensible. De la lecture des fichiers à une première approche statistique, Pandas est le compagnon de jeu de n'importe quel datascientist pythonicien. Pandas est souvent accompagné de son ami Scikit Learn, LE package de machine learning de Python. Avant de pouvoir utiliser Scikit Learn, Pandas intervient dans la première phase d'un projet de datascience, afin de lire, comprendre et préparer la donnée.

niveau 200

Nous vous proposons, à travers un exemple, de passer en revue les principales fonctionnalités de ce package, en suivant les grandes étapes du data management.

Nous allons nous intéresser à l'historique des prénoms de 2004 à 2017 dans l'agglomération parisienne (disponibles sur https://opendata.paris.fr/), et pour les plus bretons d'entre nous, nous étudierons aussi les prénoms rennais (disponible sur https://data.rennesmetropole.fr/).

Ces deux jeux de données (datasets) nous permettront d'apprendre à utiliser Pandas.

Les plus courageux pourront utiliser le dataset final pour prédire les prénoms les plus populaires dans les années suivantes et, surtout, choisir le prénom de leur futur enfant.

Il est important de mentionner que l'aide de Pandas est très bien documentée. Les fonctions présentées dans ce tutorial sont le plus souvent utilisées dans leur version par défaut, mais de nombreuses options supplémentaires existent et permettent de simplifier les démarches.

Comment lire et stocker mes données ?

La première étape, et singulièrement parfois la plus compliquée, est de lire les données. Pandas simplifie la lecture de fichiers de tous types : csv, xlsx, txt, json ou encore d'un dictionnaire Python.

Les données lues sont stockées dans des Dataframes.

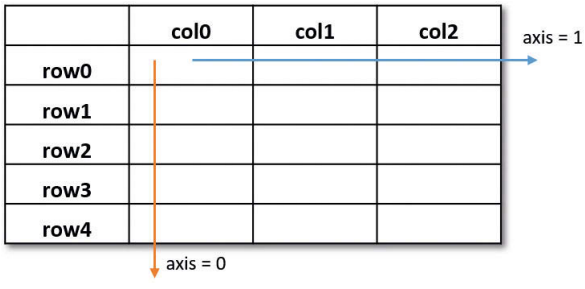
Un dataframe est une structure de données permettant de stocker les données selon deux dimensions : lignes et colonnes.

Les colonnes sont accessibles par les noms de colonnes, et les lignes sont accessibles par leur index.

Un index peut-être un entier, une date, une chaîne de caractères.

Pandas gère également le multi-index.

Revenons à notre lecture de fichier.



De multiples options permettent de gérer :

- les séparateurs ;
- l'encodage ;
- le type de chaque colonne ;
- la présence ou nom d'un en-tête de colonne ;
- la façon de traiter les valeurs manquantes ;
- la façon de traiter les dates ;
- le nombre de lignes à sauter.

Par défaut, Pandas interprète le type des données présentes dans chaque colonne en lisant les premières lignes du fichier. Rien ne permet de s'assurer que cette méthode est toujours performante (allez, disons presque toujours).

Il est donc préférable de parfaitement préparer la lecture du fichier en précisant un schéma de données (en-tête et type). Évidemment, ami pythonicien, l'ajout de quelques **try** et autres **except** est toujours aussi indispensable.

Créer un dataframe après avoir lu les données dans une base SQL, PostgreSQL, Mongo ou tout autre système d'information est aussi tout à fait possible. En fonction des différents SI, des packages et des drivers additionnels doivent être installés. Il faut donc utiliser différentes fonctions pour obtenir un dataframe à partir de listes, de dictionnaires, etc.

Passons de tout de suite à la pratique :

- Pour le fichier csv :

```
df_paris = pd.read_csv('prenoms_paris.csv', sep=';')
```

- Pour le fichier json :

```
with open('prenoms_paris.json') as f:
  data = json.load(f)

prenom_data = []
for d in data:
  prenom_data.append(d['fields'])
df = pd.DataFrame(prenom_data)
```

Si le fichier est trop gros, l'option chunksize permet de lire le fichier en plusieurs fois pour ensuite les concaténer dans un dataframe :

```
chunksize = 10 ** 6
```

```
data_chunks = list()
for chunk in pd.read_csv('prenoms_paris.csv', chunksize=chunksize, sep = ';'):
  data_chunks.append(chunk)
df = pd.concat(data_chunks)
```

Explorer mon dataframe

Une fois le fichier lu, il est temps de passer à une première analyse de ces données. Nul besoin d'un autre package, Pandas s'occupe de tout.

Afficher un échantillon du dataframe

Généralement, on préfère ne pas afficher le dataframe en entier. On se contentera d'afficher juste un sous ensemble du dataframe en utilisant la fonction head (ou tail par symétrie).

df_paris.head() par défaut retourne les 5 premières lignes

	Nombre	Sexe	Annee	Prenoms
0	6	F	2014	Deva
1	6	F	2014	Arya
2	6	F	2014	Yuna
3	6	F	2014	Djeneba
4	6	F	2014	Jihane

Filtrer sur une valeur spécifique

Si je ne veux afficher que les lignes contenant mon prénom :

```
df_paris [ df_paris ['Prenoms']=='Dora' ]
```

Prenons deux minutes pour comprendre ce que fait Pandas :

- pour chaque ligne du dataframe, l'expression entre les crochets est évaluée pour retourner un booléen ;
- seules les lignes avec valeur égale à True sont affichées.

On obtient le résultat suivant :

	Nombre	Sexe	Annee	Prenoms
1698	8	F	2004	Dora
12591	9	F	2005	Dora
14659	7	F	2006	Dora

Description du dataframe

Après avoir affiché notre dataframe, on passe à l'analyse de son contenu :

- nombre de lignes ;
- type des colonnes ;
- valeurs possibles pour chaque colonne.

Le nombre de lignes est obtenu en utilisant **shape** : df_paris.shape qui retourne le nombre de lignes et colonnes de notre dataframe.

Nous obtenons le type de chaque colonne en utilisant **dtypes**.

On obtient le résultat suivant :

```
Nombre      int64
Sexe       object
Annee       int64
Prenoms    object
dtype: object
```

Les colonnes 'Sexe' et 'Prenoms' sont des objets et les colonnes 'Annee' et 'Nombre' sont des entiers.

Afin d'éviter les éventuels conflits, nous voulons que les valeurs de la colonne 'Annee' soient de type str.

Pour cela, nous utilisons la fonction **astype** avec le type voulu en paramètre :

```
df_paris ['Annee'] = df_paris ['Annee'].astype(str)
```

Nous pouvons pour chaque colonne obtenir un résumé des données présentes en utilisant la fonction *describe()* (le nombre de valeurs, la moyenne, l'écart type, le minimum, le maximum et des quantiles). Évidemment, cette fonction ne s'applique que sur les colonnes numériques.

Pour obtenir des informations sur toutes les colonnes, l'option *include='all'* est disponible

```
df_paris .describe()
df_paris.describe(include='all')
```

	Nombre	Sexe	Annee	Prenoms
count	16641.000000	16641	16641	16641
unique	NaN	3	14	2389
top	NaN	F	2017	Andrea
freq	NaN	8449	1312	28
mean	26.232859	NaN	NaN	NaN
std	39.159518	NaN	NaN	NaN
min	5.000000	NaN	NaN	NaN
25%	7.000000	NaN	NaN	NaN
50%	12.000000	NaN	NaN	NaN
75%	26.000000	NaN	NaN	NaN
max	398.000000	NaN	NaN	NaN

Cette fonction nous permet de découvrir qu'il existe dans le dataset 2389 prénoms différents pour des années de 2004 à 2007.

Renommer une colonne

Afin de faciliter la lecture des données par nos amis anglais, nous pouvons aussi renommer une ou plusieurs colonnes en créant un dictionnaire de correspondances avec l'aide de la fonction (attention surprise) **rename**

```
newcols = {
  'Nombre': 'Number',
  'Sexe': 'Sex',
  'Annee': 'Year',
  'Prenoms': 'FirstName'}
df_paris.rename(columns=newcols, inplace=True)
```

L'argument inplace permet de préciser si on veut appliquer sur le dataframe lui-même (inplace = True) ou sur une copie du dataframe (inplace = False). Les deux lignes suivantes sont équivalentes :

```
df_paris.rename(columns=newcols, inplace=True)

df_renamed = df_paris.rename(columns=newcols, inplace=False)
```

Gérer les lignes en double

Une étape importante dans l'exploration de nos données est la suppression des lignes dupliquées via la méthode **drop_duplicates**. Celle-ci retourne un dataframe avec les lignes en double supprimées, avec possibilité de considérer seulement certaines colonnes. Elle prend en argument :

- subset : liste des colonnes sur lesquelles on va vérifier l'existence de lignes en double ;
- Option keep qui a trois valeurs possibles : 'first' : garder la première occurrence, 'last' garder la dernière occurrence et False pour supprimer toutes les occurrences.

Pour notre cas, nous voulons vérifier sur toutes les colonnes et garder la première occurrence :

```
df_paris.drop_duplicates(subset=['FirstName', 'Year', 'Sex'], keep='first')
```

Gérer les valeurs inconnues

On observe que la colonne 'Sexe' possède trois valeurs distinctes : F, M et X.

La valeur 'X' peut avoir plusieurs significations : une valeur manquante, une valeur indéterminée ou même les prénoms qui sont à la fois masculins et féminins.

Dans un monde idéal, il faudrait contacter le responsable de l'extraction pour obtenir cette information. Ici, nous ne pouvons effectuer cette démarche, donc nous allons enquêter par nos propres moyens :

On filtre sur les lignes où les valeurs de la colonne 'Sexe' sont égales à 'X'

```
df_X = df_paris[ df_paris['Sexe']=='X' ]
```

On récupère la colonne 'Prenoms '

```
df_prenoms_X = df_X['Prenoms']
```

On récupère la liste des prénoms distincts

```
Prenoms_X = df_prenoms_X.unique()
```

On peut concaténer les étapes a, b et c en une seule ligne :

```
df_paris[df_paris['Sexe']=='X']['Prenoms'].unique()
```

On obtient les prénoms suivants :

```
array(['Keziah', 'Noha', 'Jessy', 'Ely', 'Dominique', 'Yaya', 'Paris',
       'Luan', 'Dali', 'Alexy', 'Felicite'], dtype=object)
```

Au vu des prénoms affichés, l'hypothèse principale est que ce X représente en fait une valeur manquante. Nous choisissons donc de filtrer ces lignes.

```
df_paris = df_paris[ df_paris['Sexe'] != 'X' ]
```

Encore plus d'options

Pour faciliter la lecture des données, il est parfois utile de ranger les valeurs d'une colonne.

```
df.sort_values(by='Nombre', ascending = False)
```

On observe que sur les 11 premières lignes, on retrouve 9 Gabriel et 2 Adam surtout sur les dernières années. Dans le top 15, on retrouve un seul prénom féminin.

Pour étudier une seule colonne ou quelques valeurs, de nombreuses possibilités sont à notre disposition.

- Une colonne : df['Annee']
- Quelques lignes : df[0:3]
- Pour obtenir toutes les valeurs de la ligne 3 : df.iloc[3]
- Pour obtenir la valeur de la colonne A correspondant à une valeur particulière : df.loc[dates[0],'A']

Attention à ne pas confondre : loc et iloc !!

- **toujours commencer par afficher le début du dataframe ainsi que les types de colonnes :**
- **redéfinir le type des colonnes si besoin ;**
- **supprimer les lignes dupliquées ;**
- **utiliser l'option unique() sur une colonne pour vérifier les valeurs possibles ;**
- **faire attention à l'argument inplace**

Modifier mon dataframe

Gérer les valeurs manquantes

Une problématique, d'une grande importance pour les data scientists, est la gestion des valeurs manquantes.

La méthode la plus simple, mais rarement la plus efficace, est d'enlever toutes les lignes ayant au moins une valeur manquante :

```
df_paris.dropna(how='any')
```

Il est tout aussi facile de remplacer les valeurs manquantes par une valeur fixe ou par une valeur moyenne :

```
df_paris.fillna(value=5)
df_paris.fillna(subset=['Nombre'], value=df_paris.Nombre.mean())
```

Créer une nouvelle colonne

On entend souvent que les prénoms courts sont à la mode. Nous nous proposons de vérifier cette assertion. Pour cela, nous allons créer une nouvelle colonne grâce à une des nombreuses méthodes possibles. En utilisant le concept de list comprehension, on peut créer une colonne de la façon suivante :

```
df_paris ['NumberLetter'] = [len(x) for x in df_paris ['FirstName']]
```

Ceci nous permet de trouver un prénom de deux lettres : El qui est donné régulièrement ces dernières années.

On peut choisir de créer une nouvelle règle basée sur des règles de gestion. Nous proposons une toute première approche pour estimer le sexe d'un individu basé sur la dernière lettre du prénom ou la longueur du prénom.

Le package Numpy peut aussi être utile avec sa fonction where :

```
import numpy as np
df_paris ['SexePredicted'] = np.where(df_paris .FirstName.str[-1:] == 'a', 'F',
            np.where(df_paris.NumberLetter < 2,'M',
          np.where(df_paris.FirstName.str.contains('john'),'M','F')))
```

Modifier une colonne

Il peut être utile aussi de 'recoder' une colonne.

Souvent, il existe de multiples valeurs correspondant à la même occurrence (exemple : H pour Homme).

```
df_paris ['Sex'] = df_paris ['Sex'].apply({'M':'Masculin', 'F':'Féminin'}.get)
```

Ici on transforme le 'M' en masculin et le 'F' en Féminin

	Number	Sex	Year	FirstName
0	6	Féminin	2014	Deva
1	6	Féminin	2014	Arya
2	6	Féminin	2014	Yuna
3	6	Féminin	2014	Djeneba
4	6	Féminin	2014	Jihane

Ne pas oublier de checker si la colonne 'Sexe' ne présente pas d'autres valeurs qu'on aurait oublié de traiter.
On peut même appliquer une fonction custom et l'appliquer à une colonne de notre dataframe :

```
lower = lambda x: x.lower()
df_paris ['FirstName'] = df_paris['FirstName'].apply(lower)
```

Filtrer des lignes

On peut utiliser des méthodes Pandas ou des méthodes plus pythoniques :

```
df_paris [np.max(df['Number']) == df_paris ['Number']]
df_paris [(df.Number == 1) & (df_paris.NumberLetter == 6)]
df_paris [(df['Number'] <= 1) & (df_paris ['NumberLetter'] > 6)]

df_paris.pipe(lambda d: d[d[' Number '] == np.max(df_paris ['Number'])])
df_paris.query('Number>0').query('0<NumberLetter<2')
```

L'avantage des deux dernières est qu'elles permettent de chaîner un grand nombre de lignes de codes pour rendre le code plus compact.

Note

En sortie du groupby, nous obtenons un multi-index, il faut donc utiliser la fonction reset_index pour transformer le multi-index en colonne.

Groupby

Pour finir, il nous reste à vous présenter le groupby qui fonctionne de la même façon que le groupby en SQL.
Si on veut sommer les prénoms donnés sur toutes les années, on regroupe par nom et on somme sur le nombre :

```
df_paris.groupby(['FirstName'])['Number'].sum().reset_index(drop=False).sort_values('Number').tail(5)
```

	FirstName	Number
246	arthur	3631
1332	louise	3902
28	adam	3989
1887	raphaël	4142
803	gabriel	4752

Si on oublie l'option reset_index , on obtient le résultat suivant :

```
FirstName
aaliyah        80
aaron        1134
abby           11
abd            10
abdallah      129
Name: Number, dtype: int64
```

Les colonnes 'Firstname' et 'Number' sont passées en index.
Revenons à notre groupby. Nous obtenons les prénoms les plus donnés depuis 2004 dans Paris. Les vainqueurs sont Louise et Emma pour les filles et Gabriel et Raphaël pour les garçons.
Vous pouvez également ajouter la colonne 'Sex' au groupby pour avoir les top prénoms féminins et masculins.

- les valeurs NA ne sont pas toujours à supprimer, tout dépend de leurs significations et de ce qu'on veut faire avec le dataframe ;
- faire attention aux index lorsqu'on effectue des transformations sur notre dataframe.

Joindre plusieurs dataframes

En data management, une des actions les plus courantes est de joindre plusieurs tables pour regrouper les données en un seul dataframe. Nous avons à notre disposition un second dataframe, l'équivalent du premier, mais avec les prénoms donnés dans la ville de Rennes. Avant de pouvoir joindre ces deux dataframes, nous devons unifier les différentes colonnes.

```
df_rennes = pd.read_csv('prenoms-a-rennes.csv', sep =';')
df_rennes ['Année de naissance '] = df_rennes ["Année de naissance "].astype(str)
newcols = {
  'Nombre': 'Number',
  'Sexe': 'Sex',
  'Année de naissance': 'Year',
  'Prénom': 'FirstName'
}
df_rennes.rename(columns=newcols, inplace=True)
df_rennes.drop_duplicates(subset=['FirstName', 'Year', 'Sex'], keep='first')
df_rennes['Sex'] = df_rennes['Sex'].apply({'Masculin':'M', 'Féminin':'F'}.get)
df_rennes = df_rennes[ df_rennes['Sex'] != 'X' ]
df_rennes['FirstName'] = df_rennes['FirstName'].apply(lower)
```

Nous pouvons maintenant joindre ces fichiers :

```
col_to_keep_paris = ['FirstName', 'Year', 'Sex', 'Number']
col_to_keep_rennes = ['FirstName', 'Year', 'Sex', 'Number']
dfrennes_merged = df_rennes[col_to_keep_rennes].merge(df_paris[col_to_keep_paris],
 how='inner', on = ['FirstName', 'Year', 'Sex'])
```

	FirstName	Year	Sex	Number_x	Number_y
0	imran	2015	Masculin	1	37
1	jonah	2015	Masculin	1	6
2	kenan	2015	Masculin	1	6
3	liham	2015	Masculin	1	5
4	loup	2015	Masculin	1	6

Nous avons deux colonnes qui portent le même nom 'Number' mais qui n'a pas été utilisé pour la jointure. Après la jointure, nous récupérons alors deux colonnes 'Number_x' et 'Number_y'. Pour éviter les confusions, nous devons renommer les colonnes avant la jointure.

```
dfrennes_merged.sort_values(['NumberRennes','NumberParis'], ascending=False)
```

	FirstName	Year	Sex	NumberRennes	NumberParis
3217	louise	2015	Féminin	67	293
915	arthur	2014	Masculin	64	255
4783	raphaël	2015	Masculin	62	320
56	gabriel	2015	Masculin	61	355
3629	raphaël	2016	Masculin	60	340
4025	louise	2014	Féminin	59	310
1374	gabriel	2017	Masculin	58	379
2046	raphaël	2014	Masculin	58	316
4645	louise	2012	Féminin	58	306
3183	jules	2016	Masculin	58	151

Nous retrouvons les mêmes prénoms dans les top prénoms pour la ville de Rennes. Evidemment, les valeurs dans la colonne NumberRennes sont plus petites que pour Paris. On peut imaginer trouver un indicateur permettant de renormaliser ces valeurs permettant de les comparer.

Les différentes possibilités pour l'argument how sont :

- left: toutes les lignes du dataframe de gauche sont gardées auquelles sont rajoutées les informations du dataframe de droite ;
- right: symétrique à left ;
- outer: garde toutes les lignes des deux dataframes ;
- inner: garde uniquement les lignes communes dans les deux dataframes.

Pour conclure, nous pouvons chercher les prénoms les plus donnés sans qu'ils aient été donnés dans l'autre ville :

```
dfrennes_merged = df_rennes[col_to_keep_rennes].merge(df[col_to_keep_paris], how='outer', on = ['FirstName', 'Year', 'Sex'])
dfrennes_merged[np.isnan(dfrennes_merged['NumberParis'])].sort_values(['NumberRennes']).tail()
```

Pivoter un dataframe

Si vous voulez transformer (reshapping) votre dataframe, Pandas vous permet aussi de le faire.

Pivoter les colonnes en lignes, ou inversement, les lignes en colonnes, se fait avec deux fonctions qui fonctionnent de manière symétrique : pivot et melt.

La fonction pivot

La fonction pivot prend trois arguments qui déterminent la façon dont le dataframe va pivoter :

- values : les valeurs du dataframe original que vous voulez voir dans la table pivot.
- columns : cet argument permet de spécifier ce qui deviendra la colonne dans la table pivot.
- index : cet argument permet de spécifier ce qui deviendra l'index dans la table pivot.

Afin de mieux évaluer les tendances annuelles et l'évolution du choix des prénoms, nous pouvons utiliser cette fonction pour pivoter notre dataframe

```
df_pivot = df_paris.pivot_table(values = 'Number', columns = 'Year', index = ['FirstName', 'Sex'])
```

		Year	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
FirstName	Sex															
aaliyah	Féminin		NaN	NaN	NaN	NaN	NaN	8.0	7.0	NaN	11.0	13.0	12.0	11.0	9.0	9.0
aaron	Masculin		55.0	52.0	66.0	76.0	92.0	76.0	86.0	79.0	92.0	83.0	90.0	74.0	97.0	116.0
abby	Féminin		NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.0	5.0
abd	Masculin		NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	5.0	NaN	NaN
abdallah	Masculin		7.0	9.0	6.0	NaN	10.0	11.0	13.0	6.0	13.0	9.0	18.0	9.0	10.0	8.0

Le prénom et le sexe sont en index, et les valeurs de la colonne year sont les noms des nouvelles colonnes.

Nous pouvons alors étudier l'évolution du prénom, Hugo, par exemple :

```
df_pivot.query("FirstName == 'hugo'")
```

		Year	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
FirstName	Sex															
hugo	Masculin		210.0	161.0	157.0	158.0	135.0	161.0	148.0	142.0	165.0	196.0	170.0	117.0	139.0	116.0

Des agrégations peuvent aussi être effectuées :

```
df_pivot = df_paris.pivot_table(values = 'Number', columns = 'Year', index = 'FirstName', aggfunc='sum')
```

Cette dernière ligne de code permet de sommer le nombre d'occurrences des prénoms mixtes en une seule et même ligne.

Year	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
FirstName														
aaliyah	NaN	NaN	NaN	NaN	NaN	8.0	7.0	NaN	11.0	13.0	12.0	11.0	9.0	9.0
aaron	55.0	52.0	66.0	76.0	92.0	76.0	86.0	79.0	92.0	83.0	90.0	74.0	97.0	116.0
abby	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.0	5.0
abd	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	5.0	NaN	NaN
abdallah	7.0	9.0	6.0	NaN	10.0	11.0	13.0	6.0	13.0	9.0	18.0	9.0	10.0	8.0

Nous avons maintenant le dataframe df_pivot qui peut servir à effectuer des séries temporelles en utilisant scikit learn, ou le dataframe dfrennes_merged qui peut nous permettre de classifier un prénom entre Paris et Rennes.

La fonction melt

La fonction melt est la fonction inverse de la fonction pivot (ou pivot_table).

Dans le cas où les données d'une ou plusieurs colonnes sont des variables identifiantes, tandis que toutes les autres colonnes sont des valeurs numériques, nous pouvons alors transformer plusieurs colonnes en une seule qui prendra des valeurs différentes.

Dans ce cas, le nouveau dataframe aura plus de lignes et moins de colonnes.

```
df_pivot.reset_index(drop=False,inplace=False).melt(id_vars=['FirstName'],value_vars=df_pivot.columns.tolist())
```

	FirstName	Year	value
0	aaliyah	2004	NaN
1	aaron	2004	55.0
2	abby	2004	NaN
3	abd	2004	NaN
4	abdallah	2004	7.0
5	abdel	2004	NaN
6	abdelkader	2004	NaN
7	abdellah	2004	6.0
8	abderrahmane	2004	NaN
9	abdou	2004	NaN
10	abdoul	2004	NaN
11	abdoulaye	2004	15.0
12	abdourahmane	2004	NaN
13	abdramane	2004	NaN
14	abdullah	2004	NaN

Nous avons recréé le dataframe d'origine, à une exception près. Nous avons maintenant toutes les combinaisons possibles FirstName/Year, même celles qui n'ont aucune valeur.

Et pour finir, comment sauvegarder, un dataframe ?

Une fois retravaillé, nous pouvons sauvegarder notre dataframe sous différents formats, en fonction de l'objectif métier derrière cette automatisation : csv, xlsx, json ou même remplir automatiquement une base SQL-like ou PostgreSQL.
Pour les deux premiers, Pandas propose de manière native des fonctions qui permettent de le faire à peu de frais :

```
df.to_csv('myDataFrame.csv', sep='\t', encoding='utf-8')
df.to_excel('myDataFrame.csv', encoding='utf-8')
```

De nombreuses options permettent d'améliorer la lecture des fichiers de sortie, de l'encodage (toujours un piège), au séparateur des décimales en passant par l'ajout ou non de l'index dans le fichier de sortie (drop = True). En bonus, le package xlswriter permet d'ajouter couleurs, format de cellule, graphiques, etc. Dans les cas où le fichier de sortie a pour objectif une lecture facile pour les gens du métier, ce package leur change la vie.
Pour les sorties au format json, une fonction existe aussi dans le package Pandas. L'argument le plus important est orient qui permet de définir les clés du json. L'index, les colonnes et les valeurs peuvent passer sur plusieurs formats :

```
with open('df.json', 'w', encoding='utf-8') as file:
  df.to_json(file, force_ascii=False)
```

La dernière possibilité évoquée ici est de pouvoir remplir une base base SQL-like ou même une base PostgreSQL en itérant sur les lignes de notre dataframe (avec l'aide de psycopg2 par exemple). Pour ceci, rien de plus simple, comme souvent avec Python, la fonction iterrows() nous permet d'itérer par ligne en donnant des tuples (index, rows).

Conclusion

Dans cet article, nous vous avons présenté le package Pandas qui permet de réaliser l'étape majeure, et souvent négligée, d'un projet data science. Le data management avec Pandas est simple, efficace et le code résultant est surtout parfaitement lisible.
Nous vous avons fourni les éléments essentiels pour une première approche, mais il existe de nombreux paramètres et de nombreuses autres fonctions qui facilitent encore plus le traitement des données que vous pouvez retrouver sur le site d'aide de Pandas.
N'oubliez pas que, par la suite, c'est le package scikit learn qu'il faut utiliser pour analyser ces données. Peut-être dans un prochain article •

Philippe Charrière
@k33g_org
Technical Account Manager pour GitLab
Fondateur de Bots.Garden (éleveur de bots)
Associé chez Clever Cloud

Partie 4

GitLab - Déploiement continu avec OpenFaaS

Dans cette partie, nous allons découvrir les principes du déploiement continu avec GitLab CI. Pour cela nous allons avoir besoin d'une plateforme qui nous permette de déployer des applications web (c'est le cas d'usage que j'ai choisi) ; j'ai choisi la plateforme OpenFaaS qui vous permet de "faire" du FaaS (function as a service). OpenFaaS a le mérite d'être simple à déployer et facile à prendre en main. Vous pouvez bien sûr déployer sur les plateformes que vous souhaitez avec GitLab CI, mais je reste dans mon mode de fonctionnement où j'essaye de tout faire tourner sur ma machine (comme ça je continue à m'amuser même là où je n'ai pas de WiFi).

niveau 200

J'ai écrit cette partie de manière à ce que vous puissiez avancer, même si vous n'avez pas lu les articles précédents (en revanche, je ne rappelle pas toutes les bases de départ).

Pour cette partie, nous allons avoir besoin de plusieurs VM :

- Une VM GitLab ;
- Une VM OpenFaaS ;
- Une VM Docker Registry ;
- Une VM GitLab Runner avec la CLI OpenFaaS.

Installation

GitLab

Si vous avez suivi les articles précédents, vous l'avez déjà. Sinon, j'utilise **Vagrant** et **VirtualBox**, et vous pourrez trouver le Vagrantfile pour l'instance **GitLab** ici :
https://gitlab.com/tanuki-tools/vagrant-files/blob/master/gitlab-ce/Vagrantfile.
Le setup est simple :

- copiez le Vagrantfile dans un dossier ;
- dans le fichier /etc/hosts de votre ordinateur ajoutez la ligne 172.16.245.121 gitlab-ce.test (si c'est ce que vous qvez retenu comme ip et nom de domaine) ;
- dans le dossier du Vagrantfile, lancez la commande vagrant up ;
- patientez ;
- connectez-vous à http://gitlab-ce.test ;
- renseignez le mot de passe du user root ;
- vous avez "un GitLab" sur votre machine.

Serveur OpenFaaS et Docker Registry

Prérequis : *pour tester OpenFaaS, vous aurez besoin au début d'avoir le client* **Docker** *installé sur votre poste.*

Vous trouverez le Vagrantfile de création des VM **OpenFaaS** et de la **Docker Registry** ici :
https://gitlab.com/tanuki-tools/vagrant-files/blob/master/openfass-server/Vagrantfile
Une fois encore le setup est simple :

- copiez le Vagrantfile dans un autre dossier ;
- dans le fichier /etc/hosts de votre ordinateur ajoutez la ligne 172.16.245.160 openfaas.test ainsi que 172.16.245.150 registry.test (si c'est ce que vous avez retenu comme IP et nom de domaine) ;
- dans le dossier du Vagrantfile, lancez la commande vagrant up ;
- patientez ;
- durant l'installation, dans votre console vous verrez le password d'administration dans les logs d'affichage (quelque chose comme ceci password: 2b8fb3db5c0f664af7ab8e7e631c69aade1a88e1e9d8ac8e917b196b698ab7ab), copiez le quelque part pour la suite ;
- connectez-vous à http://open-faas.test:8080 ;
- utilisez le mot de passe pour vous connecter à votre instance **OpenFaaS** avec le user admin ;
- Vous avez un serveur OpenFaaS sur votre machine.

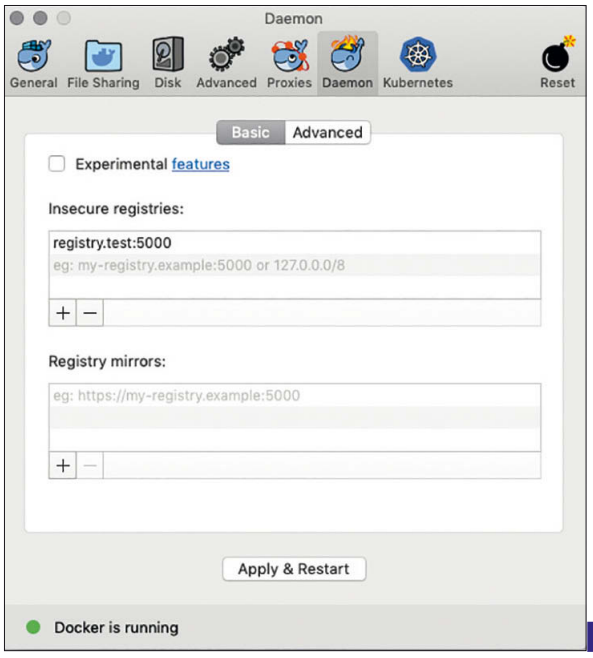
Vous pouvez trouver plus d'informations ici
https://gitlab.com/tanuki-tools/vagrant-files/blob/master/openfass-server/README.md

Docker Registry

Vous avez aussi maintenant, une **Docker Registry**, accessible ici :
http://registry.test:5000/v2/_catalog
Pour cela, allez dans les préférences de votre client **Docker** et ajoutez registry.test:5000 comme insecure registry, puis clickez sur on **Apply & Restart**. 1

OpenFaaS CLI

Dans un premier temps, pour découvrir et créer votre première fonction **OpenFaaS** vous aurez besoin d'installer la CLI **OpenFaaS**. Cette installation est simple et décrite ici : https://docs.openfaas.com/cli/install/.



1

Remarque

Vous allez devoir paramétrer votre client **Docker** *pour qu'il puisse utiliser votre* **Docker Registry**

GitLab Runner pour OpenFaaS

Avant de rentrer dans le vif du sujet, nous avons besoin d'installer un **GitLab Runner** dans une VM qui contiendra aussi un client **Docker** et la CLI **OpenFaaS**, ainsi le runner sera capable de préparer des images **Docker**, les "pousser" dans notre registry et les déployer sur notre instance **OpenFaaS**.
Le Vagrantfile est ici:
https://gitlab.com/tanuki-tools/vagrant-files/blob/master/gitlab-runner-openfaas-cli/Vagrantfile

Remarque importante

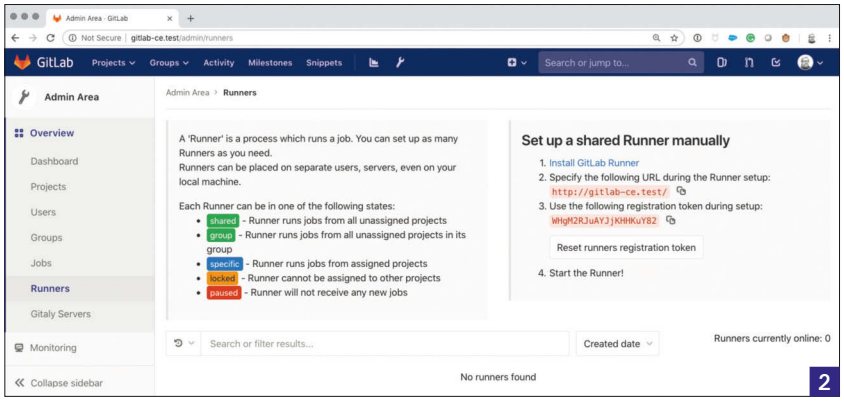
Ne lancez pas l'installation du runner tout de suite.

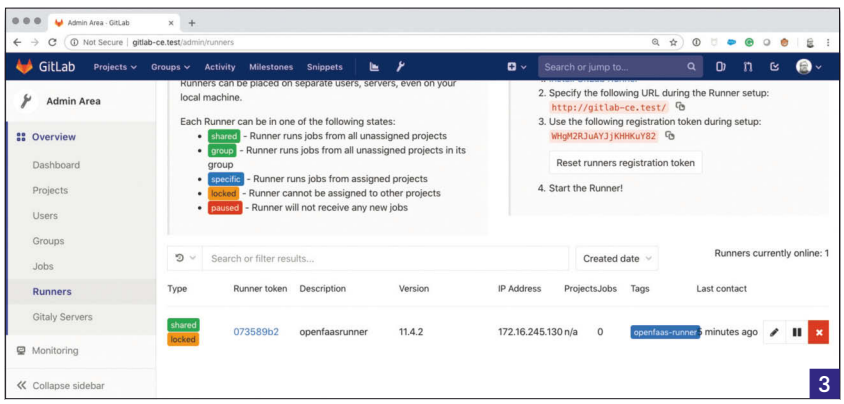
Avant de lancer le runner vous devez récupérer quelques informations sur votre instance **GitLab** pour lui permettre de s'enregistrer auprès de votre instance **GitLab** justement. Donc en tant qu'administrateur allez dans la rubrique runners de l'administration : 2 et copiez les informations (le token et l'url) dans le Vagrantfile du runner :

```
CI_REGISTRATION_URL="http://gitlab-ce.test/"
CI_REGISTRATION_TOKEN="WHgM2RJuAYJjKHHKuY82"
```

Vérifiez que les différentes ip et noms de domaines des autres VM sont corrects dans le Vagrantfile du runner :

```
REGISTRY_IP = "172.16.245.150"
REGISTRY_DOMAIN = "registry.test"
REGISTRY_PORT = 5000
REGISTRY = "registry.test:#{REGISTRY_PORT}"

OPENFAAS_IP = "172.16.245.160"
OPENFAAS_DOMAIN = "open-faas.test"

GITLAB_DOMAIN = "gitlab-ce.test"
GITLAB_IP = "172.16.245.121"
```





Sauvegardez et lancez la commande vagrant up, patientez ... Ensuite, si vous rafraîchissez votre page d'admin (au niveau de la section runner), vous verrez s'afficher votre runner : 3
Donc nous avons un runner qui a été enregistré avec la commande suivante (pour les paramètres, éventuellement adaptez) :

```
gitlab-runner register --non-interactive \
 --url "http://gitlab-ce.test/" \
 --name "openfaasrunner" \
 --registration-token WHgM2RJuAYJjKHHKuY82 \
 --tag-list "openfaas-runner" \
 --executor shell
```

Nous avons donc une "infrastructure virtuelle" qui ressemble à ceci : 4
Et nous sommes enfin prêts à travailler. Faisons un petit tour de découverte rapide de la plateforme **OpenFaaS**.

OpenFaaS - Découverte

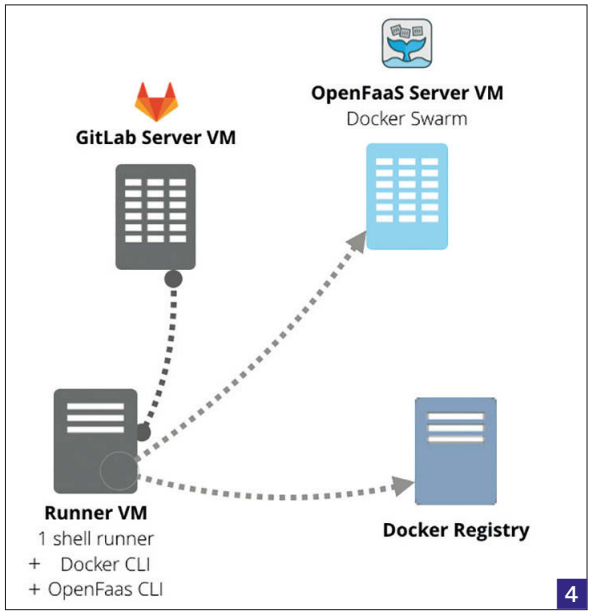
OpenFaaS vous permet de développer des fonctions dans divers langages (Java, Python, C#, JavaScript, ...) dont la liste est potentiellement infinie, puisque vous pouvez fournir vos propres templates (mais c'est un autre sujet).
Nous allons faire dans le simple et efficace, et nous allons créer une fonction **OpenFaaS** en JavaScript. Dans un terminal, entrez les commandes suivantes :

```
mkdir hello-world-project
cd hello-world-project/
faas-cli new hello-world --lang node
```

La dernière commande va vous générer le squelette de votre fonction :

```
.
├── hello-word-project
    ├── hello-world
    │   ├── handler.js
    │   └── package.json
    ├── hello-world.yml
    └── template
```

Si vous éditez le fichier hello-world.yml, vous devriez avoir ceci :



```
provider:
 name: faas
 gateway: http://127.0.0.1:8080
functions:
 hello-world:
  lang: node
  handler: ./hello-world
  image: hello-world:latest
```

Modifiez hello-world.yml comme ci-dessous pour préciser que vous déployez sur http://open-faas.test:8080 et que vous utilisez votre propre **Docker Registry** (registry.test:5000, dans le cas où vous voulez utiliser **Docker Hub**, remplacez registry.test:5000 par votre handle sur le **Docker Hub**) :

```
provider:
 name: faas
 gateway: http://open-faas.test:8080
functions:
 hello-world:
  lang: node
  handler: ./hello-world
  image: registry.test:5000/hello-world:latest
```

Le code source de votre fonction se trouve ici : hello-world-project/hello-world/handler.js. Modifiez-le de la façon suivante :

```
"use strict"

module.exports = (context, callback) => {
 callback(undefined, {message: "Hello World"});
}
```

Maintenant nous allons construire l'image **Docker** de notre fonction et la publier sur notre **Docker Registry** :

```
faas-cli build -f hello-world.yml
faas-cli push -f hello-world.yml
```

Vous pouvez vérifier en ouvrant cette url http://registry.test:5000/v2/_catalog que vous avez bien une nouvelle image dans votre registry :

```
{"repositories":["hello-world"]}
```

Nous pouvons donc enfin déployer notre fonction sur notre serveur **OpenFaaS** :

```
# préciser une 1ère fois l'url du serveur
export OPENFAAS_URL=http://open-faas.test:8080
# s'authentifier
echo -n 2b8fb3db5c0f664af7ab8e7e631c69aade1a88e1e9d8ac8e917b196b698ab7ab | faas-cli
 login --username=admin --password-stdin

# déployer
faas-cli deploy -f hello-world.yml
```

Si tout va bien, vous devriez obtenir ceci :

```
Deploying: hello-world.

Deployed. 202 Accepted.
URL: http://open-faas.test:8080/function/hello-world
```

Remarque

2b8fb3db5c0f664af7ab8e7e631c69aade1a88e1e9d8ac8e917b196b698ab7ab est le mot de passe généré lors de l'installation d'OpenFaaS.

Maintenant, si vous "appelez" http://open-faas.test:8080/function/hello-world directement avec un navigateur, vous obtiendrez {"message":"Hello World"}

Avec **curl**: curl http://open-faas.test:8080/function/hello-world vous obtiendrez le même résultat.

Ou encore vous pouvez vous connecter directement dans l'IHM web d'**OpenFaaS** et exécuter la fonction directement : 5

Déploiement continu sur OpenFaaS avec GitLab CI

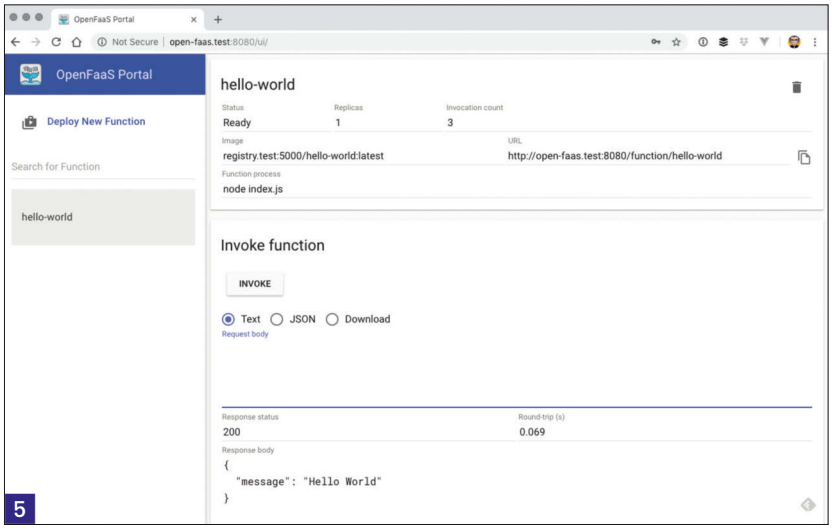
Donc vous voyez, le "**Server Less**" même avec plein de serveurs, c'est super simple. Maintenant voyons comment mettre ça en gestion de configuration sur **GitLab** et comment utiliser les runners pour déployer le projet à chaque changement.

Remarque

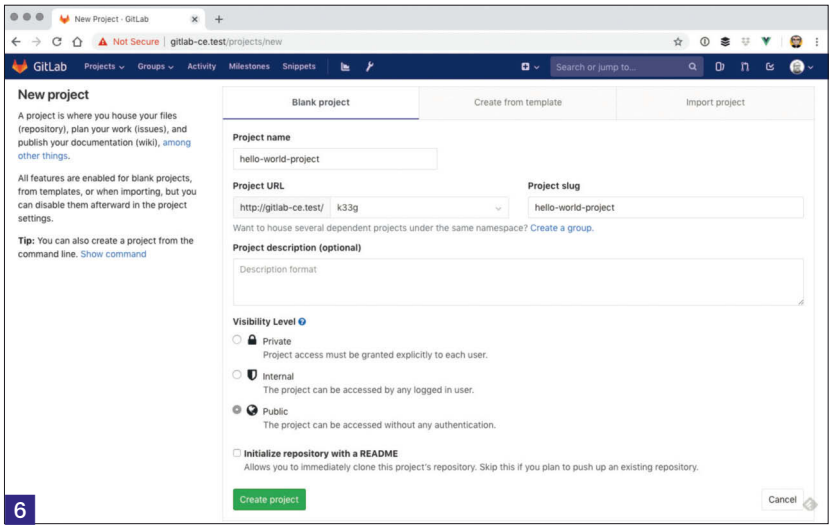
Ne créez pas de fichier README.md

Publier notre projet sur notre GitLab

Pour cela, allez sur votre instance **GitLab**, pour créer un projet hello-world-project : 6



5



6

Ensuite sur votre poste :

```
cd hello-world-project
git init
git remote add origin git@gitlab-ce.test:k33g/hello-world-project.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

Activer la partie CI

Tout d'abord, allez dans les settings CI/CD de votre projet et créez une variable ADMIN_PASSWORD avec le mot de passe du serveur **OpenFaaS**: 7

Ensuite, sur votre poste, dans votre projet, ajoutez un fichier .gitlab-ci.yml avec le contenu suivant :

```
# Nous aurons 3 étapes principales
stages:
 - build
 - package
 - deploy

variables:
 OPENFAAS_URL: "http://open-faas.test:8080"

# Build function: job de build de l'image de la fonction
function_build:
 stage: build
 tags: # seul les runners tagués `openfaas-runner` seront utilisés
  - openfaas-runner
 only: # ce job n'est déclenché uniquement que lorsqu'il y a un commit ou un push sur master
  - master
 script: # création de l'image
  - echo -n $ADMIN_PASSWORD | faas-cli login --username=admin --password-stdin
  - faas-cli build -f hello-world.yml

# Push function to Docker Registry
function_push:
 stage: package
 tags:
  - openfaas-runner
 only:
  - master
 script: # envoi de l'image sur la Docker Registry
  - faas-cli push -f hello-world.yml

# Deploy function to OpenFaaS
function_deploy:
 stage: deploy
 tags:
  - openfaas-runner
 environment:
  name: production/hello-world
  url: $OPENFAAS_URL/function/hello-world
 only:
  - master
 script: # déploiement de la fonction sur OpenFaaS
  - faas template pull
  - faas-cli deploy -f hello-world.yml
```

Remarque

Toutes les explications sont dans les remarques dans le fichier .gitlab-ci.yml

Les petites choses à retenir :

Pour spécifier un runner :

```
tags:
 - openfaas-runner
```

Pour n'exécuter les jobs lors d'un commit ou un push, uniquement sur la branche master :

```
only:
 - master
```

Maintenant, "publiez" votre fichier .gitlab-ci.yml vers votre **GitLab**

```
git add .
git commit -m "add ci file"
git push
```

Retournez dans la page des pipelines de votre instance **GitLab** http://gitlab-ce.test/k33g/hello-world-project/pipelines, vous pouvez noter qu'un nouveau pipeline a été créé : 8

Si vous sélectionnez ce pipeline, vous pourrez suivre son évolution en direct : 9

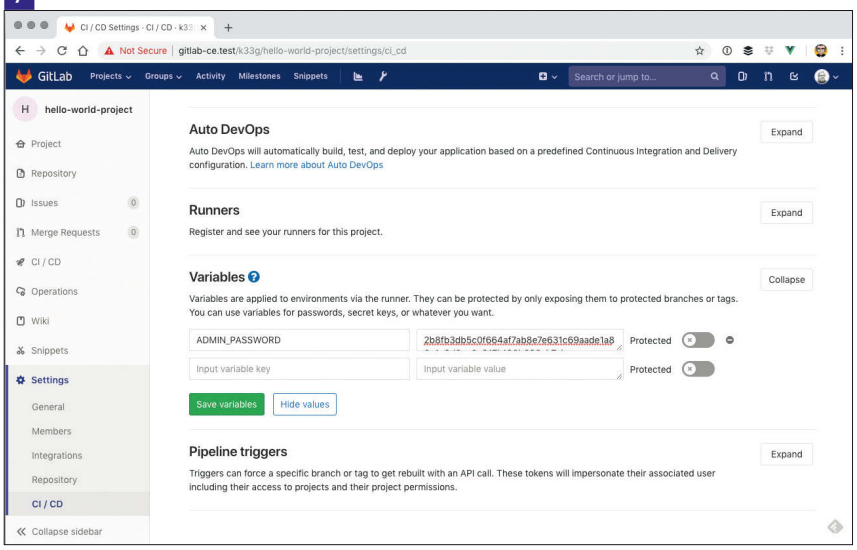
Et une fois que tout est au vert : 10

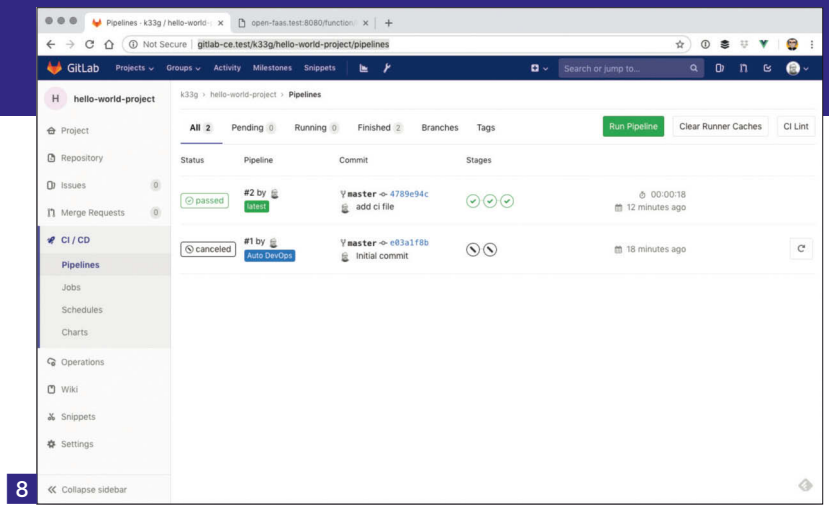
Allez sur la page environnements http://gitlab-ce.test/k33g/hello-world-project/environments : 11

Vous pouvez donc voir qu'un nouvel environnement a été créé, si vous cliquez sur le lien à droite, vous accèderez directement à la fonction. Ceci est permis grâce à la clé environment dans notre fichier .gitlab-ci.yml :

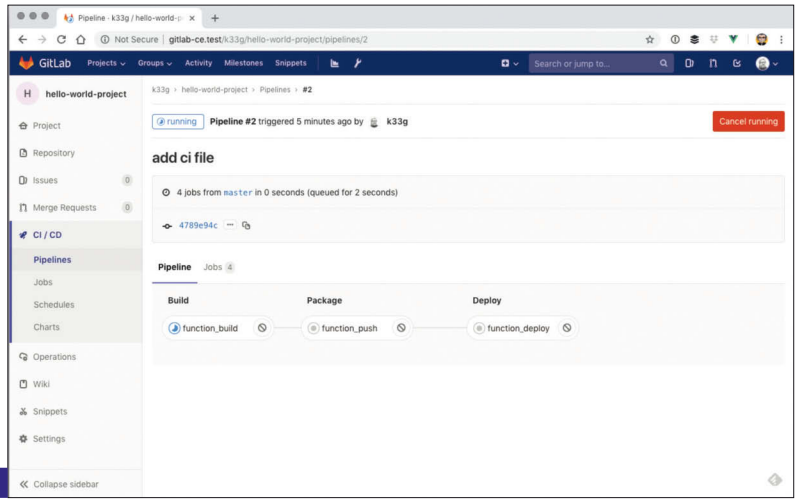
```
environment:
 name: production/hello-world
 url: $OPENFAAS_URL/function/hello-world
```

7

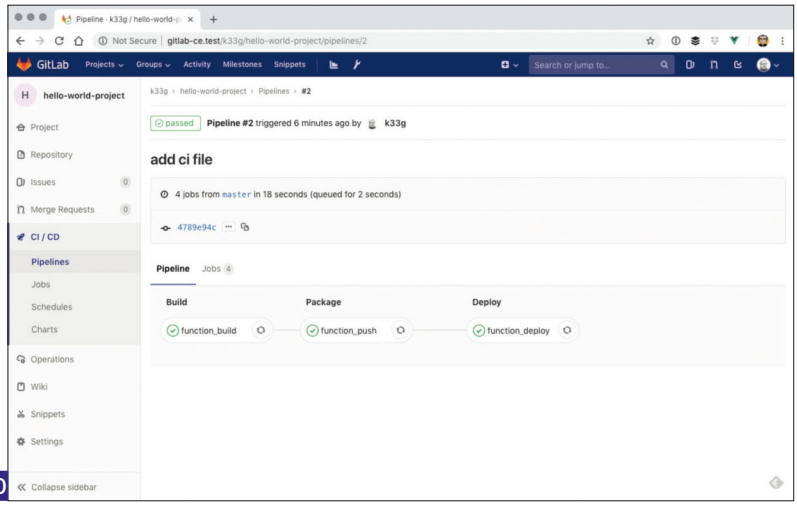




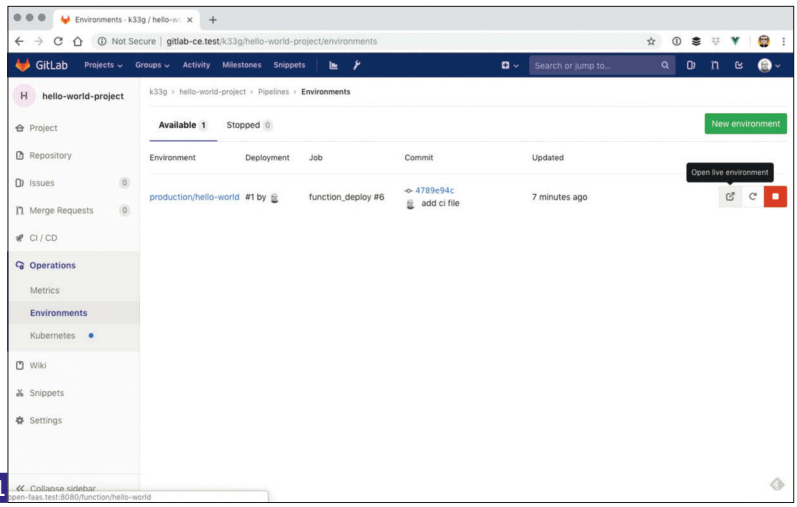
8



9



10



11

Maintenant à chaque fois que vous implémenterez une modification, votre fonction sera redéployée. Nous avons donc une 1ère version de déploiement continu pour notre fonction. Mais nous allons aller plus loin.

Review application des fonction OpenFaaS avec GitLab CI

Maintenant, je voudrais :

- pouvoir créer une branche pour faire des modifications (une "**feature branch**" ou "**topic branch**") ;
- et provisionner/déployer temporairement une nouvelle fonction **B** à partir de cette "**topic branch**" pour vérifier/tester mon développement ;
- sans toucher à ma fonction **A** qui tourne en production ;
- et enfin au moment où je merge mes modifications sur master, ma nouvelle fonction est redéployée à la place de la fonction **A** en production ;
- et la fonction de développement **B** temporaire est supprimée au moment du merge sur master et où l'on supprime la **topic branch**.

C'est ce que l'on appelle chez **GitLab** le principe de **Review Apps** (vous trouverez un peu plus de détail ici : https://docs.gitlab.com/ee/ci/review_apps/).

Préparation

Pour arriver à cela, nous devons modifier le fichier .gitlab-ci.yml en lui ajoutant quelques jobs.

Mais avant toute chose, nous allons créer un 2ème fichier de déploiement yaml preview-hello-world.yml avec ce contenu-la :

```
provider:
  name: faas
  gateway: http://open-faas.test:8080
functions:
  $FUNCTION_NAME:
    lang: node
    handler: ./hello-world
    image: registry.test:5000/$FUNCTION_NAME:latest
```

Je veux pouvoir à chaque **topic branch** avoir la possibilité de donner un nouveau nom à ma fonction et à l'image docker, comme par exemple hello-world-k33g-master-patch-75867, pour cela, je vais utiliser l'utilitaire linux envsubst (installé par défaut sous **Ubuntu**, et mon runner "tourne" dans une **Ubuntu**) pour remplacer dans le fichier preview-hello-world.yml la variable $FUNCTION_NAME par une valeur particulière (ex: hello-world-k33g-master-patch-75867), et créer temporairement un nouveau fichier yaml de déploiement avec un nom différent pour chaque **topic branch**, au hasard : hello-world-k33g-master-patch-75867.yml. Et pour donner un nom différent à chaque fois, je vais utiliser la "**GitLab CI variable**" CI_COMMIT_REF_NAME qui prend comme valeur le nom de la branche ou du tag à partir de laquelle/duquel est construit le projet (plus d'informations ici : https://docs.gitlab.com/ee/ci/variables/).

Donc nous obtiendrons un nouveau fichier yaml avec ce contenu :

```
provider:
  name: faas
  gateway: http://open-faas.test:8080
```

```
functions:
  hello-world-k33g-master-patch-75867:
    lang: node
    handler: ./hello-world
    image: registry.test:5000/hello-world-k33g-master-patch-75867:latest
```

Utiliser un cache

Les jobs **GitLab CI** sont par défaut immutables pour pouvoir être facilement rejoués. Du coup pour pouvoir conserver le fichier yaml (*.yml) généré d'un job à l'autre, nous devons utiliser un cache :

```
cache:
  key: "$CI_PIPELINE_ID"
  paths:
  # keep all yaml files between stages
    - ./*.yml
```

Remarque

Positionnez cette partie juste après la définition de la liste des étapes ou après la définition des variables (en début de fichier).

key: "$CI_PIPELINE_ID" permet de spécifier une clé pour le cache qui est unique pour le pipeline courant et qui permet de partager le cache entre les jobs du pipeline. La "**GitLab CI variable**" la plus appropriée est CI_PIPELINE_ID qui est donc l'id unique du pipeline qui changera à chaque nouveau pipeline et permettra donc de générer un nouveau cache.

Construire l'image de preview

C'est là que se passe la génération du nouveau fichier de déploiement :

```
preview_function_build:
  stage: build
  tags:
    - openfaas-runner
  except:
    - master
  script:
    - export FUNCTION_NAME="hello-world-$CI_COMMIT_REF_NAME"
    - echo -n $ADMIN_PASSWORD | faas-cli login --username=admin --password-stdin
    - envsubst < preview-hello-world.yml > hello-world-$CI_COMMIT_REF_NAME.yml
    - faas-cli build -f hello-world-$CI_COMMIT_REF_NAME.yml
```

- alors, tout d'abord, **à noter** l'utilisation du mot clé except avec la branche master, dans notre cas, notre job sera exécuté sur toutes les branches sauf la branche master ;
- ensuite avec cette commande export FUNCTION_NAME="hello-world-$CI_COMMIT_REF_NAME", je crée une nouvelle variable d'environnement FUNCTION_NAME avec un nouveau nom de fonction ;
- Et je crée un nouveau fichier yaml de déploiement avec la commande envsubst < preview-hello-world.yml > hello-world-$CI_COMMIT_REF_NAME.yml qui me permet de substituer la variable FUNCTION_NAME par sa valeur dans le fichier preview-hello-world.yml pour créer un nouveau fichier.

"Pousser" l'image de preview dans la registry

Dans cette partie, rien de spécial

```
preview_function_push:
  stage: package
  tags:
    - openfaas-runner
  except:
    - master
  script:
    - faas-cli push -f hello-world-$CI_COMMIT_REF_NAME.yml
```

Déployer la fonction (ou la Review App)

Cette fois-ci, j'ai 2 jobs : preview_function_deploy pour déployer la fonction et stop_preview_function pour supprimer la fonction :

```
preview_function_deploy:
  stage: deploy
  tags:
    - openfaas-runner
  environment:
    name: preview/hello-world-$CI_COMMIT_REF_NAME
    url: $OPENFAAS_URL/function/hello-world-$CI_COMMIT_REF_NAME
    on_stop: stop_preview_function
  except:
    - master
  script:
    - faas template pull
    - faas-cli deploy -f hello-world-$CI_COMMIT_REF_NAME.yml
```

Dans le job preview_function_deploy, la partie intéressante se trouve dans la section environment :

- je définis un nouveau nom d'environnement: name: preview/hello-world-$CI_COMMIT_REF_NAME (et j'aurais autant d'environnement que de **topic branches**) ;
- je lui affecte une nouvelle url : "url: $OPENFAAS_URL/function/hello-world-$CI_COMMIT_REF_NAME" ;
- et surtout j'ajoute ceci : "on_stop: stop_preview_function" qui me permettra au moment du merge lorsque je supprime la **topic branch** d'appeler le job stop_preview_function qui supprimera la fonction temporaire.

Et donc voici le job de suppression :

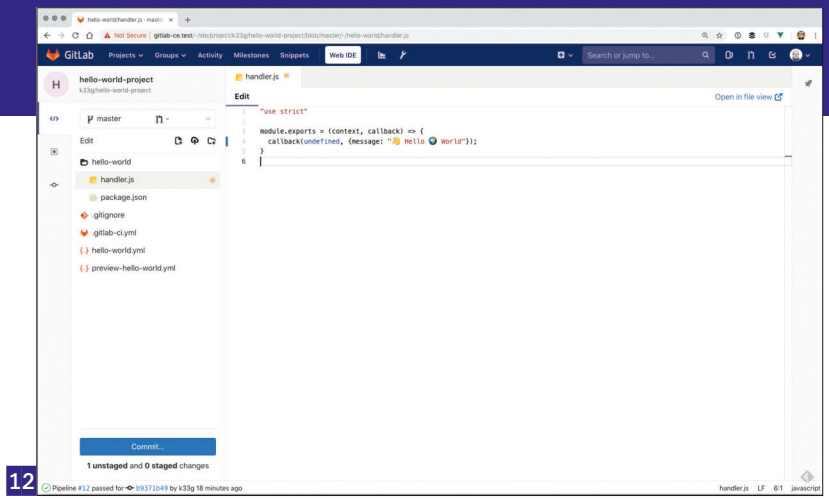
```
stop_preview_function:
  stage: deploy
  tags:
    - openfaas-runner
  script:
    - faas-cli remove -f hello-world-$CI_COMMIT_REF_NAME.yml
  except:
    - master
  when: manual
  environment:
    name: preview/hello-world-$CI_COMMIT_REF_NAME
    action: stop
```

Notez que le job est défini comme manuel avec when: manual pour ne pas être déclenché automatiquement durant le pipeline.
N'oubliez pas d'implémenter votre code et de le déployer sur votre instance **GitLab** :

```
git add .
git commit -m "review app"
git push
```

Et maintenant une Review App en image

Pour la suite, j'utilise la fonctionnalité "Web IDE" de GitLab, donc je vais tout faire directement dans l'IHM Web, mais vous pouvez très



12

bien faire autrement. Modifiez votre code : 12

Implémentez sur une nouvelle branche et créez une merge request : 13

Soumettez votre merge request : 14

Vous pouvez suivre directement l'évolution de votre pipeline dans la page de la merge request. Notez le bouton **View app** qui vous permet de visualiser votre application : 15

Voici donc votre **Review App**: 16

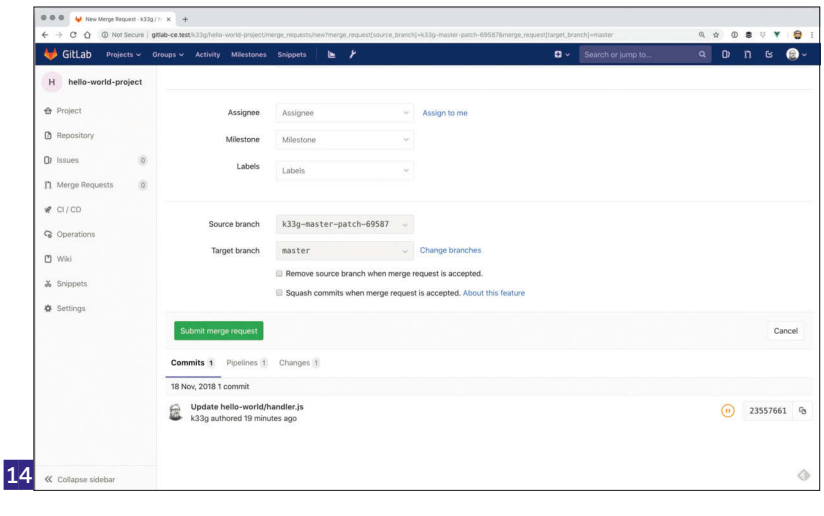
Si vous allez voir votre page environnements, vous noterez l'apparition d'un nouvel environnement : 17

Visualisez le pipeline (notez l'apparition du job de suppression) : 18

Une fois que vous mergez votre **topic branch** sur master, tout en supprimant votre branche, un nouveau déploiement (pipeline) va être déclenché pour redéployer la fonction de production ; vous pouvez vérifier dans les environnements que votre environnement de preview n'est plus là.

Et si vous "appelez" votre fonction de production, vous voyez que les modifications ont été prises en compte : 19

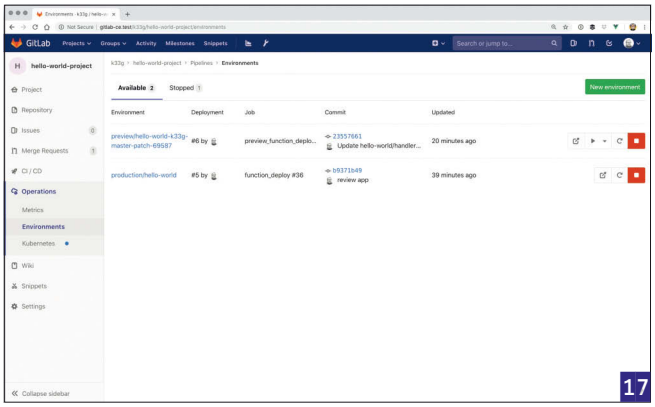
Voilà pour aujourd'hui. Comme vous avez pu voir, le concept de **Review App** est plutôt simple à mettre en œuvre, et surtout très utile dans le cadre d'un workflow de développement. La simplicité du concept des runners (ordonnancement de script shell) vous permettra d'adapter vos scripts de CI et de **Review App** facilement à vos plateformes de déploiement. •

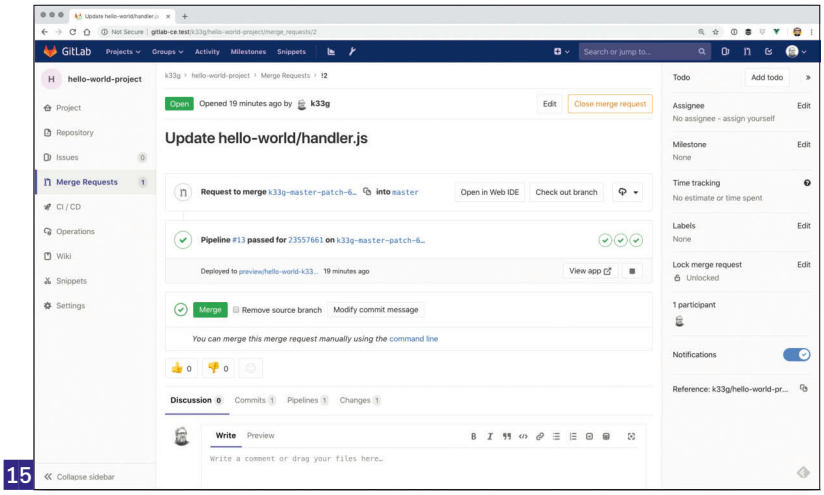


13

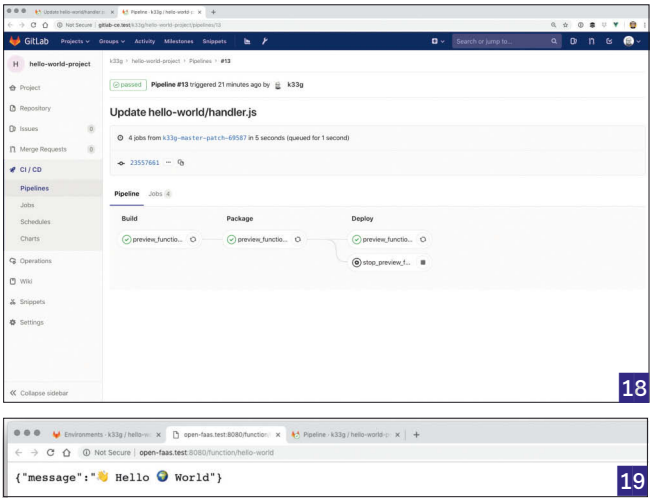
16

14


17



15



18

19

Jérôme GIACOMINI
Consultant Infinite Square
http://blogs.infinitesquare.com/
@jeromegiacomini

Tiny RestClient

niveau 100

Je suis très heureux de vous présenter un projet sur lequel je travaille depuis un petit moment : un client REST compatible .NET Standard entièrement fluent.

Le but principal est d'avoir un client REST optimisé et simple à utiliser. Il est compatible .Net Standard 1.3 et 2.0, ce qui signifie qu'il est utilisable sur le .Net Framework 4.6 ou supérieur, Xamarin et .Net Core et UWP.
Il masque la complexité inhérente à la communication entre votre application et le serveur REST.
Fonctionnalités :

- Un client moderne pour faire des appels asynchrones pour les API REST ;
- Support des principaux verbes HTTP : GET, POST , PUT, DELETE, PATCH ;
- Support des verbes personnalisés ;
- Support des "cancellation token" sur tout type de requêtes ;
- Détection automatique du désérialiseur à utiliser ;
- Support de la sérialisation/ désérialisation JSON et XML ;
- Support de sérialiseur / désérialiseur personnalisés ;
- Support du multi-part form (envoi multiples de fichiers) ;
- Appels http optimisés ;
- Exceptions typées plus faciles à interpréter ;
- Fournit une façon simple de logger toutes les requêtes : les réponses qui échouent, le temps de réponses... ;
- Possibilité de définir le timeout globalement ou par requête ;
- L'API lance une TimeoutException lorsque les requêtes sont en timeout (par défaut HttpClient lance une exception Operation CancelledException ce qui rend impossible de faire la différence entre une annulation du token et un timeout) ;
- Permet d'exporter les requêtes sous forme de collection Postman.

Créer un client

```
using Tiny.RestClient;
var client = new TinyRestClient("http://MyAPI.com/api", new HttpClient());
```

Headers

Ajouter un header par défaut pour toutes les requêtes

```
// Add default header for each calls
client.Settings.DefaultHeaders.Add("Token", "MYTOKEN");
```

Ajouter un header pour la requête courante

```
// Add header for this request only
client.GetRequest("City/All").
    AddHeader("Token", "MYTOKEN").
    ExecuteAsync();
```

Lire les headers de la réponse

```
await client.GetRequest("City/All").
        FillResponseHeaders(out headersOfResponse Headers).
        ExecuteAsync();
foreach(var header in headersOfResponse)
{
  Debug.WriteLine($"{current.Key}");
  foreach (var item in current.Value)
  {
    Debug.WriteLine(item);
  }
}
```

Créer une requête GET

```
var cities = client.GetRequest("City/All").ExecuteAsync<List<City>>();
// GET http://MyAPI.com/api/City/All an deserialize automaticaly the content

// Add a query parameter
var cities = client.
  GetRequest("City").
  AddQueryParameter("id", 2).
  AddQueryParameter("country", "France").
  ExecuteAsync<City>> ();
// GET http://MyAPI.com/api/City?id=2&country=France and deserialize automaticaly
 the content
```

Créer une requête POST

```
// POST
 var city = new City() { Name = "Paris" , Country = "France"};

// With content
var response = await client.PostRequest("City", city).
          ExecuteAsync<bool>();
// POST http://MyAPI.com/api/City with city as content

// With form url encoded data
var response = await client.
          PostRequest("City/Add").
          AddFormParameter("country", "France").
          AddFormParameter("name", "Paris").
          ExecuteAsync<Response>();
// POST http://MyAPI.com/api/City/Add with from url encoded content


var fileInfo = new FileInfo("myTextFile.txt");
var response = await client.
          PostRequest("City/Image/Add").
          AddFileContent(fileInfo, "text/plain").
```

```
        ExecuteAsync<Response>();
// POST text file at http://MyAPI.com/api/City/Add
```

Créer une requête avec un verbe http personnalisé

```
await client.
    NewRequest(new System.Net.Http.HttpMethod("HEAD"), "City").
    ExecuteAsync();
```

Définir le timeout

Définir un timeout global

```
client.Settings.DefaultTimeout = TimeSpan.FromSeconds(100);
```

Définir le timeout pour sur une requête

```
request.WithTimeout(TimeSpan.FromSeconds(100));
```

Télécharger un fichier

```
string filePath = "c:\map.pdf";
FileInfo fileInfo = await client.
        GetRequest("City/map.pdf").
        DownloadFileAsync("c:\map.pdf");
// GET http://MyAPI.com/api/City/map.pdf
```

Récupérer une réponse HttpResponse-Message brute

```
var response = await client.
        PostRequest("City/Add").
        AddFormParameter("country", "France").
        AddFormParameter("name", "Paris").
        ExecuteAsHttpResponseMessageAsync();
// POST http://MyAPI.com/api/City/Add with from url encoded content
```

Lire une réponse en tant que String

```
string response = await client.
        GetRequest("City/All").
        ExecuteAsStringAsync();
```

Effectuer des requêtes multipart

Créer des requêtes multipart est très simple avec le RestClient.

```
// With 2 json content
var city1 = new City() { Name = "Paris" , Country = "France"};
var city2 = new City() { Name = "Ajaccio" , Country = "France"};
var response = await client.NewRequest(HttpVerb.Post, "City").
await client.PostRequest("MultiPart/Test").
        AsMultiPartFromDataRequest().
        AddContent<City>(city1, "city1", "city1.json").
        AddContent<City>(city2, "city2", "city2.json").
        ExecuteAsync();
```

```
// With 2 byte array content
byte[] byteArray1 = ...
byte[] byteArray2 = ...

await client.PostRequest("MultiPart/Test").
        AsMultiPartFromDataRequest().
        AddByteArray(byteArray1, "request", "request2.bin").
        AddByteArray(byteArray2, "request", "request2.bin")
        ExecuteAsync();


// With 2 streams content
Stream1 stream1 = ...
Stream stream2 = ...
await client.PostRequest("MultiPart/Test").
        AsMultiPartFromDataRequest().
        AddStream(stream1, "request", "request2.bin").
        AddStream(stream2, "request", "request2.bin")
        ExecuteAsync();


// With 2 files content

var fileInfo1 = new FileInfo("myTextFile1.txt");
var fileInfo2 = new FileInfo("myTextFile2.txt");

var response = await client.
        PostRequest("City/Image/Add").
        AsMultiPartFromDataRequest().
        AddFileContent(fileInfo1, "text/plain").
        AddFileContent(fileInfo2, "text/plain").
        ExecuteAsync<Response>();

// With mixed content
await client.PostRequest("MultiPart/Test").
        AsMultiPartFromDataRequest().
        AddContent<City>(city1, "city1", "city1.json").
        AddByteArray(byteArray1, "request", "request2.bin").
        AddStream(stream2, "request", "request2.bin")
        ExecuteAsync();
```

Streams et byte[]

Vous pouvez faire des requêtes avec comme contenu un stream ou un bytes array.
Si vous utilisez ces méthodes aucun sérialiseur ne sera utilisé.

Streams

Lire une réponse de type Stream :

```
// Read stream response
Stream stream = await client.
        GetRequest("File").
        ExecuteAsStreamAsync();
```

Envoyer un contenu du type Stream :

```
// Post Stream as content
await client.PostRequest("File/Add").
        AddStreamContent(stream).
        ExecuteAsync();
```

byte[]

Lire une réponse de type byte[] :

```
// Read byte array response
byte[] byteArray = await client.
         GetRequest("File").
         ExecuteAsByteArrayAsync();
```

Envoyer un contenu du type byte[]

```
// Send bytes array as content
await client.
        PostRequest("File/Add").
        AddByteArrayContent(byteArray).
        ExecuteAsync();
```

Gestion des erreurs :

Les requêtes peuvent lancer 4 types d'exceptions :

- ConnectionException : lancée quand la requête ne peut pas atteindre le serveur ;
- HttpException : lancée quand la requête a atteint le serveur mais que le StatusCode est invalide (404, 500...) ;
- SerializeException : lancée quand le sérialiseur ne peut sérialiser le contenu ;
- DeserializeException : lancée quand le désérialiseur ne peut désérialiser la réponse ;
- TimeoutException : lancée lorsque la requête prend trop de temps à s'exécuter.

Rattraper un Status code spécifique

```
string cityName = "Paris";
try
{
  var response = await client.
    GetRequest("City").
    AddQueryParameter("Name", cityName).
    ExecuteAsync<City>();
}
catch (HttpException ex) when (ex.StatusCode == System.Net.HttpStatusCode.NotFound)
{
  throw new CityNotFoundException(cityName);
}
catch (HttpException ex) when (ex.StatusCode == System.Net.HttpStatusCode.Internal
ServerError)
{
  throw new ServerErrorException($"{ex.Message} {ex.ReasonPhrase}");
}
```

Formatters

Par défaut :

- Le formateur Json est utilisé par défaut ;
- Un formateur XML est ajouté dans la liste des formateurs supportés.

Chaque formateur a une liste de "Supported Media Types". Cela permet au RestClient de détecter quel formateur sera utilisé.
Si aucun formateur ne correspond, il utilisera le formateur par défaut.

Ajouter un nouveau formateur

Ajouter un formateur comme formateur par défaut.

```
bool isDefaultFormatter = true;
var customFormatter = new CustomFormatter();
client.Settings.Formatters.Add(customFormatter, isDefaultFormatter);
```

Supprimer un formateur :

```
var lastFormatter = client.Settings.Formatters.Where( f=> f is XmlSerializer>).First();
client.Remove(lastFormatter);
```

Définir un sérialiseur pour la requête courante :

```
IFormatter serializer = new XmlFormatter();
var response = await client.
  PostRequest("City", city, serializer).
  ExecuteAsync();
```

Définir un désérialiseur pour la requête courante :

```
IFormatter deserializer = new XmlFormatter();

var response = await client.
  GetRequest("City").
  AddQueryParameter("Name", cityName).
  ExecuteAsync<City>(deserializer);
```

Formateur personnalisé :

Vous pouvez créer votre propre formateur en implémentant l'interface IFormatter.
Voici par exemple l'implémentation du XMLFormatter :

```
public class XmlFormatter : IFormatter
{

  public string DefaultMediaType => "application/xml";

  public IEnumerable<string> SupportedMediaTypes
  {
    get
    {
      yield return "application/xml";
      yield return "text/xml";
    }
  }

  public T Deserialize<T>(Stream stream, Encoding encoding)
  {
    using (var reader = new StreamReader(stream, encoding))
    {
      var serializer = new XmlSerializer(typeof(T));
      return (T)serializer.Deserialize(reader);
    }
  }
```

```
public string Serialize<T>(T data, Encoding encoding)
{
    if (data == default)
    {
      return null;
    }

    var serializer = new XmlSerializer(data.GetType());
    using (var stringWriter = new DynamicEncodingStringWriter(encoding))
    {
      serializer.Serialize(stringWriter, data);
      return stringWriter.ToString();
    }
  }
}
```

Listener

Vous pouvez ajouter simplement un IListener pour "écouter" toutes les requêtes / réponses / exceptions reçues.

Debug Listener

Un listener Debug est fourni avec la librairie.
Pour l'ajouter, il suffit d'appeler la méthode AddDebug sur la propriété Listener.

```
client.Settings.Listeners.AddDebug();
```

Vous pouvez aussi créer votre propre listener en implémentant l'interface IListener.

Postman Listener

Pour ajouter un listener Postman vous avez à appeler la méthode AddPostman sur la propriété Listeners.

```
PostManListerner listener = client.Settings.Listeners.AddPostman("nameOfCollection");
```

Vous pouvez sauvegarder la collection Postman en appelant SaveAsync du listener Postman.

```
await listener.SaveAsync(new FileInfo("postManCollection.json");
```

Si vous voulez uniquement le Json de la collection, vous pouvez appeler la méthode GetCollectionJson

```
await listener.GetCollectionJson();
```

Ajout d'un listener personnalisé

```
IListener myCustomListerner = ..
client.Settings.Listeners.Add(myCustomListerner);
```

POUR CONCLURE

J'espère que cette librairie vous sera autant utile qu'à moi. N'hésitez pas à m'envoyer des idées ou à venir contribuer sur le projet pour l'enrichir. Happy coding. •

Le code est disponible sur Github :
https://github.com/jgiacomini/Tiny.RestClient
Et la librairie est sur Nuget :
https://www.nuget.org/packages/Tiny.RestClient/
Vous voulez en parler ? Passez me voir sur Gitter : https://gitter.im/Tiny-RestClient/Lobby

Valentin Koukponou
Consultant et formateur
à Zenika Paris

TensorFlow : démocratiser le Deep Learning

Quand Google dévoile une partie de son secret en novembre 2015 en rendant TensorFlow open source sous la licence Apache 2.0, un grand nombre de personnes s'y est naturellement mis. Sa popularité est justifiée de par son origine et de par sa portabilité, sa flexibilité et une large communauté (plus de 10 000 commits et 3 000 repos en un an).

C'est quoi exactement TensorFlow ?

TensorFlow est un système d'apprentissage automatique intégré à plusieurs produits du géant de Mountain View. Il se présente comme une bibliothèque dédiée au calcul numérique et en est à sa version 1.3.

Initialement, le but de TensorFlow était d'optimiser les calculs numériques complexes, mais aujourd'hui il est très connu pour résoudre des problèmes de Deep Learning. Toutefois, ce framework est assez général pour être utilisé à d'autres fins.

Les entreprises qui utilisent TensorFlow sont, entre autres, Google, DeepMind, OpenAI, Snapchat, Airbnb, Uber, Airbus, eBay, Dropbox. Aussi plusieurs projets sont menés sous ce framework. On peut citer le fameux Google Translate ou bien WaveNet de DeepMind qui génère des audios à partir de textes. Des radiologues utilisent aussi TensorFlow pour identifier des signes de Parkinson grâce aux analyses de scans.

Les deux composantes principales de TensorFlow sont : les Opérations et les Tensors.

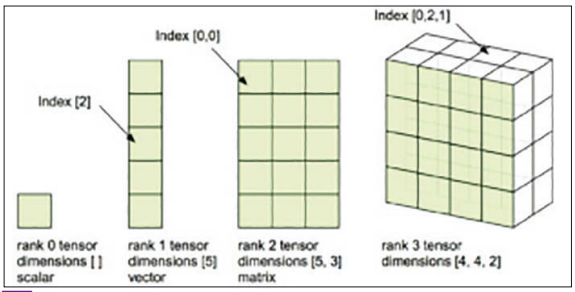
L'idée est de créer un modèle, un ensemble d'Opérations, puis d'alimenter ce modèle avec les données ou Tensors. Ces Tensors vont couler dans le modèle comme un flux (Flow): d'où le nom TensorFlow.

Dans cet article, nous allons détailler les quatre étapes importantes dans la philosophie TensorFlow à savoir : les «Tensors», les «Operations », les «Sessions» et les «Graphs». Puis avant d'aborder les réseaux de neurones dans les prochains numéros, nous finirons cet article en créant un modèle de régression simple sous TensorFlow. Pour la suite de l'article, vous pouvez l'installer pour Java, Go, Python ou C.

Les différents types de Tensors

- Un Tensor peut être vu comme un tableau de données (array) multidimensionnel à taille dynamique. Pour en créer un avec une valeur que l'on souhaite garder fixe, on utilise tf.constant() .
- Par contre, si on souhaite pouvoir modifier la valeur d'un Tensor, on le définit avec tf.Variable(). Les Variables au sens de TensorFlow sont utiles quand on souhaite maintenir l'état d'une variable durant une première exécution puis le modifier pour de prochaines exécutions.
- tf.placeholder() permet de définir un placeholder. C'est simplement une Variable dont la valeur sera spécifiée plus tard lors du run d'un calcul. Le but de ces "espaces réservés" est de pouvoir définir une série d'opérations plus ou moins complexes. Puis d'alimenter ces placeholders plus tard lors du lancement de la Session grâce au "feed dictionary" nommé feed_dict.

Il est important de noter la possibilité de transformer les données existantes sous Python (NumPy, Pandas, …) en tenseur par la méthode tf.convert_to_tensor.



1 *Quelques exemples de structures de tenseurs[/caption]*

Les «Operations»

En tant que bibliothèque de calculs numériques au même titre que Numpy, on peut effectuer plusieurs calculs sur TensorFlow comme le montrent les quelques exemples ci-dessous :

```
1 import tensorflow as tf
2 a=tf.constant([3,6])
3 b=tf.constant([2,2])
4 tf.add(a,b)
5 tf.add_n([a,b,b]) # a + b + b
6 tf.multiply(a,b)
```

Le tableau ci-dessous donne une idée non exhaustive, par groupe, des opérations possibles et imaginables via TensorFlow.

Operations groups	Operations
Maths	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal
Array	Concat, Slice, Split, Constant, Rank, Shape, Shuffle
Matrix.	MatMul, MatrixInverse, MatrixDeterminant
Neuronal Network	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool

Groupe d'opérations sur les tenseurs

Les Sessions

Définissons et affichons une chaîne de caractères hello via TensorFlow.

```
1 hello = tf.constant('Unkipple welcomes you to TensorFlow')
2 print(hello)
3 hello
```

Sortie :

```
1 Tensor("Const_2:0", shape=(), dtype=string)
```

Comme vous pouvez voir, le tenseur constant hello n'a pas été exécuté. Pour exécuter une opération, il est primordial d'ouvrir une Session grâce à la méthode Session.run(). Ce système d'exécution permet d'éviter de retourner dans du pur Python à chaque étape et

d'exécuter toutes les opérations d'un graphe en une seule fois dans un même backend optimisé.

```
1 with tf.Session() as sess:
2   print(sess.run(hello))
```

Sortie :

```
1 Unkipple welcomes you to TensorFlow
```

Toutefois, c'est un système assez contraignant, surtout quand on travaille en mode interactif, par exemple en ligne de commande ou avec des notebooks. Dans ce cas précis, il peut être plus agréable d'utiliser une InteractiveSession. On n'a plus qu'à utiliser la méthode .eval() sans passer par la Session.

```
1 sess = tf.InteractiveSession()
2 a = tf.constant(5.0)
3 b = tf.constant(6.0)
4 c = a*b
5
6 print("La multiplication de a par b donne : %i" % c.eval())
7 sess.close()
```

Sortie :

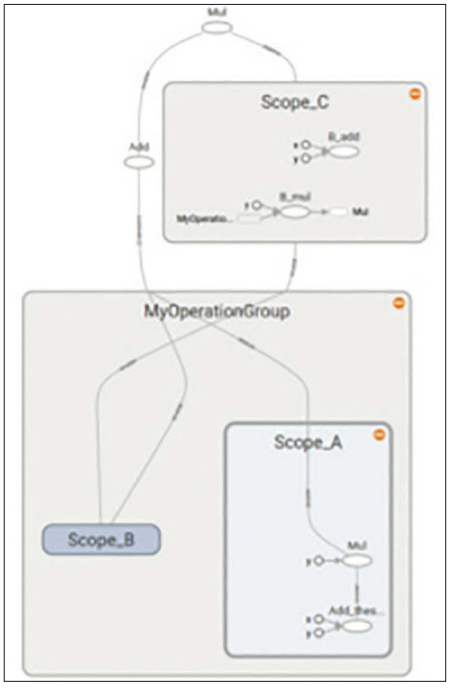
```
1 La multiplication de a par b donne : 30
```

L'exemple suivant montre l'utilisation d'une Session avec un placeholder :

```
1 # Créer un placeholder de type float 32-bit, vecteur à 3 éléments
2 a=tf.placeholder(tf.float32,shape=[3])
3 # Créer un vecteur constant à 3 éléments de type float 32-bit
4 b=tf.constant([5,5,5],tf.float32)
5 # Utiliser le placeholder comme une constante ou une variable
6 c=tf.add(a,b)  # On peut aussi faire simplement a+b
7 with tf.Session() as sess:
8   print(sess.run(c,{a:[1,2,3]})) # le second paramètre de run met en valeur les placehoders
```

Sortie :

```
1 [ 6. 7. 8.]
```



Le TensorBoard

TensorFlow nous permet de définir des opérations très complexes. TensorBoard est un formidable outil pour visualiser le graphe de ces opérations. Cet outil de TensorFlow permet aussi de debugger et d'optimiser les programmes. Notre prochain article lui sera exclusivement consacré.

2 *Exemple de visualisation de graphe via TensorBoard*

La structure du code

En général, les étapes d'un code TensorFlow peuvent être divisées en deux parties :

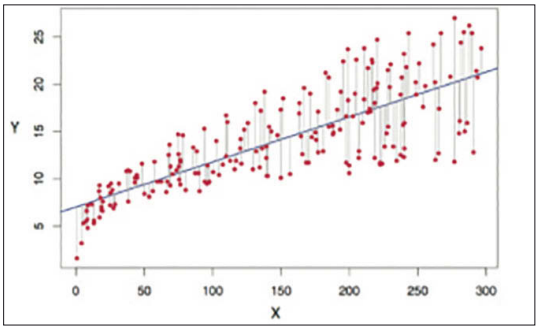
- construction et assemblage de toutes les opérations du graphe ;
- exécution des opérations sus-définies via une Session.

TensorFlow : Un premier modèle étape par étape

Dans la dernière partie, nous prenons, étape par étape, un exemple de régression linéaire simple avec TensorFlow. Une régression linéaire simple a pour but de prédire une variable continue Y grâce à une autre X via une relation linéaire de la forme Y = wX+b+e où w et b sont des coefficients à déterminer et e est le terme des erreurs.

Cela revient à faire passer une droite dans un nuage de points pour résumer au mieux ceux-ci. Autrement dit, cela revient à résoudre le problème suivant :

$$min_{w,b} \sum_{i=1}^{n} (y_i - b - wx_i)^2$$



3 *Cette figure représente la régression linéaire d'une variable X par la variable Y [1]*

Dans notre exemple, le but est d'expliquer le nombre de vols (cambriolages) Y à partir du nombre de départs de feu dans la ville de Chicago (que nous nommerons par la variable X.

Le jeu de données est disponible [2]

Phase 1 : construction du graphe

Etape 1 : charger les données depuis un fichier .xls

```
123456789101112 data_file = 'data/fire_theft.xls'
2
3 import xlrd
4 book = xlrd.open_workbook(data_file, encoding_override='utf-8')
5 sheet = book.sheet_by_index(0)
6
7 import numpy as np
8 data = np.asarray([sheet.row_values(i) for i in range(1, sheet.nrows)])
9 n=sheet.nrows - 1
10
11 import tensorflow as tf
12 n_samples = tf.constant(n,dtype=tf.float32)
```

Etape 2 : créer deux placeholders pour X (nombre de départs de feu) et Y (nombre de vols). Tous les deux de type float32

```
1 X = tf.placeholder(tf.float32, name='X')
2 Y = tf.placeholder(tf.float32, name='Y')
```

Etape 3 : créer les Variables poids w (pour weight) et biais b, et les initialiser à 0

```
12 w=tf.Variable(0.0,name='W')b=tf.Variable(0.0,name='b')
```

Etape 4 : construction du graphe du modèle de régression simple

```
1  Y_predicted= tf.add(tf.multiply(X, w),b)
```

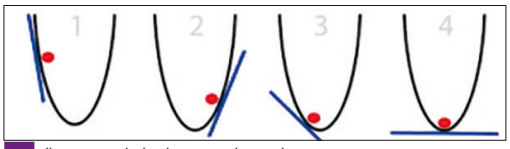
Etape 5 : utiliser le RMSE pour définir la fonction de coût
La qualité de l'ajustement est mesurée par plusieurs indicateurs. Dans notre cas, on fait le choix du RMSE (pour Root Mean Square Error) qui est défini par :

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{t=1}^{n} (\hat{y}_t - y_t)^2}{n}}.$$

Où y^t est la prédiction et yt l'observation. Cela représente la somme des longueurs (élevées au carré) des traits verticaux sur la fig 2.

```
1  loss=tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(Y,Y_predicted))))
```

Etape 6 : utiliser la descente de gradient pour minimiser la fonction de coût avec un learning_rate à 0.001
La descente de gradient est une méthode itérative d'optimisation [3]. Ici, elle est utilisée pour minimiser les erreurs commises par notre droite. À chaque itération, on choisit la meilleure pente sur notre fonction de coût pour se diriger itération après itération vers le minimum de notre fonction de coût.
Notons que le learning_rate représente la vitesse à laquelle nous souhaitons terminer nos itérations.



4 *illustration de la descente de gradient.*

```
1  optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001).minimize(loss)
```

Il existe beaucoup d'optimiseurs sous TensorFlow dont :
- tf.train.GradientDescentOptimizer
- tf.train.AdagradOptimizer
- tf.train.AdamOptimizer
- …

Phase 2 : l'entraînement du modèle

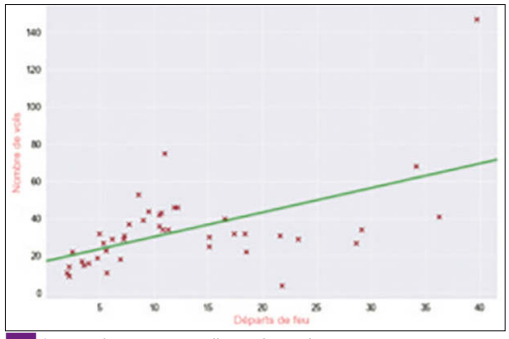
On recherche les meilleures valeurs de w et b qui minimisent notre fonction de coût.
Pour cela, on itère 1000 fois sur l'exécution de l'optimiseur. Toutes les 50 itérations, on affiche la valeur de la fonction de coût et les résultats du modèle à cette étape.

```
12345678910111213141516 display_step = 50
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for epoch in range(1000):
    for x, y in data:
      sess.run(optimizer, feed_dict={X: x, Y: y})
    if (epoch+1) % display_step == 0:
      _, c = sess.run([optimizer, loss], feed_dict={X: x, Y:y})
      print "Epoch:", '%04d' % (epoch+1), "cost=", c, \
        "W=", sess.run(w), "b=", sess.run(b)
 print "Optimization Finished!"
 training_cost = sess.run(loss, feed_dict={X: x, Y: y})
 print "Training cost=", training_cost, "W=", sess.run(w), "b=", sess.run(b), '\n'
```

Sortie :

```
12345678910111213141516171819202l
Epoch: 0050 cost= 8.97787 W= 1.89358 b= 0.956991
Epoch: 0100 cost= 8.06919 W= 1.90475 b= 1.81203
Epoch: 0150 cost= 7.48993 W= 1.84773 b= 2.66499
Epoch: 0200 cost= 6.58134 W= 1.85891 b= 3.51993
Epoch: 0250 cost= 6.00211 W= 1.80189 b= 4.37287
Epoch: 0300 cost= 5.12279 W= 1.81447 b= 5.19181
Epoch: 0350 cost= 4.68153 W= 1.76245 b= 5.88276
Epoch: 0400 cost= 4.05922 W= 1.75023 b= 6.56371
Epoch: 0450 cost= 3.43692 W= 1.73801 b= 7.24466
Epoch: 0500 cost= 3.12123 W= 1.674 b= 7.86761
Epoch: 0550 cost= 2.57061 W= 1.66638 b= 8.45479
Epoch: 0600 cost= 2.09304 W= 1.64437 b= 9.03802
Epoch: 0650 cost= 1.88439 W= 1.59176 b= 9.49921
Epoch: 0700 cost= 1.98742 W= 1.47755 b= 9.94439
Epoch: 0750 cost= 2.09045 W= 1.36334 b= 10.3896E
poch: 0800 cost= 1.9766 W= 1.29473 b= 10.8327
Epoch: 0850 cost= 1.74765 W= 1.26052 b= 11.2259
Epoch: 0900 cost= 1.52896 W= 1.23751 b= 11.555
Epoch: 0950 cost= 1.31027 W= 1.2145 b= 11.8842
Epoch: 1000 cost= 1.09158 W= 1.19149 b= 12.2133
Optimization Finished!Training cost= 1.06754 W= 1.19149 b= 12.2133
```

On peut observer la décroissance de la fonction de coût au fil des itérations successives.



5 *Nuage de points et meilleure droite de régression*

Existe-t-il vraiment une relation entre le nombre de vols et les départs de feu à Chicago ? Si oui, cette relation est-elle linéaire ? Est-ce que ce modèle peut être généralisé sur de nouvelles données ? Et le sur-apprentissage dans tout ça ? Voici quelques-unes des questions qu'on peut se poser.

•

Références

[1] Introduction to Statistical Learning with Applications in R, Trevor Hastie et al.
[2] Source du jeu de données
[3] Descente de gradient

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.

Créer une application utilisant OpenStreetMap avec le mkframework

niveau 100

Vous avez pu voir dans les derniers numéros de Programmez l'intérêt que pouvait représenter OpenStreetMap. En effet, depuis le changement de politique tarifaire de Google Maps, il devient important de trouver des solutions alternatives. Vous allez pouvoir apprécier ici la facilité d'utiliser ce projet de cartographie Opensource avec le mkframework.
Pour cet article, nous créerons un simple site de voyage qui permettra de visualiser des monuments de Paris.

Télécharger et installer le framework

Rendez-vous à l'adresse du site http://www.mkframework.com, dans la partie téléchargement et télécharger le « package contenant l'ensemble pour démarrer ».
Désarchivez-le dans le répertoire web et ouvrez votre navigateur à l'adresse http://localhost/mkframeworkPackage-1.0.XX (en fonction de votre version)
Vous voyez plusieurs répertoires :

- Builder (qui contient désormais le générateur web) ;
- Lib (qui contient la librairie du framework) ;
- Projects (qui contiendra les projets créés).

Créons la base de données

Dans une base de données Mysql, nous pouvons créer une table Monuments avec le code SQL suivant :

```
CREATE TABLE `tuto`.`Monuments` (

`id` INT NOT NULL AUTO_INCREMENT ,
`title` VARCHAR(100) NOT NULL ,
`address` VARCHAR(100) NOT NULL ,
`zip` VARCHAR(50) NOT NULL ,
`city` VARCHAR(50) NOT NULL,

PRIMARY KEY (`id`)


) ENGINE = InnoDB;
```

Créons l'application

Ouvrez votre navigateur à l'adresse du framework téléchargé, cliquez sur le répertoire Builder pour afficher le générateur web. 1

Note *Si vous avez un message rouge vous indiquant un souci d'écriture sur Projects, il vous faudra donner les droits afin que le builder puisse écrire dedans.*

Sélectionnez le type de template à utiliser pour créer cette application : Software Craftsmanship, application bootstrap, et générez.

Architecture de notre application

Prenons le temps de rapidement observer l'application créée 2.

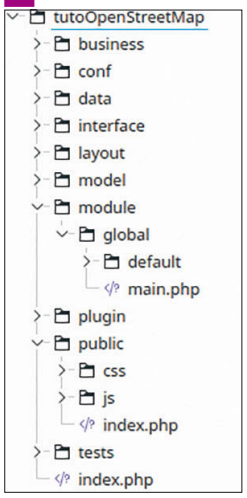
C'est un projet s'appuyant sur un template orienté architecture hexagonale, pour cela vous avez un répertoire contenant les classes métier (business) ainsi que les interfaces des dépendances de celles-ci.
Vous avez ensuite un répertoire contenant la configuration, les layouts, un autre les couches modèles, les modules, et, enfin un dernier contenant le répertoire public, aussi appelé « webroot » c'est-à-dire la racine de votre application web (à paramétrer ainsi dans votre configuration apache).

Paramétrage de notre fichier de configuration

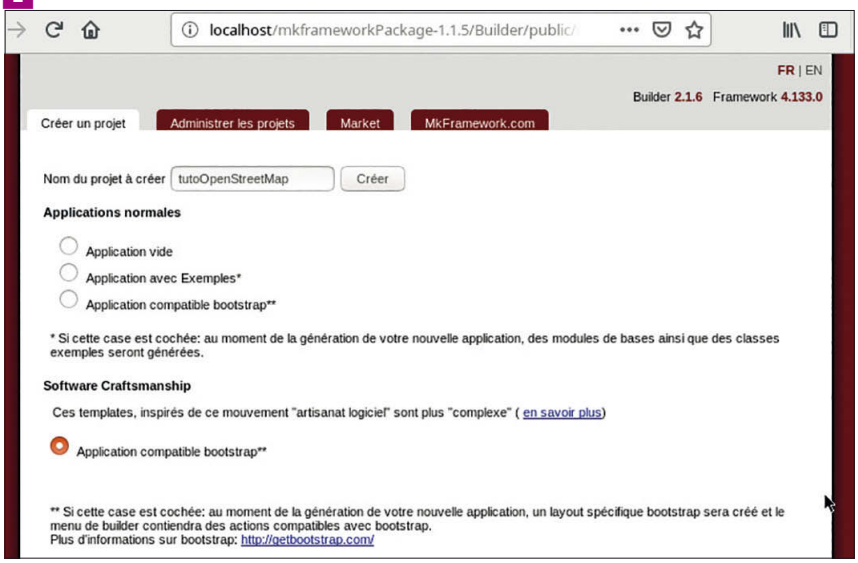
Il y a plusieurs fichiers de paramètres, modifions ici le fichier conf/connexion.ini.php afin de définir la base de données à utiliser.

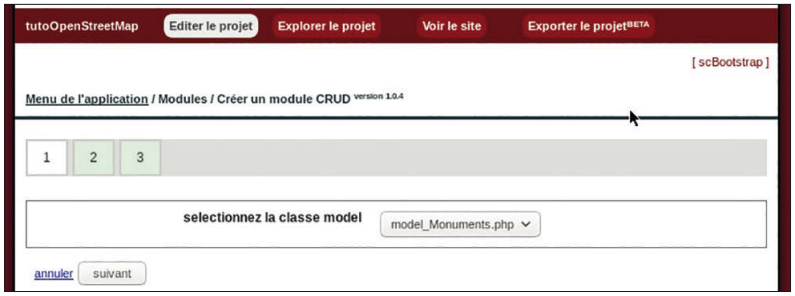
```
;<?php die()?>
[db]
tuto.dsn="mysql:dbname=tuto;host=localhost"
tuto.sgbd=pdo_mysql
tuto.username=root
tuto.password=root
```

2

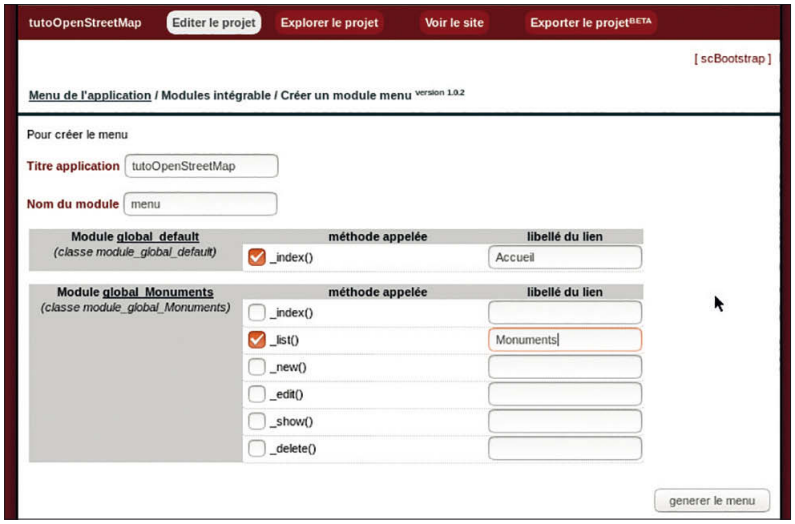


1

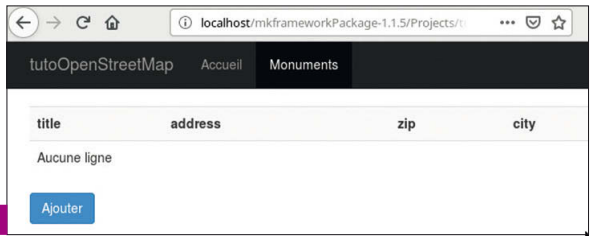




3



4



5

Génération de la couche modèle

Le framework adopte une architecture MVC (Modèle/Vue/Contrôleur). Il faut donc créer des classes pour interagir avec la base de données. Dans cette optique, le builder vous permet de les générer en quelques clics.

Vous aurez une classe model_Monuments.php de ce type :

```
<?php
class model_Monuments extends abstract_model implements interface_model{

    protected $sClassRow='row_Monuments';

    protected $sTable='Monuments';
    protected $sConfig='tuto';

    protected $tId=array('id');

    public static function getInstance(){
        return self::_getInstance(__CLASS__);
    }

    public function findById($uId){
        return $this->findOne('SELECT * FROM '.$this->sTable.' WHERE id=?',$uId );
    }
    public function findAll(){
        return $this->findMany('SELECT * FROM '.$this->sTable);
    }

}
```

Génération du module CRUD pour la table Monuments

Pour rappel, la génération d'un CRUD signifie la création de formulaires de création/modification/suppression (Create / Update / Delete) ainsi que d'un tableau listant les enregistrements (Read) d'une table.

Le Buider vous facilite cette génération via son interface web. 3

Dans le cadre de l'architecture hexagonale, le Builder vous générera :

- une classe « business » ;
- un module ;
  - avec les vues associées,
  - et un répertoire contenant les fichiers de langue.
- des tests unitaires associés.

Ajout d'un menu à l'application

Le builder permet de vous générer beaucoup de choses pour commencer et vous économiser du temps de développement.

Profitons-en pour ajouter un menu à notre application. 4

Une fois généré, il vous indique d'ajouter le code suivant dans votre module

```
$this->oLayout->addModule('menu','menu::index');
```

Ce template d'application gère la notion d'héritage, vous pouvez donc créer des modules héritant d'un autre module, ici le module « global ».

Editez donc le fichier module/global/main.php

```
<?php
class module_global extends abstract_module {

    protected $oLayout;

    public function before() {
        $sLang = _root::getConfigVar('language.default');
        if (isset($_SESSION['lang'])) {
            $sLang = $_SESSION['lang'];
        }
        _root::getI18n()->load($sLang);
        _root::getI18n()->loadFromDir(_root::getConfigVar('path.module').$this->_sModulePath.'/i18n/');

        $this->oLayout = new _layout('bootstrap');

        $this->oLayout->addModule('menu','menu::index');
    }
```

Vous aurez ensuite le menu intégré naturellement à votre application 5

Passons à OpenStreetMap

Le cœur de l'application installée, passons à l'ajout de notre module de cartographie.

Rendez-vous sur le site du framework, rubrique télécharger > modules, vous verrez une liste de modules disponibles dont « openStreetMap ».

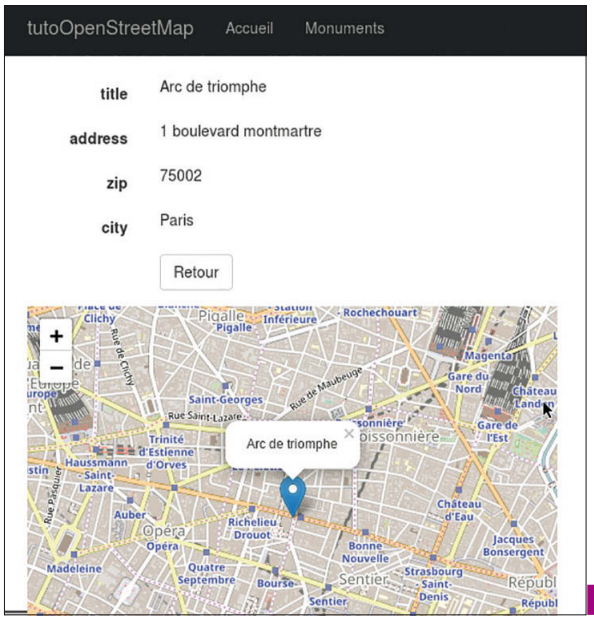
Une fois téléchargé, désarchivez-le et déplacez le module dans votre répertoire module, et le contenu du répertoire « public » dans votre répertoire public (fichier css, et js).

Pour l'utiliser, nous allons ajouter dans la page de visualisation d'un monument l'affichage d'une carte indiquant l'emplacement de celui-ci.

Dans le fichier module/global/Monuments/main.php, méthode d'affichage :

```
public function _show() {
        $oMonuments = model_Monuments::getInstance()->findById(_root::getParam('id'));

        $oView = $this->getView('show');
        $oView->oMonuments = $oMonuments;

        $this->oLayout->add('main', $oView);

        $oModuleOpenStreetMap=new module_openstreetMap();
        $oModuleOpenStreetMap->setWidth(500);
        $oModuleOpenStreetMap->setHeight(400);
        $oModuleOpenStreetMap->setZoom(14);

        //un pointer avec un peu de contenu html lorsque l'on clique dessus
        $oModuleOpenStreetMap->addPosition(
            $oMonuments->address.', '.$oMonuments->zip.' '.$oMonuments->city,
            $oMonuments->title);

        $this->oLayout->add('main',$oModuleOpenStreetMap->getMap());
    }
```

6



6

Conclusion

Vous avez pu voir dans cet article qu'il était très simple d'implémenter une carte openStreetMap avec le mkframework. En effet le framework propose via son système de modules un moyen de capitaliser sur vos développements. Par exemple ici, pour implémenter une carte nous utilisons un module php, ce qui permet de proposer des méthodes simples pour pouvoir utiliser facilement cette librairie.

Ici, il est donc très simple pour le développeur d'ajouter OpenstreetMap sans même imaginer la complexité de l'affichage d'une carte centrée sur une adresse, en ayant au préalable récupéré la longitude/latitude de l'adresse via un autre service.

C'est toute l'idée des modules du mkframework depuis 2009 : investir du temps à chercher, comprendre une technologie pour facilement la rendre utilisable par d'autres développeurs de l'entreprise ou soi-même dans d'autres projets.

C'est le cas également pour Gurrido, Datatables…

N'hésitez pas à vous appuyer sur ce principe pour développer vos propres modules. •

Yannick Grenzinger

David Panza

Du monolithe aux micro services

Partie 2

OU CE QUE NOUS AURIONS AIMÉ SAVOIR AVANT DE NOUS LANCER

Dans cet article, l'objectif est de vous faire un retour d'expérience sur notre passage de deux applications "monolithiques" à une architecture micro-services en se concentrant sur ce que nous aurions aimé savoir avant de nous lancer.

***Note importante :* nos notes sont des articles à rechercher dans votre moteur de recherche favori.**

niveau 200

3ÈME AXE : La collaboration

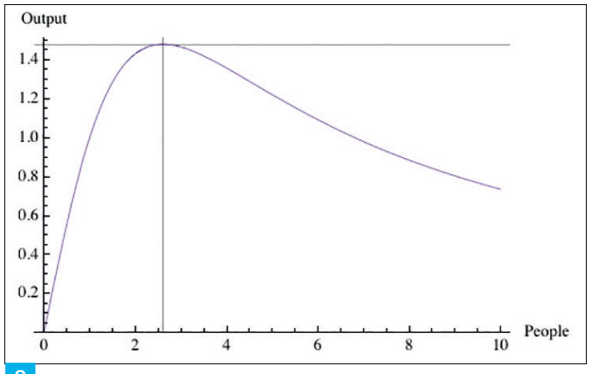
Les dérives d'une organisation

Développer une plateforme micro-services exige non seulement une rigueur technique mais aussi une organisation adaptée. Il est tout à fait naturel de répartir le développement d'une plateforme micro-services entre plusieurs équipes, étant donné l'éclatement des responsabilités et la séparation du code. C'est l'un des gros avantages de ce type d'architecture : de pouvoir grandir très rapidement. Cependant et contrairement aux aspects techniques, **l'organisation reste l'un des plus gros dangers**.

Vous connaissez sûrement cette expression : *"Neuf femmes ne font pas un enfant en un mois"*. Une expression qui nous vient de *The Mythical Man Month* de *Fred Brooks* paru en 1975. L'auteur explique les dérives causées par une mauvaise interprétation de l'unité de coût de développement : l'homme-mois. Ainsi, **doubler le nombre de développeur ne réduira pas le temps de développement de moitié**. 8

Et pour cause, la formation de nouveaux arrivants, les différentes réunions, la redescente d'information par des lignes de communication plus lentes sont des facteurs de ralentissement. Et donc **ajouter des personnes à un projet déjà en retard accentuera son retard !** 9

De plus, la manière dont l'organisation communique aura un impact significatif sur le code produit. Ce fait est expliqué par la loi de Conway qui dit que **la structure de communication sera inévitablement répliquée dans le code**. Ainsi, si deux équipes produisent une plateforme, il est fort probable que celle-ci soit divisée en deux sous-systèmes différents assez hétérogènes !



8 *Courbe de productivité, Modeling the Mythical Man-Month*

L'organisation des équipes

Sur notre plateforme, il existe deux types d'équipes ayant des motivations différentes : les équipes dites features et les équipes dites composants.

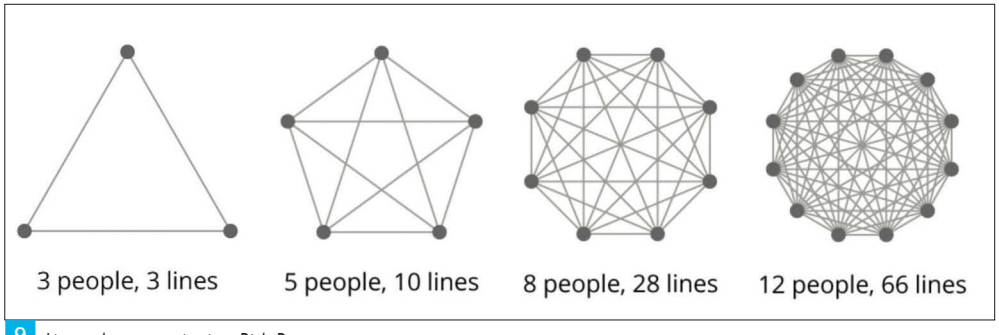
La **feature team** a pour objectif de développer les fonctionnalités d'une plateforme. Elle opérera sur plusieurs micro-services afin de concevoir au mieux le besoin. A l'image d'un micro-service, cette équipe est **autonome** et nécessite une composition avec des rôles variés : développeurs, testeurs, designer, responsable de produit, voire même un manager.

Ainsi, cette équipe sera parfaitement **alignée avec le besoin métier**. Attention toutefois à ne pas dépasser une certaine taille pour ne pas perdre en efficacité ! Le concept de pizza team illustre bien ce fait : une taille suffisante afin qu'on puisse nourrir une équipe avec deux pizzas, à savoir 8 personnes maximum.

A l'inverse, les **component teams** seront spécialisées sur un composant ou un rôle bien spécifique. Elles auront pour avantage de **maîtriser leur périmètre** et d'effectuer des changements ou tâches rapidement. Elles peuvent intervenir sur différents périmètres comme le support, un outil d'intégration et de déploiement continu ou encore la maintenance des serveurs. Cela peut faire sens lorsque le composant est **centralisé et standard** à tout une plateforme.

Néanmoins, une question se pose concernant la maintenance d'un micro-service. Si l'on considère qu'une feature team peut modifier n'importe quel **micro-service, à qui revient la responsabilité de les maintenir, en termes de design ou en cas de problème**. Cette question a suscité beaucoup de débats sur notre projet.

La première tentative a été de donner la responsabilité d'un micro-service à l'équipe l'ayant initié, ce qui est problématique



9 *Lignes de communication, Rich Rogers*

lorsque l'équipe ou ses membres d'origines ne sont plus là.
La solution la plus intéressante est issue de l'open source avec **le concept de Trusted/Untrusted Commiters**. Ainsi, les responsables d'un micro-service sont ceux qui l'ont initié ou y travaillent le plus et deviennent garants du design, de la qualité et de l'intention de départ. Quant aux autres équipes, tout développement passera par un mécanisme de validation (Pull Request) visant à revoir le code par l'équipe en charge du micro-service.

Le partage de connaissances

Le développement logiciel est un apprentissage du métier auquel il répond, sur son environnement et sur son organisation. Pour nous, concevoir une architecture micro-services est une aventure où **les échecs comme les succès font partie intégrante de notre processus de création**. Et comme le dit si bien Alberto Brandolini :

> *"Software development is a learning process, working code is a side effect"*

Notre architecture a été conçue progressivement et de manière émergente. Certaines tentatives ont été fructueuses, d'autres non. Et afin de ne pas reproduire les mêmes erreurs au fil du temps, il est primordial de **capitaliser sur ces expériences**. Tous nos choix techniques et organisationnels sont motivés par un besoin spécifique dans un contexte temporel, social, technique et politique. Mais il arrive à un moment où ces solutions ne sont plus adaptées, dû à l'évolution du projet ou de la technique. Ainsi, les "Architecture Decision Record[1]" (ou **ADR**) peuvent vous aider à **documenter ces choix** pour les justifier à vos successeurs.
Une autre pratique afin de diffuser la connaissance est de **sacraliser des ateliers ou rassemblements**. Dans notre projet, le "tech chapter" est un point récurrent qui vise à **partager les problématiques** de la plateforme, **des évolutions ou encore les expérimentations**. Les sujets abordés lors de cette réunion sont décidés en amont par les votes des membres, via un GitHub. Ainsi, toutes nos décisions stratégiques sont présentées et aboutissent le plus souvent à des standards partagés par tous.
Par le biais des **standards**, vous allez pouvoir **garantir la cohérence** de votre système. Car certes un micro-service en lui-même est une application autonome avec ses propres problématiques et briques techniques. Néanmoins certains standards doivent être posés notamment aux frontières des services. Par exemple, la manière de communiquer via un seul et même broker de message type RabbitMQ, la manière de gérer la configuration ou encore le monitoring des services. Autant de standards qui limiteront les points de friction avec d'autres équipes.
Enfin, tous ces standards peuvent vous servir à **constituer une sorte de guide ou charte du micro-service** et peut vous aider à juger de la conformité extrinsèque de celui-ci. C'est en quelque sorte un guide du "*bon citoyen*" qui stipule tous les prérequis en termes de fonctionnement et de qualité. Et afin d'appliquer cette charte, des agents de conformité automatisés ou non peuvent assurer la conformité d'un micro-service. Bien entendu, la revue de code ou le pair programming sont des moyens simples et efficaces d'assurer le respect de cette charte.

Une dynamique d'équipe

Par dynamique nous entendons la synergie entre développeurs quant à l'**harmonisation des pratiques**. Et lorsqu'une plateforme est composée de plusieurs équipes, ces pratiques ont **de fortes chances d'être hétérogènes**. Il ne s'agit pas ici d'imposer des règles de code ou encore un langage bien particulier. Pour nous, cela se concentre davantage sur la démarche de conception ou encore sur les paradigmes de programmation.
Par le biais de kata, nous maintenons une veille technologique par de simples exercices qui ont lieu le midi. Ces ateliers ont lieu de manière récurrente et sont sur la base du volontariat. Bien sûr, l'idéal serait de tenir ces ateliers pendant les heures de travail car faisant partie intégrante du développement, au même titre qu'une nouvelle fonctionnalité. On peut y découvrir un nouveau langage, une librairie, un framework ou de nouvelles manières de pratiquer le TDD par exemple.
Le binômage (ou pair programming) aide à maintenir la cohérence technique et la qualité d'un développement. Cette pratique peut être systématique pour chaque développement à entreprendre. De notre côté, nous avons choisi de l'appliquer uniquement lorsque la fonctionnalité à développer dépasse une certaine complexité. Et dans certains cas, pour un POC ou encore un nouveau micro-service, nous effectuons des ***"mob programming"***[2] réunissant toute une équipe autour d'un même écran, où le clavier change de main à intervalle régulier. Ces ateliers sont très efficaces afin d'attaquer un problème. Attention toutefois à bien les préparer, car ils réunissent plusieurs développeurs au même endroit et au même moment, et peuvent se transformer en des sessions de brainstorming sans fin !

4ÈME AXE : La production

Nous abordons le 4ème et dernier axe de maturité : la production. C'est celui qui vous offrira le plus de défis dans votre transition d'une architecture monolithique à des micro-services, surtout si vous brûlez les étapes.

Observabilité

Dans notre transition vers des micro-services, nous avons commencé par déployer nos applications sur deux instances (machine virtuelle) existantes histoire d'assurer une résilience. Cependant cette situation ne doit être que transitoire. Car afin d'assurer une meilleure résilience de votre plateforme, le mieux est d'avoir une VM par instances de services (principe d'autonomie). L'idée d'une architecture micro-services est aussi d'**avoir un système résilient à la panne**. L'autre avantage est de pouvoir mettre à l'échelle de façon transparente. Dans notre cas, nous sommes passés sur un cloud privé plus flexible et élastique.
Mais à ce moment, la production va rencontrer sa première vraie difficulté. En cas d'erreurs, il n'est plus question d'aller parcourir chaque fichier de log en se connectant sur chaque instance. Il va falloir mettre en place dès le départ une solution permettant d'explorer vos logs de façon

(1) Voir Architecture Decision Record de Mickael Nygard

(2) Voir Mob Programming, Woody Zuill, GOTO 2017

centralisée. On pense bien sûr à la suite Elastic mais cela peut être Splunk, Graylog ou des solutions SaaS comme Logmatic. 10 Au passage, de vos instances, vous devriez très rapidement remonter des métriques comme la charge CPU, mémoire, réseau, du système de fichiers, les temps de réponses de vos APIs ou encore la fréquence de passage du Garbage Collector par exemple dans le cadre d'une application Java. Pour cela, vous pourrez utiliser des agents (Metricbeat, Collectd, Telegraf…) ou directement envoyer les métriques par l'application (Spring Actuator permet de faire ça facilement). Ces métriques peuvent représenter une volumétrie importante (surtout si vous cherchez à avoir du temps réel) et il sera peut-être utile de passer sur des bases de données adaptées appelées time-series (Prometheus, InfluxDB…).

Maintenant vous pouvez mettre à jour vos tableaux de bord (quoi ? vous n'en aviez pas ?) pour **surveiller, en temps réel, votre système**. Vous pouvez installer un écran large dans l'open space en faisant attention à sélectionner les informations essentielles (nombre d'erreurs, de connexions ou encore la santé de vos builds et déploiements). Grafana est l'outil parfait pour ce besoin et il permet de sélectionner de multiples sources de données (par exemple ElasticSearch et Prometheus). 11

Alerte, on a une erreur en prod! 12

Les tableaux de bord représentent un bon moyen de suivre l'état de votre système, à condition de les avoir sous les yeux en permanence ! **L'idéal est donc d'avoir des alertes**. Celles-ci peuvent être délivrées par e-mail, sur vos channels Slack voire SMS pour les plus urgentes. Des outils comme ElastAlert ou Watcher peuvent ainsi se connecter à ElasticSearch et vous permettre de définir différentes alertes. Vous pouvez, par exemple, définir une alerte :

- en cas d'erreur ;
- si la fréquence d'un événement diminue fortement. Par exemple, si le nombre de connexions par minute diminue fortement par rapport à la normalité, il se peut que vos utilisateurs aient du mal à se connecter et donc indique un problème sur votre plateforme ;
- si la fréquence d'un évènement augmente fortement (spike). Par exemple, si le nombre de GC par minute augmente fortement, peut-être que votre application a une fuite mémoire et va finir en Out of Memory ;
- Si une valeur change entre deux évènements, comme les changements d'email pour vérifier qu'il n'y a pas une tentative de piratage.

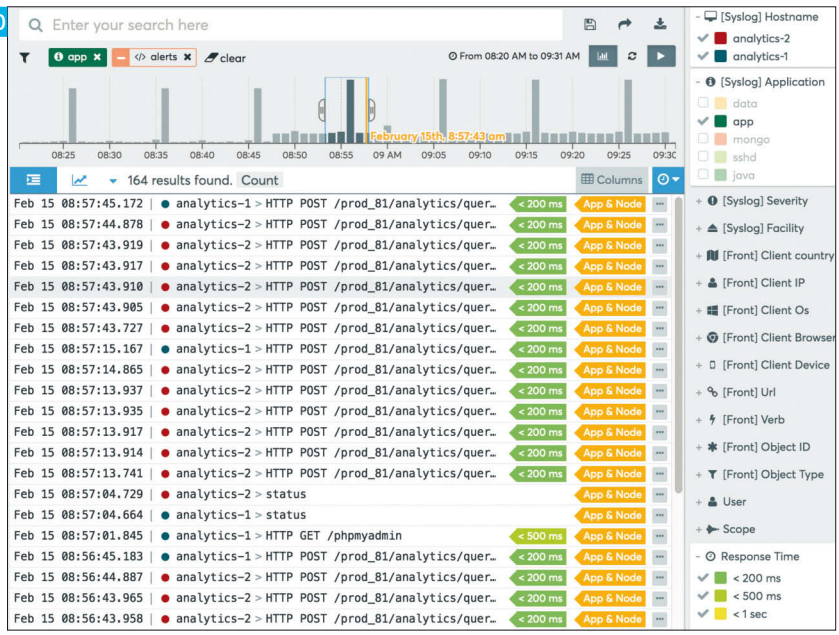
Ainsi, mieux que d'aller chercher l'information, c'est **l'information qui vient à vous**. Ces alertes sont principalement très utiles pour les logs d'erreurs et améliorer drastiquement la qualité de votre application. Néanmoins l'alerting peut être assez rude à mettre en oeuvre sur une application existante car on est rapidement noyé sous les alertes. Souvent celles-ci sont remplies de ce qu'on peut appeler avec sarcasme les "erreurs de bon fonctionnement", ces erreurs dont personne ne s'occupe car au final "ça marche". Il va falloir enfin s'en occuper soit en les résolvant (ce sont souvent les signes d'un vrai problème) soit en les passant en changeant leur niveau de criticité. C'est aussi le moment de se **poser de vraies questions sur la qualité des logs et leurs niveaux**.

Des exemples de questions à se poser :

- vos logs sont-ils compréhensibles pour tout le monde ?
- ont-ils suffisamment d'information pour leur analyse comme la trace d'exception ?
- est-ce que cette trace est utile ou au contraire nuit à la lisibilité ?
- cet évènement est-il une erreur ou un warning ? Par exemple, un utilisateur non trouvé dans votre système, est-ce une vraie erreur nécessitant une investigation et donc une alerte ? Ou un fonctionnement tout à fait raisonnable qui peut être indiqué sous forme d'un warning afin de laisser la possibilité de suivre les tentatives infructueuses de connexion ?

Améliorer la qualité de l'observabilité de votre plateforme demande un investissement constant comme la qualité du code !

Logmatic 10



Exemple de dashboard Grafana 11

Système distribué de Schrödinger

Un autre élément important de la production est la stratégie de "healthcheck"[3] surtout dans le cadre d'une production "Container as a Service". Dans le cadre d'un monolithe, vous savez facilement si votre application est vivante ou non. Dans le cadre d'un système distribué, c'est plus complexe. Chaque instance étant autonome le système doit être capable de déterminer automatiquement l'état de cette instance. Vous ne pouvez pas vous permettre d'attendre qu'un utilisateur vous prévienne quand l'application ne fonctionne plus. **Grâce aux healthchecks, un système d'orchestration de services se chargera de relancer automatiquement une application en mauvaise santé.**

Dans l'écosystème Spring Boot, la librairie Actuator vous permet de facilement mettre en place un endpoint /health pour connaître l'état de santé de votre application. Et par état de santé, nous vérifions le micro-service en lui-même ainsi que ses dépendances comme sa base de données ou son broker de message. Un bon healthcheck est donc transitif et c'est là où est toute la difficulté. Dans le cas d'orchestrateur de services, redémarrer un micro-service n'ayant plus de base de données ne fixera pas le problème… Il faudra donc adapter votre stratégie de récupération selon la cause réelle du dysfonctionnement, ou dans le cas échéant, intervenir manuellement. 13

DevOps encore !

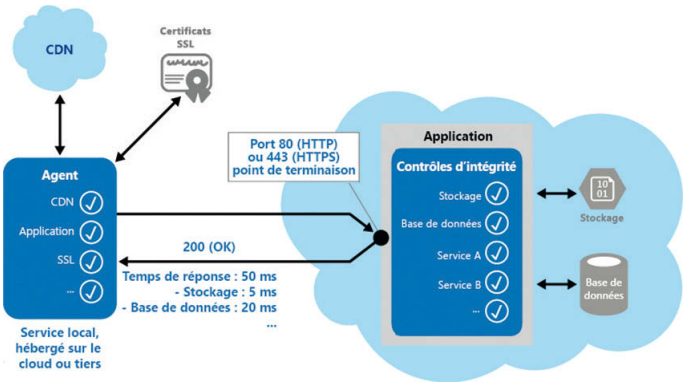
Nous voyons que gérer une production de micro-services est bien plus exigeant que gérer le typique monolithe. Là encore **il faut une vraie culture devops pour ne pas être submergé**. Si vous êtes dans une entreprise qui sépare encore prod et dev, l'équipe de production va très vite détester cette approche micro-services et tous les ennuis qu'elle leur cause. Clairement, il va falloir que la collaboration soit à son maximum, voire directement rendre les devs responsables de la production (le célèbre "You build it, you run it" d'Amazon). Dans ce cas, l'équipe de production se recentre sur la mise en place des nouveaux outils de déploiement, de monitoring voire de testing comme la pratique de "chaos engineering"[4]. Cette pratique vise à stresser un système distribué en simulant d'éventuelles pannes, qui vont de l'arrêt d'un serveur à l'extinction d'un data center. Ce genre de test a pour but de renforcer la confiance dans la capacité du système de résister à des conditions difficiles en production.

Conclusion

Dans cet article, nous avons recensé une grande partie des concepts et pratiques que nous pensons nécessaires pour gérer efficacement une architecture micro-services. Même si une partie de ses éléments (comme les tests, la livraison continue ou encore le monitoring) se retrouvent dans un monolithe, la complexité et l'énergie nécessaire de l'ensemble prend une tout autre dimension. Alors doit-on faire des micro-services ? Si on suit la mode alors "oui", et certains diront même que vous tuez des chatons si vous n'en faites pas ! (voir ce tweet de Josh Long, Spring Cloud Evangelist) 14

De notre côté, nous sommes plus modérés. Nous rejoignons ainsi les avis de Sam Newman[5], Martin Fowler[6] ou encore Simon Brown[7]. Si l'architecture micro-services présente de gros avantages, en particulier en termes d'agilité, de mise à l'échelle, d'apprentissage et de répartition des développements, il nous semble plus raisonnable de démarrer avec un monolithe adoptant une architecture très propre et permettant une extraction en services indépendants rapidement (le terme qui apparaît est "modular monolith"[8]). L'architecture micro-services demande énormément de techniques, de pratiques, de communication, et nécessitera une organisation à la hauteur. Comme le dit Martin Fowler, "you must be this tall to do micro-services"[9].

Un développeur continuant à coder malgré la prod en feu 12



Principe de "healthcheck" ou vérifier l'état de santé d'un micro-service, AZURE 13

14

De plus il est beaucoup plus risqué de démarrer une architecture micro-services de zéro qu'à partir d'un monolithe existant.

Cependant cette architecture va sûrement prendre une place majeure dans notre travail de tous les jours et nous espérons que les conseils que nous vous avons apportés vous permettront de les prendre en main facilement. Notre avis sur ce type d'architecture est qu'il n'est pas simple d'écrire un système distribué et il est surement préférable de commencer par un monolithe modulaire afin de mieux appréhender vos domaines métiers et vos besoins techniques. Mettre en place des micro-services peut vous créer beaucoup plus de problèmes qu'il n'en résout. Et soyons honnête, peu de contextes métiers peuvent vraiment justifier l'utilisation d'une telle architecture. •

(3) voir article Docker Health check, comment vérifier la santé de vos containers?

(4) Plus d'info sur le site principlesofchaos.org

(5) Voir l'article "Microservices for greenfield" sur le blog de Sam Newman

(6) Voir l'article "Don't start monolith" sur le blog de Martin Fowler

(7) Voir la conférence "Modular monoliths by Simon Brown" à Devoxx 2016

(8) Voir l'article "Modular Monolith Or Microservices?" sur le site mozaicworks.com

(9) Voir l'article "Microservice Prerequisites" sur son blog de Martin Fowler

Denis Duplan,
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Zoom hardware sur Amiga

Partie 2

Le choc de l'arrivée de la Super Nintendo fin 1991 pour les amateurs de micros 16 bits, ce fut le Mode 7. La console était capable d'appliquer une rotation et un redimensionnement à un bitmap de grande taille dans la trame, et il était possible de trafiquer pour produire un effet de perspective.

niveau 200

Ajuster le zoom horizontal hardware, un vrai challenge

En modifiant `ZOOM_MOVE` dans le programme précédent, on indique l'indice du MOVE auquel on souhaite que le Copper écrive `ZOOM_BPLCON1` dans BPLCON1, sachant que BPLCON1 est initialisé à `$00FF`. Il est alors possible d'observer le résultat à l'écran.

Pour que ce dernier soit bien visible, l'effet est répété sur une bande de `ZOOM_DY` de hauteur, en l'occurrence 20 pixels, précédée et suivie d'une bande de même hauteur sans effet. Par ailleurs, un motif blanc sur fond rouge est dessiné dans le bitplane : c'est une succession de groupes de 16 pixels : dans le 1er groupe, le dernier pixel est blanc ; dans le 2ème groupe, les deux derniers pixels sont blancs ; etc. Ce motif permet de réaliser un effet du genre : « *si je me positionne au niveau du MOVE correspondant au groupe comportant 4 pixels blancs et que je réduis BPLCON1 de 4, est-ce que j'observe la disparition de ces 4 pixels ?* »

Ce qui peut être observé, c'est que très généralement, les résultats sont décevants. Par exemple, passer BPLCON1 à `$0022` au 18ème MOVE (ie : `ZOOM_MOVE` valant `17`) produit le résultat reproduit sur la figure 6.

Toutefois, ils peuvent être satisfaisants. A titre de contre-exemple, passer BPLCON1 à `$00EE` au 4ème MOVE produit le résultat montré sur la figure 7.

C'est en procédant de la sorte, mais à l'aide d'un programme plus élaboré qui permet de désigner plusieurs MOVE devant réduire le retard de 1 dans BPLCON1, qu'il est possible d'établir une liste des 40 MOVE permettant de réduire la largeur de l'image de 1 pixels, de 2 pixels, de 3 pixels et ainsi de suite jusqu'à 15 pixels, pour une valeur initiale de BPLCON1 donnée.

« Une valeur initiale de BPLCON1 donnée » ? Ce n'est pas `$00FF` ? Non, car il faut gérer une contrainte : lorsqu'une colonne est dissimulée, les colonnes qui suivent sont décalées à l'écran d'un pixel sur la gauche. A ce train-là, une image dont 15 colonnes sont dissimulées se trouve tassée de 15 pixels sur la gauche. Or ce n'est pas ce qui est attendu lors d'un zoom réduisant la largeur d'une image de 320 à 0 pixels : comme évoqué en présentant le zoom vertical hardware, l'image doit rester centrée à l'écran.

Pour y parvenir, il faut adopter le scénario décrit par la figure 8.

6 *Les aléas de la modification de BPLCON1 en cours de ligne.*

7 *La dissimulation réussie d'une colonne par modification de BPLCON1 en cours de ligne.*

Sur cette figure, chaque ligne représente une étape du zoom. A gauche, la valeur initiale de BPLCON1. A droite, sa valeur finale. Entre les deux, les 20 groupes de 16 pixels composant une ligne d'une image de 320 pixels de large :

- les groupes dont les derniers pixels ont déjà dissimulés aux étapes précédentes sont en rouge clair ;
- le groupe dont le dernier pixel doit être dissimulé à cette étape est en rouge foncé.

A chaque groupe rouge clair ou rouge foncé, la valeur de BPLCON1 est réduite de 1 pour dissimulée un pixel.

Comme il est possible de le constater, le scénario vise à compenser autant que possible le tassement sur la gauche de l'image en la décalant d'un pixel sur la droite chaque fois que deux pixels ont été dissimulés. C'est un scénario qui se construit en remontant les étapes depuis la dernière, car il faut que la valeur initiale de BPLCON1 soit alors `$00FF` pour pouvoir dissimuler 15 pixels.

Adopter ce scénario a une conséquence capitale. Pour le comprendre, il faut désormais bien distinguer quatre choses :

- **L'écran**. C'est l'écran physique de l'ordinateur : s'agissant du repère de référence, il n'est jamais décalé, et sa résolution horizontale est toujours de 320 pixels.
- **L'image**. C'est ce que le spectateur voit à l'écran : une colonne de couleur 0 sur la gauche créée par la valeur initiale de BPLCON1, puis les bitplanes dont la partie visible est amputée d'une colonne de même largeur sur la droite.
- **Les bitplanes**. Ce sont des espaces en mémoire de 320 pixels de large dont une partie seulement est visible dans l'image à l'écran.
- **Le dessin**. C'est ce que l'on souhaite donner à voir au spectateur, à savoir quelque chose dont la largeur se réduit progressivement tout en restant centré dans l'image.

C'est l'enchâssement de multiples repères (le dessin dans le bitplane, le bitplane dans l'image, l'image dans l'écran – mais on peut considérer que leurs repères sont confondus) qui fait toute la com-

$0077	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0077
$0088	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0077
$0088	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0066
$0099	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0066
$0099	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0055
$00AA	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0055
$00AA	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0044
$00BB	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0044
$00BB	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0033
$00CC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0033
$00CC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0022
$00DD	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0022
$00DD	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0011
$00EE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0011
$00EE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0000
$00FF	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	$0000

8 *Scénario du zoom horizontal hardware.*

plexité de la gestion du zoom horizontal hardware.

Quand aucune colonne n'est dissimulée, il faut que le dessin soit centré à l'écran. Or BPLCON1 vaut alors `$0077`. Pour qu'un dessin décalé de 7 pixels sur la gauche apparaisse centré dans l'écran de 320 pixels de large, il faut que la largeur de ce dessin ne dépasse pas 306 pixels. En effet si 7 pixels sont inutilisables à gauche, il faut que 7 pixels le soient à droite : 14 pixels en tout, à soustraire de 320, ce qui donne 306.

C'est donc un dessin de 306 pixels de large, calé sur la gauche dans des bitplanes de 320 pixels de large, qui peut être au plus utilisé pour produire le zoom, la colonne de 14 pixels de large sur la droite devant être en couleur 0 (Figure 9).

Le scénario débouche sur une liste qui donne la valeur de BPLCON1 en début de ligne et identifie les MOVE qui décrémentent cette valeur pour dissimuler la dernière colonne de N groupes de 16 pixels, réduisant d'autant de pixels la largeur de l'image (Figure 10).

Cette liste est totalement spécifique :

- la série de 40 MOVE d'une ligne doit être précédée d'un WAIT à la position horizontale $3D ;
- les groupes ont été déterminés en appliquant le scénario, qui n'est qu'un scénario parmi d'autres.

Reconnaissons cette liste comme une des « magic list » de l'Amiga. Une autre « magic list » est celle qui indique comment organiser la Copper list pour produire des boucles qui répètent, sur 8 lignes, des séries de 40 MOVE, et ce sur toute la hauteur d'un écran PAL, c'est-à-dire sur 256 lignes.

Retour à notre liste. Sa mise en œuvre dans le programme `zoom2.s` produit le résultat suivant, où un dessin de 306 pixels de large (lignes blanches sur fond gris) centré dans un bitplane de 320 pixels de large (fond rouge) est réduit progressivement en dissimulant les derniers pixels de 15 groupes de 16 pixels les uns après les autres. Pour que le zoom semble progressif s'il devait être animé, les dissimulations alternent entre la gauche et la droite du groupe de pixels du milieu. Le dessin est intact dans la moitié supérieure de l'écran ; il subit le zoom dans la moitié inférieure de l'écran, perdant un pixel toutes les 8 lignes (Figure 11).

Au-delà de 15 pixels, passer de la dissimulation à la suppression

BPLCON1 permet d'introduire un retard initial de 15 pixels au plus au début de chaque ligne. Partant, c'est au plus 15 pixels qu'il est possible de dissimuler sur cette ligne, ce qui est clairement insuffisant pour produire un zoom où la largeur de l'image passerait de 320 à 0 pixels.

Quand les possibilités du zoom horizontal hardware sont épuisées (BPLCON1 valant `$00FF` au début d'une ligne, et étant réduit progressivement jusqu'à `$0000`), nulle autre solution que de prendre le relai avec un zoom horizontal software. Il s'agit alors de reproduire dans les bitplanes ce que le spectateur voit à l'écran avant de réinitialiser le zoom hardware (BPLCON1 valant `$0077` au début d'une ligne, et n'étant pas réduit). Cela revient à dire qu'il faut supprimer véritablement les colonnes de pixels dissimulées dans les bitplanes, et recentrer horizontalement le dessin ainsi réduit dans ces derniers.

Recentrer de combien de pixels ? Comme cela vient d'être rappelé, au moment où il faut prendre le relai du zoom hardware, la valeur de BPLCON1 est `$00FF`, ce qui correspond à un décalage de 15 pixels sur la droite. Par ailleurs, la valeur de BPLCON1 quand aucun pixel n'est dissimulé est de `$0077`, ce qui correspond à un décalage de 7 pixels sur la droite. En conséquence, il faut produire un dessin qui, lorsque les bitplanes qui le contiennent seront ainsi décalés, présentera les caractéristiques suivantes :

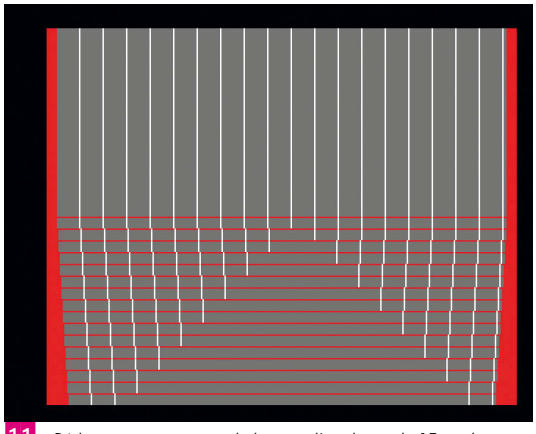
- il fera 320 - 15 = 305 pixels de large ;
- il apparaîtra décalé de 15 pixels sur la droite.

Bref, une fois réduit, le dessin doit être décalé de 15 - 7 = 8 pixels sur la droite dans les bitplanes (Figure 12). On parle bien du dessin et non des bitplanes. En effet, il est bien entendu que la largeur de ces derniers ne change pas ; c'est le dessin qu'ils contiennent

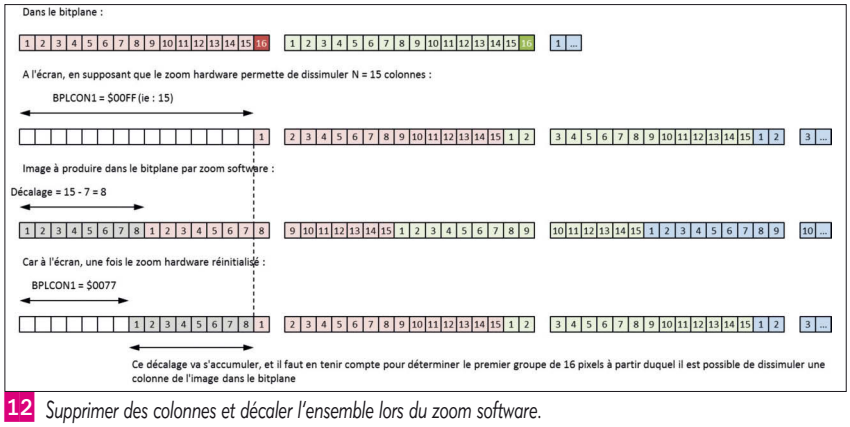
9 *Le dessin de 306 pixels de large dans un bitplane de 320 pixels de large.*

Départ	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77
$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77	$77
$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$88	$77	$77	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66
$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$88	$88	$77	$77	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66	$66
$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$99	$88	$88	$77	$77	$66	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55
$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55	$55
$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44
$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44	$44
$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$BB	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$44	$44	$33	$33	$33	$33	$33	$33	$33	$33	$33	$33	$33	$33	$33
$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$BB	$BB	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$44	$44	$33	$33	$33	$33	$33	$33	$33	$33	$33	$33	$33	$33	$33
$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$CC	$BB	$BB	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$44	$44	$33	$33	$22	$22	$22	$22	$22	$22	$22	$22	$22	$22	$22
$DD	$DD	$DD	$DD	$DD	$DD	$DD	$DD	$DD	$DD	$DD	$DD	$CC	$CC	$BB	$BB	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$44	$44	$33	$33	$22	$22	$22	$22	$22	$22	$22	$22	$22	$22	$22
$DD	$DD	$DD	$DD	$DD	$DD	$DD	$DD	$DD	$DD	$DD	$DD	$CC	$CC	$BB	$BB	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$44	$44	$33	$33	$22	$22	$11	$11	$11	$11	$11	$11	$11	$11	$11
$EE	$EE	$EE	$EE	$EE	$EE	$EE	$EE	$EE	$EE	$DD	$DD	$CC	$CC	$BB	$BB	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$44	$44	$33	$33	$22	$22	$11	$11	$11	$11	$11	$11	$11	$11	$11
$EE	$EE	$EE	$EE	$EE	$EE	$EE	$EE	$EE	$EE	$DD	$DD	$CC	$CC	$BB	$BB	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$44	$44	$33	$33	$22	$22	$11	$11	$00	$00	$00	$00	$00	$00	$00
$FF	$FF	$FF	$FF	$FF	$FF	$FF	$FF	$EE	$EE	$DD	$DD	$CC	$CC	$BB	$BB	$AA	$AA	$99	$99	$88	$88	$77	$77	$66	$55	$44	$44	$33	$33	$22	$22	$11	$11	$00	$00	$00	$00	$00	$00	$00

10 *La « magic list » du zoom horizontal hardware.*



11 *Réduire progressivement la largeur d'un dessin de 15 pixels, tout en le centrant horizontalement.*



12 *Supprimer des colonnes et décaler l'ensemble lors du zoom software.*

qui est réduit, pas eux. Comment supprimer ? Il est possible d'utiliser le Blitter, qui s'entend à merveille pour décaler des mots sur la droite (en mode ascendant) comme sur la gauche (en mode descendant) après les avoir éventuellement masqués, ou le CPU, voire les deux. Une optimisation est possible – en fait, elle est même nécessaire. En effet, au fil du zoom software, la largeur du dessin dans les bitplanes se réduit, jusqu'au point où le dessin n'empiète plus sur certains groupes de 16 pixels qui se trouvent à sa gauche et à sa droite. Partant, il devient inutile de supprimer des colonnes dans ces groupes. De même, ces colonnes n'ont plus à être dissimulées. Ces optimisations sont obligatoires, car le spectateur ne comprendrait pas que le zoom semble faire une pause parce qu'une étape est consacrée à la dissimulation d'une colonne qui est vide. Il faut donc déterminer quelles colonnes doivent être dissimulées durant un cycle de zoom hardware, et supprimées par zoom software au terme de ce cycle, en fonction de la largeur du dessin au début de ce cycle. C'est ce qui a été fait pour réaliser le programme `zoom3.s` à l'aide de la feuille Excel qui figure dans `zoom.xlsx` (Figure 13).

Comme le nombre de colonnes à supprimer se réduit, en mode débogage (constante `DEBUG` passée à `1`), il apparaît que le temps pris par la suppression diminue tandis que le dessin occupe de moins en moins de groupes dans les bitplanes. Au début, ce temps est assez considérable pour que la suppression ne tienne pas dans la trame sur Amiga 500. En fait, il faudrait simplement sortir la suppression de la boucle principale et l'utiliser avant pour précalculer les versions successives du dessin. Elles devraient toutes pouvoir tenir en mémoire. Au final, le code de la suppression est assez complexe, et il serait trop long d'en présenter le détail ici. Qui veut en savoir plus peut se référer au source du programme indiqué. On y utilise le Blitter uniquement pour recopier en les décalant et en les masquant les colonnes de mots qui forment le dessin, et celles-là uniquement.

Le choix des lignes et des colonnes à dissimuler : un dilemme d'informaticien

Les techniques présentées pour dissimuler / supprimer des lignes et des colonnes fonctionnent bien, mais elles reposent sur des hypothèses dont il faut avoir conscience. Ces hypothèses portent sur la manière dont il convient d'identifier les lignes et les colonnes à dissimuler à une étape du zoom donnée. Elles ont notamment déterminé le scénario du zoom horizontal hardware présenté plus tôt.

S'il faut dissimuler N lignes et colonnes à une étape du zoom donnée, le premier réflexe est de chercher à répartir également ces lignes sur la hauteur et ces colonnes sur la largeur du dessin. A bien y réfléchir, cela repose sur l'hypothèse que le zoom sera plus réaliste si la dissimulation est diffuse, car le spectateur devrait toujours disposer d'assez d'informations en tout point du dessin pour reconnaître sinon le dessin initial, du moins le dessin de l'étape précédente. Toutefois, ce n'est qu'une hypothèse. S'il s'agissait de zoomer un texte, il y a fort à parier que les caractères traités aussi arbitrairement deviendraient vite méconnaissables. En toute rigueur, la réduction d'un dessin, c'est le filtrage d'un message, lequel ne peut déboucher sur un résultat acceptable qu'à condition de tenir compte un minimum de la signification du message en question, quitte à reformuler ce dernier.

Etape	Largeur	Décalage	Décalage cumulé	1er groupe occupé	Groupes occupés	Groupes zooomables	Groupes zoomés	BPLCON1 initial	BPLCON1 final
1	306	0	0	0	20	19	15	7	15
2	291	8	8	0	19	18	15	7	15
3	276	8	16	1	18	17	15	7	15
4	261	8	24	1	17	16	15	7	15
5	246	8	32	2	16	15	15	7	15
6	231	8	40	2	15	14	14	7	14
7	217	7	47	2	15	14	14	7	14
8	203	7	54	3	14	13	13	7	14
9	190	7	61	3	13	12	12	7	13
10	178	6	67	4	12	11	11	7	13
11	167	6	73	4	11	11	11	7	13

13 *Calcul des groupes à traiter à chaque cycle de zoom horizontal hardware.*

Cette considération sémantique peut paraître stratosphérique dans le cas du zoom en temps réel d'un dessin sur Amiga, tant il est vrai que la puissance de calcul disponible interdira d'en tenir compte rigoureusement – si tant est qu'il soit seulement possible d'en tenir compte ! Toutefois, il est bon de l'évoquer pour échapper à une remise en question sans issue de la pertinence de l'hypothèse adoptée. Par là, on veut dire qu'en cherchant à diffuser les dissimulations, on cherche à faire au mieux pour emballer le spectateur avec les moyens du bord. Ce qui fonde l'hypothèse, ce ne sont pas de savants calculs ; c'est que le résultat qui en découle est, pour le spectateur, assez convaincant.

Enfin, il faut tout de même questionner l'hypothèse du fait d'une contrainte technique qui limite sérieusement cette application. Cette contrainte porte sur le zoom horizontal hardware, dont on a vu qu'il n'est pas aussi souple que le zoom vertical hardware. En effet, il est impossible de choisir les colonnes à dissimuler aussi facilement que les lignes à traiter pareillement : c'est parmi les derniers pixels de groupes de 16 pixels qu'il faut choisir.

Par exemple, pour réduire un dessin de 320 pixels à 318 en dissimulant 2 colonnes, il pourrait paraître cohérent de dissimuler les colonnes 106 et 214, l'espace entre les colonnes dissimulées étant alors régulier :

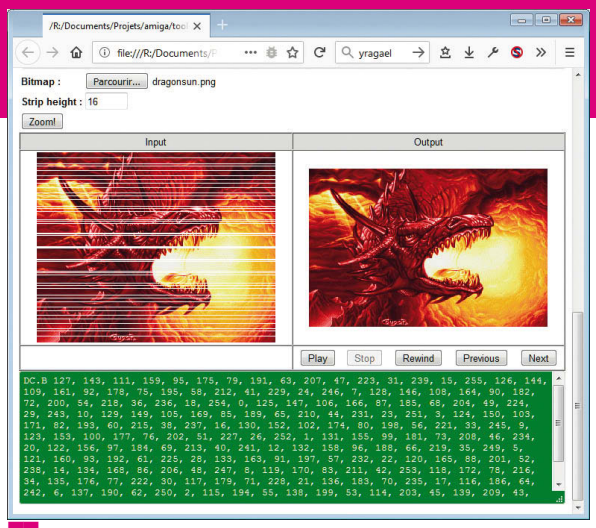
- 106 pixels entre les colonnes 0 et 105 ;
- 107 pixels entre les colonnes 107 et 213 ;
- 106 pixels entre les colonnes 215 et 320.

Or la colonne 106 correspond au 10ème pixel du 6ème groupe de 16 pixels d'une ligne. Autrement dit, elle tombe mal, car elle ne tombe pas sur le pixel d'un groupe qu'il est possible de dissimuler en réduisant le retard de 1 avant l'affichage de ce dernier. Bref, elle gêne car elle ne correspond pas au 16ème pixel d'un groupe de 16 pixels.

La distribution des colonnes de pixels qu'il est possible de dissimuler est contrainte. Qui veut réduire progressivement la largeur d'un dessin de 320 à 0 pixels en dissimulant une nouvelle colonne de pixels à chaque étape peut au mieux répartir les colonnes qu'il dissimule tous les 16 pixels. Et encore, ce n'est pas tous les 16 pixels du dessin, mais tous les 16 pixels des bitplanes qui contiennent le dessin, comme on l'a vu.

Enfin, quelle que soit l'hypothèse adoptée, il faut tout de même aussi la questionner sur un autre point. Continuons sur l'exemple du zoom horizontal. Indépendamment de la contrainte qui vient d'être évoquée, s'il faut enchaîner des dissimulations de colonnes, comment vaut-il mieux procéder :

- en considérant qu'une colonne dissimulée ne doit pas réapparaître, et donc en dissimulant une nouvelle colonne quitte à ce que les colonnes supprimées ne soient pas également réparties sur toute la largeur du dessin ?
- en considérant qu'une colonne dissimulée peut réapparaître, et donc en dissimulant de nouvelles colonnes, qui seront nécessairement différentes, mais qui présenteront l'intérêt d'être toujours également réparties sur toute la largeur du dessin ?



14 *Un outil réalisé en HTML pour tester un scénario de zoom vertical.*

Ici encore, c'est un questionnement sur lequel on passerait un temps infini, car au regard des moyens disponibles, il est impossible de retenir l'une ou l'autre de ces hypothèses sans tenir compte de l'effet que le résultat qui en découle produit sur le spectateur.

Pour un esprit cartésien, cette histoire de zoom hardware, c'est un peu la déconvenue à chaque étape. L'identification des MOVE pour modifier BPLCON1 au bon moment sur une ligne ? A déterminer empiriquement. L'identification des colonnes et des lignes à dissimuler ? A déterminer empiriquement. Le choix de faire réapparaître ou non des colonnes et des lignes dissimulées ? A déterminer empiriquement.

Mais c'est que l'esprit cartésien s'est trompé de domaine. Rappelons que le zoom hardware est employé dans une démo. Or, ainsi que l'a fort bien expliqué le formidable Navis / ASD, le principe de toute démo est de faire illusion. Il est bon qu'on nous rappelle régulièrement à quoi nous sommes censés nous consacrer. C'est le meilleur moyen pour ne pas être victime d'une illusion, celle de la poursuite d'une perfection sans intérêt pour celui auquel nous destinons les fruits de notre labeur, à savoir le spectateur. Et dans une démo, il s'agit de faire du Hollywood pour attirer les foules, pas du cinéma français pour les tenir à distance… Si vous ne travaillez que pour vous-même, à moins d'être un génie – mais qui se pose cette question a déjà la réponse –, vous ne resterez pas dans l'Histoire : le génie subjugue ; le tiers doit convaincre. Sans donc nous faire d'illusion sur la catégorie dont nous relevons, concluons benoîtement notre exploration des délices du zoom hardware.

Le programme `zoom4.s` rajoute le zoom vertical hardware à la combinaison des zooms horizontaux hardware et software. Dans le cas du zoom vertical, le scénario adopté est exactement le même que celui retenu pour la dissimulation des colonnes, imposé par les limitations du zoom horizontal hardware. Un outil en HTML5, réalisé pour l'occasion, a permis de générer les indices des lignes à dissimuler tandis que le zoom progresse (Figure 14).

Le zoom horizontal hardware, une trouvaille inutile ?

Il faut saluer la ténacité de ceux qui se sont lancés dans le zoom hardware pour en faire quelque chose, tant l'effet est pénible à maîtriser. Toutefois, il ne faut pas surestimer le gain qu'ils ont pu en tirer. Ce n'est pas parce qu'il y a zoom dans la trame bientôt en plein écran que ce dernier est hardware. En fait, quand le zoom horizontal est très réussi, il y a tout lieu de penser qu'il n'est pas hardware. Ainsi, la fameuse démo *Elysium* de Sanity s'ouvre par zoom irréprochable d'une image sur 4 bitplanes (Figure 15). Mais le désassemblage de la Copper list ne donne à voir que des WAIT à chaque ligne qui débouche sur des modifications de BPL1MOD et BPL2MOD. Autrement dit, le zoom horizontal hardware n'est pas utilisé ici, mais uniquement le zoom vertical hardware.

On trouve un exemple de zoom horizontal hardware dans une autre production de Sanity, le music disk *Jesterday*. Ce type de zoom est utilisé pour produire le rouleau que l'utilisateur fait tourner pour sélectionner un morceau de musique à écouter.

Encore plus original, le scrolling à la Star Wars dans *The Fall* par The Deadliners & Lemon. Le désassemblage de la Copper list montre que non seulement le zoom horizontal hardware est utilisé, mais que l'effet est combiné à des changements d'adresse de bitplanes pour effectuer des sauts dans ces derniers.

Ce scrolling doit être mentionné pour l'ingéniosité dont a fait preuve le codeur, mais aussi à l'inverse pour sa médiocre qualité visuelle, tant les lettres sont déformées (*). On a vu des scrollings de ce type bien plus agréables à contempler, tout particulièrement celui de l'invraisemblable jeu *Stardust*, qui n'utilise ni zoom horizontal hardware, ni zoom vertical hardware, en HiRes qui plus est !

Il y a quelques leçons à tirer de tout cela. D'abord, c'est que la focalisation sur les coprocesseurs peut conduire à faire perdre de vue le fait qu'il est parfaitement possible d'accomplir bien des effets au CPU, éventuellement beaucoup plus réussis. C'est comme toujours : on se focalise sur les accessoires au détriment du principal, parce qu'aspiré par la technique, on en vient à négliger l'algorithmique.

« *Legends never dies* », comme l'a dit Bacchus / Fairlight. The Fall remonte à… avril 2018 ! Franchement, qui aurait dit que plus de 30 ans après sa sortie, il se trouverait encore des codeurs pour imaginer de nouvelles exploitations du hardware de l'Amiga 500 ? Ici encore, c'est comme toujours : épuise-t-on jamais les possibilités d'une vieille casserole, dont le proverbe dit qu'on y fait toujours la meilleure confiture ?

(*) Cela n'enlève rien au mérite du codeur, qui a réussi une prouesse technique. Par ailleurs, la démo contient des effets visuellement très réussis, en particulier des enchaînements dont la réalisation a dû nécessiter un travail fou. Sur les enchaînements, lire d'ailleurs une interview de Chaos / Sanity, publiée dans *The Jungle* #2 en 1993. Codeur à juste titre très réputé, il prenait visiblement tout le monde de haut – lire « *Chaos - Only superlatives please* » publié dans *R.A.W.* #7 pour en rire –, mais son propos était pertinent. •

15 *Zoom irréprochable d'une image en 4 bitplanes dans la démo Elysium de Sanity.*

Pour en savoir plus

World of Commodore 92 de Sanity :
https://www.pouet.net/prod.php?which=2938
Helmet for Sale par Jason / Kefrens :
http://janeway.exotica.org.uk/release.php?id=8261
Conférence de Navis / ASD :
https://www.youtube.com/watch?v=3GNVBuVHKso
Elysium de Sanity : https://www.pouet.net/prod.php?which=1505
Jesterday de Sanity : https://www.pouet.net/prod.php?which=2894
The Fall de The Deadliners & Lemon :
https://www.pouet.net/prod.php?which=75773
Interview de Chaos / Sanity dans *The Jungle* #2 :
http://janeway.exotica.org.uk/target.php?idp=480&idr=9440&tgt=1

Si le code fonctionne...

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
Nos experts techniques : C. Pichaud, Dr P. Guichard, K. Vavelin, D. Poulin, J. Thiriet, J. Dubois, D. Dhouib, P. Charrière, J. Giacomini, V. Koukponou, M. Bacci, Y. Grenzinger, D. Panza, D. Duplan
Couverture : JHipster - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@programmez.com.
Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34

Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, janvier 2019

Abonnement :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com*
Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30.

Tarifs
Abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter.

PDF
35 € (monde entier) souscription sur www.programmez.com

INFORMER
pour transformer l'entreprise

La dématérialisation, le Cloud,
les communications unifiées,
les nécessités de la cybersécurité
transforment le travail et toute l'entreprise,
les services publics.
Le magazine, le site, ses newsletters
vous informent sur cette actualité mouvante
et vous aident à décoder les tendances

Abonnez-vous

www.solutions-numeriques.com/abonnement/

4 sites thématiques, pour répondre à vos besoins d'information

- Vous êtes responsable informatique ou bien dirigeant ou cadre d'entreprise ? *2 sites répondent à votre profil*
- La cybersécurité vous concerne ? *Cliquez sur l'onglet. Vous trouverez les infos, l'annuaire, le lexique, etc*
- L'emploi, les salaires, les formations, les offres vous intéressent ? *Le site sur l'Emploi dans le numérique est à votre disposition*

www.solutions-numeriques.com