CAFETIÈRE | INTEL | JENKINS X | C++ | REDIS

programmez.com

Programmez!

Le magazine des développeurs

229 MAI 2019

MongoDB

Le NoSQL surfe sur le cloud

C# 8.0 & Visual Studio 2019

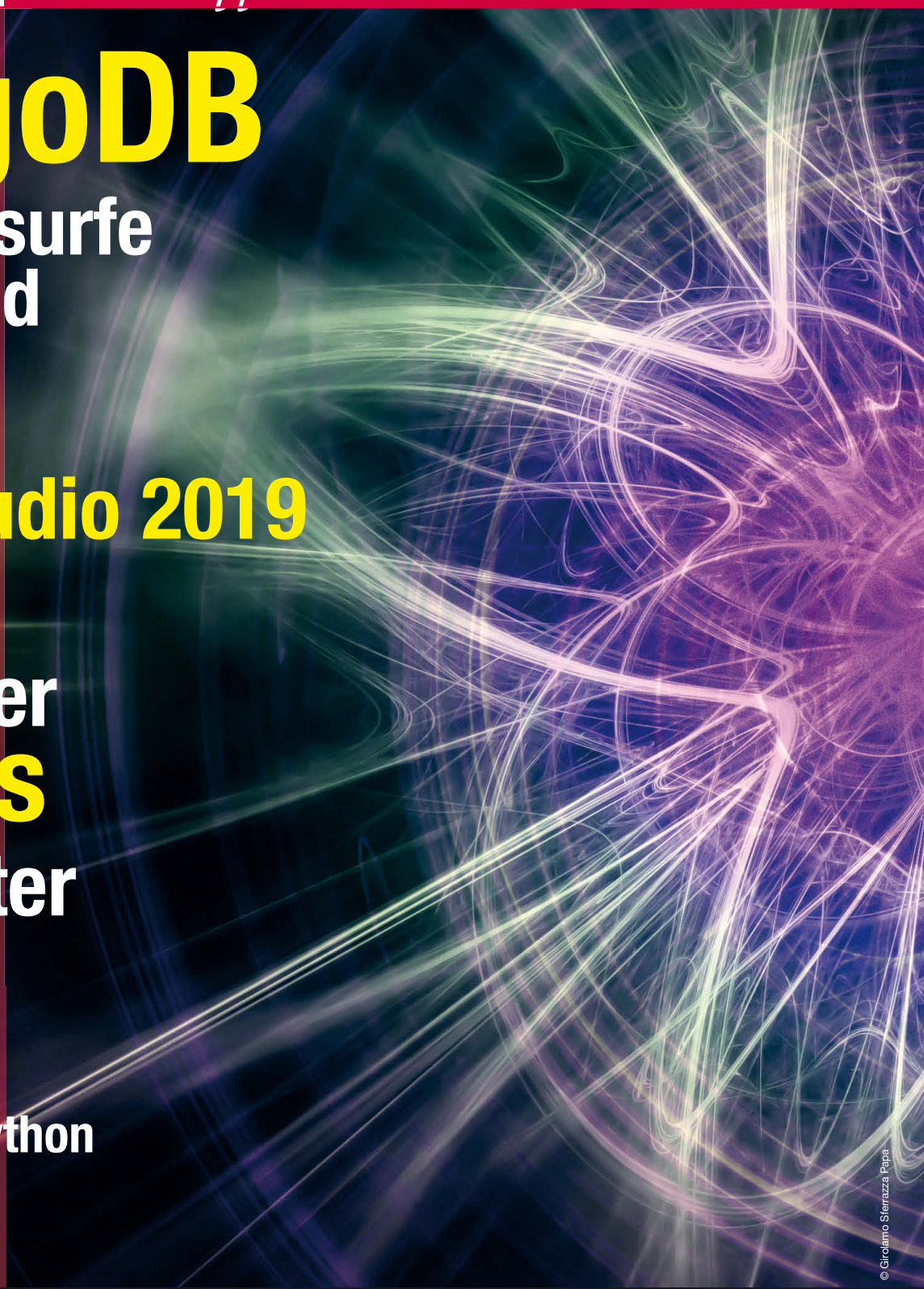
Les nouveautés

Développer des apps avec Flutter

PYTHON

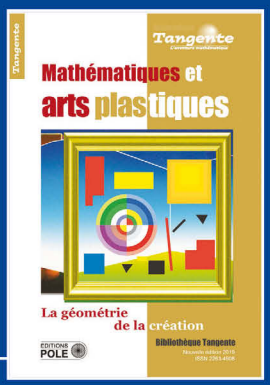
Programmation quantique en Python

M 04319 - 229S - F: 6,50 € - RD



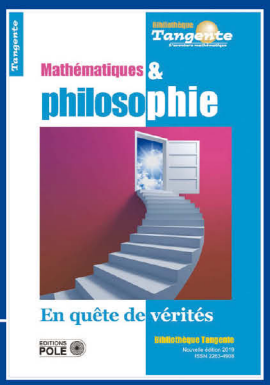
LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

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



Mathématiques et arts plastiques

Les clés de toute œuvre, artistique comme mathématique, sont la créativité, l'originalité, la beauté... Le dialogue entre les deux activités est d'autant plus fécond que les techniques mathématiques peuvent se mettre au service de l'art.
Voyage documenté et visuellement plaisant vers cette complicité.



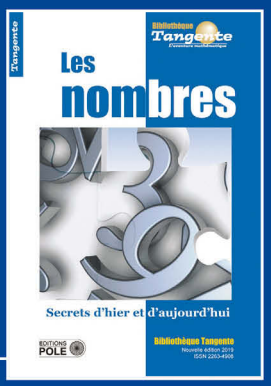
Mathématiques et philosophie

Philosophie et mathématiques sont nées simultanément du regard porté sur le monde qui nous entoure. Ce sont souvent les mêmes qui ont réfléchi aux notions d'infini, de logique, de hasard, de paradoxes... Merci Aristote, Platon, Descartes, Leibniz, Poincaré...

La plus belle bibliothèque mathématique au monde

vient de rééditer 5 de ses succès

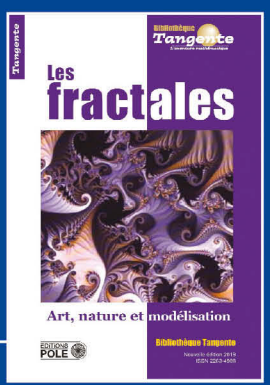
Entiers positifs ou négatifs, zéro, nombres fractionnaires, irrationnels, algébriques ou transcendants, imaginaires...
Ils ont mis des siècles à s'imposer, à cohabiter.
Découvrez leurs secrets.



Les nombres et leurs secrets

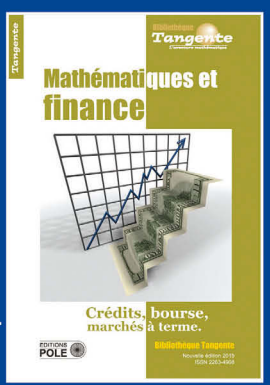
Bibliothèque Tangente
L'aventure mathématique

Des taux d'intérêt à l'inflation, de la bourse aux marchés à terme et options, de la notion de risque à la nécessité de régulation, les mathématiques sont partout dans la finance. Cette nouvelle édition est enrichie par les conséquences de la crise financière et l'apport de l'intelligence artificielle.



Les fractales

Les fractales, identiques à elles-mêmes à toutes les échelles, se rencontrent aussi dans la nature, dans l'art et dans de nombreux domaines techniques. Retrouvez dans cet ouvrage très visuel la théorie et l'histoire, très liée à Benoît Mandelbröt, de ces magnifiques objets géométriques.

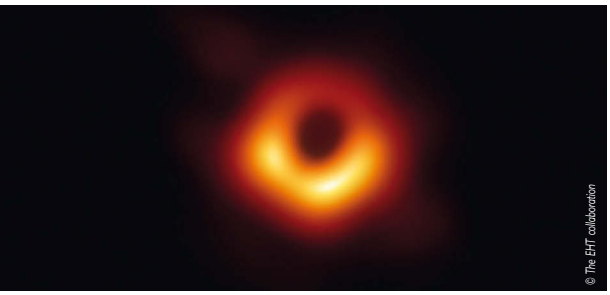


Mathématiques et finance

Disponibles sur Internet à l'adresse **www.infinimath.com/librairie**

EDITO

M87* de la galaxie M87



Ce n'est pas encore la réponse ultime à la question ultime mais le 10 avril dernier, une photographie en ultra haute définition a permis de « voir » pour la première fois un trou noir super massif : M87* de la galaxie M87. Le trou noir est une des entités cosmiques les plus étonnantes et mystérieuses. Ce monstre, dont la masse représente 6,5 milliards de fois notre soleil, se situe à 55 millions d'années-lumière de la Terre.

Pour réaliser ce cliché exceptionnel, c'est un ensemble de téléscopes, répartis sur l'ensemble de la planète, qui a été utilisé : Event Horizon Telescope. Il a fallu ensuite récupérer et traiter l'énorme quantité de données et de photos pour construire l'image de M87*. Les données récoltées en 2017 représentaient plus d'un Petaoctet qu'il a fallu injecter dans les outils et les modèles. Cette recherche a mobilisé plus de 200 scientifiques.

Pour construire la photo, il a fallu développer de complexes algorithmes. Deux noms ressortent : Katie Bouman et Andrew Chael.

Loin des polémiques sur la contribution réelle de Katie Bouman, et des commentaires parfois honteux sur la chercheuse, il faut au contraire se féliciter de cette avancée et du travail des chercheurs.

Si vous découvrez LV-223 ou LV-426, n'oubliez pas de nous prévenir(1)…

Mais revenons sur cette Terre. Ce mois-ci, nous allons explorer MongoDB. Rappelons que cette base NoSQL est taillée pour le big data, les traitements et le cloud computing qui sera un des principaux thèmes de ce numéro.

Côté développement mobile, nous allons parler de Flutter. S'il fut présenté assez discrètement par Google, aujourd'hui, c'est un des environnements les plus puissants pour développer des apps mobiles multiplateformes. Oui la concurrence est forte mais Flutter expose de solides arguments techniques.

Autre excellent thème (tous les thèmes sont excellents chez Programmez!) : réaliser des fractales Mandelbrot avec du Web Assembly. Cela va envoyer du lourd ! Dans la même veine, nous allons parler de programmation quantique avec… Python ! Ou encore comment hacker facilement une cafetière (faut bien se détendre un peu). Nous évoquerons aussi l'excellent outil Redis, les "promises" en JavaScript.

Une des grosses annonces d'avril a été lancement de Visual Studio 2019. Focus sur cette nouvelle version qui embarque de belles nouveautés. Et nous faisons aussi un point sur C# 8 qui arrive dans quelques mois.

Bon code !

François Tonic
ftonic@programmez.com

(1) Vous savez les sympathiques planètes des Aliens

SOMMAIRE

Dans le prochain numéro !
Programmez! #230, dès le 31 mai 2019

Cloud & développeur :
comment utiliser des services cloud ?
Quel impact pour vous ? Les bonnes pratiques

996ICU : sur Github, les développeurs chinois donnent de la voix

Sur Github, un groupe de développeurs chinois a lancé un dépôt Github d'un genre un peu spécial. Le but n'est pas vraiment ici de partager du code, mais plutôt de rassembler les développeurs autour d'un thème sensible, celui des conditions de travail. Les auteurs du projet protestent notamment contre la popularisation du rythme 996, qui signifie « From9am to 9pm, 6days a week. » Sur le dépôt, on retrouve ainsi des listes d'entreprises ayant adopté ce rythme de travail ou encore un projet de licence logicielle visant à protéger les droits des développeurs. Sans grande surprise, l'accès à la page a été bloqué sur plusieurs navigateurs « maison » conçus par Tencent ou Xiaomi. L'ambiance est au beau fixe, comme on l'imagine.

Arrivederci Mounir Mahjoubi, hello Cedric O

Le secrétaire d'État au numérique Mounir Mahjoubi a quitté son poste à l'occasion du remaniement ministériel à la fin du mois de mars. Mounir Mahjoubi avait en effet fait part de son intention de se porter candidat pour les élections municipales de Paris sur la liste LREM. Il a donc été remplacé par Cedric O, ancien cadre de chez Safran et conseiller proche de plusieurs figures du PS, devenu conseiller d'Edouard Philippe et Emmanuel Macron lors de la victoire de ce dernier à l'élection présidentielle de 2017.

LE BITCOIN BOUGE ENCORE

Alors qu'on avait perdu tout espoir de voir un jour le cours du bitcoin repartir à la hausse, voilà que le cours de la fameuse monnaie virtuelle a soudainement rebondi dans la nuit du 1er avril pour brièvement atteindre les 5 000 dollars, soit son plus haut niveau depuis plusieurs mois. On ne prendra pas le risque de s'avancer sur les raisons de cette hausse, ni sur la pérennité de celle-ci. Mais qu'on se le dise : en ce début du mois d'avril, le cours du bitcoin a rebondi, entraînant d'autres cryptomonnaies avec lui. Jusqu'au prochain crash ?

Taxe Gafa : les députés s'emparent du sujet

L'Assemblée nationale se penche ce mois-ci sur le projet de loi relatif à la taxation des géants du numérique. Le projet de loi vise à taxer certaines entreprises sur les activités de vente de publicité en ligne, de vente de données personnelles et de mise en relation des utilisateurs. La taxe ne vise que les entreprises réalisant plus de 25 millions d'euros de chiffre d'affaire en France, ce qui en fait selon ses auteurs un outil parfait pour viser les Google, Apple, Facebook et consorts.

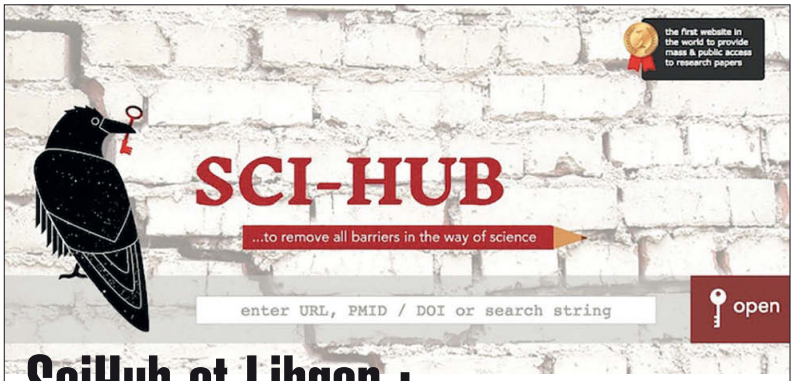
#NPMLayoffs : les lignes bougent chez NPM

NPM, la société qui gère le gestionnaire de paquet JavaScript npm, s'est doté d'une nouvelle direction dans le courant de l'année dernière et affiche des objectifs de croissance ambitieux. Malheureusement, cette nouvelle impulsion ne se fait pas sans casse et plusieurs développeurs et membre de l'équipes ont annoncé sur Twitter avoir été renvoyés sèchement par la nouvelle direction sur le hashtag #npmlayoffs. L'ambiance est donc ici aussi au beau fixe.

Facebook :

quand ça veut pas, ça veut pas

Malgré toutes les promesses de Zuckerberg, qui jure que Messenger et Whatsapp évolueront vers plus de confidentialité, Facebook enchaîne les bourdes en matière de sécurité. Au cours des dernières semaines, le réseau social a ainsi découvert qu'il avait stocké en clair plusieurs centaines de millions de mots de passe. Puis quelques jours plus tard, des chercheurs ont découvert des serveurs AWS dépourvus d'authentification qui contenaient les données d'un peu plus de 500 millions d'utilisateurs Facebook. Et comme si ça ne suffisait pas, Facebook s'est ensuite fait taper sur les doigts pour avoir demandé à ses nouveaux inscrits de donner leurs mots de passe d'adresse email pour vérifier leur compte. On attend la suite avec une certaine impatience.



SciHub et Libgen :

au tribunal, les éditeurs obtiennent gain de cause

Dans le monde de la recherche universitaire, SciHub et Libgen sont deux sites bien connus utilisés par la communauté des chercheurs pour diffuser leurs articles de recherche en dehors des circuits traditionnels des éditeurs de revues scientifiques. Du piratage en somme, et un manque à gagner pour les éditeurs qui facturent l'abonnement à leurs revues à des prix parfois élevés. Mais Elsevier et Springer Nature, deux éditeurs importants du secteur, ont obtenu gain de cause face à la justice française et les deux sites seront dorénavant bloqués par les fournisseurs d'accès français. Ouf, on s'inquiétait presque pour eux.

PARIS 16 & 17 MAI

+ JOURNÉE WORKSHOPS LE 15 MAI

{ newcrafts }

April Wensel
fondatrice de Compassionate Coding
@aprilwensel

Alberto Brandolini
créateur de l'Event Storming
@ziobrando

Emily Bache
auteur de The Coding
Dojo Handbook @emilybache

Brian Marick
signataire du manifeste Agile
et auteur @marick

et bien
d'autres...

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

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

{ TECHNOLOGIES }
{ PRATIQUES }
{ ARCHITECTURES }

PASS 2 OU 3 JOURS

toutes les infos sur ncrafts.io
NOTRE CFP EST OUVERT ! cfp.ncrafts.io

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

RETROUVEZ LA RÉDACTION DE PROGRAMMEZ! SUR NEWCRAFTS, BEST OF WEB, DEVFEST LILLE

Mai

Du 15 au 17 : Riviera Dev / Sophia Antipolis

http://rivieradev.fr

16 & 17 : Ncrafts / Paris

Pour la sixième année consécutive, ne manquez pas LE rendez-vous indépendant et international des développeuses et développeurs professionnel.le.s passionné.e.s autour des technologies, pratiques et architectures modernes !
https://ncrafts.io/
@ncraftsConf

22 mai : CloudBees Days / Paris

Lors de cette journée, vous pourrez :

- **Découvrir les dernières technologies,** telles que Kubernetes, qui formeront l'avenir de DevOps, grâce aux experts de CloudBees et Jenkins.
- **Vous renseigner sur les tendances du développement logiciel** auprès d'analystes de premier plan. James Governor, analyste et cofondateur de RedMonk, interviendra notamment.
- **Participer aux ateliers techniques** sponsorisés par CloudBees, Sonatype, GitHub, CyberArk, Gradle et Google Cloud, et découvrir Jenkins X, CloudBees Core for Kubernetes CD, Declarative Pipeline on CloudBees Core, mélangeant les environnements de déploiement continu SaaS avec Jenkins, et bien plus.
- **Faire du networking** avec les experts de DevOps à la Sponsor Expo, pendant le déjeuner et lors de la réception.

Site : https://www.cloudbees.com/cloudbees-days#paris

23 & 24 : Mixit/Lyon

https://mixitconf.org/fr/

Juin

4 : Paris Container Day/Paris

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

5 : Google Cloud Summit Paris

Venez découvrir la plateforme cloud de Google le 5 juin prochain à la Porte de Versailles. L'éditeur fera les choses en grand : +30 sessions sur tous les sujets chauds du moment, 4 salles en parallèles ! Tous les experts Google et les meilleurs partenaires seront présents sur scène. Bien entendu, de nombreuses entreprises viendront parler de leur projet Google Cloud Platform !
Site : https://cloudplatformonline.com/2018-Summit-Paris-speakers.html

7 : Best of Web/Paris

Une conférence qui réunit le Best Of des présentations données pendant l'année, sur une journée à Paris ! Les organisateurs de meetups se réunissent pour proposer une journée avec les meilleurs talk qui ont été donnés dans leurs meetups respectifs. Contrairement à ce que le nom pourrait laisser entendre, on n'y parle pas exclusivement de Web. On y parle de tous les sujets présents parmi les meetups fédérés, dont Human Talks, Duchess France, Scala UG, Software Crafters, ... qui ne sont pas des communautés spécialisées dans le Web.

Venez échanger avec les communautés ! +500 personnes attendues.

Pour accéder aux formations du jeudi, il faut avoir un billet pour la conférence du vendredi et s'inscrire pour une somme symbolique de 10€ afin de réserver sa place. Il y a au maximum une quinzaine de participants pour que le formateur puisse travailler dans les meilleures conditions.

Les places sont relativement accessibles pour une conférence : 50€ le billet regular, et moins encore pour les billets early.

Site : http://bestofweb.paris

11 au 14 juin : INFORSIF/Paris

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

11-13 juin : OW2 Con'19/Paris

La conférence annuelle OW2 est une rencontre internationale d'experts, d'architectes, de développeurs et de chefs de projets open source. **Communauté open source internationale et indépendante,** OW2 est dédiée au développement de logiciels d'infrastructure de qualité industrielle. OW2 regroupe des entreprises, des collectivités et des organismes de recherche de premier plan dont Orange, l'Inria, la Mairie de Paris, Cap Gemini et Airbus Défense.

Le thème central de OW2con cette année est : **"Open Source : vers la maturité industrielle"**. Alors que l'open source se généralise, les développeurs informatiques, les fournisseurs, les utilisateurs et même les organisations open source telles qu'OW2 doivent s'adapter. Les fusions à coups de milliards de dollars annoncées en 2018 démontrent que l'open source atteint sa maturité. Les projets open source deviennent de plus en plus concurrentiels et de plus en plus critique et tous les développeurs, fournisseurs et utilisateurs informatiques doivent avoir une stratégie open source.

Plusieurs thèmes technologiques seront couverts lors de la conférence : *les applications d'entreprise, la sécurité, l'accessibilité, l'open cloud, l'IoT, la blockchain, l'intelligence artificielle, la qualité et les tests logiciels, les modèles économiques et la gouvernance des logiciels libres, open source dans les grandes villes.*

OW2con est ouvert à tous, l'évènement est gratuit et les conférences ont lieu en anglais.
Site : https://www.ow2con.org/view/2019/

MEETUP PROGRAMMEZ! #5

SPÉCIAL

programmation quantique

Tout savoir ou presque sur l'informatique quantique

25 juin à partir de 19h

Où

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

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

Inscription et informations sur **www.programmez.com**

Meetups organisés par [Programmez!] Le magazine des développeurs

14 : DevFest Lille

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

24 au 28 juin : COMPAS 2019/Bayonne

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

27 & 28 : Sunny Tech/Montpellier

Plusieurs conférences et ateliers en parallèle sur 2 journées, sujets transverses allant des outils, méthodologies de conception et développement logiciel jusqu'à la gestion de produit et notamment l'expérience utilisateur.
https://sunny-tech.io

27 : JHipster Conf / Paris

Ne ratez pas la 2ème édition de la JHipster Conf le 27 Juin à Paris ! Cette année, la JHipster Conf possédera deux tracks (français / anglais) ! Les plus grands contributeurs et passionnés de la communauté JHipster seront présents :)

Les speakers : Julien Dubois, Deepu K Sasidharan, Pia Mancini, Matt Raible, Josh Long…
Site : https://jhipster-conf.github.io/

Septembre

16-17 : Agile en Seine 2019/Paris

Agile en Seine se veut une conférence multipratique, ouverte à tous, dont l'objectif est de mettre en avant toute la diversité de ce que l'on nomme communément « Agilité » : du Scrum à l'agilité à l'échelle, en passant par le Design Thinking, le Kanban, le Lean Startup, le Lean, l'entreprise libérée, l'Holacracy … Pour cette édition 2019, nous vous proposons un nouveau format, sur 1,5 jour, le 16 septembre après-midi et le 17 septembre 2019.
La première demi-journée sera consacrée à l'animation d'ateliers de mise en œuvre des pratiques Agile et la seconde journée à des conférences de partage et des retours d'expérience.
Site : https://www.agileenseine.com

Octobre

3 : DevFest Toulouse 2019/Toulouse

Le DevFest Toulouse aura lieu, pour sa 4e édition, le 03 octobre 2019 au Centre des Congrès Pierre Baudis de Toulouse et réunira 900 développeurs, personnes travaillant dans les métiers techniques de l'informatique et étudiants. En attendant le jour J, le CFP (Call For Papers / Appel à Orateurs) du DevFest Toulouse est désormais ouvert ! Cette année, vous pouvez soumettre du 25 mars jusqu'au 25 mai (2 mois).À mi-CFP, une présélection de nos "early speakers" sera effectuée, ne tardez donc pas à soumettre vos propositions ! ;) La délibération finale se fera ensuite pendant le mois de juin.
Site : https://devfesttoulouse.fr/fr/

Novembre

15 : BDX I/O/Bordeaux

Site : https://www.bdx.io/

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

CONFÉRENCES ORGANISÉES PAR XEBIA

27 juin : 2e édition du DataXDay

=> https://dataxday.fr/

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

7-8 Octobre : 4e édition de la FrenchKit

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

28 novembre : XebiCon, Build The Future => https://xebicon.fr/

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

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 pouvons que soutenir !
Pour en savoir plus : https://gcc.prologin.org/

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

Offres 2019

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

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

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

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

** Offre limitée à la France métropolitaine*

Nos classiques

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

** Tarifs France métropolitaine*

Abonnement numérique

PDF ..... 35€
1 an - 11 numéros

Option : accès aux archives 10€

Souscription uniquement sur
www.programmez.com

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

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

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

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

☐ Vidéo : Symfony 3
☐ Vidéo : Raspberry Pi

PROG 229
Valable jusqu'au 31 mai 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**

Roadmap 2019 des langages

Chaque mois, Programmez! vous propose la roadmap des sorties des langages, frameworks, etc.

DÉJÀ DISPONIBLE

React 16.8 : sortie en février. La version se caractérise par l'arrivée des hooks. Les hooks sont un mécanisme logiciel bien connu. Ce bon vieil Emacs fonctionne sur le principe des hooks ;-) Ici, les hooks permettent d'utiliser l'état ou d'autres fonctionnalités de React, sans écrire de classes. Le développeur peut aussi construire ses propres hooks pour partager une logique dynamique entre les composants (comme on peut le faire entre des modules Drupal par exemple).

Visual Studio 2019 : la dernière version de l'IDE de Microsoft est désormais disponible en version finale.

SWIFT 5 : depuis fin mars, la v5 du langage est disponible. Cette version est cruciale pour le langage car il s'agira de la première version à stabiliser l'Application Binary Interface (ABI). L'ABI décrit l'interface entre les applications et l'OS, entre l'application et les bibliothèques, etc. Cette stabilité ABI permettra d'avoir une compatibilité entre les applications et les bibliothèques, utilisant différentes versions du langage. La v5 promet aussi de réduire la taille des binaires générés. On aura droit à un nouvel attribut (@dynamicCallable), des nouveautés sur la gestion des packages ou encore dans le compilateur.

Java 12 : disponible depuis mars.

Kotlin 1.3.30 : il avait été demandé à la communauté d'évaluer plusieurs éléments ; vue séparée des variables dans la fenêtre outil, trace Async dans les coroutines, mode interactif pour les fichiers Scratch, possibilité de faire des commentaires TODO sur plusieurs lignes. Tous les détails : https://blog.jetbrains.com/kotlin/2019/04/kotlin-1-3-30-released/

Rust 1.34.0 : cette nouvelle version apporte les registres alternatifs (cargo), le support de ? dans les tests de documentation, des améliorations diverses notamment sur TryFrom. fn before_exec est déprécié et il est remplacé par unsafe fn pre_exec

1ER SEMESTRE 2019

Angular 8

Date de sortie : mai.

Les principales nouveautés attendues / annoncées : Angular arrive très vite et les versions se succèdent à une cadence élevée, tous les 6 mois environ. La v8, qui arrive maintenant, introduit plusieurs améliorations :

- Sur la CLI : possibilité d'utiliser ES5 et ES2015+ avec une amélioration sur le chargement et le build du code JS ;
- Pré-version d'Ivy ;
- Web Worker : amélioration sur les performances et la parallélisation des applications ;
- Mise à jour des dépendances, notamment outils.

Sur la compatibilité, attention : seuls les codes provenant d'Angular 7 sont pris en compte (dans la note officielle).

Et aussi : retrait de @angular/http, ngcc -make disponible, diverses corrections de bug. Les équipes préviennent aussi d'un changement provocant une non-compatibilité de code : rajouter impératif de @blazel/typescript dans le package.json

Au-delà : la v9 est prévue pour octobre – novembre.

SWIFT 5.1

Date de sortie : printemps.

Les principales nouveautés attendues / annoncées : ce sera la première mise à jour de SWIFT 5. Cette version doit apporter des corrections de bugs et terminer la stabilité des librairies et des modules. La stabilité de l'ABI est désormais en place, reste à la renforcer et à la confirmer avec les prochaines versions. On doit s'attendre à quelques changements dans ce que les équipes appellent la « module stability » comme l'apparition du .swiftinterface pour le fichier d'interface à la place de .swiftmodule. Quelques éléments ici : https://forums.swift.org/t/plan-for-module-stability/14551

Attention : la stabilité ABI n'est pas encore totalement disponible sur les plateformes non-macOS.

Au-delà : la prochaine version majeure devrait être la v6.

Rust 1.35

La version 1.35 est attendue pour fin mai. La 1.36 pour début juillet. Pour suivre la roadmap : https://forge.rust-lang.org

Ruby on Rails 6

Date de sortie : fin avril (sous réserve).

Les principales nouveautés attendues / annoncées : parallèlement au langage, Rails continue à évoluer de son côté. Les grosses nouveautés attendues concernent les actions sur les mailbox, les évolutions sur les fonctions de tests, particulièrement sur le testing action cable et les tests parallèles. Pour la migration, il faudra partir d'une version 5.2 pour faire la transition.

2ER SEMESTRE 2019

React 16.9

Date de sortie : sans doute durant l'été.

Les principales nouveautés attendues / annoncées : en attendant la prochaine version « majeure », React 16.8 est régulièrement mis à jour. La version 16.8.6 a été déployée fin mars avec différents bug fix.

Pour suivre les sorties : https://github.com/facebook/react/releases .

Kotlin 1.3.40

Date de sortie : été ?

Les principales nouveautés attendues / annoncées : le développement de cette nouvelle version est en cours. Nous savons d'ores et déjà que les appels trimIndent et trimMargin seront traités par le compilateur et non le runtime. Plus de détails dans les semaines à venir.

Goland 1.13

Date de sortie : août/septembre.
Les principales nouveautés attendues / annoncées : cette version doit activer le mode module par défaut (le mode par défaut d'auto à activé) tout en déconseillant le mode GOPATH. On aura aussi des nouveautés sur les génériques, la gestion des erreurs.
Au-delà : Go 2 sera LA grosse évolution du langage Go même si les équipes ne veulent pas faire une version de rupture, mais des évolutions au fil des versions. Un des changements sera la manière de définir les évolutions et comment la gouvernance fonctionne. L'idée est que la communauté soit plus impliquée. La compatibilité avec Go 1.x sera un des aspects cruciaux. Pour le moment, le planning de Go2 reste à préciser.

C# 8.0

Date de sortie : été / automne.
Les principales nouveautés attendues / annoncées : cette nouvelle évolution du langage phare .Net doit arriver avec .Net Core 3.0. Parmi les nouveautés attendues :

- Les types de références nullables doivent en finir avec les exceptions null. Pour cela, elles vous empêchent de mettre null dans des types de référence ordinaire comme string. Par défaut, ces types seront non nullables ! Cela pourrait avoir un impact sur les codes existants ;
- Les flux asynchrones ;
- Types range et index ;
- Expressions Switch.

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

Python 3.8

Date de sortie : octobre
Les principales nouveautés attendues / annoncées : plusieurs nouveautés devraient plaire aux développeurs. Tout d'abord, on bénéficie du nouveau paramètre pythonpycacheprefix. Il permet d'utiliser le cache bytecode dans une branche séparée du système de fichiers parallèle. Il sera à préférer au __pycache__ présent dans les sous-répertoires des dossiers sources. On disposera aussi de la méthode as_integer_ratio(), dans le type int. Le contrôle-C, SIGINT, sera modifié pour éviter les problèmes liés à l'exception KeyboardInterrupt. Plusieurs modules auront droit à des améliorations comme asyncio avec ProactorEventLoop ou encore dans gc avec de nouveaux paramètres dans le get_objects().
Pour plus de détails : https://docs.python.org/dev/whatsnew/3.8.html .
Au-delà : en toute logique, la prochaine version majeure sera la 3.9. Pour le moment, pas grand-chose n'a été communiqué. Cette version ne devrait pas arriver avant 2021.

PHP 7.4

Date de sortie : décembre (?).
Les principales nouveautés attendues / annoncées : la 7.4 doit apporter le préchargement (preloading), au cœur de PHP pour pouvoir améliorer les performances. Ce mécanisme permet de charger l'ensemble des fichiers PHP dès le démarrage et ainsi améliorer les accès et assurer une disponibilité constante de ceux-ci. Par contre, en cas de changement des fichiers, il faudra redémarrer le serveur. On notera aussi la disponibilité des propriétés typées, l'extension FFI (Foreign Function Interface) pour appeler du code C, le nouvel opérateur Null Coalescing, l'apparition de nouvelles méthodes (__serialize & __unesrialize).
On notera aussi le retrait de PEAR ou la dépréciation de ext/wwdx.
Attention : la 7.4 n'est pas encore totalement figée. Des changements peuvent intervenir.
Au-delà : après la 7.4, difficile d'y voir clair. Il était question d'une v8. Pour le moment, rien de très précis.

Ruby 2.7

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

COURANT 2020

C++20

Date de sortie : 2020.
Les principales nouveautés attendues / annoncées : les spécifications de C++20 sont désormais figées, et un premier document complet sera disponible cet été. Cette future version du langage mettra en avant deux nouveautés : les modules et les coroutines. Les modules constituent une nouvelle alternative aux fichiers d'en-tête qui apportent un certain nombre d'améliorations clés, notamment en isolant les effets des macros et en permettant des compilations évolutives, explique Herb. Il ajoute qu'en 35 ans c'est première fois que C ++ ajoute une nouvelle fonctionnalité permettant aux utilisateurs de définir une limite d'encapsulation nommée.
Les coroutines sont elles aussi une fonctionnalité à remarquer. Une coroutine est une unité de traitement qui s'apparente à une fonction (ou routine), avec cette différence que si une sortie du corps d'une fonction met fin à l'exécution de celle-ci, la sortie de la coroutine suspend seulement son traitement qui peut ensuite reprendre, l'état du traitement à la sortie étant conservée. Une coroutine peut être vue comme un morceau de programme qui conserve son état, mais qui n'a pas de thread d'exécution. Les coroutines peuvent notamment être utilisées pour des itérateurs et des générateurs.

LTS

Il n'est pas toujours facile de comprendre le support d'une version, et donc de sa durée de vie officielle et des mises à jour qui y seront appliquées. Aujourd'hui, on parlera souvent de LTS et de non-LTS. LTS signifie support long terme. C'est-à-dire que cette version est supportée officiellement durant x années et recevra les mises à jour et les patchs de sécurité nécessaires. Une version non-LTS a une durée de vie très courte, quelques mois. Seules les versions issues d'une forte communauté, d'une fondation, d'un éditeur sont LTS. Chacun fait un peu ce qu'il veut sur le support. Angular possède un cycle de 6 mois pour versions majeures. Le support d'une version.

Versionning

On parlera de version majeure et de versions mineures. Ces dernières sont souvent des versions de bug fix, de sécurité, introduisant peu ou pas de nouveautés. React explique ainsi la structure des versions avec x.y.z :
X = une version majeure introduisant une rupture ;
Y = nouvelle fonction, version mineure (ex. : 15.6) ;
Z = bug fix. On corrige les bugs, les problèmes importants.

Notes = RTFM

N'oubliez pas de lire les notes de version (release notes). Elles indiquent les changements, les modifications, les nouveautés, la migration entre les versions.

François Tonic

MSI Meg X299 Creation : une carte mère au top !

MSI est une référence en cartes mères pour tous les passionnés de PC en kit et les gamers. La Meg X299 Creation est un véritable monstre taillé pour les plus exigeant(e)s. Dès le déballage, elle impressionne par sa qualité de finition, le socket supportant les processeurs Intel Core série X et l'énorme bloc connectique !

La Meg X299 Creation est une des cartes mères références du marché. D'emblée, elle en impose par sa taille et l'ensemble de sa connectique, ainsi que les nombreux connecteurs disponibles. Le bloc connectique est énorme. Il comprend : port PS/2 (si, si), USB 2, USB 3.1 Gen 1, USB-C, 2xEthernet (2,5 Gb / 1 Gb), connecteurs antenne WiFi, ports audio.

Deux manques sont à noter : la faiblesse des ports USB-C (1 seul) et l'absence de Thunberbolt 3. La carte Thunderbolt M3 (MSI) permet de rajouter 2 ports Thunderbolt 3 à 40 Gb. Bien entendu, compatible USB-C à 10 Gb. D'autre part, cette carte dispose de 2 connecteurs DisplayPort 1.2 pouvant supporter 2 écrans 4K. La carte est petite mais occupe un connecteur PCIe.

Prévoyez aussi une solide alimentation.

Nous avons utilisé un bloc Corsair 450W. Les premiers branchements sont très simples : le gros connecteur ATX, le CPU_PWR et le PCIE_PWR. Pour ce dernier, attention : le connecteur est horizontal et non vertical et si le bloc d'alimentation est trop large, cela peut gêner le connecteur. Et on connecte le câble du bouton d'alimentation du boîtier. Vous pouvez aussi installer la carte PCIe portant le gros ventilateur, ne pas oublier de le connecter à l'alimentation. Si vous voulez installer du SSD M.2, il faut retirer le dissipateur thermique qui le masque.

Nous avons beaucoup apprécié la carte et son évolutivité. Bien entendu, la qualité des composants jouera dans la puissance que vous aurez.

Prix : -600 € la carte seule

Les +

- Finition
- Jusqu'à 128 Go de RAM
- Ethernet 2,5 Gb
- SSD NVMe format M.2
- 8 connecteurs SATA
- Documentation

Les –

- Le connecteur power PCIe trop près du bloc d'alimentation
- Bien choisir son boîtier
- Attention au refroidissement
- De la patience pour tout monter
- Thunderbolt 3 absent par défaut
- Les jeux de lumières (bon ok c'est sympa mais ce n'est pas ça qui fait tourner les apps)
- Pas d'accès direct aux ports M.2 (sous un bouclier)

OPTIX MPG27 CQ : L'ÉCRAN INCURVÉ SELON MSI

Seulement 27'' ? Honnêtement, vue la qualité de l'écran, 27'' suffira dans la plupart des usages que l'on en fera, c'est à dire, avant tout le gaming. Son design est élégant et sobre. Le pied en impose par taille et son poids. On installe très facile la dalle.

L'image est très belle, vive avec un joli contraste même en ambiance sombre. L'angle de vue est très bon. Surtout, l'écran se règle facilement en hauteur. Ensuite ce sera une question de balances de couleurs. Pour animer l'ambiance, des LED s'éclairent en bas mais cela reste discret.

La dalle, de technologie WQHD (là aussi suffisant à beaucoup d'utilisateurs), offre une résolution de 2560x1440, correcte mais on aurait pu espérer mieux. Par contre, elle possède un taux de rafraichissement élevé, 144 Hz et un temps de réponse moyen de 1 ms. Nous n'avons pas constaté de sauts d'images ou de problèmes de fluidité même en usage intensif. La qualité de la connectique et du GPU jouera pour beaucoup. En contexte jeu, vous pourrez aligner trois écrans. L'application Gaming OSD offre une configuration poussée.

Côté connectique, on dispose de : 2xHDMI 2, 1xDisplayPort 1.2, sortie casque / entrée micro, 2xUSB 3.1 Gen1 A et 1xUSB 3.1 Gen1 B. On pourra regretter l'absence de l'USB-C et du Thunderbolt 3. Dommage que la connectique vidéo se situe sous l'écran et non à l'arrière. Pas pratique pour brancher et débrancher les câbles.

Côté tarif, nous sommes dans la fourchette haute, +/- 500 €. On peut trouver des écrans du même type moins cher, avec de meilleures dalles mais pas toujours avec une finition à la hauteur. MSI fournit ici un écran de grande qualité, très bien fini, avec des fonctions souvent bien pensées.

Tarif : 480-500 €

macOS & le trim

Le TRIM est un mécanisme propre aux disques SSD pour optimiser l'usage des mémoires Flash et améliorer la durée de vie. Les disques Flash internes sont TRIM par défaut, pas de souci. Par contre, pour les disques externes c'est une autre histoire.

Pour pouvoir utiliser TRIM sur un Flash externe, vous devez obligatoirement utiliser un boitier Thunderbolt 3, la connectique USB 3.1 / C ne le permet plus. Bref, si vous utilisez du NVME (qui offre les meilleurs débits actuels), vous devrez investir dans un adaptateur Thunberolt 3 !

Pack maker : ESP8266 Wemos + Attiny85

VERSION PRINTEMPS 2019

NOUVEAU !

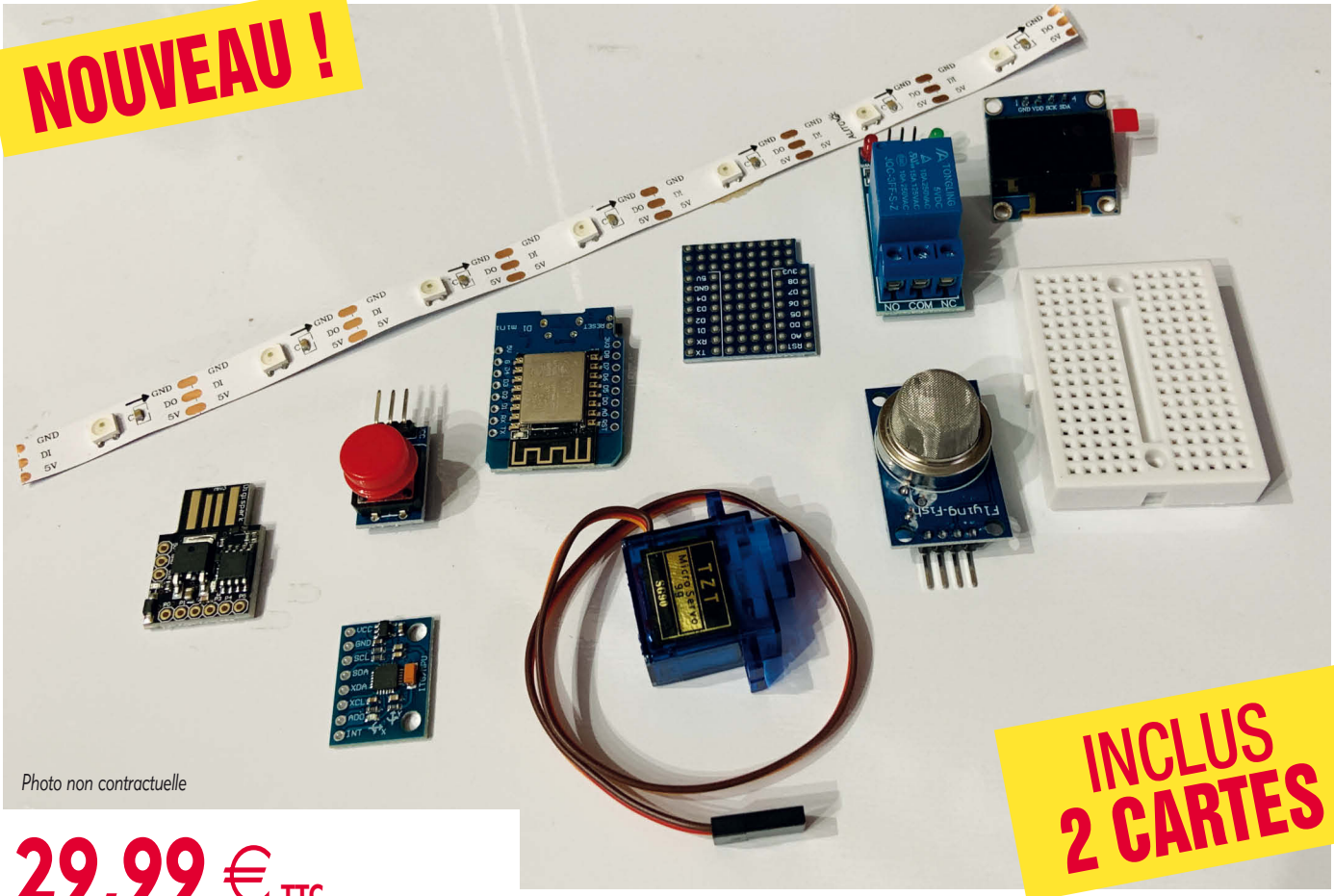


Photo non contractuelle

INCLUS 2 CARTES

29,99 € TTC

(+ 3 € de frais de port)

Attention ! Quantité limitée
Livrée sans documentation

Les composants

Wemos D1 mini (ESP8266)
Attiny85 (compatible Arduino)
Shield de prototypage Wemos
Mini planche à pain
Capteur de gaz MQ2
Bouton poussoir
Ecran OLED 0,96"
Relais
Micro servo-moteur SG90
Capteur GY-521 (gyroscope + accéléromètre 3 axes)
1 mini bande LED configurable
Capteur d'humidité

☐ **Pack maker ESP8266 Wemos + Attiny85 :**
29,99 € (+3 € de frais de port) = 32,99 €

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

PROG 229

☐ 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

Nutanix met en avant ses solutions pour l'IoT au .Next Paris

Lors de son évènement parisien Next On Tour, l'éditeur a rassemblé plus de 1 000 personnes au Palais des Congrès de la Porte Maillot à Paris.

Nutanix prend de plus en plus d'ampleur sur le marché français. Le 9 avril dernier, le spécialiste du cloud computing d'entreprise a rassemblé près de 1 000 personnes au Palais des Congrès de la Porte Maillot contre 350 l'an dernier. Clients, partenaires et curieux ont ainsi massivement convergé vers l'évènement pour en apprendre plus sur les produits de l'éditeur américain dont plusieurs étaient en démonstration.

Le premier d'entre eux, et le plus visible, était Xi IoT, avec une démonstration mêlant une caméra à une solution de reconnaissance faciale, capable de détecter les émotions des passants. « Avec Xi IoT, nous sommes en mesure de délivrer tout un panel de services d'infrastructure et applicatifs sur le Edge », expliquait d'ailleurs Satyam Vaghani, vice-président et directeur général de Nutanix en charge de l'IoT et de l'IA lors de la conférence. Concrètement, l'architecture de Xi IoT permet d'allouer de la puissance de calcul et des solutions analytiques au niveau du Edge mais aussi d'administrer les flux de données pour déterminer lesquelles doivent être traitées en bordure ou sur une infrastructure centralisée. À cet effet, Xi IoT permet d'ailleurs la mise en place d'IoT Gateways et de data pipelines pour assurer un transport sécurisé des données.

La solution Era, pour la gestion des bases de données, présente, elle aussi, bien des avantages pour les développeurs. D'abord pensée pour les DBA, elle permet notamment de cloner, de dupliquer et de déployer des bases de données en quelques clics sur différentes infrastructures. Fonctionnant pour l'instant avec des solutions SQL Server, Oracle, Postgres SQL et Maria DB, elle permet ainsi de copier des bases de données avec leur VM et leurs paramètres sur divers environnements, on premise ou sur le cloud, par exemple, pour le test et le développement de nouvelles applications.

D'autres produits étaient également en démonstration, notamment la solution de micro segmentation Flow ainsi que Frame, qui permet le déploiement d'application en mode Desktop-as-a-Service depuis le cloud. « Notre ambition est aujourd'hui de rendre l'infrastructure invisible et transparente pour les utilisateurs », rappelle Satyam Vaghani, mettant en avant l'impact que peuvent représenter les solutions précédemment évoquées.

Les projets des clients étaient par ailleurs largement mis à l'honneur pendant l'évènement notamment Compass avec son système de caisse intelligente intégrant de l'IA — avec la startup française Deepomatic — et du Edge pour automatiser l'encaissement des plateaux-repas (voir encadré), ainsi que la Société Générale, l'ICAM, ASL Airlines.

L'évènement était aussi l'occasion pour Nutanix de mettre en avant son écosystème de partenaires technologiques, massivement représenté notamment avec BitDefender, Cloudian, ControlUP, Exagrid, Hycu, Interxion, Juniper, Lenovo, Mellanox, Rubrik, Suse, Unitrends, Veritas et Wipro, tous sponsors du .Next On Tour Paris. Au cours de l'évènement, Nutanix a par ailleurs annoncé la signature d'un nouveau partenariat mondial avec HPE (voir encadré).

HPE et Nutanix se retrouvent autour du cloud hybride

HPE et Nutanix enterrent la hache de guerre autour de l'hyperconvergence. Pourtant concurrent depuis l'acquisition de Simplivity par HPE en 2017, les deux firmes viennent d'annoncer un partenariat mondial pour la distribution des solutions de Nutanix via GreenLake, l'offre de consommation d'infrastructure on premise as-a -Service d'HPE. Les clients de GreenLake vont ainsi pouvoir accéder aux offres de Nutanix sur leurs solutions de serveurs HPE Proliant et Apollo, y compris sa solution maison d'hypervision gratuite, AHV. Dans le cadre de ce même partenariat, Nutanix acquiert par ailleurs le droit de vendre directement ses solutions en bundle sur des serveurs HPE qui rejoint ainsi Dell, Lenovo, Fujitsu ou encore IBM. L'ambition affichée derrière ce partenariat est de faciliter pour les utilisateurs de solutions HPE le passage au cloud hybride en profitant des passerelles qu'apportent les solutions Nutanix entre infrastructures on-premise, cloud privé et cloud public. Ce partenariat va aussi permettre à HPE de se renforcer sur un marché du HCI où il est devancé, dans l'ordre par Dell et Nutanix d'après IDC et Gartner.

CAS CONCRET : Mise en place du Magic Mirror avec Xi IoT

Sur la scène du .Next On Tour, Christophe Jauffret, architecte solution de Nutanix a fait la démonstration de la mise en place du Magic Mirror en s'appuyant sur Xi IoT. Voici le compte rendu de comment il s'y est pris, photos à l'appui.

Avec Xi IoT, l'utilisateur est d'abord capable de repérer les différents terminaux et capteurs connectés sur le Edge via la console. 1

Christophe Jauffret sélectionne alors le boîtier Paris, correspondant à une machine installée sur l'évènement et identifie ensuite la source de l'image, ici une webcam braquée sur la salle. 2

Il choisit ensuite une fonction correspondant à un bout de code à appliquer à cette source d'image, toujours via l'interface Xi IoT. Dans l'exemple montré sur scène, Christophe Jauffret choisit une fonction de reconnaissance d'objet. 3

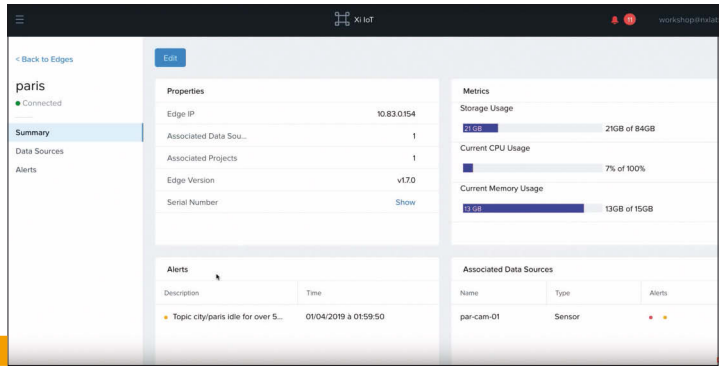
S'ouvre alors une page de code, ici du Python, qui permet de travailler en mode serverless directement sur la fonction. À noter qu'il est possible d'intégrer directement des frameworks, comme TensorFlow, à la plateforme, toujours en mode serverless. 4

Une fois la fonction validée, et accessible avec d'autres fonctions précédemment développées via un menu déroulant, il choisit un data pipeline sur lequel l'appliquer. 5

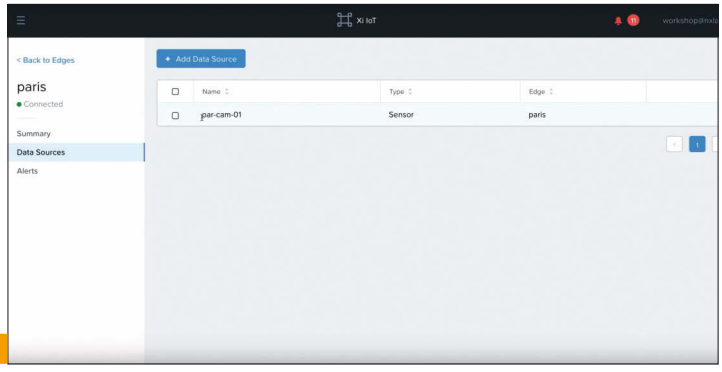
La mise en application est immédiate et le flux vidéo est directement traité sur le Edge. 6

Il est ensuite possible d'ajouter d'autres fonctions au flux vidéo, dans l'exemple une fonction de reconnaissance d'émotion. 7

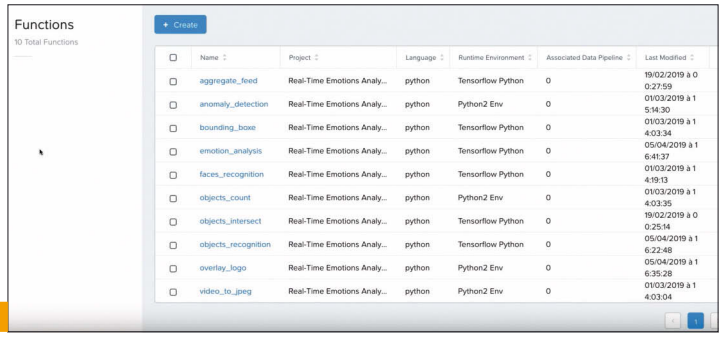
Comme la plupart des produits de Nutanix, Xi IoT est développé avec des API Rest et il est donc possible de l'intégrer à d'autres solutions ou au sein des pipeline CI/CD.



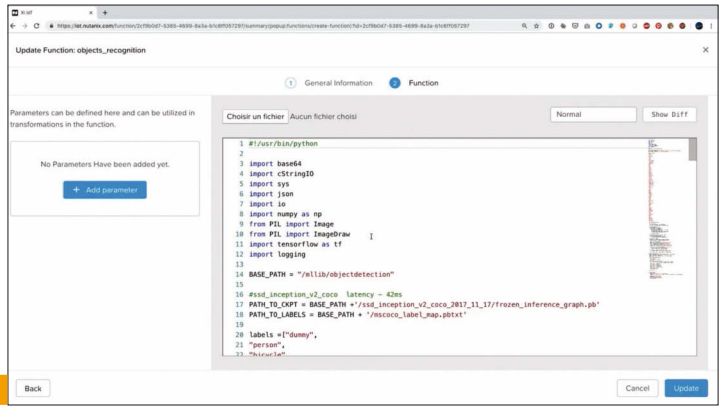
1



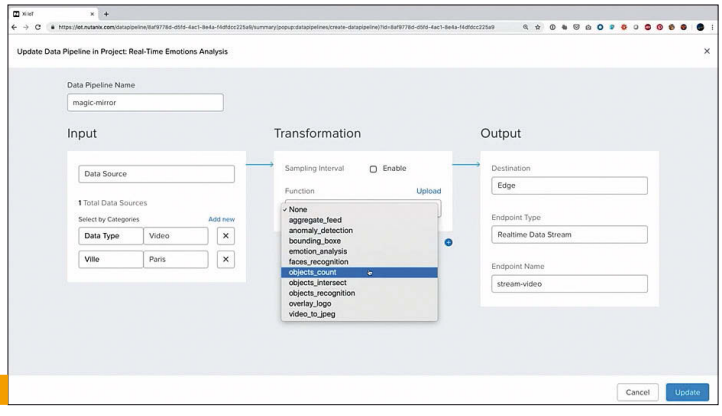
2



3

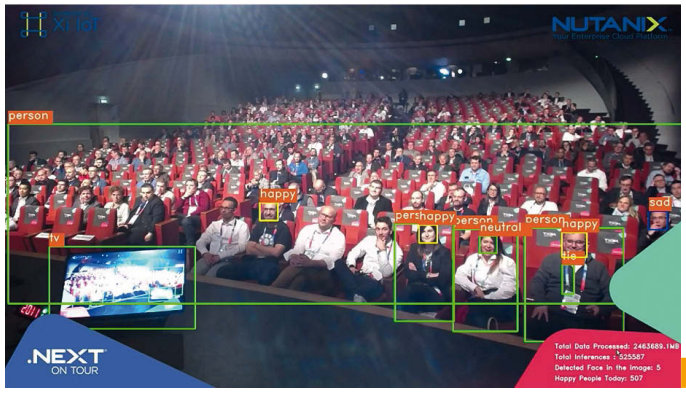


4



5

6



7

3 Questions à James Karuttykaran

Directeur Technique de Nutanix Europe du Sud

On voit beaucoup de nouvelles solutions arriver chez Nutanix. Comment peuvent-elles aider les développeurs à mettre en place de nouveaux services ?

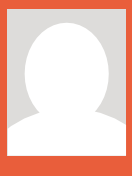
Aujourd'hui, une de nos premières préoccupations concerne la modernisation des applications et nous avons mis en place beaucoup d'outils qui peuvent servir aux développeurs. Nous avons notamment lancé Era, solution qui permet de déployer, de dupliquer, de rafraîchir ou de déplacer rapidement des bases de données. En un clic, il est ainsi possible de copier une base de données, avec ses paramètres et sa VM. Cela permet de faciliter de nouvelles solutions en simplifiant la mise à disposition de copie de base de données. Era peut également servir à migrer les bases de données des environnements de développement vers ceux de production. Parmi les autres produits qui vont faciliter le travail des développeurs, on peut citer Karbon, qui est une distribution certifiée de Kubernetes qui, alliée à Era et Buckets, permet la mise en place d'infrastructures de conteneur entièrement automatisées sur des environnements Nutanix. Cette architecture permet notamment de développer nativement les applications rapidement sur des environnements on premise pour les pousser ensuite sur le cloud sans avoir à en modifier le code, ou l'inverse.

Et qu'en est-il de l'IoT et du Edge computing qui devient un sujet majeur aujourd'hui

Avec notre offre Xi IoT, on propose aujourd'hui tout un ensemble de services qui permettent d'apporter et de déployer massivement de l'intelligence sur le Edge. Les développeurs peuvent accéder à une plateforme de services managés regroupant des conteneurs, des connecteurs IA, ou des data services comme Kafka ou Elastic Search. En s'appuyant sur nos API, ils peuvent développer leurs propres applications sur le Edge en bénéficiant des solutions de management et de gestion du cycle de vie des applications de Nutanix.

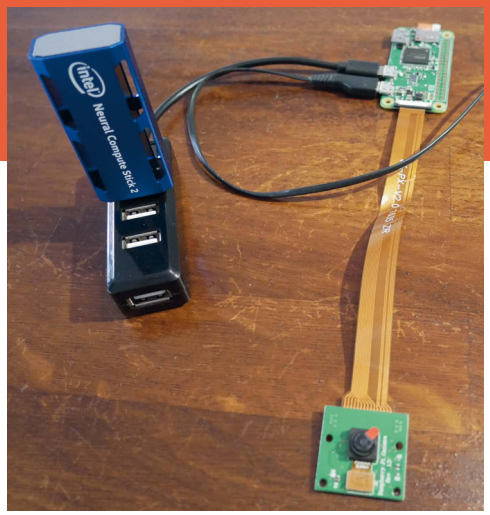
Satyam Vaghani votre vice-président en charge de l'IoT et de l'IA parlait aussi de simplifier la création des applications dans ce domaine.

C'est aussi un de nos objectifs. Pour mettre en œuvre de la reconnaissance faciale au niveau d'une caméra c'est compliqué. Notre idée est de permettre aux utilisateurs de développer des applications IoT et IA plus facilement sans qu'ils aient à rentrer en profondeur ou à maîtriser la technologie. Du coup, nous faciliterons la mise en place des solutions, mais aussi leur déploiement à l'échelle avec nos solutions Nutanix. Mais pour l'instant je ne peux pas en dire plus.



Aymeric Weinbach
Développeur

Intel présente le Neural Compute Stick 2



A l'Intel AI DevCon de janvier dernier, j'ai pu disposer d'un NCS 2 et le tester pour voir comment il se place vis-à-vis de la concurrence. L'idée a l'air séduisante. Dans le monde de l'informatique embarqué de la robotique, si on veut faire de l'IA, il y a 3 options : embarquer une grosse machine avec un gros CPU et un GPU, mais cela va coûter très cher et consommer beaucoup d'énergie. Ce n'est utilisable que pour un gros robot.

Si on veut un petit device qui consomme peu d'énergie, a priori la seule solution est le « cloud » et utiliser un service du type Azure IoT Edge et des services cloud pour déporter le traitement. C'est une excellente solution, mais parfois on n'a pas la possibilité d'avoir une connexion internet suffisamment fiable. Parfois on veut simplement pouvoir profiter des deux comme par exemple une partie traitement qui doit toujours fonctionner même déconnectée et une autre qu'on déporte dans le cloud.

Intel propose avec sa puce Myriad X d'avoir une puissance de traitement pour la reconnaissance visuelle. Le NCS 2 est une clé USB qui contient cette puce pour le prototypage qu'on peut brancher en usb et utiliser directement sur une petite carte type Raspberry Pi, avec un petit processeur.

Branchons le NCS2 sur un Raspberry Pi 3 et voyons s'il tient ses promesses. Est ce qu'on pourrait se passer de cloud pour de la reconnaissance visuelle et avoir l'équivalent de la puissance d'un laptop ?

Il vous faut pour commencer un Raspberry Pi et une caméra Pi pour démarrer.

Sur le Raspberry Pi on utilise une SDcard comme stockage. La première étape est d'installer un Raspbian OS (un Linux Debian adapté au Raspberry Pi). Le plus simple pour ça est d'utiliser NOOBS, un installateur simple pour PC/ MAC qui vous formate correctement la SDcard directement avec l'OS. J'ai installé la version non graphique de Raspbian.

Etape suivante, installer Openvino, le toolkit Intel pour le NCS 2 sur le Raspberry. Pour cela connectons-nous au terminal. Soit vous branchez le Raspberry Pi directement sur un écran, soit vous le branchez sur un réseau filaire et vous vous connectez en ssh au Raspberry Pi (mon option).

Télécharger le toolkit :

```
wget https://download.01.org/openvinotoolkit/2018_R5/packages/l_openvino_toolkit_ie_p_2018.5.445.tgz
```

Décompresser l'archive :

```
tar -xf l_openvino_toolkit_ie_p_<version>.tgz
```

Modifier le script di'nstall avec le dossier local :

```
sed -i "s|<INSTALLDIR>|$(pwd)/inference_engine_vpu_arm|" inference_engine_vpu_arm/bin/setupvars.sh
```

et exécuter ce script pour fixer les variables d'environnement :

```
source inference_engine_vpu_arm/bin/setupvars.sh
```

Les variables d'environnement sont supprimées. A chaque redémarrage pensez à exécuter le script au démarrage.

```
Autoriser l'usb
sudo usermod -a -G users "$(whoami)"
sh inference_engine_vpu_arm/install_dependencies/install_NCS_udev_rules.sh
```

Le sdk openvino contient un exemple de détection de visage qu'on peut utiliser pour notre test.

Commençons par faire le build de l'exemple de détection d'objet.

```
cd inference_engine_vpu_arm/deployment_tools/inference_engine/samples
mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_CXX_FLAGS="-march=armv7-a"
make -j2 object_detection_sample_ssd
```

Télécharger les modèles préentrainés :

```
wget --no-check-certificate https://download.01.org/openvinotoolkit/2018_R4/open_model_zoo/face-detection-adas-0001/FP16/face-detection-adas-0001.bin
wget --no-check-certificate https://download.01.org/openvinotoolkit/2018_R4/open_model_zoo/face-detection-adas-0001/FP16/face-detection-adas-0001.xml
```

Prenez une photo avec la caméra ou récupérez une photo pour un test et tentez :

```
./armv7l/Release/object_detection_sample_ssd -m face-detection-adas-0001.xml -d MYRIAD -i <path_to_image>
```

Il est aussi possible d'utiliser la reconnaissance fournie par opencv avec des modèles pré-entrainés fournis également. Les résultats sont plutôt bons et on arrive à des résultats en termes de puissance comparable pour de la reconnaissance de visage avec un laptop.

QUELQUES AUTRES SOLUTIONS

Le marché des boards IA s'anime brutalement depuis quelques mois. Outre Intel, nous avons deux grandes plateformes :

- NVidia Jetson Nano : une carte complète basée sur Maxwell et ARM A57 4 cœurs. Il dispose d'un port Ethernet 1 Go et d'un connecteur Edge. 109 € si vous arrivez à le trouver.
- Google Edge TPU / Coral : Google propose la Dev Board notamment pour faire du Machine Learning et du prototypage rapide. Il supporte TensorFlow Lite. 149,99 $. Disponibilité limitée pour le moment.

Thomas Thiry
Développeur et scrum master chez Ingenico. Professeur en gestion de projet agile à la haute école EPHEC. Auteur du livre « Les pratiques de l'équipe agile ».

L'approche agile de la programmation

Les développeurs qui veulent adopter l'agilité doivent faire évoluer leur manière de programmer en découpant le travail de leur journée en de très petits éléments qui fonctionnent à tout instant. Ils peuvent alors se concentrer sur des petits objectifs qui ont de la valeur et peuvent être livrés directement au client.

Pratiquement toutes les méthodologies agiles recommandent explicitement aux équipes qui décident de faire évoluer leur manière de fonctionner de ne pas sous-estimer l'importance des bonnes pratiques techniques. Sans cela, il est fort probable qu'elles n'obtiennent pas la flexibilité et les autres avantages que la réorganisation du travail en processus agiles est censée apporter.

Il s'agit principalement des pratiques d'Extreme Programming, mais d'autres éléments en découlent et il est important d'en avoir également conscience et de les prendre en compte.

Développement progressif

Au cœur de cette approche se trouve le principe de base agile de tout faire en petites étapes validées. C'est-à-dire de faire une petite chose à la fois, et d'avoir le moyen de vérifier ce nouvel élément ainsi que tout ce qui a été fait avant. Cela réduit les risques, assure un feedback plus rapide, et permet de commencer à délivrer de la valeur plus vite. **On vise alors toujours plus petit, toujours plus rapide, toujours plus sûr.**

Cette logique a été appliquée aux releases, réduisant les projets de plusieurs années à des déploiements tous les quelques mois ou moins. Le travail de chaque release est lui-même délivré en itérations de plus en plus courtes : un mois, deux semaines, une semaine… Chaque journée est alors elle-même une mini-itération qui commence par un daily stand-up qui définit les objectifs du jour.

Cette approche s'applique aussi au déroulement de la journée, dans le travail de développement lui-même : on vise à effectuer des commits très fréquents, plusieurs fois par jour. Cela apporte évidemment les avantages de ce genre de système (par ex. ne pas risquer de perdre ces changements), mais le véritable objectif est en réalité un changement de dynamique qui aura un impact très important. Cela force, en effet, le développeur à envisager ses changements sur un objectif bien plus proche. Il doit se limiter à des modifications qui ne « cassent » pas le reste de l'application pendant plus de quelques heures ou moins, c'est-à-dire que l'ensemble doit compiler et que tous les tests automatisés doivent passer.

Enfin, le concept de refactoring apporté par Extreme Programming mène à aller encore plus loin et envisager chaque petit changement de code (créer une méthode, renommer une variable, extraire un algorithme dans son propre objet…) comme une boucle qui nécessite de revenir à un état stable où tout fonctionne (compilation et tests) avant de passer au changement suivant qui nous rapproche du produit désiré.

Des outils spécialisés permettent d'améliorer son efficacité dans ce genre de technique. La plupart des IDE (comme Eclipse ou Visual Studio) proposent des fonctionnalités de base pour appliquer les refactorings automatiquement au lieu de s'aventurer dans des modifications manuelles par définition plus hasardeuses. L'objectif est que le développeur se concentre sur le changement/refactoring qu'il veut appliquer (« Je pense qu'on pourrait extraire cette partie dans une classe à part », « Ce nom serait plus explicite pour cette méthode » …) et qu'il laisse la machine le faire de façon correcte et systématique. Les outils de JetBrains (Resharper et Rider) poussent le concept toujours plus loin et regorgent de petites automatisations insoupçonnées qui permettent au développeur de se concentrer de plus en plus sur la réflexion / design et moins sur la syntaxe. Malheureusement le petit effort d'apprentissage nécessaire en rebute plus d'un…

Le développement est également une activité qui doit se faire progressivement.

Le code est ajouté et modifié ligne après ligne, tout en gardant un état fonctionnel et livrable au client. Le produit évolue petit à petit vers le fonctionnement qu'on souhaite obtenir, sans grand bouleversement.

Rapidité

Les méthodologies agiles visent à produire un maximum de valeur pour un coût minimum. Cela veut dire que si le résultat pour l'utilisateur correspond à ce dont il a besoin, la qualité du code n'a pas d'importance en tant que telle.

Cependant, ces méthodologies apportent également une vision à long terme dans laquelle le coût des changements futurs est également à prendre en compte. En effet, selon le code, un changement à faire dans un programme peut être très simple ou très compliqué, prendre du temps et provoquer de nombreux bugs qui vont eux-mêmes consommer beaucoup de temps et d'énergie plus tard.

On comprend alors qu'une attention particulière aux choix faits pendant la programmation peut garantir une livraison des fonctionnalités de façon régulière et éviter une augmentation quasi exponentielle du temps nécessaire comme on le voit trop souvent.

C'est pour cette raison que de nombreux agilistes promeuvent – et développent également – une multitude de principes et bonnes pratiques de programmation.

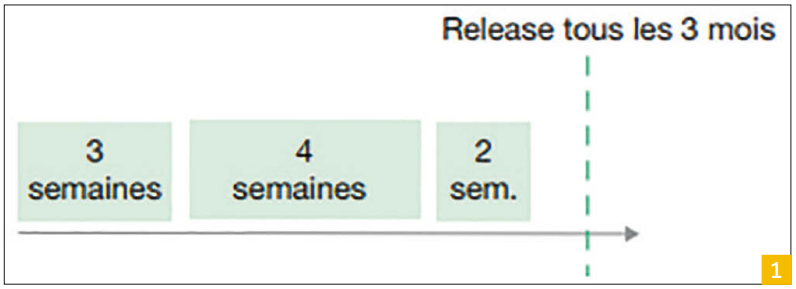
Parmi ceux-ci, on retrouve le clean code, les principes SOLID, les design patterns, don't repeat yourself (DRY), keep it simple, stupid (KISS), you aren't gonna need it (YAGNI)… et bien d'autres. De nombreux ouvrages existent sur ce thème, et le livre de Robert

C. Martin « *Clean Code: A Handbook of Agile Software Craftsmanship* » est une excellente référence pour aborder le sujet.

> ***« Je ne suis pas un programmeur excellent, je suis juste un bon programmeur avec d'excellentes habitudes. »***
> Kent Beck

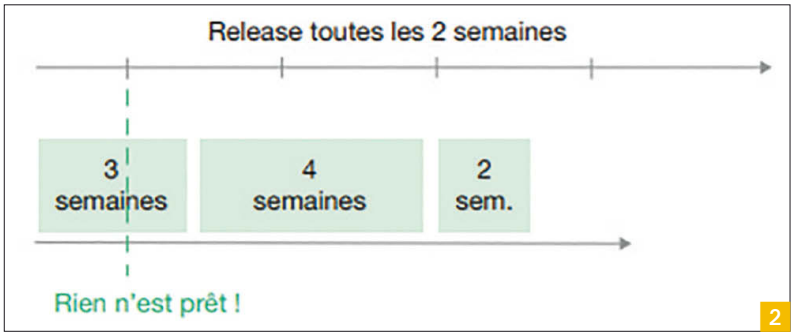
Petites user stories

Pour arriver à délivrer des fonctionnalités de façon régulière comme défini dans le manifeste agile, il est essentiel d'arriver à découper les grosses fonctionnalités en de plus petites qui peuvent être finies et livrées progressivement. Avec la popularité de la livraison continue, des solutions techniques existent pour démultiplier le nombre de déploiements et obtenir ainsi un délai plus court entre le début d'un développement et la livraison effective au client en production. Cependant, si le développement de chaque fonctionnalité prend énormément de temps, on perd une grande partie de l'intérêt et le délai pour le client reste long.
Si, par exemple, on prévoit une release tous les 3 mois et que les fonctionnalités à délivrer représentent chacune plusieurs semaines de travail, on peut espérer que tout soit prêt à être délivré à temps. 1



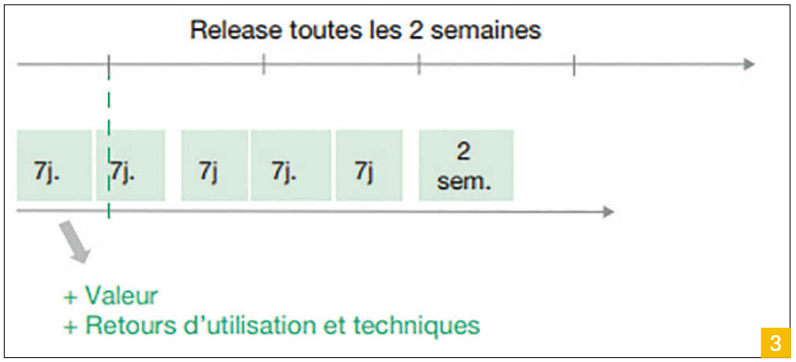
1

Si, par contre, on a la capacité de délivrer toutes les 2 semaines, à la prochaine date de livraison rien n'est prêt, il n'y a rien à déployer. Le client ne reçoit rien, donc aucune valeur ajoutée pour lui pour le moment et aucun retour pour nous. 2



2

Il est possible de résoudre cela si on arrive à découper le travail à faire en plus petites parties **qui ont déjà de la valeur ou un intérêt pour le client.**
Si on découpe nos 3 fonctionnalités, peut-être qu'une première partie pourra déjà être livrée lors de la première release prévue dans seulement 2 semaines. 3



3

L'idée est bien de fournir une véritable fonctionnalité à l'utilisateur, pas de déployer un morceau de code qui ne change pas grand-chose ou ne sert encore à rien. Cela peut sembler difficile à première vue, mais en se forçant, on peut voir une version simplifiée de la fonctionnalité demandée qui pourra être déployée dans un premier temps, en attendant que les améliorations soient apportées par la suite.

La question à se poser face à une fonctionnalité est :

« Quel est le plus petit changement qu'on peut faire et qui apporte déjà de la valeur au client ? »

Pour bien comprendre de quoi on parle et ne pas rester sur des principes théoriques, voici un exemple réel. Dans l'industrie du paiement électronique, mon équipe a reçu la demande de réaliser un envoi d'email quotidien à chaque marchand avec un fichier Excel reprenant l'ensemble des paiements réalisés pendant la journée. Il fallait que chaque marchand puisse configurer cet envoi de rapport pour indiquer notamment les adresses de destination et la fréquence d'envoi (quotidien, hebdomadaire, mensuel…). Nous aurions pu délivrer l'ensemble de la fonctionnalité d'un coup, mais cela allait prendre un certain temps car nous avions plusieurs limitations techniques à surmonter dans le système existant. Ce n'aurait pas été un problème si les releases n'étaient pas prévues que tous les 2 mois, avec la prochaine opportunité de déployer qui arrivait à grands-pas et certains clients très impatients d'obtenir ces nouveaux rapports. Il nous fallait donc une solution rapidement. Fort heureusement, notre approche agile nous a permis d'identifier des sous-fonctionnalités et de construire le système progressivement. On commence toujours par la même question : « Quel est le plus petit changement qu'on peut faire et qui apporte déjà de la valeur au client ? » L'envoi d'email semblait évidemment indispensable, avec une exécution chaque nuit. Par contre, notre système d'envoi d'email

ne supportait pas encore plusieurs pièces jointes, on décida donc d'envoyer plusieurs emails si plusieurs fichiers étaient présents. La flexibilité de changer la fréquence via une interface web n'était pas non plus indispensable, exit l'interface web. Cela nous a permis de délivrer en production cette première fonctionnalité très basique. Il n'était même pas possible pour un marchand de l'activer lui-même puisqu'aucune interface web n'était disponible. Mais nous pouvions l'activer nous-mêmes pour nos quelques marchands privilégiés qui désiraient ardemment recevoir ces rapports. Et la configuration qu'ils voulaient était justement un envoi quotidien, donc pas besoin de flexibilité à ce niveau-là. Au final, pour ces quelques clients la valeur était là, à un coût très bas et dans un délai plus rapide que ce qu'on aurait proposé autrement. Un autre avantage insoupçonné fut que l'absence d'interface web visible pour les utilisateurs nous a permis de configurer le système avec nos propres adresses email pour des gros clients et d'utiliser la fonctionnalité en production avant eux, avec de vrais volumes de données. On a pu ensuite ajouter les autres possibilités progressivement : d'abord une interface pour que le marchand puisse activer le système et indiquer l'adresse email de destination, ensuite regrouper les emails en un seul avec plusieurs pièces jointes, puis proposer plusieurs fréquences possibles…

Quelques bonnes pratiques à retenir :

- Avant de commencer une fonctionnalité, se demander quelle est la plus petite chose qui pourrait déjà servir à au moins un utilisateur (pas forcément tous !).
- Découper chaque développement en une série de commits qui compilent et passent les tests, et cela plusieurs fois par jour.
- Investir dans la qualité (vérifier que tout marche de façon systématique et le plus tôt possible, clarifier le code pour faciliter les prochains changements…) permet de maintenir un rythme de développement constant (cela limite l'effet « code legacy très lourd à modifier ») .
- Essayer ces pratiques dès aujourd'hui ne coûte rien !

Transformer une fonctionnalité à réaliser en une liste de plusieurs petites fonctionnalités représente une flexibilité inédite. En effet, il arrive souvent qu'un changement de priorité soit imposé à une équipe en plein milieu de développement d'une fonctionnalité qui prend plusieurs semaines. Il en résulte généralement beaucoup de code mis «de côté » à attendre une éventuelle reprise, qui n'arrive souvent jamais, pour diverses raisons. Avec de petites fonctionnalités qui ne prennent que quelques jours à développer et mettre en production, lorsque le changement de priorité arrive on ne perd pas tout : ce qui est déjà en production apporte déjà de la valeur au client. De plus, ce qui est gaspillé est bien moins important car il ne s'agit que du travail des derniers jours, et peut-être même qu'on pourra simplement terminer la petite fonctionnalité en cours puisqu'il ne reste qu'un jour ou deux de travail pour pouvoir la délivrer au client. Avec une fonctionnalité qui nécessitait encore 2 semaines de travail cela n'aurait pas été possible. En fournissant les bases de la fonctionnalité progressivement aux utilisateurs, vous leur donnez la possibilité de réaliser qu'ils ont peut-être assez avec ce que vous avez déjà fourni et qu'il n'est pas nécessaire de faire le reste. L'effort pourra alors être réaffecté à des fonctionnalités à plus haute valeur ajoutée. •

Cet article est extrait du livre « Les pratiques de l'équipe agile » (Thomas THIRY, Les pratiques de l'équipe agile, De Boeck Supérieur, 2019). Cet ouvrage vise à aider à comprendre en profondeur les différentes approches issues de l'agilité : Agile, DevOps, Kanban, Scrum, Extreme Programming, Lean Software Development, Livraison Continue, Lean Startup, Agile à Grande Échelle et Projets au Forfait Agiles.

Alexandre FAURÉ
Développeur .Net chez
Edisys
@alexousky
https://alexandrefaure.fr/

EDISYS
VOTRE PARTENAIRE DEMATERIALISATION

Dernières nouveautés du langage C# 7 et arrivée de C# 8.0

Depuis l'arrivée de la version C# 7.0, déployée avec la nouvelle version 2017 de Visual Studio en mars 2017, de nombreuses autres versions(1) sont sorties depuis. La dernière version en date est la version 7.3, sortie en septembre 2018.

niveau 100

En parallèle, Microsoft s'est occupé de produire la dernière version majeure C#8.0 qui sera livrée avec la nouvelle version du framework .Net Core 3.0. Cependant, les nouvelles fonctionnalités du langage sont d'ores et déjà testables avec les versions preview de Visual Studio 2019.
Dans cet article, nous passerons rapidement en revue les nouvelles fonctionnalités qui sont apparues dans les dernières versions de C#7 avant de nous attaquer aux nouveautés apportées par C#8.0(2).

Dernières nouveautés de C#7

C#7.1

Le point d'entrée d'une application peut avoir le modificateur async

Jusqu'alors, il n'était pas possible de qualifier la méthode *Main* comme asynchrone. Il fallait alors feinter en appelant la méthode *GetAwaiter()* permettant d'attendre le retour de la tâche :

```
static void Main(string[] args)
{
  Console.WriteLine("Hello World!");
  Task.Delay(2000).GetAwaiter().GetResult();
  Console.WriteLine($"This is a new feature from 7.1 C# language");
}
```

Désormais l'utilisation du modificateur *async* permet d'éviter cette surcharge de code et d'appeler directement l'instruction du code asynchrone avec *await* :

```
static async Task Main(string[] args)
{
  Console.WriteLine("Hello World!");
  await Task.Delay(2000);
  Console.WriteLine($"This is a new feature from 7.1 C# language");
}
```

Amélioration des expressions de valeur par défaut

Une expression de valeur par défaut *default(T)* permet d'obtenir la valeur par défaut associée au type générique *T* utilisé. Par exemple, la valeur par défaut du type *bool* est *false*.

Dès lors, au lieu d'écrire l'expression suivante :

```
bool myBooleanValue = default(bool);
```

On pourra donc s'affranchir de la spécification du type qui sera automatiquement déterminé lors de l'appel du booléen *myBooleanValue* :

```
bool myBooleanValue = default;
```

Inférence des noms des éléments de tuple

Les *tuples* ont été introduits dans la 7.0. Ce sont des structures qui permettent de retourner plusieurs valeurs dans un seul conteneur.
Pour initialiser un *tuple* avec plusieurs valeurs, il fallait le faire en spécifiant les noms de chaque élément le composant :

```
var myTuple = (first: "MyFirstValue", second: "MySecondValue");
```

Dans cette nouvelle version, le nom des éléments du tuple sont « inférés » à partir du type des variables utilisées et il n'y a donc plus besoin de les expliciter :

```
var myTuple = ("MyFirstValue", "MySecondValue");
```

C#7.2

Ajout de techniques d'écriture de code « safe et efficace »

On entend par code « safe et efficace » du code sécurisé et fiable pour l'amélioration des performances des applications. Notamment, cela permet de promouvoir l'utilisation des références à des types valeurs afin de limiter la copie de types valeurs en mémoire. On gagne donc potentiellement en performance en suivant ces quelques recommandations.
Comme exemple de ces améliorations, on pourra noter l'utilisation du modificateur *in* pour spécifier qu'un argument n'est pas modifiable par la méthode appelée :

```
static void Main(string[] args)
{
  int myReadOnlyNumber = 10;
  InArgumentMethod(myReadOnlyNumber);
  Console.WriteLine(myReadOnlyNumber); // La valeur est toujours 10
}

public static void InArgumentMethod(in int number)
{
```

(1) https://docs.microsoft.com/fr-fr/dotnet/csharp/whats-new/csharp-version-history
(2) https://docs.microsoft.com/fr-fr/dotnet/csharp/whats-new/csharp-8

```
    //number = 19;
  }
```

Nommage de l'ordre des arguments

Les arguments nommés permettent d'éviter de mémoriser ou rechercher l'ordre des paramètres des méthodes lors de leur utilisation. Considérons par exemple la méthode suivante :

```
  public static void MyNewMethod(int firstNumber, string firstString, bool firstBool)
  {
    Console.WriteLine($"This is my New Method with arguments {firstNumber}, {firstString}
and {firstBool}");
  }
```

Actuellement, l'appel de la méthode *MyNewMethod* s'écrit sans que l'on sache quel argument est derrière chacune des valeurs (à moins d'aller consulter la définition de la méthode en question) :

```
MyNewMethod(2, "Test", false);
```

Avec la nouvelle version, il sera possible d'appeler la méthode en spécifiant explicitement chaque argument :

```
MyNewMethod(firstNumber: 2, firstBool: false, firstString: "Test");
```

Notons aussi qu'ainsi, il n'est pas nécessaire de respecter l'ordre de déclaration des arguments. Le nommage est également plus explicite et fournit une documentation supplémentaire lors de la relecture du code.

Underscore comme caractère de début dans les valeurs numériques

Il s'agit d'une petite amélioration qui permet un confort de lecture appréciable. Une valeur numérique est un nombre fixé qui sera assigné à une variable ou utilisé dans des calculs. Cette valeur peut être de plusieurs types comme par exemple un type binaire.
La nouveauté réside dans la possibilité d'ajouter le caractère *underscore* au début de l'expression afin de passer d'une écriture de nombre binaire classique :

```
int binaryValue = 0b0101_0101;
```

A une écriture plus aérée en permettant une séparation claire du préfixe 0b :

```
int binaryValue = 0b_0101_0101;
```

A noter que cette amélioration concerne également l'écriture d'un hexadécimal.

Ajout du modificateur d'accès private protected

Pour rappel, le modificateur *private* permet l'accès dans la même classe ou structure tandis que le modificateur *protected* permet l'accès dans la même classe ou structure et leurs dérivés.
On connaissait déjà l'utilisation du modificateur *protected internal* qui permettait l'accès à la classe mère et aux classes dérivées, présentes dans la même assembly.
Private protected laissera, quant à lui, l'accès à la classe mère seulement et toujours dans la même assembly.

Attribution du résultat d'une expression conditionnelle à une variable référence

Il est désormais possible d'utiliser des conditions in-line et d'en attribuer le résultat à un type référence. Encore une fois, nous gagnons en lisibilité du code avec par exemple le passage de cette écriture :

```
if (player1.GetDistance(ref position) < player2.GetDistance(ref position))
    Update(ref player1);
else
    Update(ref player2);
```

à celle-ci :

```
Update(ref player1.GetDistance(ref position) < player2.GetDistance(ref position)
        ? ref player1
        : ref player2);
```

C#7.3

Plus d'épinglage nécessaire lors de l'indexation des champs fixed

L'instruction *fixed* permet d'indiquer au garbage collector de ne pas s'occuper d'une variable. Celle-ci est considérée comme « fixée » ou « épinglée » et empêche sa modification de manière imprévisible. Ainsi, l'utilisation de cette instruction est uniquement autorisée dans un contexte dit *unsafe* (c'est-à-dire un contexte non sécurisé où l'on devra manipuler des pointeurs).
Dans les versions précédentes, il fallait épingler la variable *fixed* avant de l'utiliser, rendant l'écriture de code assez fastidieuse :

```
static MyClass class = new MyClass();

unsafe public void Method()
{
  fixed (int* ptr = class.myFixedField)
  {
    var p = ptr[5];
  }
}

unsafe struct MyClass
{
  public fixed int myFixedField[10];
}
```

Cela a donc été remplacé par une écriture plus légère où l'attribution de la valeur de p est faite immédiatement en appelant la variable *fixed*.

```
static MyClass class = new MyClass();
```

```
unsafe public void Method()
{
  int p = class.myFixedField[5];
}

unsafe struct MyClass
{
  public fixed int myFixedField[10];
}
```

Réaffectation des variables locales ref

Il est désormais possible de réassigner une variable locale *ref* pour faire référence à d'autres instances (après avoir été initialisée).

```
int myInt = 0;
ref int myFirstVariable = ref myInt;

int anotherInt = 1;
myFirstVariable = ref anotherInt;
```

Initialisation in-line des variables stackalloc

Le mot-clé *stackalloc* permet d'allouer un bloc de mémoire sur la pile de la mémoire. Un tableau est théoriquement de type référence et ne pourrait être alloué sur la pile. Ce mot-clé permet donc cette opération.
Auparavant, il fallait séparer l'initialisation d'une variable *stackalloc* de sa déclaration :

```
int* array = stackalloc int[3];
array++;
```

Tandis que maintenant, il est possible d'initialiser ce type de variable directement lors de leur déclaration :

```
int* array = stackalloc int[3] { 0, 1, 2 };
```

Augmentation de la prise en charge des types fixed

Jusqu'alors, l'utilisation de l'instruction *fixed* n'était réservée qu'à certains types dédiés (tableaux, chaînes, mémoires tampon de taille fixe ou variables non-managées).
Il est désormais possible d'utiliser cette instruction pour tous les types contenant une méthode *GetPinnableReference()* qui retourne *ref T* (ou *ref readonly T*).
Citons par exemple les types *Span<T>* et *ReadOnlySpan<T>* qui implémentent ce modèle et peuvent donc être désormais utilisés comme *fixed* :

```
static S s = new S();

unsafe public void Method()
{
  Span<int> mySpanVariable = new Span<int>(new int[3]{1,2,3});
  int p = s.myFixedField[5];

  fixed (int* ptrToRow = mySpanVariable)
  {
    /* ...*/
  }
}

unsafe struct S
{
  public fixed int myFixedField[10];
}
```

Ajout de contraintes génériques supplémentaires

En C#, il est possible d'ajouter des contraintes de base pour les classes génériques. Comme par exemple :

```
class MyClass<T> where T: List<int>
{

}
```

Les contraintes de type *Enum* ou *Delegate* s'ajoutent aux types de contraintes déjà existants.
Enfin, on notera l'ajout de la contrainte de type *unmanaged* pour indiquer que l'on souhaite utiliser un type non managé qui ne sera donc pas automatiquement géré par la mémoire.

La mise à jour C#8.0

Pour nous faire patienter jusqu'à la livraison de Visual Studio 2019 début avril 2019, Microsoft nous a concocté deux pré-versions avec leurs lots de nouvelles fonctionnalités.
Nous détaillerons ci-après l'ensemble des améliorations fournies avec C#8.0.

8.0 - Préversion 1

Sortie avec la préversion de Visual Studio 2019 Preview 1 début Décembre 2018, cette mise à jour nous permet de profiter des premières fonctionnalités de C#8.

Types de références non-nullables

On ne compte plus le nombre d'exceptions à cause des références *null* que comportent nos applications et toute la gestion spécifique qu'elles entraînent dans notre code.
Pour pallier cela, C#8.0 empêche tout simplement tous les types références d'avoir une valeur nulle. Ainsi, pour qu'un type référence soit nullable, il faudra le spécifier explicitement dans votre code.
Pour illustrer cette fonctionnalité, on pourra voir que le code suivant génère un message d'avertissement de Visual Studio indiquant que l'on vient d'assigner une valeur nulle à un type non-nullable :

```
string myString = null; // Un message d'avertissement apparaît
```

Il est alors aisé d'expliciter le fait que cette variable doit être de type nullable en spécifiant le type avec un point d'interrogation :

```
string? myString = null; // Aucun message d'avertissement
```

Les flux asynchrones

Depuis C#5.0, il est possible de récupérer des résultats asyn-

chrones avec des méthodes sans retour (*void*). Cependant, cela n'est pas très utile pour récupérer un flux continu de résultat, qui nécessiterait justement un retour de méthode (par exemple une méthode qui retournerait un *IEnumerable*).
Microsoft l'a bien compris et a introduit l'interface *IAsyncEnumerable<T>* qui correspond à ce besoin. Il s'agit en quelque sorte d'une version asynchrone de l'interface *IEnumerable<T>*.
Exemple avec une méthode qui va renvoyer un nombre compris entre 0 et 99, avec une pause entre chaque renvoi :

```
public static async IAsyncEnumerable<int> GetSequence()
{
  for (var i = 0; i < 100; i++)
  {
    await Task.Delay(100);
    yield return i;
  }
}
```

Pour appeler cette méthode, il faudra utiliser l'instruction *await foreach* afin de récupérer le résultat asynchrone de la méthode GetSequence() :

```
await foreach (var number in GetSequence())
{
  Console.WriteLine(number);
}
```

Amélioration de la manipulation des index et plages

Les plages et les index permettent à l'utilisateur de manipuler facilement les éléments d'un tableau. La plage désigne un ensemble d'éléments du tableau tandis que l'index permet de désigner les éléments d'un tableau par leur ordre d'ajout.
Des nouveautés ont été introduites côté syntaxe. Principalement au niveau de la récupération des plages d'un tableau et de la récupération d'éléments via leur index.
Considérons le tableau de string suivant :

```
var cities = new string[]
{
  "Paris",
  "Londres",
  "Madrid",
  "Rome",
  "Berlin",
  "Genève"
};
```

Pour récupérer chaque élément du tableau, il est possible d'utiliser l'index du tableau. Par exemple, on récupère le premier élément de la façon suivante :

```
Console.WriteLine($"Première ville : {cities[0]}"); // Renvoi Paris
```

On récupère donc l'élément en partant du début du tableau avec l'index 0. C#8 permet désormais de récupérer des éléments en partant de la fin du tableau. Pour cela, on considère que l'index de fin « commence » à la valeur 1. On ajoutera également le signe ^ pour indiquer que l'index commence par la fin du tableau. Par conséquent, pour récupérer la dernière valeur du tableau, on écrira :

```
Console.WriteLine($"Dernière ville : {cities[^1]}"); // Renvoi Genève
```

En ce qui concerne les plages d'éléments, le principe est le même. Pour récupérer les éléments de « Londres » à « Rome », on va devoir récupérer les éléments *cities[1]* à *cities[3]*.
Le code à appeler pour récupérer cette plage sera le suivant :

```
var plage1 = cities[1..4]; // Renvoi Londres à Rome
```

L'élément 4 de la liste n'est donc pas inclus dans cette plage mais on doit indiquer qu'il faut aller jusqu'à l'index 4… exclu !
Du coup, le principe est identique pour la sélection d'une plage récupérant les valeurs « Berlin » et « Genève » :

```
var plage2 = cities[^2..^0]; // Renvoi Berlin à Genève
```

En faisant toujours attention de mentionner l'index ^0 qui sera exclu de la sélection.

8.0 - Préversion 2

La préversion 2 de C#8 est sortie fin janvier 2019 accompagnée de la nouvelle version preview de Visual Studio 2019. Un nombre de fonctionnalités plus important a été ajouté pour cette mise à jour.

Amélioration des patterns

L'amélioration des patterns consiste à simplifier l'écriture du code. Ceci lors de l'utilisation de *switch case*, des modèles de propriétés des objets, des tuples et de positions. Nous ne rentrerons pas dans le détail de tout, mais décrirons l'amélioration en question dans le cas des switch case.
Cette amélioration est purement syntaxique. La nouveauté consiste à améliorer la lisibilité de déclaration de tous les cas possibles d'une condition switch.
Actuellement, on décompose chaque cas du switch par des *case*, correspondant à chaque cas possible :

```
public static string GetCapitaleFromPays(string pays)
{
  switch (pays)
  {
    case "France":
      return "Paris";
    case "Espagne":
      return "Madrid";
    case "Allemagne":
      return "Berlin";
    default:
      throw new ArgumentException());
  };
}
```

Désormais, les différents cas (cases) ne nécessitent pas l'écriture des mots-clés case et return pour simplifier l'écriture du switch case, augmentant considérablement le confort de lecture du code :

```
public static string GetCapitaleFromPays(string pays) => pays switch
{
  "France" => "Paris",
  "Espagne" => "Madrid",
  "Allemagne" => "Berlin",
};
```

Chaque cas y est exposé directement par la valeur du cas suivi de l'opérateur => permettant d'assigner directement sa valeur correspondante en sortie du switch case.

Déclarations using

Pour rappel, la déclaration *using* permet (entre autres !), de créer des instructions « disposables » qui garantissent que les objets traités seront correctement nettoyés par la mémoire en fin d'utilisation. Leur utilisation est possible lorsque la durée de vie d'un objet est limitée à une seule méthode. En effet, à la fin de l'instruction *using*, la méthode Dispose (de *IDisposable*) est appelée afin de libérer la mémoire associée à cet objet. De plus, même si une exception survient, *using* nous garantit que *Dispose()* sera appelée.
Parfois, nous utilisons *using* pour gérer une seule et unique instruction. L'écriture du block *using* se révèle alors assez lourde pour cette seule utilisation :

```
static void Main(string[] args)
{
  using (var options = Parse(args))
  {
    if (options["verbose"])
    {
      Console.WriteLine("Verbose option is activated");
    }
  }
}
```

C#8 permet de simplifier son utilisation en permettant l'ajout de la ligne à exécuter directement après la déclaration du using (comme cela est déjà le cas pour les instructions conditionnelles) :

```
static void Main(string[] args)
{
  using (var options = Parse(args))
  if (options["verbose"]) { Console.WriteLine("Verbose option is activated"); }
}
```

Fonctions locales statiques

Lorsque l'on déclare des fonctions dans un bloc, celles-ci accèdent aux variables déclarées dans le bloc en question. Actuellement, pour déclarer une fonction qui procède à l'addition de deux nombres, on devra écrire :

```
int Addition()
{
  int membreA = 0;
  int membreB = 0;
  int result;
  LocalFunction(membreA, membreB);
  return result;

  void LocalFunction(int membreA, int membreB) => result = membreA + membreB;
}
```

La variable locale *LocalFunction* ne peut utiliser que les variables déclarées précédemment dans la méthode (membreA, membreB et result).
On pourra désormais déclarer une fonction locale de manière statique afin de ne plus dépendre des variables de la portée de cette façon :

```
int Addition()
{
  int membreA = 0;
  int membreB = 0;
  return LocalFunction(membreA, membreB);

  static int LocalFunction(int gauche, int droite) => gauche + droite;
}
```

Les variables utilisées par la fonction sont génériques et ne sont aucunement définies avant l'utilisation de la fonction (*gauche* et *droite*).

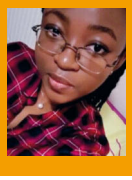
Structs ref disposable

Les variables de type *struct*, lorsqu'elles sont déclarées avec le mot clé *ref* ne peuvent pas implémenter d'interfaces. En effet, l'ajout du modificateur *ref* à un type *struct* indique que les instances de ce type doivent être allouées par la pile (contrairement au type *struct* de base qui est géré par le tas).
Par conséquent, celles-ci ne peuvent pas implémenter l'interface *IDisposable*.
Avec C#8, les types structures *ref* possèderont déjà la méthode *Dispose()* leur permettant d'être correctement gérés par la mémoire à la fin de leur utilisation.

Conclusion

Nous avons vu que depuis C#7.0, de nombreuses fonctionnalités très utiles ont été ajoutées au langage. Beaucoup de ces améliorations concernent le confort d'écriture des développeurs afin de simplifier la lecture du code et son écriture.
Au moment où je finis d'écrire ces quelques lignes, Visual Studio 2019 est disponible depuis quelques semaines. Vous pouvez donc d'ores et déjà profiter des premières fonctionnalités de C#8 décrites dans cet article en attendant la sortie du framework .Net Core 3.0 prévue dans les prochains mois.

•



Lucette FAGNON
Ingénieur Concepteur Développeur
SQLI DIGITAL EXPERIENCE

Nouveautés dans visual studio 2019

La suite Microsoft Visual Studio 2019, disponible depuis le 2 avril, est la version améliorée de Visual Studio 2017, aussi bien sur le plan visuel que des fonctionnalités. Faisons ensemble le tour des nouveautés.

UN VISUEL OPTIMISÉ

Interface de bienvenue

L'interface de bienvenue de **Visual Studio 2019** est plus intuitive et fait gagner du temps. Le processus d'accueil est simplifié ; il propose directement les actions suivantes au lancement :

→ *Récupérer du code depuis un dépôt en ligne ;*
→ *Ouvrir une solution, un projet existant ou un dossier local ;*
→ *Créer un projet.*

Un modèle de sélection beaucoup plus simple d'utilisation avec le choix du langage, de la plateforme et du type de projet. Il propose également une barre de recherche afin de retrouver rapidement un projet donné. 1 2

Barre de titre

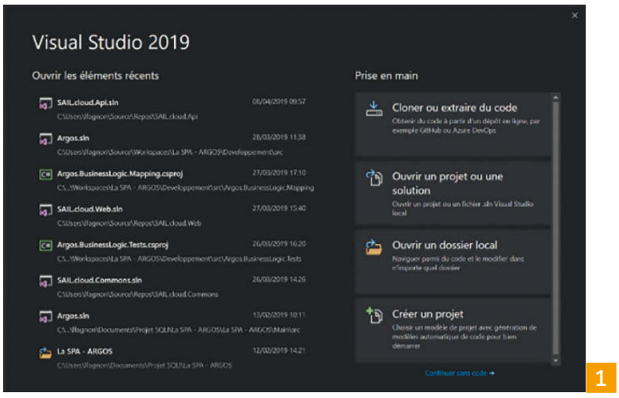
La barre de titre est plus réduite que celle de Visual Studio 2017, avec la suppression de nom du projet/solution, offrant ainsi un plus grand espace pour coder. 3

Elle inclut une **barre de recherche « intelligente »** qui remplace la barre de lancement rapide de Visual Studio 2017. Selon le contexte, cette barre de recherche offre différentes solutions avec les mêmes mots-clés. Par exemple, lorsqu'on se place au niveau de la solution en tapant le mot-clé « ajouter », la barre de recherche propose diverses options telles que « ajouter un projet » et « ajouter un dossier ». En revanche, lorsqu'on se place au niveau d'un projet en tapant le même mot-clé, la barre propose des options telle que « ajouter une classe, une référence ». 4

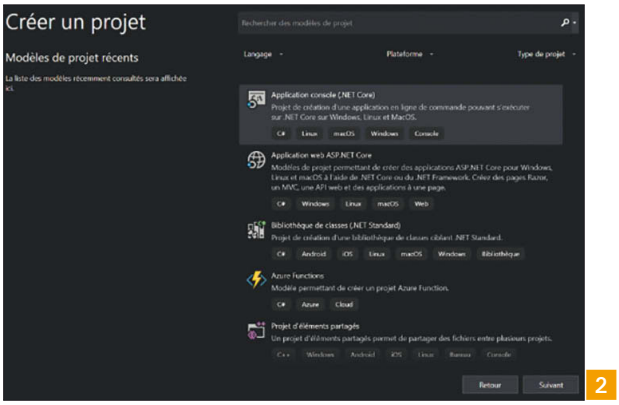
NAVIGATION DANS LE CODE

Filtres de solution

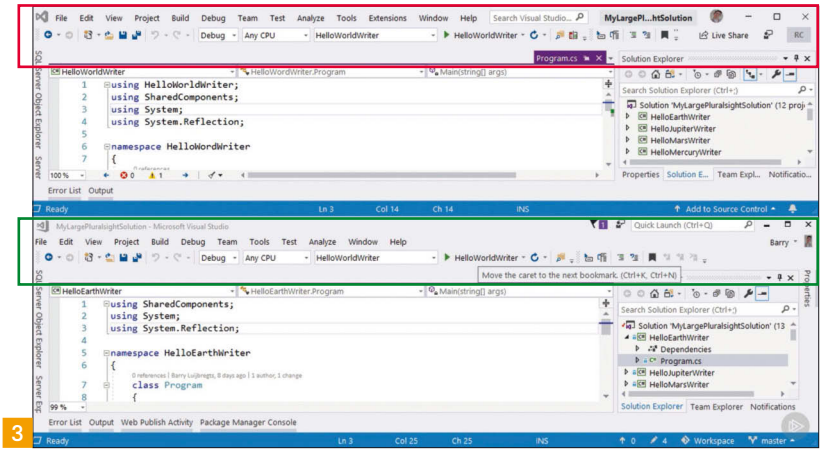
La version 2019 permet d'ouvrir une solution sans charger les projets contenus dans celle-ci. Il suffit lors de l'ouverture d'un fichier .sln de cocher la case « Ne pas charger les projets ». On peut donc choisir le projet à charger. Visual Studio 2019 permet également d'enregistrer un filtre dans un fichier au format .slnf. Ce fichier représente une version filtrée de la solution. 5

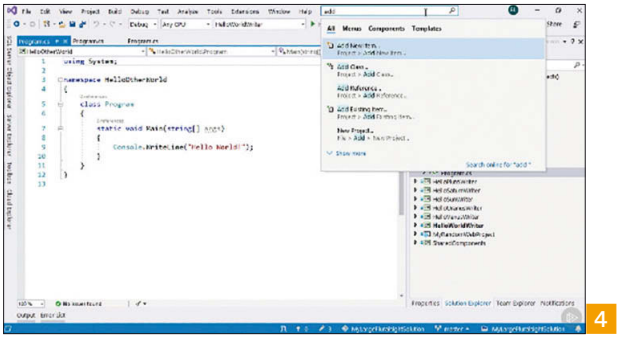


1



2


3


4

Visual Studio 2019 Visual Studio 2017

Azure Devops – Pull Request

Contrairement à la version 2017, on retrouve ici une extension permettant de gérer les ***pull requests*** (*PR* ou requêtes de tirage) d'une solution hébergée sur un dépôt Git (Azure Devops) distant. 6 7

Dans Team Explorer > Projet > Requête de tirage, on a la possibilité d'ouvrir une PR (une fois connecté à son repository) à partir de son ID, de visualiser les pull requests qu'on a émises et celles qui sont assignées, et même de créer une nouvelle pull request.

En cliquant sur une pull request, on visualise tous les détails : description, relecteurs, éléments de travail et modifications dans les fichiers.

Tout comme sur la plateforme Azure Devops, on peut non seulement visualiser les commentaires des autres développeurs et le code associé, mais aussi ajouter les siens et réagir aux commentaires de ses coéquipiers par des likes.

Lors de l'ajout des commentaires, on peut ajouter des émoticônes pour exprimer son ressenti par rapport au code et taguer ses collaborateurs. On peut aussi joindre des fichiers ou d'autres éléments. 8 9

Après avoir enregistré ses modifications, on peut approuver, rejeter ou faire toute autre action par rapport à la pull request.

Cette fonctionnalité, qui est une grande première dans Visual Studio, facilitera non seulement le travail d'équipe, mais aussi le travail des développeurs souhaitant gérer leur code entièrement dans l'application sans avoir à s'appuyer sur l'interface web d'Azure Devops.

AU NIVEAU DU CODE

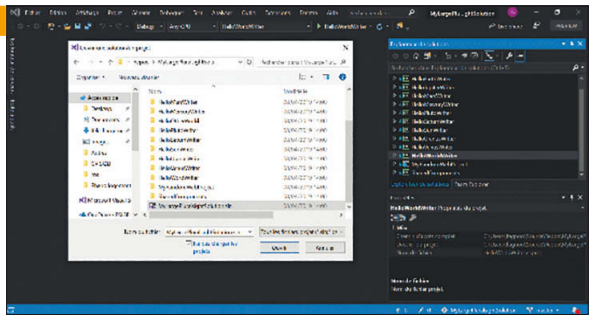
Code Cleanup

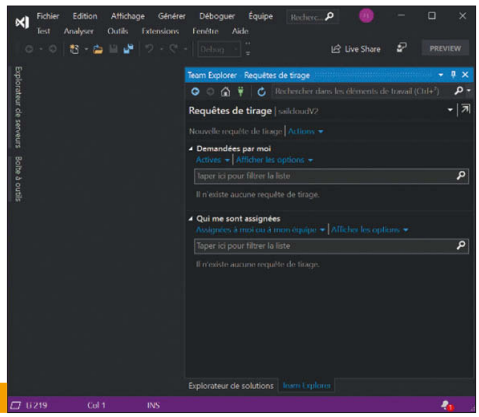
Dans la barre des tâches, située en bas de la fenêtre, Visual Studio 2019 permet de visualiser rapidement le nombre d'erreurs et de warnings dans le code. 10

En cliquant sur l'une de ces icônes, l'application affiche en détails les erreurs et les warnings soulignés. Afin d'enrichir cette liste, on peut paramétrer en un clic les règles de son Cleanup. Le cadre du haut correspond aux correcteurs inclus et celui du bas aux correcteurs disponibles.

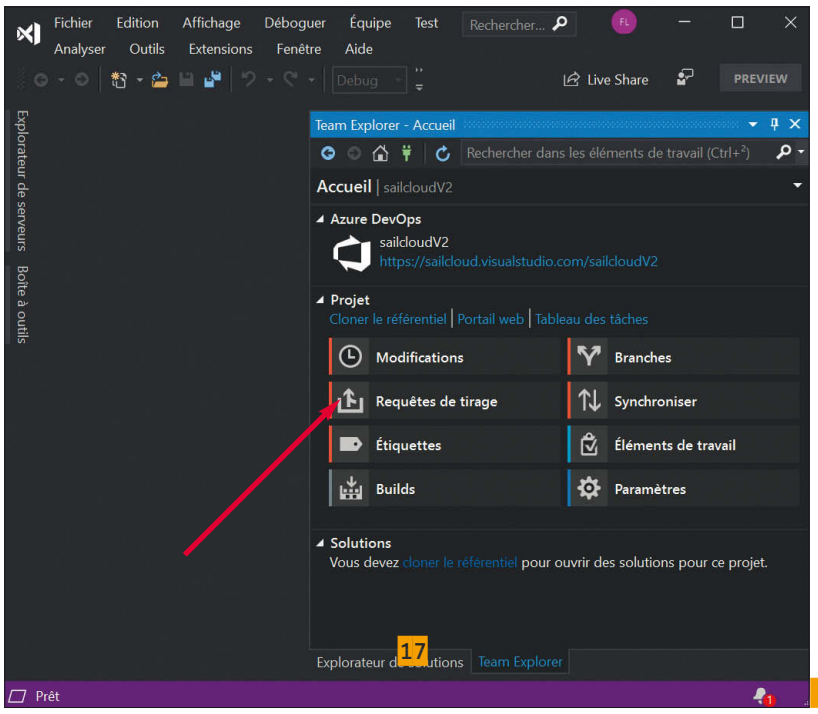
10

5

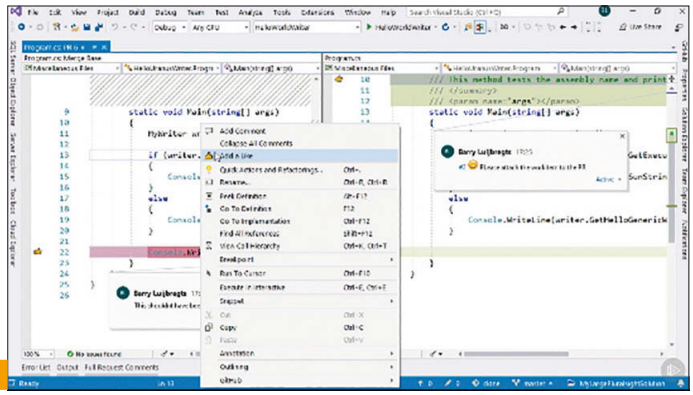




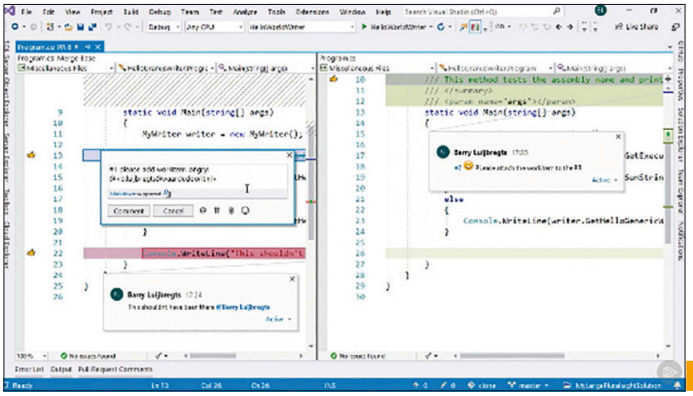
7



6



8



9

Dans cette fenêtre, on peut choisir les correcteurs à inclure dans le Cleanup. On peut configurer jusqu'à deux profils et ainsi exécuter le Cleanup configuré pour chacun des deux profils, directement à l'aide de l'icône dans la barre des tâches. 11 12 13

Par ailleurs, pour définir une règle générale pour les équipes, les développeurs peuvent créer leur propre style de code et l'enregistrer dans un fichier de configuration partageable avec toute l'équipe. Ainsi, tous auront les mêmes types d'erreurs et de warnings. Pour cela, dans la barre de recherche située en haut à droite dans la barre de titre, il suffit de taper « code style » et de cliquer sur la solution proposée. Après avoir défini toutes ses règles de codage, les cas d'erreurs et de warnings, on a plus qu'à enregistrer dans un fichier.

.Net Refactorings

Nombreux sont les *helpers* de *Refactorings* de code inclus dans Visual Studio 2019. En voici quelques-uns :

→ *Modifier la signature d'une méthode*

Une méthode avec plusieurs paramètres se présente en général comme suit : 14

Avec plusieurs paramètres, la signature de la méthode devient difficile à lire sans scroller l'écran. En surlignant la méthode, une **ampoule jaune** apparaît à gauche sur la même ligne. Elle représente l'ensemble des actions rapides et refactorisations du code. En cliquant dessus, on a plusieurs options, dont « Wrapper chaque paramètre » qui permet d'aligner les paramètres et/ou de les mettre en retrait. 15

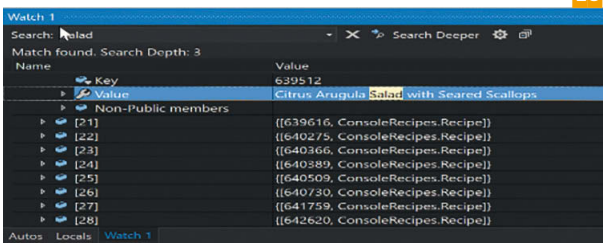
Lors de la création d'une interface pour une classe préexistante, un développeur peut reporter les méthodes de sa classe jusqu'à l'interface grâce à une action rapide. Cela lui permet de ne pas avoir à recopier la signature de ces méthodes dans son interface et d'optimiser son temps. 16

Par ailleurs, Visual Studio 2019 permet au développeur de modifier la signature de sa méthode, en supprimant certains paramètres, ou en modifiant l'ordre des paramètres. Pour cela, il lui suffit de placer son curseur sur le nom de la méthode, de cliquer ensuite sur la petite brosse qui apparaît sur la même ligne, puis de choisir « *Changer la signature* ». 17 18

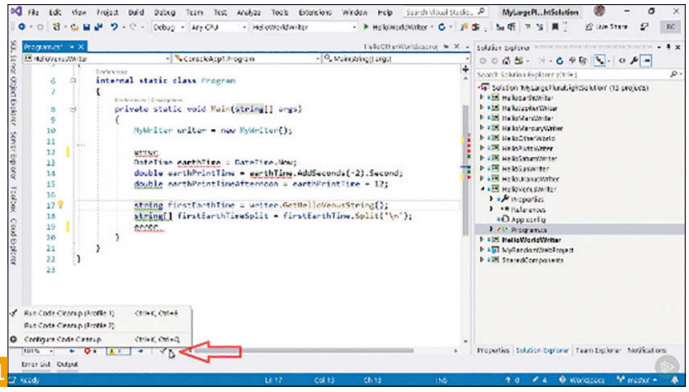
→ *Convertir une boucle foreach en syntaxe LINQ*

Le développeur, grâce à une action rapide, convertit facilement une boucle *foreach* utilisant un *IEnumerable* en requête LINQ ou en formulaire d'appel LINQ (aussi connu sous le nom de méthode LINQ), dans le but de réduire la quantité de code dans le

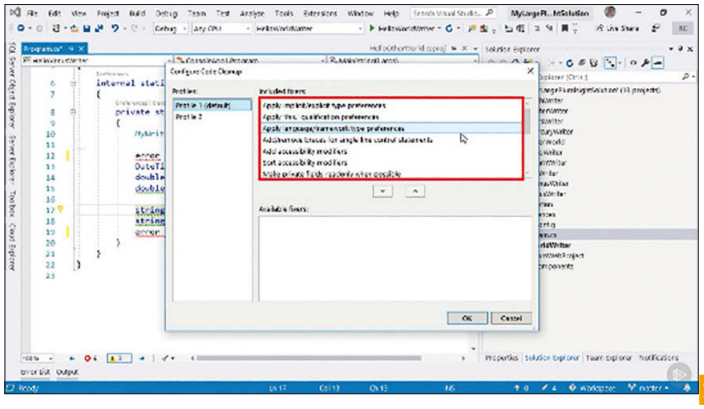
18



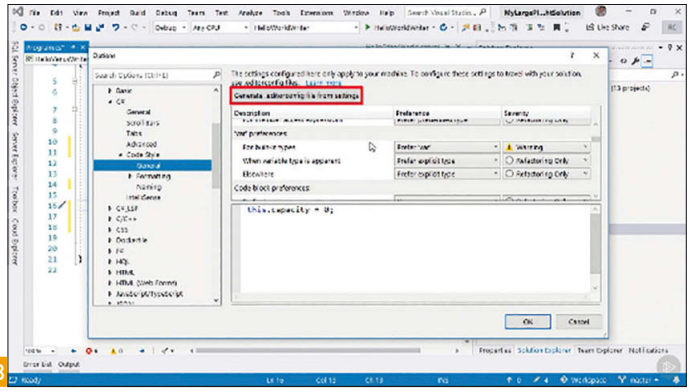
11



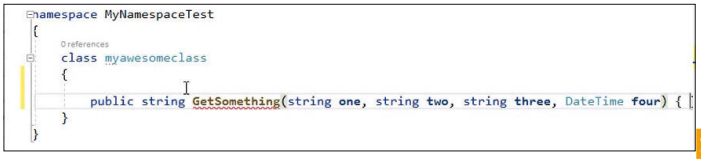
12



13

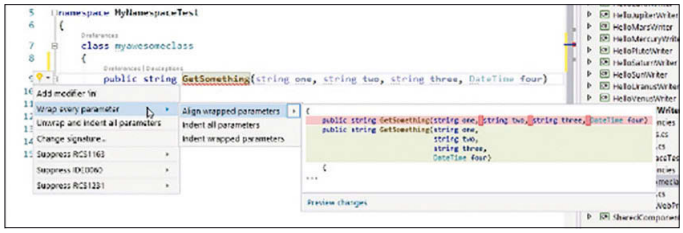


14

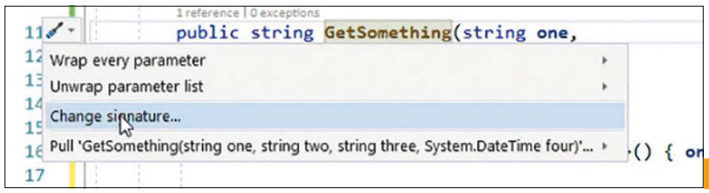


16

15



17



fichier. Pour cela, il doit placer son curseur sur le mot clé *foreach*. Une sorte de brosse minuscule apparaît sur la même ligne que la boucle. Il suffit de cliquer sur cette brosse, puis de sélectionner « Convertir en LINQ ». 19

Visual Studio IntelliCode

Depuis les versions précédentes, Microsoft utilise l'intelliSense pour proposer automatiquement des compléments aux mots-clés, des méthodes, et même des variables selon le contexte, le langage et la référence utilisées. Voici un exemple avec le type DateTime. 20

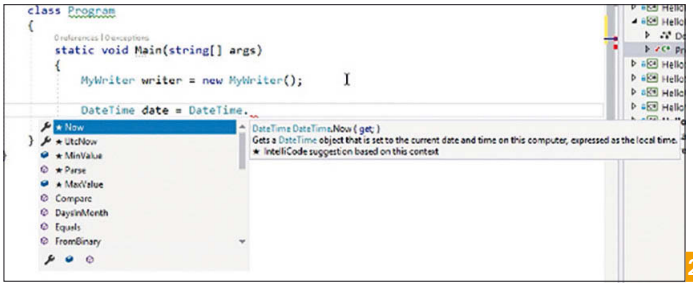
Pour augmenter la productivité du codage, et proposer des compléments de code encore plus précis, Microsoft a eu l'idée d'intégrer des algorithmes basés sur le machine learning et l'Intelligence Artificielle dans une extension : c'est l'**IntelliCode** !

L'ayant rendue disponible à la fois pour Visual Studio Code et pour Visual Studio 2019, l'objectif est de développer davantage de nouvelles fonctionnalités sous forme d'extensions afin d'en faire bénéficier les utilisateurs des deux IDEs. **Visual Studio IntelliCode** est téléchargeable depuis la marketplace de Visual Studio. Après téléchargement, il suffit de procéder à son installation dans chacune des applications. Cette extension permet de faire un apprentissage sur son code (solution) de sorte que, selon le contexte (boucle, type de variable, requêtes…), l'application fasse des propositions intelligentes en se basant sur le contenu, et le type de méthodes et de variables. Le but est d'identifier la façon dont est codée la solution afin d'en extraire les habitudes pour favoriser la cohérence du code dans toute la solution. 21 22

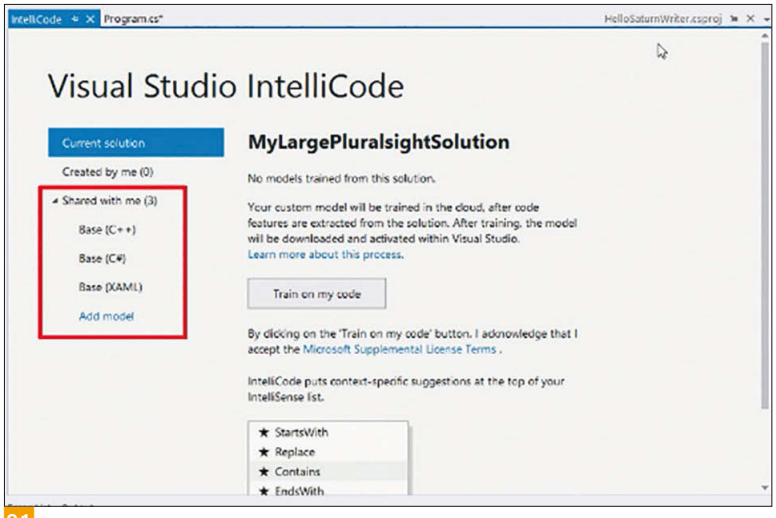
Pour l'essayer, il suffit d'ouvrir l'extension grâce la barre de recherche située en haut à droite dans la barre de titre. Dans la fenêtre qui s'ouvre, on voit les modèles partagés avec soi, et ceux créés localement. En se basant sur sa solution, on peut entraîner son propre modèle en cliquant sur « **Effectuer l'entraînement sur mon code** ». Une fois l'entraînement terminé en quelques minutes, on peut le partager, le supprimer et même l'entraîner à nouveau après une modification du code. On peut visualiser dans les détails du modèle, le nombre de classes prises en compte dans le modèle et quelques recommandations sur les divers contextes dans lesquels utiliser les méthodes. En testant le modèle, on remarque bien que selon les méthodes appelées (contexte), l'application propose les variables correspondantes.

19

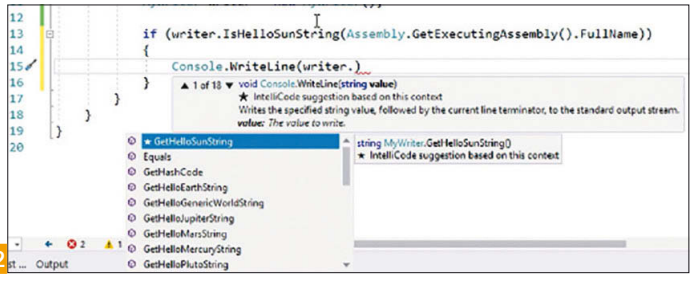
20



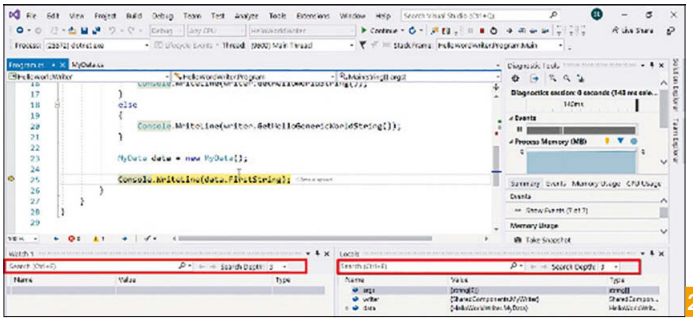
21



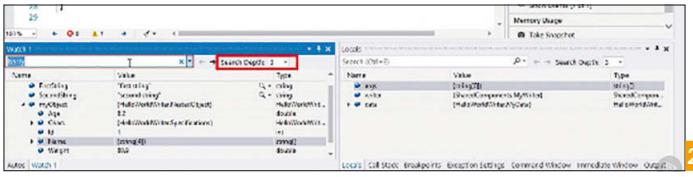
22



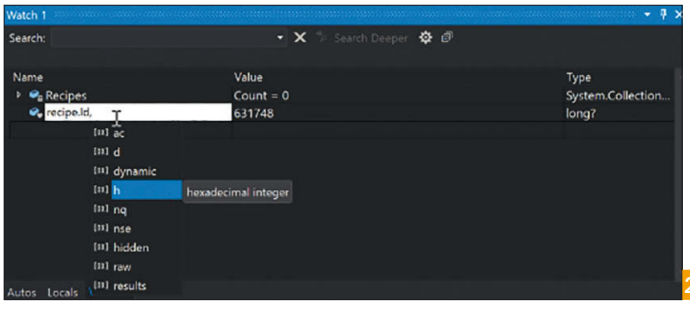
23



24



25



Faster Debugging & Search in Watch and locals' windows

Microsoft a considérablement amélioré le débogage des codes par rapport aux versions précédentes. Après avoir défini un point d'arrêt et lancé le débogage, on constate dans la fenêtre des diagnostics que le débogage est plus rapide qu'auparavant. 23 24

Une autre plus-value de cet outil de débogage est qu'il est maintenant possible de lancer une recherche dans les fenêtres *Watch (Espion)* et *Locals (Variables locales)* pendant le débogage. Ceci n'était pas possible dans les versions précédentes. Préciser le niveau ou la profondeur de la recherche est également faisable. Autrement dit, on peut définir jusqu'à quel niveau de la hiérarchie on souhaite faire la recherche.

En outre, on peut visualiser les objets, les variables et leurs contenus. Il suffit d'utiliser des mots-clés pour trouver des éléments de code, les mettre en surbrillance et y accéder directement.

On peut également afficher une liste déroulante de spécificateurs et d'options pour définir le format des données dans les fenêtres Espion, Automatique et Variables locales, en ajoutant une virgule à un élément de la liste (voir figure <-). 25

Visual Studio Live Share

Live Share permet une collaboration en temps réel entre deux développeurs. Plus besoin de récupérer le code depuis une branche, ni de mettre à jour ses packages, ni de pousser le code pour qu'il puisse être modifié par un autre membre de l'équipe. Avec Live Share, les collaborateurs peuvent faire du *pair programming* en temps réel c'est-à-dire qu'ils peuvent depuis leurs ordinateurs respectifs modifier un même code en live. Chacun a la possibilité de lancer un débogage, qui sera visible en temps réel sur l'ordinateur du second collaborateur. De plus, on est connecté par audio, même à distance. 26

Cette fonctionnalité existe sous forme d'extension pour Visual Studio 2017, mais est une fonctionnalité native de Visual Studio 2019.

Démo sur Live Share

Pour démarrer Live Share, il suffit de cliquer sur le bouton dédié situé en haut à droite dans la barre de titre en dessous du profil utilisateur : 27

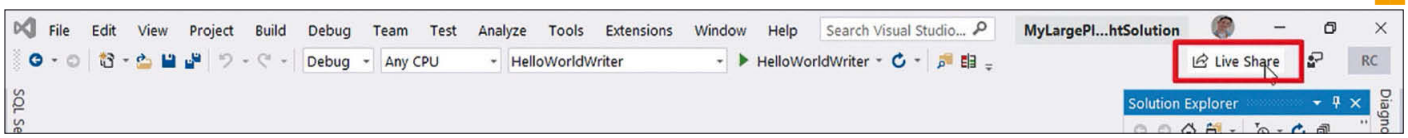
Une fenêtre s'ouvre, accompagnée d'un warning juste au-dessus. Elle propose de faire un partage en lecture seule, de copier à nouveau le lien d'invitation à envoyer à l'autre collaborateur, ou de prendre connaissance des mesures pour un partage sécurisé. Dans la fenêtre figure le lien d'invitation à envoyer. 28

Dans **Visual Studio Code,** Live Share est une extension à installer.

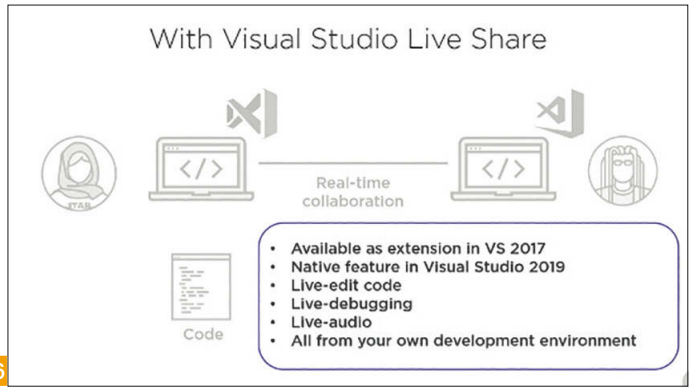
Le second collaborateur n'a plus qu'à utiliser l'url pour accéder au code. Par exemple, dans Visual Studio Code, après avoir installé l'extension, il lui suffit de procéder comme suit : 29

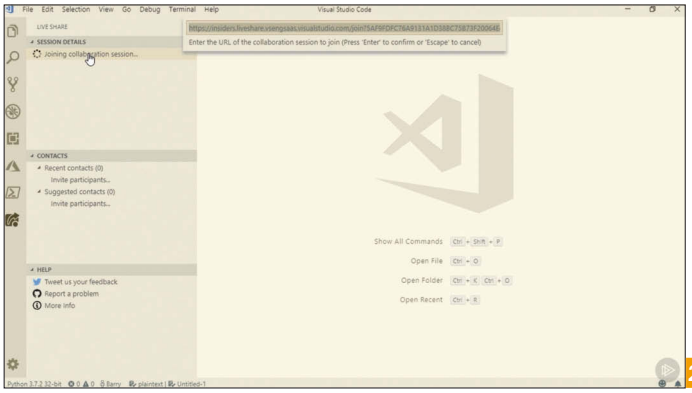
Une fois le chargement terminé, il aura accès au code et pourra visualiser en temps

27



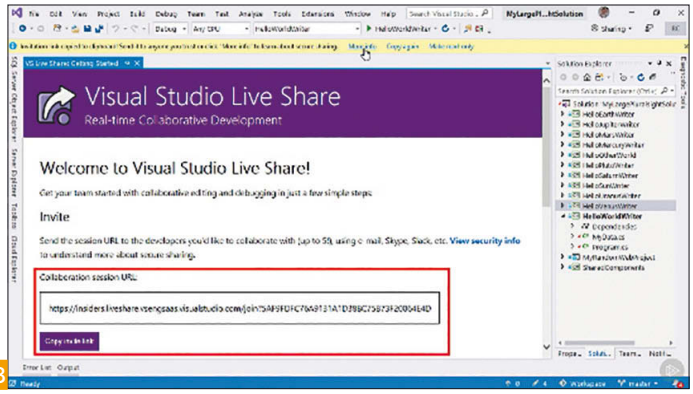
26

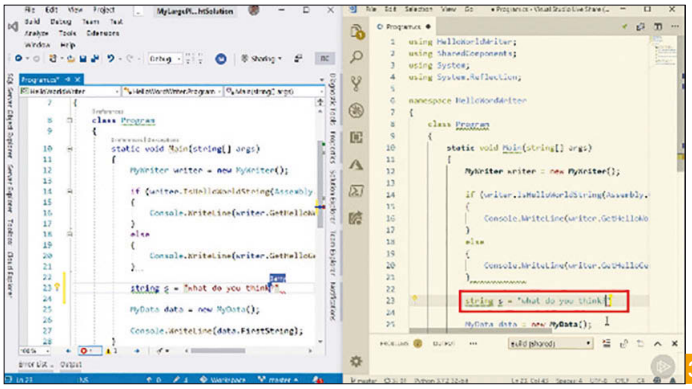




29

28





30

réel les modifications apportées par l'autre collaborateur. Les deux collaborateurs, bien qu'ils aient chacun un certain contrôle sur le code écrit par l'un ou l'autre, ne sont pas tenus de travailler sur le même fichier. Ils peuvent donc coder n'importe où dans la solution au même moment. Chacun a la possibilité de suivre le curseur de l'autre développeur dans les différents fichiers. 30

Dans Visual Studio Code, grâce à une icône : 31

Et dans Visual Studio 2019, en cliquant sur l'option : 32

Ils peuvent aussi lancer le débogage et même le contrôler depuis l'un ou l'autre des postes. Autrement dit, une fois le débogage lancé sur l'un des postes, l'autre collaborateur peut le visualiser sur son poste, ainsi que les points d'arrêts. Ils peuvent tous les deux mettre des points d'arrêts indépendamment sur leurs postes respectifs. De plus, ils peuvent utiliser les nouvelles fonctionnalités apportées par Visual Studio 2019 pour un meilleur débogage (voir ***Faster Debugging & Search in Watch and locals' windows***). 33

Pour les applications web, après lancement du débogage, chaque utilisateur visualise l'application dans son navigateur et peut interagir avec l'application différemment sur son ordinateur sans impacter sur l'autre. 34

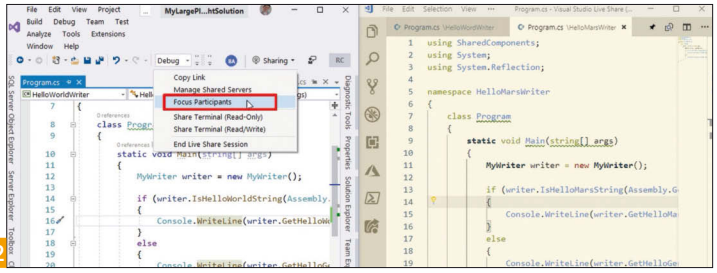
Tout cela est possible car Live Share offre un partage de serveurs pour les applications web, configurables depuis Visual Studio 2019. 35

On peut également partager le terminal en lecture seule ou en lecture/écriture depuis Visual Studio 2019 : 36

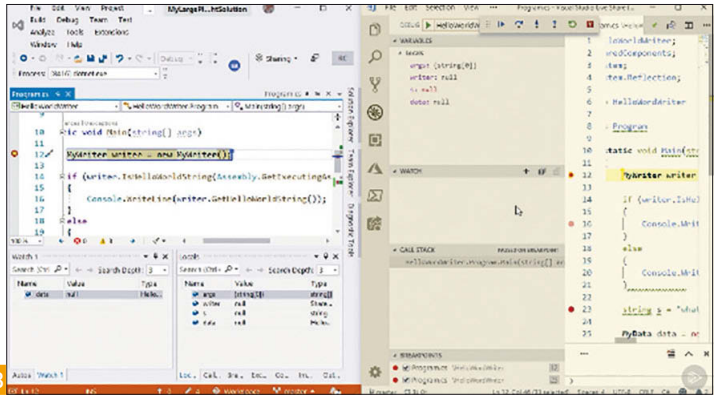
Cependant, les commandes dans le terminal ne fonctionneront que si elles sont écrites depuis le terminal du premier collaborateur (celui qui a émis le partage). Le contraire ne saurait fonctionner car le terminal est ouvert dans le répertoire courant du premier collaborateur et n'exécutera les commandes que sur son ordinateur. Mais les résultats sont visibles depuis les deux terminaux.

Vous savez tout maintenant, ou presque, des nouveautés dans Visual Studio 2019. Ces fonctionnalités augmenteront votre productivité et vous permettront de vivre une toute nouvelle expérience en tant que développeur ! •

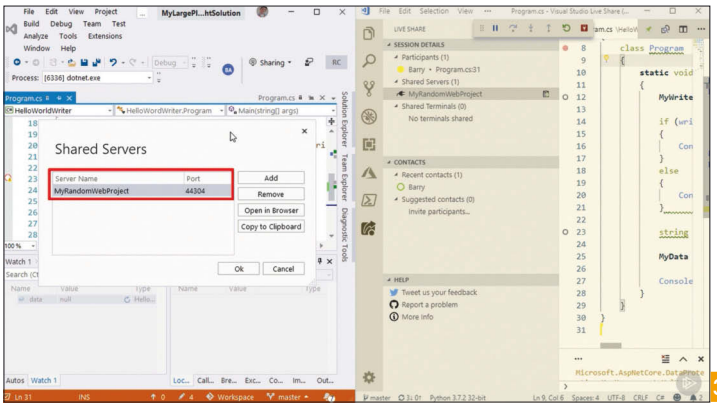
32



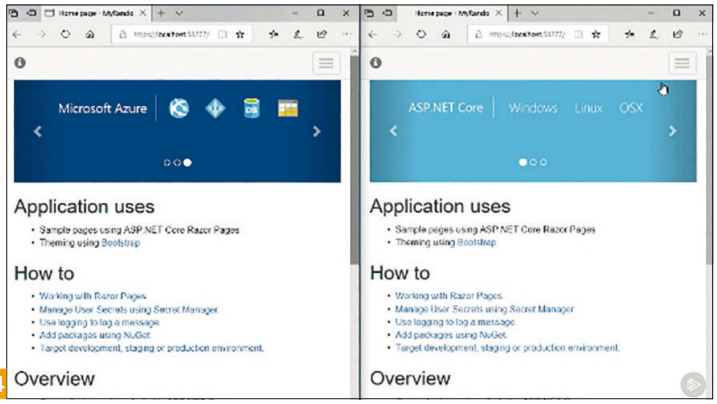
33



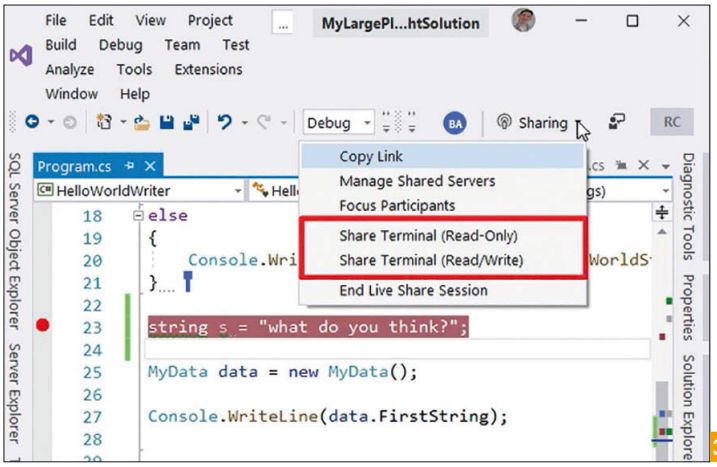
35



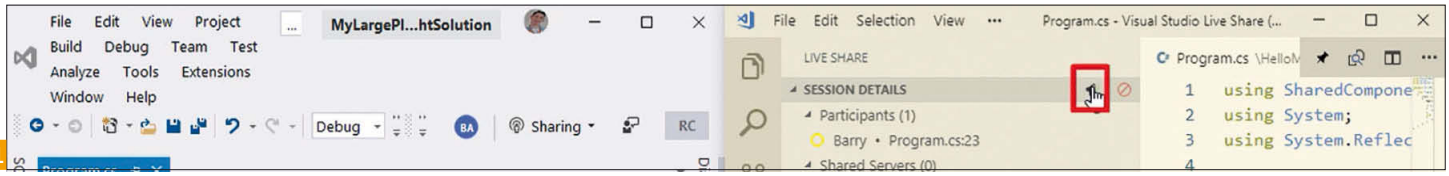
34



36



31



Maxime Beugnet
Maxime travaille avec MongoDB depuis 5 ans et dans l'industrie informatique depuis 8 ans. Il est formateur MongoDB et Java, avec les certifications DEV et DBA de MongoDB. Maxime a rejoint MongoDB il y a un an en tant que Developer Advocate EMEA pour partager son expérience avec la communauté. Il aime le code propre, la plongée sous-marine, les défis de code et les vikings !
["@MBeugnet", "@mongodb"], "github" : "MaBeuLux88"

MongoDB : le NoSQL bien dans son cloud

mongoDB

niveau 200

MongoDB est un système de base de données (BDD) NoSQL orienté documents. Cette BDD stocke et manipule des documents JSON, par opposition aux BDD relationnelles qui stockent des lignes sous forme de tableaux.

Le modèle documentaire JSON tire sa puissance du fait qu'il supporte de très nombreux types de valeurs (date, timestamp, booléen, double, entier, chaîne de caractères, etc.), mais aussi les tableaux et les sous-documents comme nous pouvons le voir dans l'exemple fig. 0. De plus, nous pouvons combiner toutes ces forces ensemble.

Dans l'exemple fig. 0, l'objet métier "*person*" est ainsi représenté dans un seul document JSON car nous pouvons stocker ses voitures dans un tableau "*cars*" qui contient les sous-documents qui représentent une voiture.

La taille du document n'est donc pas définitive et pourra changer au cours de la vie du document dans la BDD : nous pouvons ajouter ou supprimer des voitures ou encore même d'autres clés-valeurs comme l'âge de la personne par exemple.

Si une personne ne possède pas de voiture, je pourrais donc très bien stocker un document qui ne contient pas le champ "*cars*", chose qu'il est impossible de faire dans des systèmes tabulaires standards. En effet, le schéma de la table étant défini à l'avance dans ces systèmes, l'espace de stockage pour chaque ligne est déjà réservé à l'avance ce qui n'est pas le cas du côté de MongoDB. 1

Il est donc parfaitement possible de stocker ces produits par exemple dans une seule et même collection MongoDB.

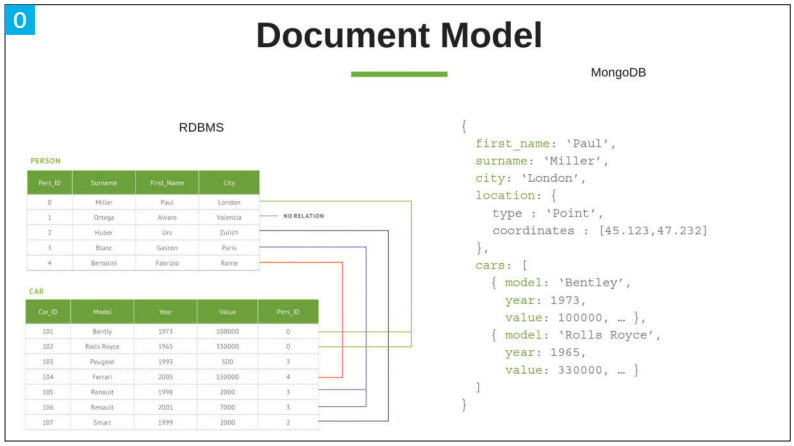
MongoDB est donc "libre de schéma" : nous avons la liberté de ne pas respecter un schéma de stockage au sein d'une même collection MongoDB ce qui n'empêche pas de s'en imposer un si besoin, et surtout de réfléchir à la structure des documents pour tirer parti de la puissance du moteur MongoDB en fonction des usages que l'on fera des données. MongoDB supporte ainsi la norme "*JSON Schema*" pour définir les règles auxquelles les documents doivent obéir (e.g. le prix est un champ obligatoire et doit être un nombre positif). Plus d'information ici : http://json-schema.org/.

Requêtage

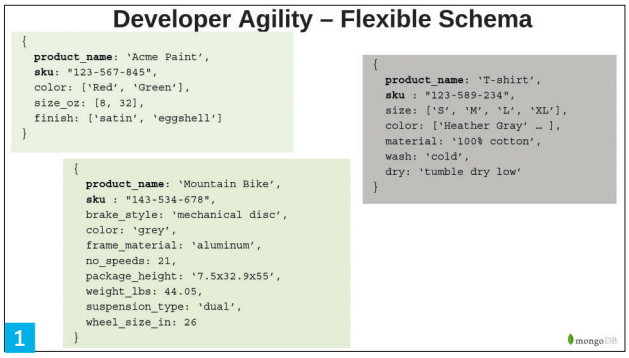
Du côté du requêtage, MongoDB propose son propre langage de requêtage MQL (MongoDB Query Language) qui est plus adapté au format JSON que ne l'est le SQL, de facto limité par le format tabulaire (un champ existe toujours et contient une valeur scalaire unique). Voici quelques exemples des opérations CRUD (Create - Read - Update - Delete) de base. Vous pourrez constater au passage que le MQL ressemble plus à une API qu'à un langage de requêtage car la philosophie centrale de MongoDB est d'être "*developer-friendly*".

```
> db.myCollection.insert({"firstname":"Maxime"});
WriteResult({ "nInserted" : 1 })
> db.myCollection.findOne()
{ "_id" : ObjectId("5ca4fd49cbcaebc698f65753"), "firstname" : "Maxime" }
```

0



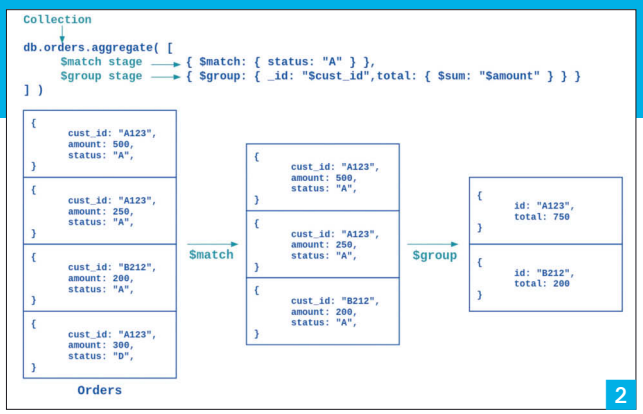
1



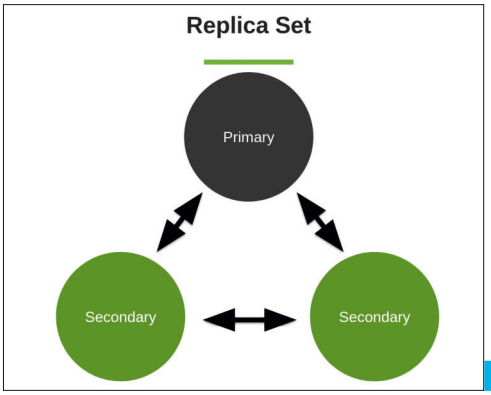

```
> db.myCollection.update({"firstname":"Maxime"}, {$set: {"lastname":"Beugnet"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.myCollection.find({"firstname":"Maxime"})
{ "_id" : ObjectId("5ca4fd49cbcaebc698f65753"), "firstname" : "Maxime", "lastname" :
"Beugnet" }
> db.myCollection.remove({"firstname":"Maxime"})
WriteResult({ "nRemoved" : 1 })
```

Pour compléter les opérations de manipulation basiques, MongoDB utilise un framework d'agrégation qui permet de faire des requêtes analytiques beaucoup plus poussées. Ce framework fonctionne comme un pipeline Unix. 2

Il existe en tout 25 différents "*stages*" qui permettent d'effectuer des manipulations sur les données, et plus d'une centaine d'opérateurs pour appliquer des transformations (mathématiques, dates, etc.). Si vous souhaitez vous aider d'un outil pour manipuler la donnée dans MongoDB et construire des pipelines d'agrégation avec un builder qui vous aidera au fur et à mesure de la construction, je vous recommande le client MongoDB Compass dont la version communautaire est téléchargeable gratuitement sur le site de MongoDB: https://www.mongodb.com/products/compass



2



3

De plus, si vous souhaitez en apprendre plus sur MongoDB, je ne peux que vous recommander le MOOC officiel MongoDB University https://university.mongodb.com. Vous y trouverez des cours en ligne gratuits pour tous les niveaux, pour les développeurs comme pour les DBA.

Les index

Comme toute BDD qui se respecte, MongoDB possède toute une panoplie d'index pour répondre à différents types de besoin. Les index MongoDB sont basés sur des B-Tree, rien de bien original, mais c'est une structure qui a fait ses preuves.

Nous pouvons citer comme types d'index :

- Les index primaires : toute collection dans MongoDB possède un index primaire unique sur le champ _id ;
- Les index simples : ondexation de la valeur d'un champ au sein d'un document ;
- Les index composés : indexation de plusieurs champs au sein d'un document. On peut donc obtenir un tri par nom, prénom puis âge par exemple ;
- Les index MultiKey : ce sont des index sur des champs dont les valeurs sont des tableaux. Chaque entrée de chaque tableau est indexée ;
- Les index texte : support de la fonction de recherche full-text ;
- Les index géo-spatiaux :
  - Les index "2d" indexent des coordonnées sur un plan,
  - Les index "2dsphere" indexent des coordonnées sur une sphère,
  - Ces deux types d'index permettent ensuite d'effectuer des recherches géo-spatiales rapides.
- Les index hashés : indexation de la valeur hashée plutôt que la valeur elle-même. C'est utile dans le cas des clés de sharding - voir le chapitre sur la scalabilité horizontale plus loin.

Ces index possèdent des options :

- TTL : Time To Live
  - La donnée et l'entrée dans l'index sont supprimées après le temps défini,
  - Fonctionne uniquement si le champ indexé est de type Date ou Timestamp.
- Unique : vérifie l'unicité de la valeur avant chaque insertion et mise à jour ;
- Partiel : indexe uniquement les valeurs qui correspondent au filtre fourni ;
- Epars (Sparse) : index uniquement lorsque la valeur est présente dans le document ;
- Insensibilité à la casse : pour comparer des chaînes de caractères, les index possèdent une collation qui peut être sensible ou non à la casse.

Note

Les index doivent pouvoir tenir en RAM dans un cluster MongoDB et vous devez aussi idéalement garder de la RAM pour le "working set". Si vous ne respectez pas ces prérogatives, vous n'aurez pas des performances optimales.
https://docs.mongodb.com/manual/tutorial/ensure-indexes-fit-ram/

Haute disponibilité et cohérence forte

MongoDB assure une haute disponibilité grâce à un système de réplication appelé *Replica Set* qui permet au cluster de continuer à fonctionner tout en réduisant à quelques secondes la période d'indisponibilité et en assurant la restauration automatique d'un noeud après une coupure. 3

En cas de panne du serveur primaire, MongoDB est capable en quelques secondes de réagir à cette panne en déclenchant une élection. Les serveurs disposant d'un droit de vote éliront le serveur le plus à même de devenir primaire pour que le service reste opérationnel.

Lorsque le serveur défectueux est remis sur pied, il rejoindra à nouveau le *Replica Set* en tant que membre secondaire après s'être synchronisé avec le nouveau primaire pour rattraper les écritures qui ont eu lieu durant sa coupure.

Cette fonctionnalité est possible grâce au système de réplication de MongoDB qui utilise une réplication basée sur les opérations d'écriture (*"statement based replication"*). MongoDB partage en permanence les instructions d'écritures du primaire vers les secondaires pour permettre la réplication. Ces informations sont stockées dans une collection spéciale appelée *"oplog.rs"* qui signifie *"Operations Log"*.

MongoDB lit et écrit sur le serveur primaire uniquement et les index sont mis à jour atomiquement avec les écritures : MongoDB est fortement cohérent par défaut.

Cela étant dit, MongoDB permet les lectures sur les serveurs secondaires grâce à l'option *"ReadPreference"* mais cela signifie que vous n'êtes pas certain de lire la donnée la plus à jour à cause du lag de réplication entre les serveurs. On dit qu'on fait alors des lectures "finalement consistantes" (de l'anglais *"eventually consistent"* qui signifie que la donnée finira par être cohérente mais peut ne pas l'être à un instant donné à cause de la réplication en cours).

D'autre part, MongoDB permet de s'assurer qu'une écriture est persistée sur plusieurs secondaires (en plus du primaire) via l'option *"WriteConcern"*. Cela permet d'éviter qu'une donnée soit perdue ou "rolled back" en cas de perte du primaire avant que la réplication n'ait pu avoir lieu. 4

Pour plus d'information sur ces 2 options veuillez vous reporter à la documentation : https://docs.mongodb.com/manual/applications/replication/

La scalabilité avec les Sharded Clusters

MongoDB est scalable par nature contrairement aux systèmes tabulaires historiques. Ces systèmes ont plus de mal à maintenir un

haut niveau de performance lorsqu'il faut distribuer de grandes quantités de données sur plusieurs dizaines voire centaines de machines. Cela provient du fait que MongoDB a été conçue dès la première ligne de code dans l'optique de répondre aux problématiques apportés par le Big Data et le Web 2.0 alors que les systèmes tabulaires historiques ont été écrits dans les années 1980 et peinent aujourd'hui à intégrer une scalabilité robuste dans leur solution existante. Un cluster shardé dans MongoDB consiste donc à "diviser pour régner". Nous allons toujours stocker la donnée sur 1 serveur primaire et la dupliquer sur plusieurs secondaires pour assurer la haute disponibilité de l'ensemble, mais cette fois-ci nous allons le faire sur plusieurs Replica Set en partitionnant les données. Chacun d'eux se verra attribuer une portion de la donnée en fonction du nombre de Replica Set mis en place.

Si nous mettons 5 Replica Sets en place - que nous appelons "*Shards*" - chacun de ces shards aura la responsabilité de 20% de la donnée totale.

Sans trop aller dans les détails - car cela pourrait être un article de 10 pages en soi - les shards seront accédés par les serveurs applicatifs via les drivers MongoDB et des instances "Mongos" (signifie Mongo Shard) qui sont en fait des services de routage qui permettent aux drivers d'aller lire ou écrire la donnée directement sur le bon shard.

Et pour terminer cette architecture, il manque les serveurs de configuration qui sont garants de la distribution des données et qui sont donc responsables de savoir où se trouve chaque donnée sur les shards. Cette distribution est définie par la clef de sharding qu'il faut choisir avec soin : https://docs.mongodb.com/manual/core/sharding-shard-key/#choosing-a-shard-key.

Voici un schéma qui résume tout cela. 5

Comme vous pouvez le constater, l'infrastructure se complique quelque peu et elle deviendra d'autant plus importante à mesure que votre cluster grossira.

La production

Que vous ayez en production un Replica Set ou un Cluster Shardé, il n'y a pas de place pour le doute dans un environnement de production, donc il va falloir résoudre un certain nombre de problématiques pour sécuriser votre environnement de production.

Les premières choses évidentes à mettre en place seront :

- L'authentification pour maîtriser qui peut se connecter
  - Il existe 4 méthodes d'authentification : SCRAM, x.509 Certificate Authentication, LDAP et Kerberos.
  - https://docs.mongodb.com/manual/core/authentication/
- L'autorisation pour maîtriser les privilèges de chacun de vos utilisateurs.
  - MongoDB implémente le Role Based Access Control (RBAC).
  - https://docs.mongodb.com/manual/core/authorization/

Il va falloir mettre en place une solution de monitoring pour suivre l'évolution de vos noeuds MongoDB (oui, tous les noeuds). Cela vous permettra de garder un oeil sur l'utilisation de la RAM, du disque et ainsi de prévenir les incidents, ou dans le pire des cas, de recevoir une alerte en cas d'incident pour réagir au plus vite. Idéalement, votre solution de monitoring pourra faire un suivi des métriques systèmes ainsi que des métriques MongoDB tels que le nombre d'écritures par secondes par shard, le lag de réplication entre le primaire et un secondaire, etc.

Ensuite, si une catastrophe absolue arrive, vous devrez être en état de pouvoir redémarrer un nouvel environnement de production et faire une restauration de la donnée à partir d'une sauvegarde… Ce qui sous-entend évidemment qu'il faut faire des sauvegardes complètes du système et prévoir de quoi permettre une restauration la plus rapide possible.

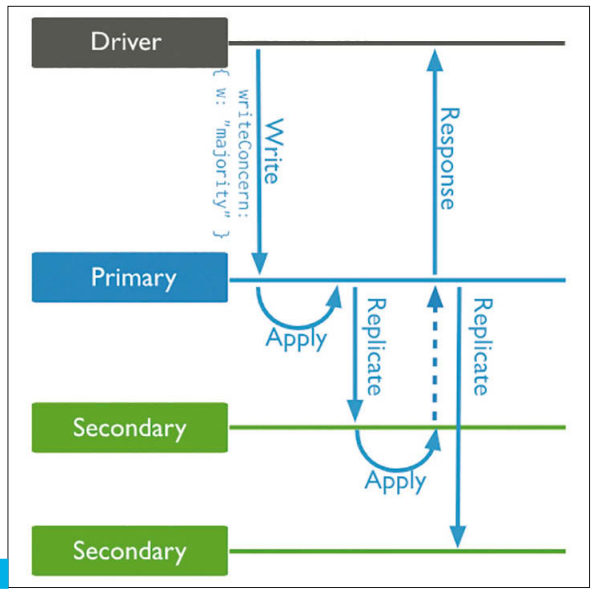
Finalement, pour votre confort, il serait de bon ton d'avoir un système d'automatisation pour :

- administrer vos noeuds MongoDB ;
- faire des déploiements de clusters ;
- faire des mises à jour ;
- faire des maintenances sur ces noeuds.

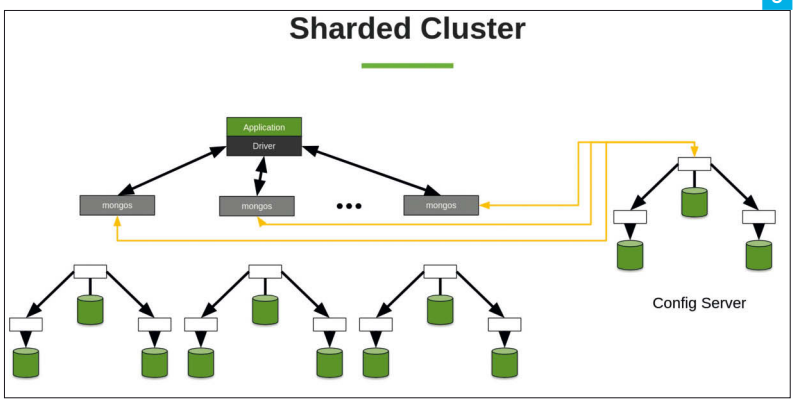
Parfois, c'est juste un nouveau projet qui démarre et il faudra mettre en place un environnement de prod, qualification et développement.

Parfois il faut changer une barrette de RAM ou changer des disques durs, et parfois il faut juste faire une montée de version pour passer à la dernière version de MongoDB car vous êtes un DBA responsable :-) ! Cela paraît simple dit comme ça… Mais quand on a un cluster shardé de plusieurs centaines de machines, ce n'est pas une opération à prendre à la légère car une erreur peut faire du dégât.

Cela dit, comme nous l'avons vu précédemment, MongoDB utilise



4



5

les Replica Sets et permet l'élection d'un nouveau primaire dès qu'un serveur est hors ligne (volontairement ou pas). Toutes les procédures de maintenance de MongoDB utilisent donc les Replica Sets pour permettre de faire des opérations de maintenance avec zéro temps d'indisponibilité… Encore faut-il faire les opérations dans le bon ordre et bien faire attention à ne pas impacter plus de la majorité des noeuds votants d'un Replica Set pour conserver l'intégrité du quorum. https://docs.mongodb.com/manual/replication/

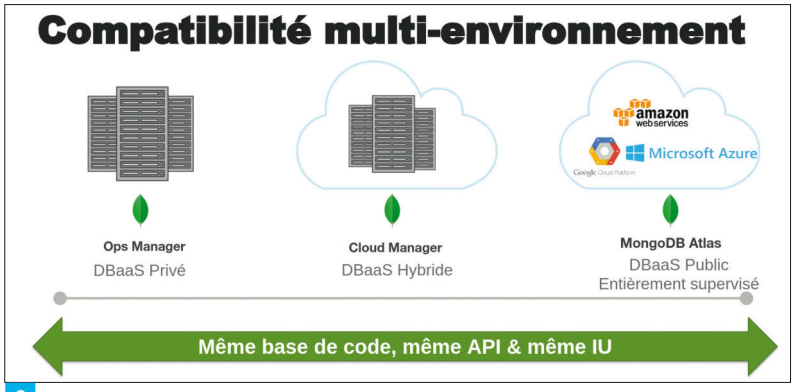
Au passage, oui, j'ai bien dit "zéro temps d'indisponibilité" car même lorsqu'une élection est en cours et que je n'ai donc plus de serveur primaire sur lequel je peux écrire, les drivers dont l'URI contient *"retrywrites=true"* sont capables d'effectuer de nouvelles tentatives d'écriture en cas d'erreur de ce type donc aucune opération n'est perdue ni ne termine en erreur.

Voir https://docs.mongodb.com/manual/core/retryable-writes/index.html.

Bref, un environnement de production n'est pas quelque chose à prendre à la légère, surtout quand la donnée est critique - et bien souvent elle l'est - et représente une grande valeur commerciale pour l'entreprise.

Si vous avez bien lu ce que j'ai écrit ci-dessus, vous avez dû voir que 4 grandes problématiques ressortent:

- Sécurité ;
- Surveillance ;
- Sauvegarde ;
- Automatisation.



6

7

Ne vous inquiétez pas plus, MongoDB a évidemment pensé à vous et il y a plusieurs solutions pour répondre à ces problématiques que ce soit sur vos environnements internes ou dans le Cloud.

Les solutions MongoDB

MongoDB propose 3 produits pour vous accompagner dans ces tâches qui sont très proches en termes de fonctionnalité et d'apparence car ils permettent de résoudre les 4 grosses problématiques mentionnées précédemment mais dans 3 environnements différents.

• MongoDB Ops Manager

C'est la version que vous utiliserez pour vos déploiements locaux. C'est le choix des clients MongoDB qui souhaitent conserver l'entièreté de leurs données et des outils permettant de gérer leurs clusters MongoDB au sein de leur entreprise dans leurs datacenters privés. Ce mode de déploiement implique le déploiement d'une infrastructure supplémentaire qui permet la gestion des backups notamment. Il faudra aussi des serveurs MongoDB supplémentaires qui permettront de stocker les métriques collectées et les topologies des clusters déployés. Bien évidemment, la taille de l'infrastructure d'Ops Manager devra grandir au fur et à mesure que le nombre de noeuds à administrer est grand.

• MongoDB Cloud Manager

Ce mode de déploiement est un compromis entre le Cloud et un déploiement local. Bon nombre d'entreprises ne souhaitent pas confier leurs données à une entreprise tiers mais ne souhaitent pas pour autant s'embarrasser d'une infrastructure de gestion supplémentaire dont ils devraient prendre la charge et la responsabilité. MongoDB Cloud Manager est donc un MongoDB Ops Manager disponible dans le Cloud qui fera le même travail de surveillance, sauvegarde et automatisation mais depuis le Cloud.

Un avantage particulier de Cloud Manager réside dans l'infrastructure de sauvegarde qui est totalement pris en charge par la plateforme et ne nécessite pas de provisionner de stockage.

• MongoDB Atlas

MongoDB Atlas est l'offre PaaS ou DBaaS de MongoDB. On retrouvera ici les mêmes fonctionnalités que précédemment mais en plus on pourra y provisionner à la demande des clusters MongoDB dans les 3 plus grands fournisseurs Cloud : Amazon Web Services, Google Cloud Platform et Microsoft Azure. MongoDB Atlas possède une offre d'essai gratuite, l'idéal pour tester et apprendre. 6

MongoDB Atlas

Une fois que vous aurez créé un compte sur https://cloud.mongodb.com, vous aurez accès à l'interface de MongoDB Atlas et vous pourrez déployer vos premiers clusters en seulement quelques clics.

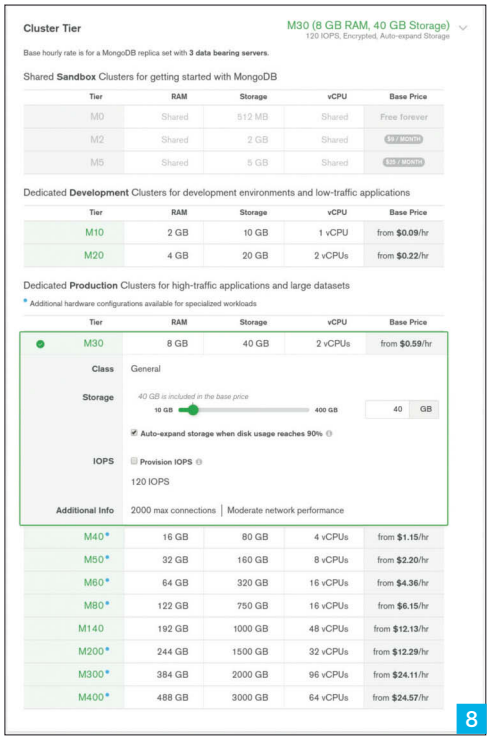
Pour cela vous devrez choisir votre fournisseur de Cloud et la région de déploiement. 7

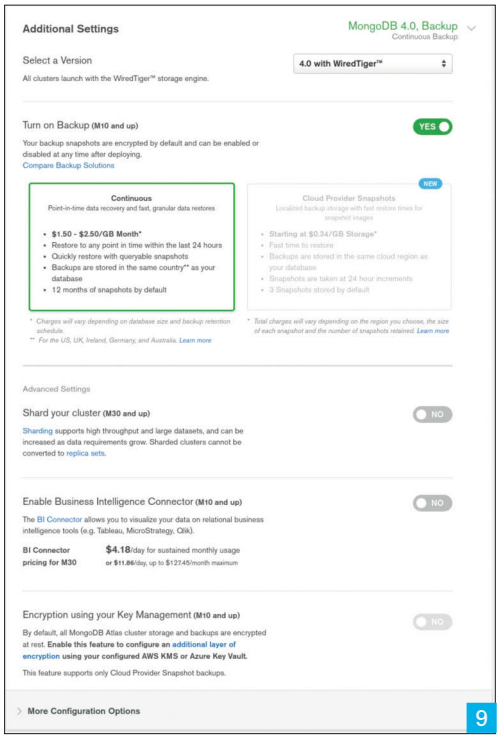
Notez que vous pouvez aussi faire des déploiements multi-régions. Vous pourrez ensuite choisir la taille de votre cluster. Notez bien ici que le prix à l'heure est pour un Replica Set de 3 machines (1 primaire et 2 secondaires). 8

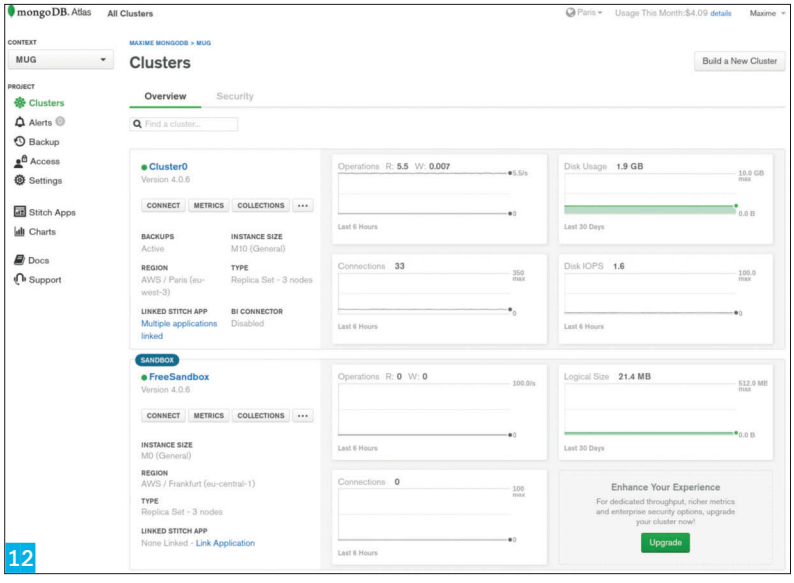
On notera aussi la présence de quelques fonctionnalités pour vous simplifier la vie telles que l'augmentation automatique de la taille des disques durs dès que vous êtes à 90% de remplissage du disque ou la possibilité de choisir les IOPS des disques.
Ensuite vous pourrez choisir la version de MongoDB que vous souhaitez utiliser. Vous pourrez choisir votre stratégie de sauvegarde dont la sauvegarde continue qui permet de faire de la restauration à un point précis dans le temps (Point-In-Time) et d'envoyer une requête directement sur une sauvegarde sans avoir besoin de la restaurer entièrement.
Vous trouverez aussi l'option pour transformer votre Replica Set de 3 noeuds en un cluster shardé (1 clic…), l'option de provisionner un serveur supplémentaire pour avoir un BI Connector qui permet de connecter des outils tels que Tableau, MicroStrategy ou Qlik et pour finir l'option pour chiffrer votre donnée sur disque. 9
Pour finir, il vous faudra choisir le nom de votre cluster - facile ! 10
Au bas de l'écran, vous pourrez consulter le prix mensuel qui inclut tout sauf les choses que nous ne pouvons pas prévoir à l'avance car cela dépendra de votre utilisation du cluster :

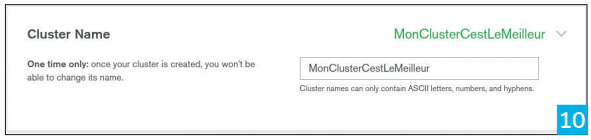
- Le coût des sauvegardes ;
- Le coût des transferts de données.

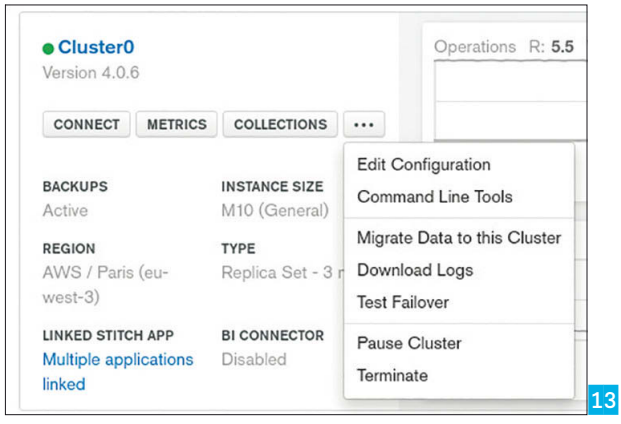
D'après notre expérience, la combinaison de ces 2 coûts combinés se situe entre 10 et 20% du prix mensuel. Pour en savoir plus :
https://www.mongodb.com/cloud/atlas/pricing. 11
Une fois que vous aurez cliqué sur "Create Cluster", la magie opère :-) ! Au bout de 7 à 10 minutes, votre cluster sera prêt à l'emploi et vous le retrouverez sur l'écran d'accueil. 12
Dans cet écran, vous aurez accès aux alertes que vous pouvez configurer et personnaliser ainsi que les sauvegardes où vous pourrez faire vos restaurations.
Vous trouverez aussi les options de gestion des utilisateurs de MongoDB Atlas car vous n'êtes surement pas seul dans votre société et chaque responsable de projet aura son projet Atlas dédié pour ne pas interférer avec les clusters des autres.
Dans les options, vous trouverez un bouton "Edit Configuration" qui vous permettra de revenir vers le formulaire précédent pour faire une montée de version ou pour déménager d'une région de votre Cloud à une autre, par exemple, à chaud et sans interruption de service. 13
Vous y trouverez aussi un outil pour migrer votre cluster MongoDB

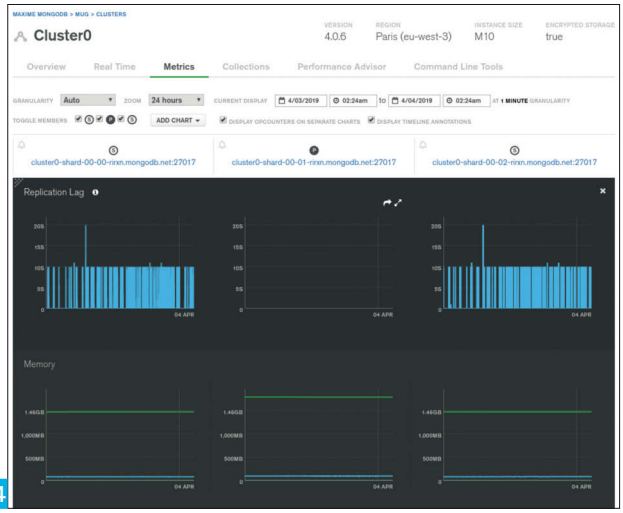




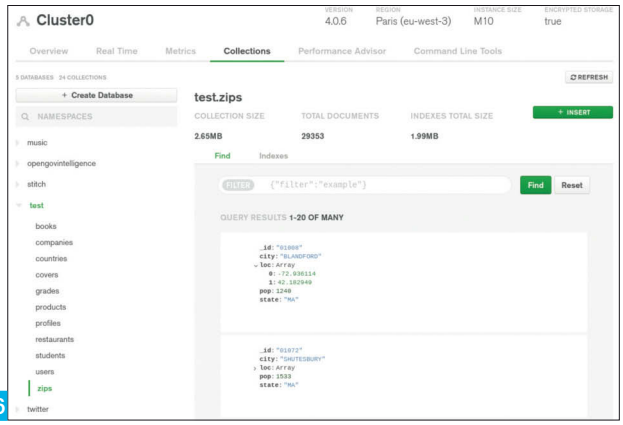




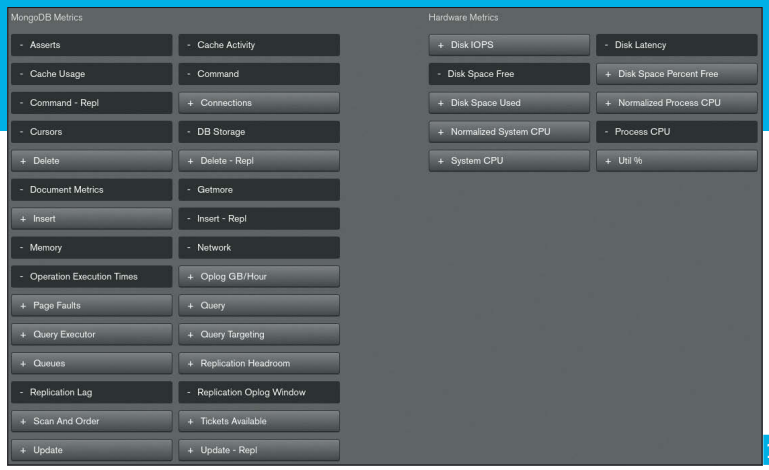




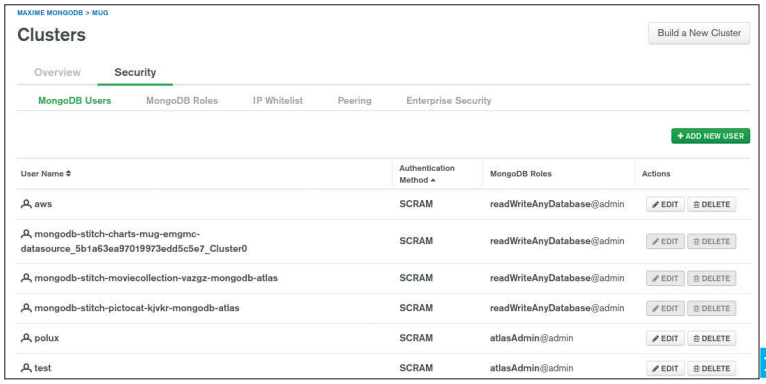
14



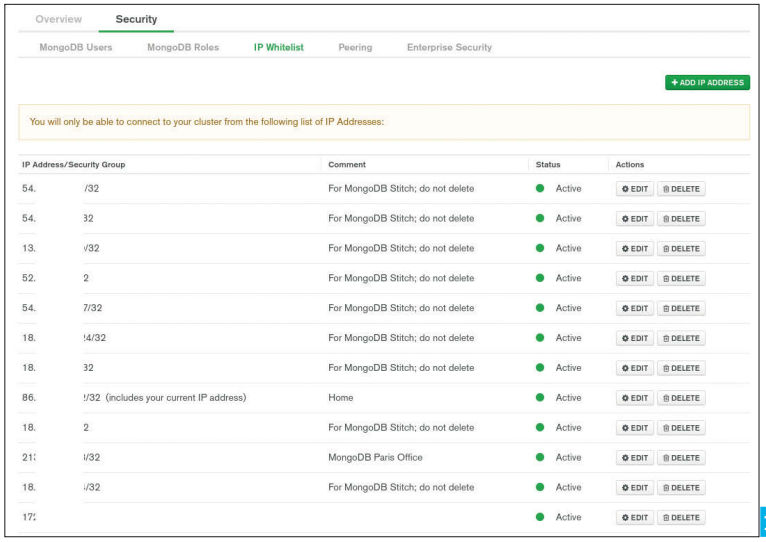
16



15



17



18

actuel vers ce nouveau cluster dans le Cloud et l'option pour mettre en pause votre cluster lorsque vous ne l'utilisez pas pour limiter vos coûts d'infrastructure - pratique pour des serveurs de développement ou de tests. Dans l'onglet "*Metrics*", vous aurez accès à toutes les métriques possibles : 14

Et même plus si vous le souhaitez ! 15

Dans l'onglet "*Collections*" vous pourrez explorer directement la donnée dans votre cluster et la mettre à jour si besoin. Même chose pour les index. 16

Vous trouverez aussi un onglet "*Real Time*" qui vous permet de voir toutes les activités en cours sur votre cluster et l'onglet "*Performance Advisor*" qui vous recommandera les meilleurs index à créer sur votre cluster en se basant sur les requêtes lentes que MongoDB a détectées et tout cela est fait automatiquement.

De retour sur l'écran principal, vous trouverez l'onglet sécurité qui vous permettra de gérer les utilisateurs de vos clusters MongoDB que vous utiliserez pour vous connecter à vos clusters. 17

Vous y trouverez aussi les "*IP Whitelist*" qui définissent les IP/réseaux des serveurs qui ont le droit d'accéder à vos clusters MongoDB. Par défaut, aucune IP n'est paramétrée pour ne laisser aucune chance aux hackers ! 18

Pour finir, vous trouverez aussi ici l'onglet "*Peering*" qui permet de faire de l'appairage avec les réseaux d'AWS et GCP (Azure arrive courant 2019). Vous pourrez connecter directement vos instances MongoDB Atlas avec vos serveurs applicatifs dans votre fournisseur de Cloud sans avoir à exposer vos instances MongoDB à Internet, via des adresses IP locales et le système de sécurité du fournisseur (IAM et Security Groups dans AWS par exemple).

Sécurité maximale ! L'onglet "*Enterprise Security*" vous permettra de configurer LDAP ou encore d'activer les fonctionnalités d'audit pour finaliser la sécurité de vos clusters et répondre à vos obligations RGPD. 19

Si vous souhaitez tester MongoDB Atlas, utilisez le formulaire sur ce lien pour vous inscrire sur la plateforme et vous bénéficierez de 200$ de crédit offert : https://www.mongodb.com/meetatlas.

MongoDB Charts

Depuis peu, MongoDB Charts est aussi disponible depuis l'écran d'accueil. Cet outil vous permettra de construire des tableaux de bord et de les partager pour rendre votre donnée toujours plus vivante. 20

MongoDB Stitch

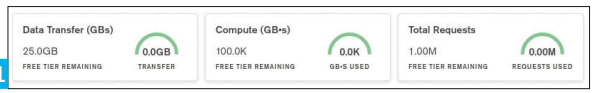
MongoDB Stitch est aussi accessible depuis MongoDB Atlas et cette fois-ci MongoDB propose une offre de *"backend as a service"* en mode *"serverless"* qui vous permettra de déployer une API REST par dessus votre cluster MongoDB Atlas. Vous pourrez aussi utiliser des *"Triggers"* qui vous permettront de déclencher des fonctions Javascript dans un environnement *"serverless"* pour effectuer des traitements à chaque fois qu'un document est inséré, supprimé ou mis à jour dans votre BDD. Et puisque tout cela s'exécute dans un environnement *"serverless"*, la répartition des charges est assurée automatiquement pour assurer une scalabilité sans faille automatiquement. Vous pouvez aussi héberger votre site web directement dans MongoDB Stitch et aller chercher et manipuler des données via le Stitch Javascript SDK une fois que vous serez authentifié via une API d'authentification telle que Google ou Facebook qui sera aussi gérée par MongoDB Stitch :-). Vous lancez votre nouvelle application mobile ? MongoDB Mobile est une version légère de MongoDB qui tourne sur IOS et Android qui est capable de se synchroniser automatiquement via MongoDB Stitch avec le cluster central sur MongoDB Atlas en cas de coupure réseau. Envie d'une bonne nouvelle supplémentaire ? Vous pouvez essayer MongoDB Stitch gratuitement aussi ! 21

Conclusion

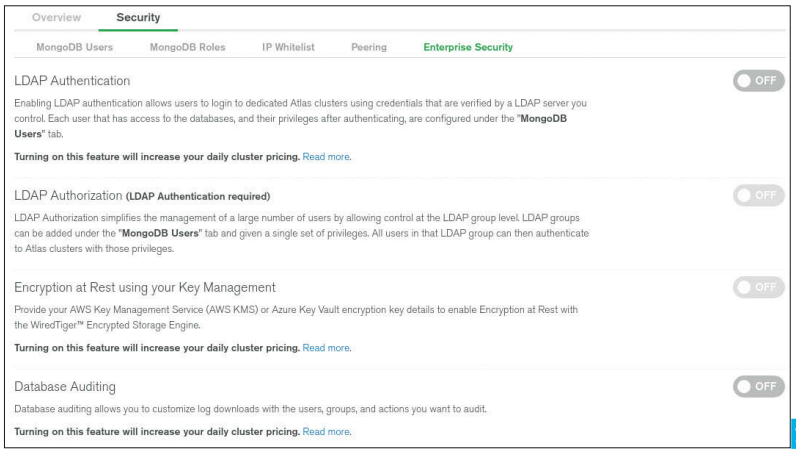
MongoDB est plus qu'une simple base NoSQL, c'est une véritable plateforme de données opérationnelle professionnelle qui a su garder son ADN d'origine : faciliter la vie des développeurs et permettre de lancer rapidement des nouveaux services. Le monde IT ne s'y est pas trompé ;l'adoption de MongoDB est massive et en perpétuelle augmentation, comme en témoignent les téléchargements (45 millions à fin 2018), la dernière étude Forrester Wave 2019 BigData NoSQL qui position MongoDB comme leader du NoSQL, et le classement DB-Engines https://db-engines.com qui place MongoDB cinquième SGBD au classement général toutes BDD - SQL et NoSQL - confondues. MongoDB a fêté ses 10 ans cette année et c'est aussi la première BDD depuis plus de 20 ans à entrer en bourse. C'est un excellent indicateur de bonne santé tout comme le fait qu'il bénéficie d'une excellente notoriété auprès des développeurs et d'une communauté très active. C'est une technologie qui a su convaincre, qui continue d'innover et d'essayer de rendre la vie des développeurs et des porteurs de projets plus simple au quotidien. La version 4.0 de MongoDB a secoué le monde des RDBMS en annonçant le support des transactions ACID multi-documents au sein des Replica Sets et la version 4.2, en cours de développement, a annoncé le support de cette fonctionnalité pour les clusters shardés. MongoDB organise aussi plusieurs conférences annuelles où vous pouvez venir en apprendre encore plus sur MongoDB et son environnement qui évolue tous les jours. MongoDB World - https://www.mongodb.com/world - aura lieu cette année du 17 au 19 Juin 2019 à New York et de nombreuses grandes villes ont une conférence d'une journée appelée MongoDB.local : https://www.mongodb.com/local.

MongoDB organise aussi un Hackathon en ligne mondial autour de MongoDB World : https://www.mongodb.com/world/hackathon avec 42100$ de prix à gagner ! Bonne chance à tous !

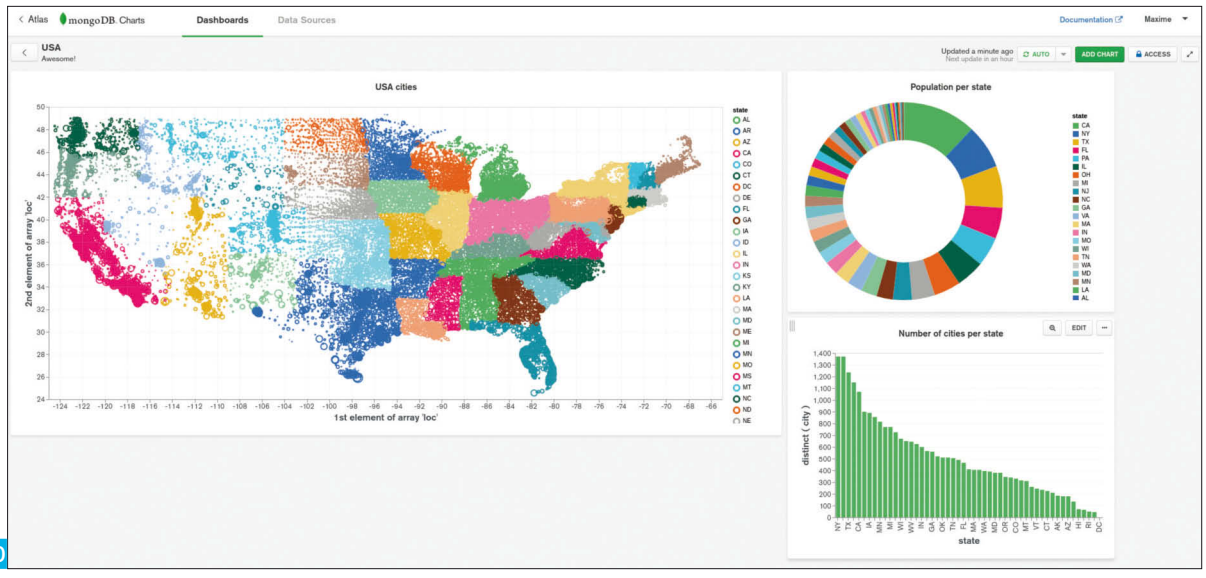
Merci de m'avoir lu, j'espère que cet article vous aura intéressé. Si vous avez des questions n'hésitez pas à me contacter directement : maxime@mongodb.com. •



21



19



20

Cédric Michel
Développeur .Net et Scrum Master, est un collaborateur de la société Satellit. Celle-ci est une entreprise de 70 consultants spécialisée dans le développement Microsoft ainsi que la transformation business, digitale notamment à travers l'AI. Pour en savoir davantage, n'hésitez pas à aller consulter le site : www.satellit.be

Créer votre propre build step pour Azure DevOps Services.

Partie 1

niveau 100

À travers cet article, nous allons premièrement apprendre comment réaliser une extension pour Azure DevOps. Cette extension contiendra un build step qui sera capable de lancer un exécutable (ici, une application console écrite en .NET) et de lui passer des paramètres en argument. Ensuite, nous allons voir comment récupérer le retour de l'exécution afin de visualiser les logs dans votre Azure Pipelines.

Tout au long de cet article, j'utilise le nom MongoSync. Il s'agit du nom d'une extension que j'ai créée et publiée précédemment sur le store et qui me sert d'exemple. Cette extension contenant un build step développé en .NET permet de synchroniser les fonctions Mongo d'une DB source vers une DB cible.
https://marketplace.visualstudio.com/items?itemName=Cedric-Michel.build-release-task
Vous pouvez trouver tout le code source sur GitHub via ce lien :
https://github.com/michelcedric/MongoDbSynchronisation

Créer votre profil (Publisher)

Afin de pouvoir déployer une extension sur la plateforme Azure Devops(Pipelines), il faut tout d'abord créer un compte via le lien suivant : https://marketplace.visualstudio.com/manage/createpublisher
Une fois connecté, compléter le formulaire de création de votre profil. Avant de pouvoir publier votre première extension, il faut attendre que votre profil soit validé : cela peut prendre entre 24 à 48 heures. Lorsque celui-ci est validé vous devriez voir ceci : 1

Préparer votre environnement

Comme nous allons créer une application console, vous aurez besoin de Visual Studio.
Pour créer une extension VSTS, il faut installer le package tfx (TFS Cross Platform Command Line Interface (tfx-cli)) qui est un package npm. Pour ce faire installer node.js. https://nodejs.org/.
Ensuite, ouvrez un command prompt. (Win+R), taper cmd puis enter. Utiliser la commande suivante :

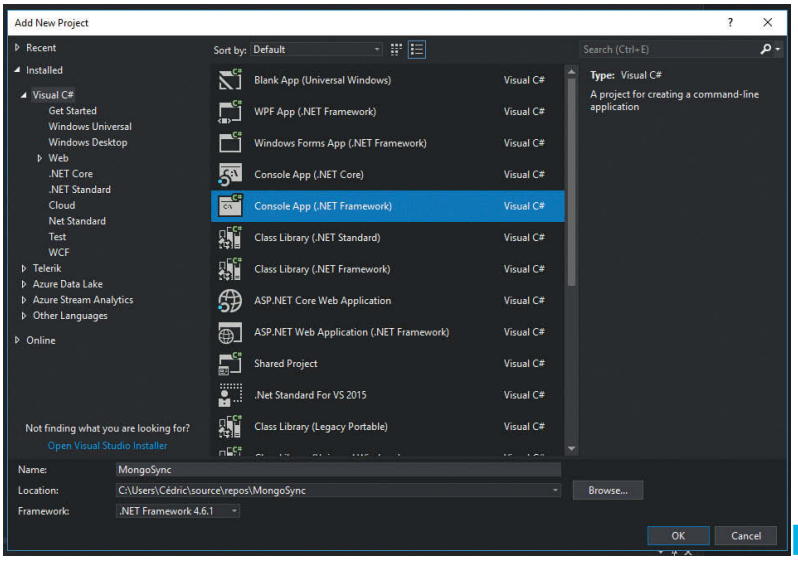
```
npm install -g tfx-cli
```

Grâce à ceci, vous aurez les outils nécessaires à la création de votre extension.

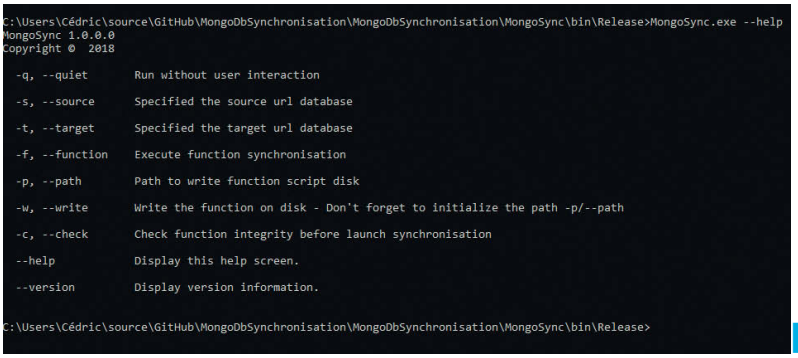
Créer une application console

Premièrement, nous allons créer une solution Visual Studio pour contenir l'exécutable qui devra être exécuté par le build step. La solution contiendra également la définition de l'extension. Ensuite, nous automatiserons la génération du package qui devra être uploadé sur le marketplace afin d'être publié. Nous allons commencer par créer une solution vide dans Visual studio. Ensuite nous allons y ajouter une application console(.NET Framework). 2
Vous êtes libre de développer tout ce dont vous désirez afin que le Build Step l'exécute. L'avantage de faire une application console est que votre application est entièrement indépendante. Vous serez libre de l'utiliser comme build step mais également où vous le souhaitez. Lorsque l'application sera exécutée depuis le build step, nous pourrons lui passer des paramètres.
Pour pouvoir facilement lire les paramètres (arguments), je vous recommande d'utiliser la librairie CommandLineParser.
https://www.nuget.org/packages/CommandLineParser/
Cette librairie vous simplifiera la tâche pour lire chaque argument et vous permettra de générer le –help de votre application console. 3
Pensez également que votre application ne doit pas demander d'inter-

1



2



3

action avec l'utilisateur (-q). Celle-ci pouvant être effectivement lancée depuis un serveur de build.
N'hésitez pas à mettre des logs dans votre application. Ceux-ci seront exploités plus tard et affichés dans votre build step et log.

```
Console.WriteLine("Your Log");
```

Exemple de résultat visible depuis Azure Devops (Pipelines) : 4

Créer un build step

Au niveau de votre .csproj développé précédemment, créez un répertoire

```
Vsts-Extension
```

Dans ce répertoire, composez l'arborescence suivante :

```
|---home
      |--- buildAndReleaseTask
            |--- task.json
            |--- ps_modules
            |--- tool|---tool.ps1
      |--- images
            |--- extension-icon.png
      |--- overview.md
      |---package.bat
      |--- vss-extension.json
```

Fichier task.json

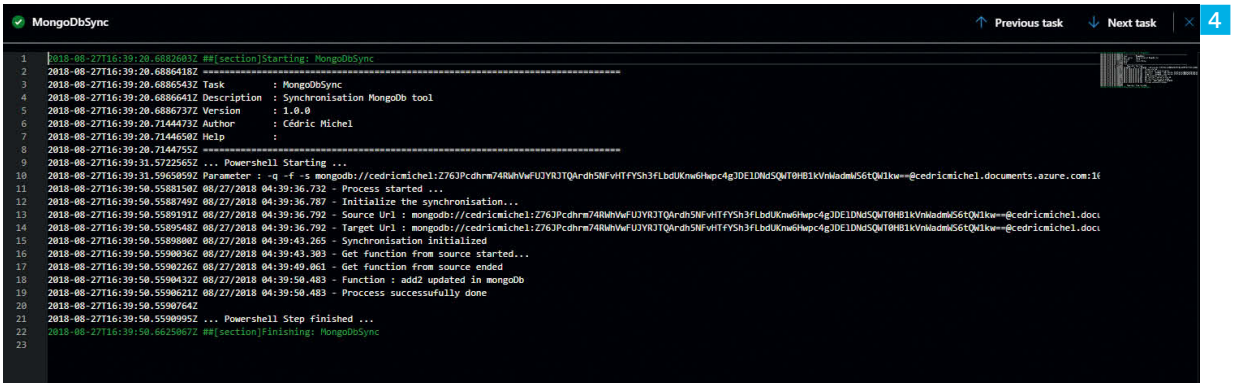
Il s'agit du fichier qui décrit votre build step

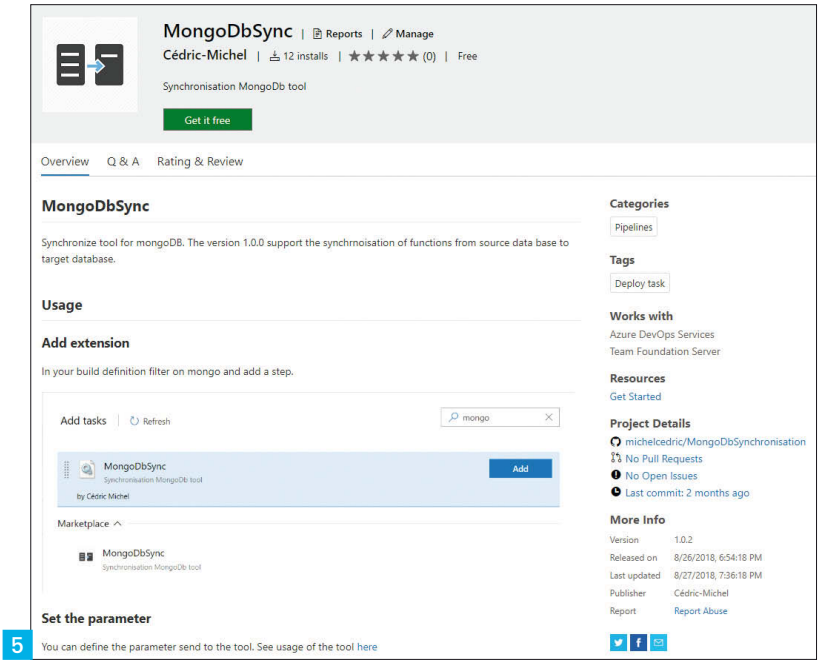
```
{
  "id": "d8cb5786-a111-439e-901d-355062b2941e",
  "name": "MongoDbSync",
  "friendlyName": "MongoDbSync",
  "description": "Synchronisation MongoDb tool",
  "helpMarkDown": "",
  "category": "Deploy",
  "visibility": [
    "Build",
    "Release"
  ],
  "author": "Cédric Michel",
  "version": {
    "Major": 1,
    "Minor": 0,
    "Patch": 0
  },
  "groups": [
   {
     "name": "MongoDbSync",
     "displayName": "MongoDbSync",
     "isExpanded": false
   }
  ],
  "inputs": [
       {
     "name": "parameter",
     "type": "string",
     "label": "Parameter to launch the Synchronisation MongoDb tool",
     "defaultValue": "-q -f",
     "required": true,
     "helpMarkDown": "Parameter send to tool"
   }

  ],
  "execution": {
    "PowerShell3": {
      "target": "tool.ps1",
      "argumentFormat": ""
    }
  }
}
```

Pour l'ID, utiliser un générateur de Guid : (Ex : avec Visual Studio à Tools à Create Guid, https://www.uuidgenerator.net/guid et https://www.guidgenerator.com/, https://www.guidgen.com/)
La partie « inputs » définit les champs qui seront visibles dans votre Azure Pipelines et qui devront être configurés pour utiliser l'extension dans le build step.
Dans la partie « execution », nous ne pouvons pas lancer directement notre application (.exe), nous allons donc lancer un script PowerShell. Ce script doit être du PowerShell V3, il s'agit d'un prérequis pour utiliser le VSTS Task SDK for PowerShell dont nous allons avoir besoin.
https://github.com/Microsoft/azure-pipelines-task-lib/tree/master/powershell/Docs

Répertoire ps_modules

C'est dans ce répertoire que nous allons ajouter le SDK nécessaire.
Ouvrez PowerShell et placez-vous sous le répertoire ps_modules.
Exécutez la commande suivante :





5

```
Save-Module -Name VstsTaskSdk -Path .
```

Acceptez l'ajout du nuget provider.
Vous allez avoir une structure comme ceci :

```
|--- buildAndReleaseTask
       |--- ps_modules
              |--- VstsTaskSdk
                     |--- (numéro de version exemple 0.11.0)
                            |--- (repertoire et fichier du SDK)
```

Malheureusement, avec cette arborescence le script PowerShell ne trouvera pas le SDK. Il faut reprendre les fichiers et répertoires du dernier niveau et supprimer le répertoire avec le numéro de version. Après cette étape vous devriez avoir une structure comme ceci :

```
|--- buildAndReleaseTask
       |--- ps_modules
              |--- VstsTaskSdk
                     |--- (répertoire et fichier du SDK)
```

Grâce à cette étape, nous pourrons utiliser le SDK (VSTS Task) dans notre script PowerShell.

Répertoire tool.

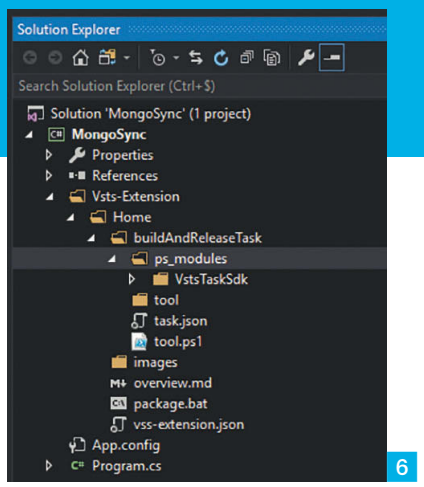
C'est dans ce répertoire que l'on viendra mettre notre exécutable ainsi que toutes les librairies (.Dll) nécessaires à son bon fonctionnement.

Fichier tool.ps1

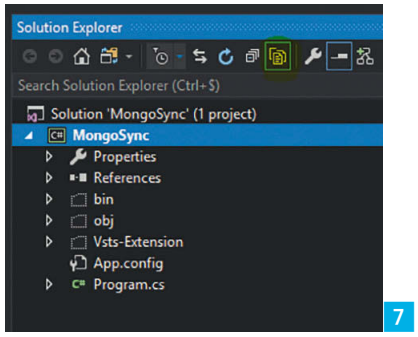
Il s'agit du fichier PowerShell détaillé un peu plus loin.

Fichier overview.md

Il s'agit du fichier Markdown qui sera utilisé sur la page du marketplace. (Fig. 5).



6



7

Répertoire Images

Il s'agit du répertoire dans lequel vous allez devoir déposer l'icône (aux dimensions de 128*128) qui sera affichée sur le marketplace.

Fichier package.bat

Créer un fichier texte package.txt et placez-y ces 2 lignes.

```
echo create extension
tfx extension create --manifest-globs vss-extension.json
```

Changer l'extension en .bat
Il s'agit de la commande qui sera utilisée pour créer le package de votre extension. Cette commande se base sur le manifeste de votre extension (vss-extension.json). La commande génèrera un fichier .vsix. Après la génération vous pouvez vérifier le contenu de votre .vsix. Pour ce faire, changez l'extension de votre .vsix en .zip que vous pourrez dès lors parcourir. Nous automatiserons l'appel à cette commande lors de la compilation de Visual studio.

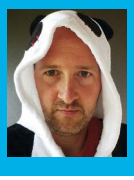
Fichier vss-extension.json

Il s'agit du fichier manifeste de votre extension. Ce fichier contient toutes les informations sur votre extension. Il comprend aussi des liens vers vos fichiers, y compris vos dossiers de tâches et vos images. Ces informations seront disponibles sur la page de présentation de votre extension sur le marketplace. Fig. 1
Vous pouvez retrouver ici l'ensemble de tous les attributs requis et optionnels. https://docs.microsoft.com/en-us/azure/devops/extend/develop/manifest?view=azure-devops&viewFallbackFrom=vsts#required-attributes

Code complet sur programmez.com & github

Maintenant que la structure de notre extension est prête, nous allons l'intégrer dans la solution Visual studio.
Pour ce faire, cliquez sur le nom de votre Projet Console créé précédemment. Puis cliquez sur afficher tous les fichiers (ici en jaune). Le répertoire Vsts-Extension devrait apparaître. Faites un clic droit sur le répertoire et choisissez "Inclure dans le projet". 6
Voici ce qui devrait s'afficher : 7

La suite de cet article dans le prochain numéro



Nicolas De Loof
Bricoleur touche à tout
chez **CloudBees**

niveau 200

A la découverte de Jenkins X

Lorsqu'on parle d'intégration continue, on pense inévitablement à Jenkins. Depuis plus de 10 ans, le projet open-source a conquis toutes les cibles possibles de développement, avec un écosystème de plugin qui permet de couvrir tous les besoins. Un peu comme sur les App Stores, quel que soit votre besoin "il y a un plugin pour ça". Est-ce que la messe est dite ? Non, bien sûr. Des solutions alternatives sont proposées, pour beaucoup en mode Software-as-a-Service, offrant une configuration simple et fonctionnelle dès le premier lancement. Qu'en est-il de Jenkins ?

Jenkins X est né de ce constat : aujourd'hui, mettre en place un service de CI/CD prend du temps, et nécessite de (trop) nombreux choix. Pourtant, les bonnes pratiques sont bien établies, et la domination de Kubernetes pour ce qui est de gérer l'hébergement des applications "cloud-native" (i.e. les trucs modernes bien faits) renvoie bon nombre de plugins Jenkins aux oubliettes.
Jenkins X s'est donc donné pour mission de proposer un service de CI/CD totalement intégré, avec un workflow préétabli, ciblant le développement et le déploiement d'applications cloud-native sur Kubernetes. Dans cet article je vous propose de découvrir Jenkins X par la pratique. Mais dans un premier temps, jetons un coup d'oeil au menu :

- Mise en bouche "batteries included"
- Entrée : Carpaccio de Kube sur lit de Cloud-Native
- Plat : Risotto de CI/CD façon GitOps
- Dessert : fondant de CI à la sauce Serverless
- Café Gourmand
- Digestif Tektonique

Mettre en place une chaîne de Continuous Delivery, c'est déjà un exercice en soit. Lorsqu'on ajoute comme cible un déploiement Kubernetes, on épaissit sensiblement le trait : à la complexité intrinsèque pour gérer le build des projets et leur promotion des environnements de test et préproduction vers la cible finale, on ajoute la difficulté de packager des images Docker, de les héberger ainsi que les artefacts de déploiement propres à Kubernetes.
Dans le monde traditionnel de Jenkins 2.x, vous seriez amené à éplucher de nombreux articles pour trouver la bonne combinaison de plugins et de fragments de Jenkinsfile adaptée à votre besoin. Votre consultant en CI/CD sera d'ailleurs ravi de vous faire un devis dans ce sens. Mais Jenkins X propose une autre approche : offrir une solution "batteries included". Voyons en quoi cela consiste.

Jenkins X se focalise sur l'expérience utilisateur pour rendre l'adoption d'un process de CI/CD sur Kubernetes aussi simple et rapide que possible. Son installation consiste à - tout simplement - télécharger un binaire que nous utiliserons en ligne de commande. Rendez-vous donc sur jenkins-x.io, cliquez le bouton "Get Started" qui vient aguicher votre regard en plein milieu de la page, et choisissez "Get jx".
Notez au passage qu'une documentation plutôt complète vous guide sur les divers aspects de Jenkins X et de son utilisation, mais comme vous êtes en train de lire cet article laissons là de côté pour le moment.Je vous laisse le soin d'installer jx selon votre système.
Pour la suite de cet article, nous allons avoir besoin :

- D'un compte GitHub - allez, vous êtes lecteur de Programmez, je ne peux pas croire que vous n'en ayezez pas déjà un !
- D'un compte Google Cloud - si vous n'en avez pas, Google offre 300$ d'essai, ça ne vous engage à rien.

Nous utiliserons ici Google Cloud, mais Jenkins X peut tout aussi bien utiliser Amazon, Azure, Oracle Cloud, et tout un tas d'autres qui n'attendent qu'une pull-request pour être supportés (je pense à toi, OVH…) aussi bien qu'un cluster Kubernetes que vous gérez vous-même, aventuriers que vous êtes.

Mise en bouche

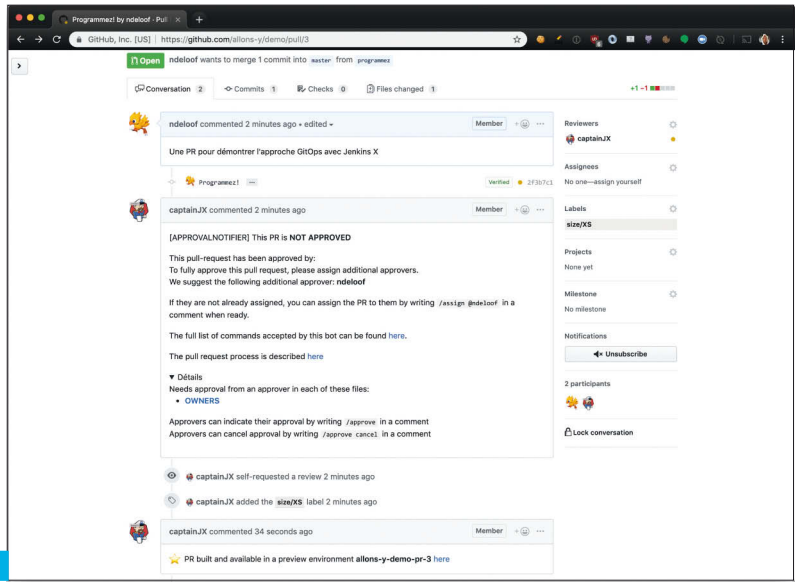
Nous voilà fin prêts, lançons le grand bal ! Nous allons donc :

- Créer un cluster Kubernetes ;
- Installer un registry Docker pour les images … Docker ;
- Installer un registry ChartMuseum pour les Helm Charts Kubernetes ;
- Installer le service de build Jenkins X ;
- Configurer tout ce petit monde pour réagir aux commits GitHub ;
- Définir des namespaces Kubernetes pour nos environnements : staging et production ;
- Configurer des dépôts Git pour gérer en version l'état de ces environnements ;
- C'est déjà pas mal non ?

Ai-je précisé que ce n'est que l'amuse-bouche ? Non ne fuyez pas pour éviter l'indigestion : Jenkins X a été conçu dans un esprit "batteries included", c'est à dire que tout cela vient pré-cablé sans que vous ayez besoin de mettre le nez dedans. Une simple commande à passer :

```
→ jx create cluster gke
```

Jenkins X va vous demander de vous authentifier sur le service Google Cloud, puis va vous poser un ensemble de questions pour lesquelles vous n'avez en général qu'à choisir la valeur par défaut. Précisons au passage, que ce "batteries included" vient avec son corollaire "but replaceable": si vous êtes allergique au Docker Registry, le chef vous propose un assiette spéciale en remplaçant cet ingrédient par celui de votre choix. Le sujet étant vaste je vous



1

laisse le soin de creuser la documentation pour trouver les options disponibles dans ce sens.

Lors du process d'installation, Jenkins X va vous demander le type d'installation Jenkins que vous désirez. Dans cet article, nous allons sélectionner "Serverless Jenkins" parce que nous sommes hype, et aussi parce que ce sera rapidement le choix par défaut. L'autre option consiste à gérer pour vous des Jenkins master "classiques", certes sur Kubernetes, mais pourquoi faire les choses à moitié ?

Quelques minutes plus tard, après avoir créé le cluster et installé tout un tas de trucs de plomberie, notre infrastructure Jenkins X est en place. Nous allons pouvoir passer aux choses réellement intéressantes.

Créer un projet

La commande jx create, que nous venons d'utiliser, permet de mettre en place diverses ressources de notre cluster Jenkins X. Entre autres … une nouvelle application ! Parce que oui, aussi élégante soit l'architecture de notre cluster, tant qu'il n'héberge pas d'application il ne sert à rien. Jenkins X propose une intégration avec la plupart des wizards de création d'application, du désormais courant JHipster (voir l'édition de Programmez consacrée au sujet) à des choses moins conventionnelles comme Lile… Et il n'attend que vos pull-request pour en supporter d'autres. Pour le reste (!) l'équipe Jenkins X maintien des "quickstart", qui sont des squelettes de projets préconfigurés pour un déploiement Kubernetes.
Pour illustrer cet article je vais utiliser un quickstart nodejs. Essayez avec la stack qui vous correspond ça ne changera pas le déroulé de ce tutoriel. Entrons donc la commande :

```
→ jx create quickstart
```

Jenkins X vous propose alors les quickstarts à sa disposition. Ils sont directement issus du projet https://github.com/jenkins-x-quickstarts auquel vous pouvez contribuer si vous voyez un élément critique manquant pour votre stack préférée, ou bien si vous avez de nouvelles idées de quickstart. Et oui, bien sûr, vous pouvez configurer votre propre source pour vos quickstarts custom ! Dans mon cas, je me contenterai du quickstart nodeJS standard.

```
About to create repository on server https://github.com with user ndeloof
? Which organisation do you want to use? allons-y
? Enter the new repository name: demo

Creating repository allons-y/demo
? select the quickstart you wish to create nod [Use arrows to move, type to filter]
> node-http
```

Quelques secondes plus tard, nous voilà avec un squelette de code généré, poussé sur GitHub - rien de transcendant. Mais si nous attendons encore quelques instants, nous pourrons passer la commande suivante :

```
→ jx get applications
APPLICATION STAGING PODS URL
demo      0.0.1  1/1 http://demo.jx-staging.timey-wimey.io
```

Comme promis, Jenkins X a mis en place la CI/CD pour notre tout nouveau projet, ce qui inclut le build du code, son packaging dans un conteneur Docker, le déploiement sur Kubernetes à l'aide d'un Helm Chart, et le tout entièrement géré en version. Mais nous n'en sommes qu'à l'entrée, passons donc au plat principal.

CI/CD en mode GitOps

L'approche "GitOps" consiste à gérer toute, absolument toute intervention sur un environnement, via un commit Git - tout est donc tracé, versionné, et en principe réversible - et via l'automatisation complète des déploiements. Le nom de ce point de vue n'est d'ailleurs pas très révélateur, car l'important n'est pas de stocker dans Git, mais plus de pouvoir à tout instant appliquer à l'identique la même configuration via une automatisation complète.

Plutôt que de longs discours, attaquons le plat de résistance par la pratique. Nous allons donc nous placer dans la peau d'un développeur (si vous lisez les pages de ce magazine et que vous n'êtes pas ma Maman, ça ne devrait pas être très dur) et développer une nouvelle fonctionnalité super géniale pour notre application.
Edition du code, petit test en local, et hop, je commit mes modifications, que je pousse dans une branche avant d'ouvrir une majestueuse pull-request. Pour ce tutoriel je me contente de modifier le texte de la page d'accueil pour qu'il salue le magazine qui nous édite, je laisse à votre imagination le soin d'inventer de *vraies* fonctionnalités pour rentabiliser vos heures de travail. 1

La création de cette Pull-Request a déclenché un ensemble d'actions sur notre infrastructure Jenkins X. D'une part, CaptainJX (le petit nom que j'ai donné à mon bot-user Jenkins X) nous confirme la bonne prise en compte de cette PR, via un premier commentaire qui nous rappelle au passage le processus de validation.

Entre autres, Il nous informe que les personnes autorisées à approuver ce code sont listées dans un fichier OWNER. Cette subtilité permet de subdiviser un gros dépôt en plusieurs sous-parties

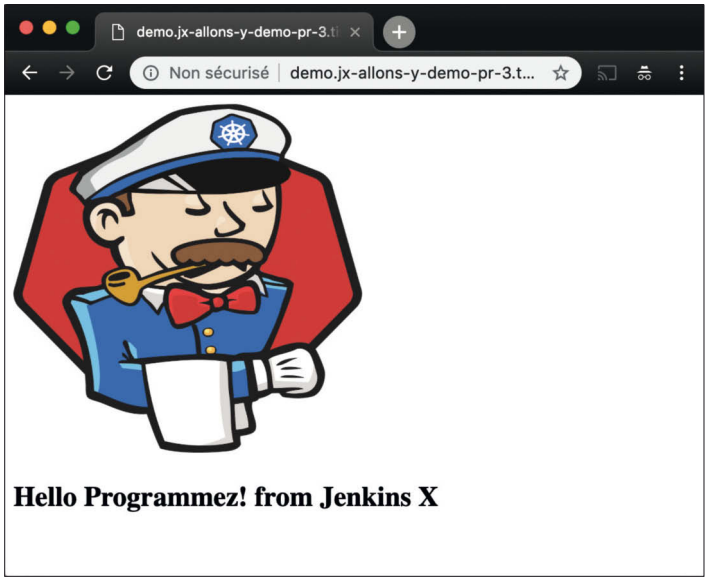
avec des responsables dédiés. C'est un petit pas vers une approche mono-repo qui vous est proposée. Ensuite, Jenkins X nous informe d'un ensemble de *commandes* que nous pouvons utiliser.

Rappelez-vous qu'un des éléments de l'approche GitOps est que **tout** est tracé dans Git. Ici Jenkins X utiliser les commentaires GitHub comme UI pour vous permettre de déclencher des actions, et de faire avancer la Pull-Request dans le processus de validation. La liste des commandes est généreuse, aussi nous ne l'explorerons pas en détail, et elle est de plus extensible avec vos propres commandes (ce qui dépasse largement le cadre de ce tutoriel :P). Je vous renvoie donc vers la documentation officielle sur https://prow.k8s.io/command-help.
Ces commandes permettent de modifier le statut de la Pull-Request, typiquement de l'approuver si vous faites parties des *owners* enregistrés dans le fichier du même nom, de l'assigner à une personne, de lancer des tests précis, ou d'afficher une image de chat - indispensable. Toutes ces choses, vous pourriez les faire en configurant finement votre dépôt GitHub, l'intérêt ici est que cette configuration est implicite, au travers du fichier OWNER, et versionnée puisque celui-ci est stocké dans le dépôt lui-même. Toute modification de statut est tracée par un commentaire, et donc vous pouvez à tout moment auditer l'historique complet du projet.

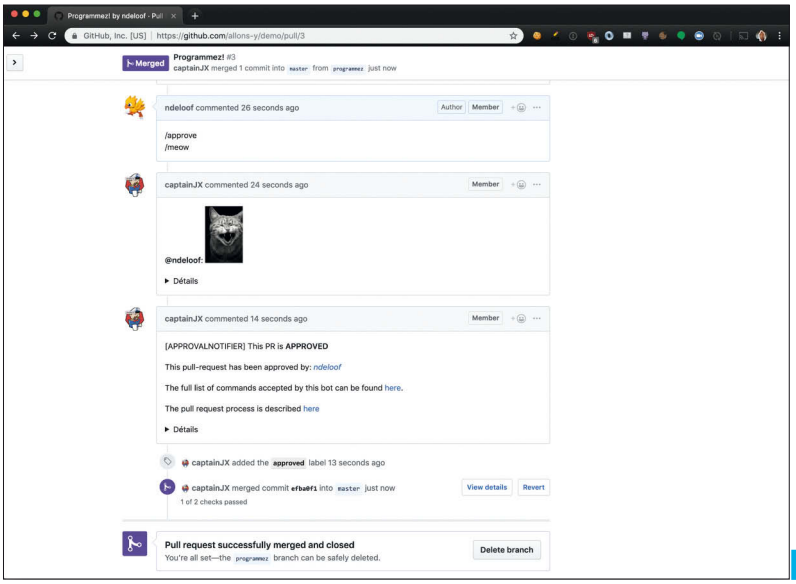
Le commentaire suivant nous propose d'aller inspecter la Pull-Request sous un angle différent de la revue de code, en accédant à un environnement de pré-visualisation. Parce que la revue de code sur les styles CSS ça peut être pertinent, mais souvent on a tout de même besoin de voir ce que ça fait *en vrai* pour être totalement confiant. Notre application est donc déployée sous une URL dédiée, que nous pouvons utiliser pour inspecter le résultat époustouflant de mon travail : 2

Dans le squelette de code produit par notre quickstart, Jenkins X a en effet inclus un ensemble de fichiers de déploiement propres à Kubernetes (Helm Charts), et ce en deux variantes : une pour un déploiement "normal", et une pour un déploiement de prévisualisation. Jenkins X a donc déployé notre application dans un petit coin du cluster, dans un namespace Kubernetes dédié au développement, conformément à la variante *preview* du Helm Chart de déploiement. Ce distinguo peut être utilisé pour ajuster l'environnement de prévisualisation afin de le rendre pertinent pour la revue de code, par exemple avec une base de données dédiée, utilisée à des fins de tests.

Le *commit status* de notre pull-request révèle l'état courant : le build du projet a fonctionné, par contre un *check* reste bloqué avec l'indication "*Needs approved label*". Comme expliqué précédemment, Jenkins X a totalement instrumenté le process de revue de Pull-Request et nous interdit d'aller plus loin tant qu'un *owner* n'aura pas indiqué son accord. Ce que nous faisons donc immédiatement, confiant dans la vérification effectuée sur l'environnement de prévisualisation. Je saisis donc le commentaire-commande /approve. Et j'y ajoute un chat, parce que ça me semble totalement indispensable: /meow. 3



2



3

Jenkins X réagit immédiatement, d'une part en allant piocher une image de chat au hasard, ce qui égaye ma journée, et d'autre part en traçant l'approbation de cette Pull-Request, via un commentaire et un label, ce qui déclenche enfin la fusion de mon code sur la branche *master* du dépôt.

Bon, mais encore ? Mon morceau de code tout neuf est désormais dans *master*. Ce nouveau lot de commits a déclenché un nouvel événement GitHub, qui a été intercepté par Jenkins X pour lancer un nouveau traitement associé à cette branche : builder l'application, et la déployer en environnement *staging* (préproduction).

Contrairement à l'environnement *preview* que nous avons déjà rencontré, le *staging* est un environnement permanent, dont l'état est entièrement géré par configuration, laquelle est entièrement stockée dans Git. Lors de l'installation, Jenkins X a créé un dépôt GitHub dédié à la gestion de cet environnement. Sur l'événement

"*push sur master*" Jenkins X lance donc un nouveau traitement :

- Construire et versionner l'application (pose d'un tag) ;
- Packaging et déploiement d'une image Docker ;
- Mise à jour de la définition de l'environnement staging (le Helm Chart associé à notre application) via un commit dans le dépôt associé.
- Application des modifications en conformité avec l'état du dépôt environment-staging.

Chacune de ces étapes repose sur la combinaison événement Git => traitement automatisé par Jenkins X. Chaque traitement reste ainsi simple et générique. Par exemple, la modification d'un environnement est lancée dès que le dépôt associé est mis à jour, que ce soit via le merge d'une Pull-Request, ou par un administrateur qui ferait une modification directe. Cela permet au final de venir faire évoluer le processus sans devoir éditer un script long et complexe.

Suite à l'enchaînement de ces traitements, notre environnement "*staging*" a été mis à jour avec la toute dernière version de notre application. Cet environnement va nous permettre de faire les derniers contrôles sur notre application, avant de l'envoyer en production.

Pour cette dernière étape, on parle de "promotion". Il existe deux approches courantes pour cette phase finale. La première consiste à disposer d'une branche "production" dans le dépôt Git, et de construire l'application à partir du code source pour l'installer dans l'environnement cible. L'autre approche consiste à utiliser les binaires, tels qu'ils ont été validés dans les étapes précédentes, à l'identique. Jenkins X a choisi la seconde option, qui embrasse complètement l'immutabilité et la portabilité des images Docker. Bien évidemment, vous l'aurez compris, la nature événementielle de Jenkins X, centrée sur Git, vous permet de venir ajuster l'ensemble du processus pour vos besoins.

Les environnements *staging* et *production* de Jenkins X ne sont pas des branches d'un même dépôt Git, mais bien deux dépôts séparés. Vous pouvez d'ailleurs créer d'autres environnements à votre convenance pour mettre en place un processus plus riche, avec des phases de QA supplémentaires. Effectuer la promotion de notre application d'un environnement à un autre va donc consister à répliquer la description de notre application telle que définie dans le dépôt *staging*, dans les fichiers de configuration du dépôt *production*. Croyez-vous vraiment que Jenkins X vous laisserait gérer manuellement une opération de cet ordre sans vous fournir l'outillage nécessaire ? Sur l'environnement *production* ? Voyons voyons, il me semble qu'il est temps de prendre un peu de recul sur ce délicieux mais un peu lourd plat "GitOps"

Trou normand : promotion

La promotion d'un environnement à l'autre est entièrement automatisée par Jenkins X, sous la forme d'une commande dédiée. Celle-ci va dérouler une suite d'événements Git et les traitements associés comparable à ce que nous avons vu précédemment, mais cette fois associée à l'environnement *production*.

```
→ jx promote --version 0.0.7
? Pick environment: production
Promoting app demo version 0.0.7 to namespace jx-production
pipeline allons-y/demo/master
Created Pull Request: https://github.com/allons-y/environment-production/pull/7

pipeline allons-y/demo/master
Pull Request https://github.com/allons-y/environment-production/pull/7 is merged at sha
 2ba6a33d217f04444aff32a3ccd5c979bbc9e3e5
```

Notez au passage un autre intérêt de l'approche GitOps : au travers de ces commandes et process outillés par Jenkins X, nous donnons la possibilité à nos développeurs de déployer leur application - sous réserve de respecter le processus - sans pour autant leur donner un accès au dépôt de référence de l'environnement cible, ni évidemment à l'environnement cible.

Précisons au passage que pour ce tutoriel l'environnement cible est basé sur un namespace Kubernetes dédié, "jx-production". Dans des environnements plus réalistes, nous ciblerions probablement un cluster Kubernetes différent, pour offrir une isolation physique entre les activités de développement / QA et la production.

Dessert : Serverless

Après ce bon repas, vous prendrez-bien une petite part de Serverless ? Oui, ce fameux buzzword dérivé à toutes les sauces pour embellir toutes les promesses marketing.

Commençons par clarifier les choses. Par *Serverless*, nous allons ici prendre la définition telle que proposée par le Serverless Working Group de la Cloud Native Computing Foundation. Des gens qui a priori savent de quoi ils parlent, plus en tout cas que le service marketing de la société AchetezMonTruc.com.

"Serverless computing refers to the concept of building and running applications that do not require server management."

Pour être plus concret, deux éléments phare sont considérés par ce working group comme piliers fondamentaux d'une solution Serverless :

- Pas de frais, de délais, d'interruption pour maintenance : "Zero Server Ops" ;
- Pas de coûts lorsqu'aucune charge n'est demandée : "No Compute Cost When Idle".

Toute personne qui a un jour géré un serveur Jenkins sait qu'on est très loin de cette promesse, pourtant nous avons au début de ce tutoriel choisi l'option "Serverless Jenkins". Alors, coup de bluff ?

Vous l'aurez sans doute compris, Jenkins X partage peu de code avec le Jenkins traditionnel, même s'il utilise la même syntaxe de définition des Pipelines au travers d'un fichier Jenkinsfile, créé pour nous dans notre squelette de projet. En mode Serverless, Jenkins X remplace bon nombre de composants internes de Jenkins par des briques issues de l'écosystème Kubernetes. Le résultat, vous l'expérimentez depuis le début de ce tutoriel : à aucun moment nous

n'avons installé de plugin Jenkins, géré la taille d'un pool d'agents, l'espace de stockage de nos logs, ou programmé une mise à jour…

Comme nous l'avons vu, Jenkins X utilise un mécanisme événementiel basé sur son interaction avec nos dépôts Git. En conséquence, il n'a plus besoin d'héberger à temps plein un service de build en attente de travail, mais peut simplement provisionner des exécuteurs temporaires en fonction de la charge, bien isolés au fond d'un conteneur. Ceux-ci sont chargés d'exécuter le script Jenkinsfile, avec juste ce qu'il faut de Jenkins pour assurer ce service. Et ce Jenkins allégé est maintenu par l'équipe Jenkins X. Vous pouvez bien sûr prendre la main dessus si vous désirez exploiter un plugin particulier dans vos Jenkinsfile.

Par ailleurs, l'interaction utilisateur étant centrée dans GitHub, plus besoin de la fastidieuse UI de Jenkins. Ce qui ne veut pas dire que vous êtes aveugle - il serait sinon bien délicat de diagnostiquer une erreur de build ! Quid de vos logs, mais aussi des résultats de test, des rapports de couverture de test ?

Pendant l'exécution, nous pouvons récupérer les logs de notre build via la CLI :

```
→ ~ jx get build logs allons-y/demo/master
Build logs for allons-y/demo/master #10
getting the log for pod 499273ec-40c5-11e9-be57-6c40089f1c7e-pod-a402ac and init container
 build-step-jenkins
Picked up _JAVA_OPTIONS: -Xmx400m
Started
Running in Durability level: PERFORMANCE_OPTIMIZED
[Pipeline] node
Running on Jenkins in /tmp/jenkinsTests.tmp/jenkins12636774558226371test/workspace/job
…
```

Après exécution, ce même log est stocké par Jenkins X. Sauf configuration d'un mécanisme de stockage dédié, Jenkins X va tout simplement pousser le log dans la branche gh-pages de notre dépôt GitHub. C'est la configuration par défaut, choisie parce qu'elle est portable sur toutes les plateformes. Mais vous pouvez évidemment configurer Jenkins X pour utiliser un stockage autre, adapté à votre infrastructure - Amazon S3, Google Cloud Storage, ou Azure Blobs…

```
→ ~ jx get storage
CLASSIFICATION LOCATION
coverage    current git repo
logs        current git repo
tests       current git repo
```

Si vous n'êtes pas fan de la ligne de commande, ou que vous avez pris l'habitude de disposer d'un dashboard des builds de votre équipe, vous vous dites sans doute que tout cela est un peu *roots*. Soit. Tout le monde n'a pas les mêmes goûts, ni les mêmes besoins.

Café Gourmand

Jenkins X apporte un ensemble de composants essentiels à la mise en place de l'approche GitOps. Mais dans beaucoup de cas pratiques il va manquer un élément déterminant - à commencer par une web UI !

Jenkins X a donc été conçu pour être extensible, utilisant les mécanismes de partage de données dans Kubernetes basé sur des *Custom Resources Definitions* (CRD). Ainsi, les données d'exécution d'un Pipeline sont stockées et accessibles à des outils tiers, qui sont appelées dans le vocabulaire Jenkins X des "*apps*" (vous lirez peut-être "*add-ons*" dans d'anciennes documentations).

Ces apps permettent par exemple d'installer sur notre Cluster un service Sonar pour effectuer de la qualimétrie, Anchore pour analyser la sécurité des images Docker produites, ou bien d'enrichir notre cluster avec istio, qui permettra de mettre en place des méthodes de déploiements plus subtiles comme du Canary testing.

Jenkins X héberge plusieurs apps sur https://github.com/jenkins-x-apps, et vous pouvez évidemment définir les vôtres - ce que CloudBees s'est empressé de faire pour proposer une web UI digne de ce nom, qui devrait être accessible en freemium à l'heure où vous lirez ces lignes.

```
→ jx add app cloudbees --basic
Updating Helm repository...
(...)
Helm repository update done.
App installed successfully.

 jx cloudbees Open the app in a browser
```

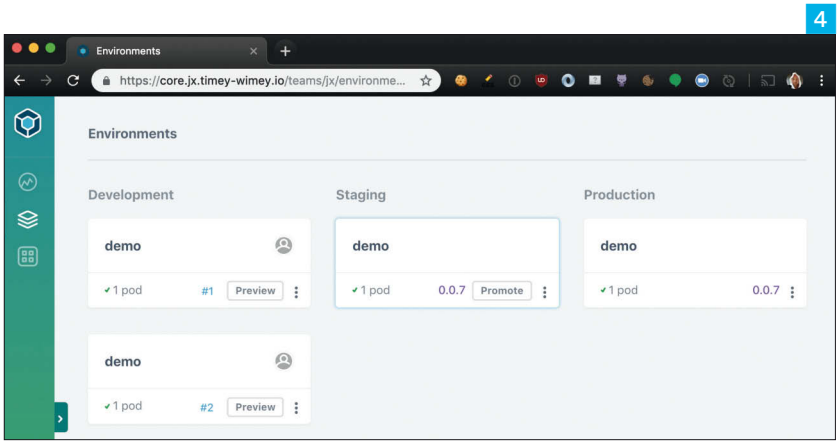
Soyez patient le temps que l'appli soit téléchargée et déployée sur votre cluster, puis lancez la commande :

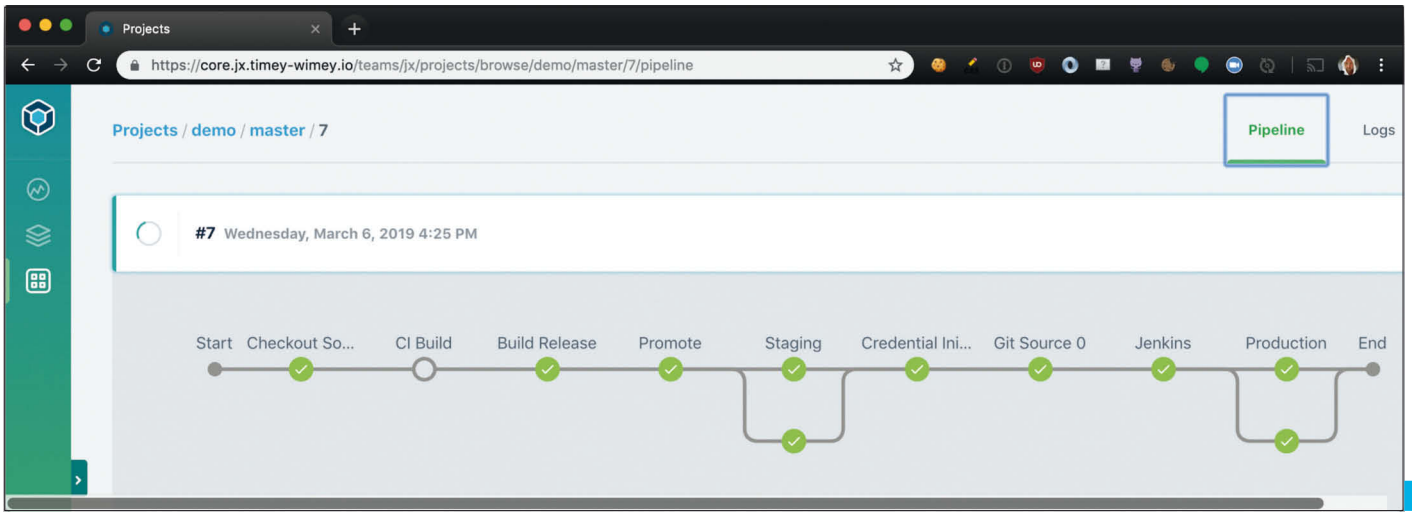
```
→ jx cloudbees
```

4

Cette application web vous donne accès aux composants Kubernetes gérés par Jenkins X : les Teams, dont nous n'avons pas parlé dans ce tutoriel, qui permettent de gérer des ressources dédiées par équipes, les Environnements sur lesquels vous pouvez suivre les applications déployées, et l'activité de vos Pipelines sur les diverses branches des dépôts projets. On retrouve ici l'interface Blue Ocean issue de Jenkins, qui permet aussi bien d'accéder au log que pour avoir une vision globale du Pipeline exécuté. 5

4





5

Digestif

Pour terminer ce repas qui, j'espère, vous a plu, je vous propose un petit Digestif pour parler de l'avenir.

Le petit monde du Continuous Delivery est riche et en plein essor. Jenkins X est l'un des projets qui a rejoint la **Continuous Delivery Foundation**, entité indépendante rattachée à la Linux Foundation, et financée par les principaux acteurs du marché.

Dans cet écrin, Jenkins X côtoie un autre projet, initié par Google : Tekton. Il s'agit d'un mécanisme de Pipelines natif Kubernetes, basé comme toujours sur la définition de CRD permettant diverses intégrations. Et comme la collaboration permet de proposer quelque chose qui soit plus que la simple somme des parties, Jenkins X travaille déjà à l'intégration de Tekton, comme alternative au Jenkinsfile traditionnel de Jenkins, via une syntaxe yaml simple et purement déclarative.

De la même manière, l'extensibilité de Jenkins X via des *apps* va permettre d'intégrer des outils tiers, se focalisant sur des aspects plus spécifiques du Continuous Delivery. La "coopétition" au sein de la CDF va permettre de faire émerger de nouvelles idées et des outils toujours plus riches, plus ouverts, et mieux intégrés pour permettre à chacun de nous d'implémenter en un tour de main toutes les bonnes pratiques du Continuous Delivery.

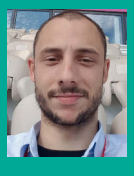
Le Continuous Delivery, si le concept n'est pas nouveau, est en plein développement et atteint un niveau de maturité qui en fait un domaine stratégique de notre métier. Jenkins X a fait le choix de moderniser l'architecture de Jenkins sans concession, et offre aujourd'hui une solution moderne, élégante, et ouverte pour préparer l'avenir.

« Vers l'infini, et au-delà !
--- Buzz l'éclair »

Oui, le Cognac ça tape un peu, et après cette envolée lyrique, j'espère que vous aurez apprécié ce rapide tour de Jenkins X. Rapide,

parce qu'il y aurait encore tant de choses à dire ! Aussi, si vous voulez en savoir plus, je vous invite à tout simplement consulter les tutoriels et la documentation publiés sur jenkinsx.io, et à rejoindre le slack #jenkins-x-user de la communauté Kubernetes.

N'hésitez pas par ailleurs à me contacter si vous voulez partager avec moi votre expérience et vos interrogations au sujet de Jenkins X :)



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

Créez une application mobile multiplateforme de suivi du cours des cryptomonnaies avec Flutter

Lancé sous le nom de code "Sky" en 2015 lors du sommet des développeurs Dart, le framework qui allait devenir Flutter ne fonctionnait alors que sur Android. Le but avoué était la création d'applications mobiles multiplateformes à partir d'une base de code unifiée tout en affichant des performances de rendu graphique dignes du code natif. Quatre ans après, Flutter vient tout juste d'atteindre sa première version stable avec la 1.0 sortie en décembre 2018. L'occasion pour nous de tester Flutter en conditions réelles via la création d'une application mobile multiplateforme de suivi des cours des cryptomonnaies.

niveau 100

Lancé discrètement en 2015 au cours du sommet annuel des développeurs Dart, un langage maison de chez Google, le framework Flutter, a accompli du chemin depuis lors. Sa première version stable officielle, sortie en fin d'année 2018, en fait désormais une solution de choix pour la création d'applications mobiles multiplateformes. Contrairement à des solutions concurrentes comme React Native, Flutter n'utilise pas les composants d'interface graphique natifs des systèmes d'exploitation hôtes. Les ingénieurs de chez Google ont adopté une approche différente avec Flutter.

Flutter implémente lui-même les composants en interne via un système de rendu graphique lui permettant de les dessiner directement sur le système hôte. Tout ceci confère à Flutter des performances de rendu graphique exceptionnelles, et une fluidité comparable aux applications des systèmes natifs. En outre, cela permet également d'alléger le poids des applications mobiles utilisant Flutter ce qui est un argument important en faveur du framework de Google. Pour le développement des applications, Flutter s'appuie donc sur le langage Dart avec une construction de l'interface graphique basée sur une approche déclarative inspirée du framework Web React. Les éléments graphiques sont appelés des Widgets dans la dénomination Flutter. Ils ne sont rendus que lorsque cela est nécessaire à la manière d'un DOM virtuel.

Pour terminer ce rapide tour d'horizon de Flutter, il paraît important de mentionner que les ingénieurs en charge du framework ont eu l'excellente idée d'inclure des fonctionnalités de rechargement à chaud. Si celles-ci sont communes dans le monde du développement Web, cela n'est pas si évident lors du développement d'applications natives. Cela donne donc la possibilité au framework Flutter de reconstruire dynamiquement l'arbre des Widgets ce qui permet aux développeurs de voir rapidement les effets de leurs changements sur le périphérique mobile cible. Un gain de productivité indéniable lors du développement d'applications mobiles.

Installation

La première étape va consister à installer Flutter sur votre machine de développement. Pour ce faire, il faut aller à l'adresse suivante : https://flutter.dev/docs/get-started/install/. Il s'agit alors de vérifier que votre système est assez puissant pour installer Flutter puis de télécharger l'archive ZIP du SDK Flutter. En prenant l'exemple de macOS, vous aurez alors sur votre machine une archive ZIP nommée *flutter_macos_v1.2.1-stable.zip*. Une fois l'archive extraite, il faudra ajouter à votre *PATH* le chemin vers le répertoire *bin* de Flutter. Lancez ensuite un terminal et exécutez la commande suivante :

flutter doctor

L'exécution de cette commande vous permettra de faire le point sur votre installation de Flutter et de vérifier les éventuelles dépendances que vous avez à installer pour la terminer correctement (figure 1).

La commande *flutter doctor* vous donne également les commandes à exécuter pour finaliser l'installation. Il ne vous reste donc plus qu'à vous laisser guider pas à pas et votre installation sera opérationnelle.

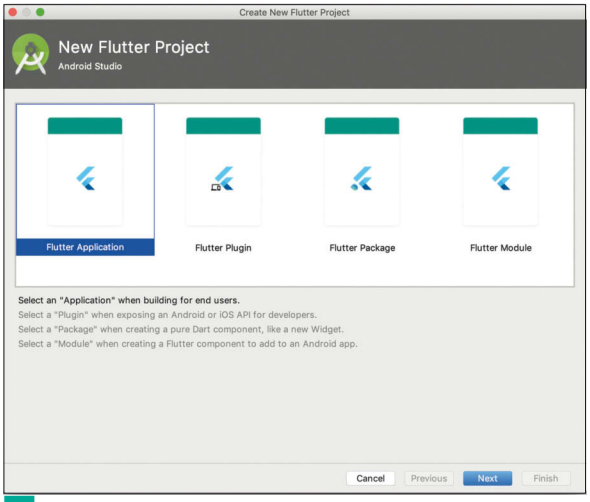
Configuration de l'environnement de travail

Pour développer des applications Flutter, un simple éditeur texte peut vous suffire mais il est beaucoup plus pratique de recourir à un éditeur de code intégré. Des plugins Flutter dédiés pour Android Studio et IntelliJ IDEA sont disponibles. Pour développer notre application, nous allons utiliser Android Studio. Lancez-le et allez ensuite dans *Preferences* > *Plugins* puis sélectionnez *Browse Repositories*. Il ne nous reste plus qu'à sélectionner le plugin Flutter et accepter ensuite l'installation du plugin Dart. Redémarrez Android Studio, et au prochain lancement, vous devriez être en mesure de créer un nouveau projet Flutter (figure 2).

Diagnostic de votre installation Flutter

1

```
ssaurel — -bash — 108×24
MacBook-Pro-de-Sylvain:~ ssaurel$ flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, v1.2.1, on Mac OS X 10.14.3 18D109, locale fr-FR)
[✓] Android toolchain - develop for Android devices (Android SDK version 28.0.3)
[!] iOS toolchain - develop for iOS devices (Xcode 10.2)
    ✗ Verify that all connected devices have been paired with this computer in Xcode.
      If all devices have been paired, libimobiledevice and ideviceinstaller may require updating.
      To update with Brew, run:
        brew update
        brew uninstall --ignore-dependencies libimobiledevice
        brew uninstall --ignore-dependencies usbmuxd
        brew install --HEAD usbmuxd
        brew unlink usbmuxd
        brew link usbmuxd
        brew install --HEAD libimobiledevice
        brew install ideviceinstaller
[✓] Android Studio (version 3.1)
[!] IntelliJ IDEA Community Edition (version 2016.3)
    ✗ Flutter plugin version 0.1.5 - the recommended minimum version is 16.0.0
    ✗ This install is older than the minimum recommended version of 2017.1.0.
[✓] Connected device (1 available)

! Doctor found issues in 2 categories.
MacBook-Pro-de-Sylvain:~ ssaurel$
```



2 *Création d'un projet Flutter sous Android Studio*

Création d'un premier projet Flutter

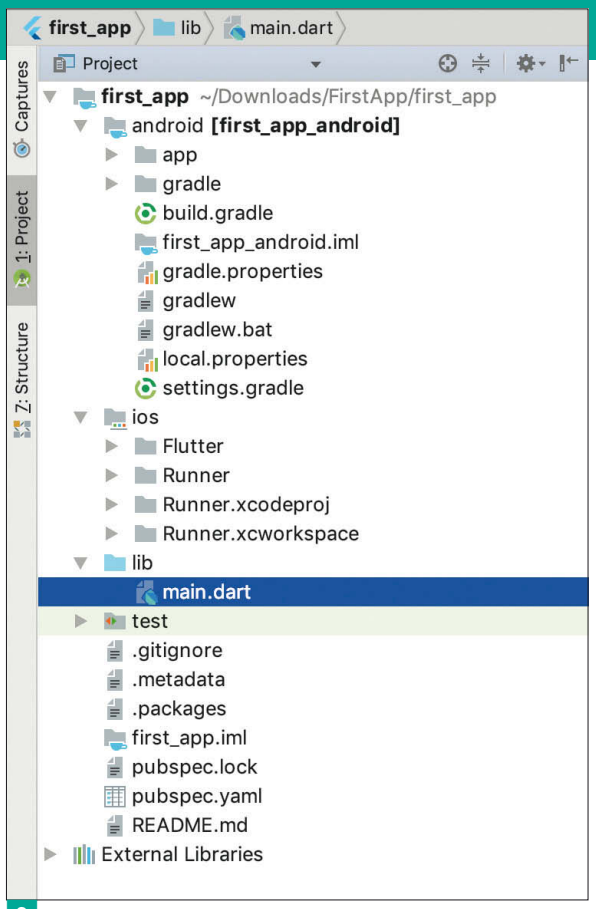
Notre premier projet d'application Flutter va consister à afficher un compteur à l'écran avec un bouton sur lequel il sera possible de cliquer. Enfin, notre application devra garder l'état courant du compteur pour qu'après chaque clic, on puisse afficher le nombre de clics réalisés par l'utilisateur. Une fois le projet créé avec Android Studio, on a l'arborescence présentée à la figure 3.

Au sein de cette arborescence, on remarque la présence de répertoires *android* et *ios*. Ce sont les points d'entrée respectifs pour la construction de l'application à destination des plateformes Android et iOS. Il est bon de noter que les éventuels codes spécifiques à l'une ou l'autre de ces plateformes pourront être ajoutés dans ces répertoires. Le but étant néanmoins de limiter ce type de code qui rendra la maintenance future d'une application plus compliquée. Le dossier *lib* contient quant à lui la base de code unifiée du projet écrite en langage Dart. Enfin, le fichier *pubspec.yaml* contient les dépendances spécifiques au projet. Il sera nécessaire de mettre à jour ce fichier lorsque vous souhaiterez utiliser des bibliothèques spécifiques de Dart pour le développement d'une application Flutter.

Première application Flutter

Notre première application n'ayant pas de code spécifique à la plateforme Android ou à la plateforme iOS, nous allons travailler seulement au sein du fichier *main.dart*. Au sein d'une application Flutter, le point d'entrée est une méthode *main()* dans laquelle on va appeler la fonction standard *runapp()* en lui passant en entrée la classe représentant l'application :

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

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

class MyApp extends StatelessWidget {
 // Ce Widget est la racine de l'application
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   title: 'Flutter on Programmez!',
```



3 *Arborescence d'un projet Flutter*

```
   theme: ThemeData(
    primarySwatch: Colors.green,
   ),
   home: MyHomePage(title: 'Flutter on Programmez!'),
  );
 }
}
```

La classe *MyApp* est un Widget sans état ce qui implique qu'il doit étendre la classe standard *StatelessWidget*. Il s'agit de la racine de notre interface graphique et comme nous choisissons d'avoir une application au look Material Design de Google, nous renvoyons un objet *MaterialApp* sur lequel nous avons appliqué une couleur et un titre. Vous remarquez également la propriété *home* qui correspond au coeur de notre interface graphique et qui va être défini avec une classe *MyHomePage* spécifique.

Le Widget *MyHomePage* aura un état puisqu'il devra conserver le nombre de clics réalisés sur le bouton de notre application. Il va donc étendre la classe *StatefulWidget*. Un Widget graphique à état doit retourner un objet étendant la classe *State*. Pour cela, nous allons devoir créer une classe *_MyHomePageState* qui aura comme propriété un entier *_counter* initialisé à zéro. Nous définissons la méthode *_incrementCounter()* qui signifiera au framework Flutter le changement d'état en appelant la méthode *setState()* au sein de laquelle on place l'instruction d'incrément de notre compteur. Le fait d'avoir appelé *setState()* va permettre un rappel automatique à la méthode *build()* de notre classe et donc un rafraîchissement de l'interface graphique.

Il nous reste à définir le contenu de la méthode *build()* de notre

classe *_MyHomePageState* qui renverra le contenu de l'interface graphique de notre application. Nous utilisons la classe prédéfinie *Scaffold* afin de nous faciliter la tâche. Nous définissons une barre d'application pour cet objet en instanciant un objet *AppBar* puis passons au corps de l'interface graphique défini au sein de la propriété *body* de notre instance de *Scaffold*. Nous utilisons un Widget *Center* pour centrer notre texte à l'écran. Dans ce Widget, nous ajoutons deux Widgets de type *Text* alignés verticalement et dont le deuxième sera utilisé pour afficher dynamiquement le nombre de clics réalisés par l'utilisateur. Enfin, il ne nous reste plus qu'à définir le bouton d'action flottant attendu par l'objet *Scaffold* via sa propriété *floatingActionButton*. Nous définissons notre méthode *_incrementCounter()* comme handler de clic sur le bouton.
Tout ceci nous donne le code suivant :

```
class MyHomePage extends StatefulWidget {
 MyHomePage({Key key, this.title}) : super(key: key);

 final String title;

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

class _MyHomePageState extends State<MyHomePage> {
 int _counter = 0;

 void _incrementCounter() {
  setState(() {
   _counter++;
  });
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text(widget.title),
   ),
   body: Center(
    child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
     children: <Widget>[
      Text(
       'Number of click on the Button:',
      ),
      Text(
       '$_counter',
       style: Theme.of(context).textTheme.display1,
      ),
     ],
    ),
   ),
   floatingActionButton: FloatingActionButton(
    onPressed: _incrementCounter,
    tooltip: 'Increment',
    child: Icon(Icons.add),
   ),
  );
 }
}
```

Pour tester l'application, il nous suffit ensuite de cliquer sur le bouton run d'Android Studio en ciblant le simulateur iOS puis ensuite un périphérique virtuel Android. On a alors le rendu de notre application sur iOS et Android présenté à la figure 4.

Application multiplateforme de suivi du cours des cryptos

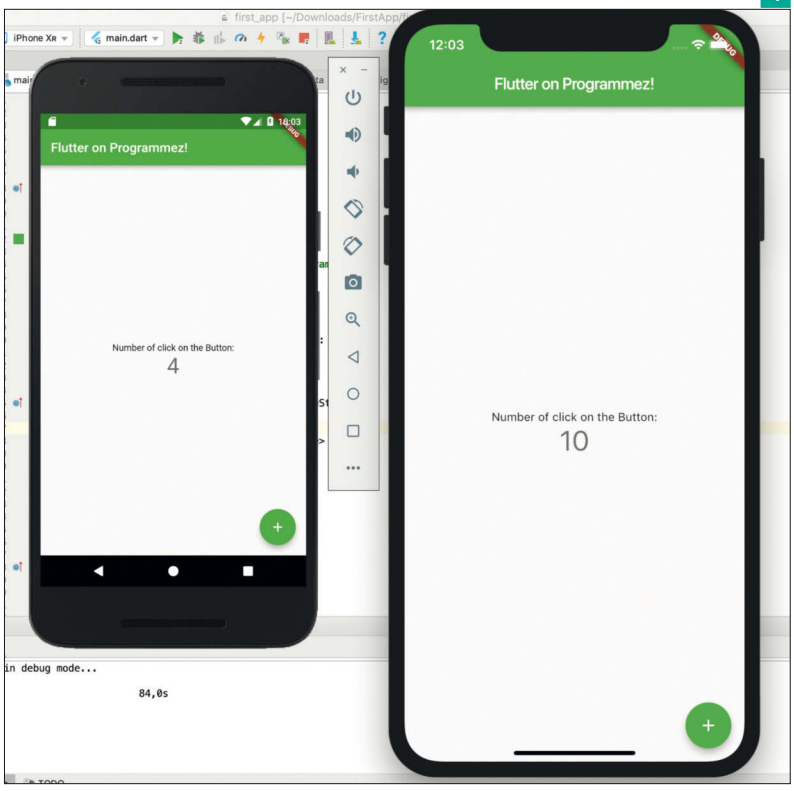
Maintenant que nous avons pu réaliser une première application Flutter avec succès, il est temps de passer aux choses sérieuses en développant une application mobile multiplateforme de suivi du cours des cryptomonnaies. Pour ce faire, nous allons utiliser l'API du site de référence CoinMarketCap qui permet d'obtenir les cours en temps réel des principales cryptomonnaies. Ce sera également l'occasion de découvrir comment consommer des Web Services au format JSON avec Flutter.

Appel d'un Web Service JSON avec Flutter

Nous commençons donc par appeler l'API CoinMarketCap dans un navigateur afin de localiser les informations qui nous intéressent. Le résultat de l'appel au Web Service https://api.coinmarketcap.com/v1/ticker/ est affiché à la figure 5.
Notre application va afficher la liste des 50 principales cryptomonnaies avec leur cours donné par le champ *price_usd*, leur nom donné par le champ *name* ainsi que le pourcentage d'évolution sur une heure donné par le champ *percent_change_1h*.

Notre première application sur Android et iOS

4



Afin de pouvoir appeler le Web Service via une requête HTTP, nous devons rajouter certaines dépendances à notre projet d'application Flutter. Nous ajoutons ainsi la dépendance *http* au fichier *pubpec.yaml* :

```
dependencies:
 flutter:
  sdk: flutter

 cupertino_icons: ^0.1.2
 http: any
```

Il ne reste plus qu'à faire les imports de bibliothèques de code nécessaires au début de notre fichier main.dart :

```
import 'dart:async';
import 'dart:convert';
import 'dart:math';
import 'package:http/http.dart' as http;
import 'package:flutter/material.dart';
```

Le package *dart:async* va nous permettre d'accéder à la classe *Future* alors que les packages *dart:convert* et *dart:math* vont respectivement nous donner la possibilité de décoder la réponse au format JSON et d'utiliser la classe *Random* pour associer des couleurs de manière aléatoire à chaque cryptomonnaie de la liste.
Nous pouvons ainsi faire une requête à notre Web Service en écrivant la fonction getPrices() suivante :

```
Future<List> getPrices() async {
 String apiUrl = 'https://api.coinmarketcap.com/v1/ticker/?limit=50';
 http.Response response = await http.get(apiUrl);
 return json.decode(response.body);
}
```

La fonction indique qu'elle va retourner un objet de type *Future<List>* à un certain moment dans le futur. Notre fonction doit donc être marquée comme asynchrone avec le mot clé *async*. L'appel à la fonction *get* de la classe *HTTP* est asynchrone et va immédiatement retourner un objet *Future<Response>*. Comme nous souhaitons avoir un comportement quasiment synchrone ici et patienter jusqu'au retour effectif de cet appel au Web Service, nous utilisons le mot clé *await* juste avant l'appel. Enfin, on utilise la méthode *decode* de la classe *JsonCodec* pour transformer la réponse en objet *Future<List>*.

Création de l'Interface Graphique

Maintenant que nous sommes en mesure de récupérer les cours des principales cryptomonnaies, il nous reste à les afficher au sein d'une liste. Pour ce faire, nous allons encore une fois appeler la fonction *runApp()*, en lui passant en entrée un objet de type *MaterialApp* au sein duquel nous définissons un Widget *CryptoListWidget* comme propriété home. Ce dernier n'ayant pas besoin de conserver un état, il va étendre la classe *StatelessWidget*.
Le Widget *CryptoListWidget* va contenir le coeur de notre application. Il va prendre en entrée une liste de cryptomonnaies qu'il se chargera d'afficher. Pour donner un petit côté amusant à notre Interface Graphique, nous allons créer une liste contenant des couleurs issues du look and feel Material Design de Google et nous les

https://api.coinmarketcap.com/v1/ticker/?limit=50


```
[
    {
        "id": "bitcoin",
        "name": "Bitcoin",
        "symbol": "BTC",
        "rank": "1",
        "price_usd": "5044.82729675",
        "price_btc": "1.0",
        "24h_volume_usd": "15324518931.5",
        "market_cap_usd": "88945536728.0",
        "available_supply": "17631037.0",
        "total_supply": "17631037.0",
        "max_supply": "21000000.0",
        "percent_change_1h": "0.36",
        "percent_change_24h": "0.75",
        "percent_change_7d": "23.17",
        "last_updated": "1554563067"
    },
    {
        "id": "ethereum",
        "name": "Ethereum",
        "symbol": "ETH",
        "rank": "2",
        "price_usd": "165.410729722",
        "price_btc": "0.03270811",
        "24h_volume_usd": "6534139312.15",
        "market_cap_usd": "17458613578.0",
        "available_supply": "105547044.0",
        "total_supply": "105547044.0",
        "max_supply": null,
        "percent_change_1h": "0.66",
        "percent_change_24h": "0.75",
        "percent_change_7d": "16.54",
        "last_updated": "1554563058"
    },
    {
        "id": "ripple",
        "name": "XRP",
```


5 *Retour de l'API CoinMarketCap*

utiliserons de manière aléatoire pour chaque cryptomonnaie de la liste. La construction du Widget se passe comme de coutume au sein de la méthode *build()* en sortie de laquelle nous renvoyons un objet *Scaffold*. Dans ce dernier, nous définissons la couleur de fond de notre Widget ainsi qu'un bouton d'action flottant qui pourrait être utilisé dans le futur pour proposer une fonctionnalité de rafraîchissement aux utilisateurs. La construction du corps de l'objet *Scaffold* est déportée au sein d'une méthode *_buildBody()*. Nous utilisons alors un objet *Container* nous permettant de définir des marges ainsi qu'un alignement vertical pour le Widget titre, qui sera de type *Text*, et le Widget liste, qui sera de type *ListView*.
Tout ceci nous donne le début de code suivant pour la classe *CryptoListWidget* :

```
class CryptoListWidget extends StatelessWidget {
 final Random _random = new Random();
 final List<MaterialColor> _colors = [Colors.green, Colors.blue, Colors.red,
     Colors.orange, Colors.brown, Colors.yellow, Colors.cyan, Colors.purple];
 final List _prices;

 CryptoListWidget(this._prices);

 @override
 Widget build(BuildContext context) {
  return new Scaffold(
   body: _buildBody(),
   backgroundColor: Color(0xFF43a047),
   floatingActionButton: new FloatingActionButton(onPressed: () {
     // Amélioration à faire ...
   }, child: new Icon(Icons.refresh),
     backgroundColor: Colors.lightGreen
   ),
  );
 }

 Widget _buildBody() {
  return new Container(
   margin: const EdgeInsets.fromLTRB(8.0, 56.0, 8.0, 0.0),
```

```
    child: new Column(
      children: <Widget>[_getAppTitleWidget(), _getListViewWidget()],
    ),
  );
}

Widget _getAppTitleWidget() {
  return new Text(
    'Crypto Prices',
    style: new TextStyle(
        color: Colors.white, fontWeight: FontWeight.bold, fontSize: 24.0),
  );
}

// ... Manque la liste ...
}
```

Rendu de la liste des cryptomonnaies

A la lecture du code de la classe *CryptoListWidget* présenté précédemment, vous aurez sans nul doute remarqué qu'il manque la définition de la méthode *_getListViewWidget()* en charge du rendu graphique de la liste des cryptomonnaies. Afin d'avoir un affichage vertical prenant la totalité de l'espace disponible à l'écran, nous allons encapsuler notre *ListView* au sein d'un objet *Flexible*. Ensuite, nous configurons notre *ListView* en définissant le nombre d'éléments et le code du callback associé à la propriété *itemBuilder* de cette *ListView*. Pour chaque élément, on va ainsi récupérer l'élément correspondant dans la liste des cryptomonnaies, lui associer une couleur aléatoirement puis retourner le résultat de l'appel à la méthode *_getListItemWidget()*.

La construction de chaque item de notre *ListView* se fait donc au sein de la méthode *_getListItemWidget()*. Celle-ci retourne un *Container* pour lequel nous définissons une marge en haut et un objet *Card* comme contenu. Cet objet *Card* permettant d'obtenir un rendu similaire à l'objet *CardView* bien connu des développeurs Android. Pour notre application, il n'y aura qu'un seul enfant à cet objet à savoir une instance de la classe *ListTile*. Ce Widget donne la possibilité d'afficher un premier élément à gauche via la propriété *leading*. Nous allons en tirer avantage pour afficher la première lettre du nom de la cryptomonnaie concernée au sein d'un cercle dont la couleur de fond sera celle passée en entrée de la méthode *_getListItemWidget()*.

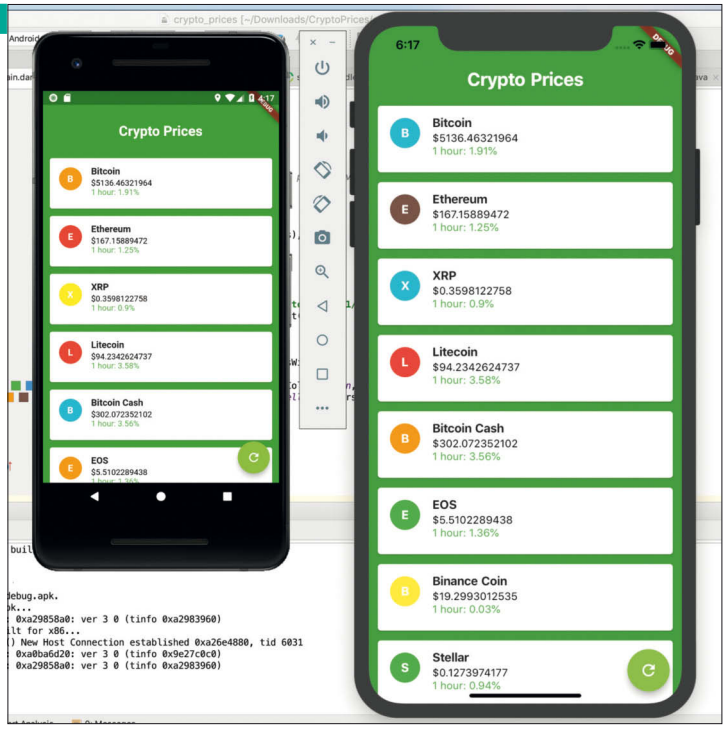
La suite consiste à utiliser la propriété *title* du Widget *ListTile* pour afficher le nom de la cryptomonnaie avant de recourir à sa propriété *subtitle* pour renvoyer le cours de la crypto monnaie en dollars ainsi que son pourcentage d'évolution sur une heure. Pour rendre notre Interface Graphique plus attractive, on applique une couleur verte si ce pourcentage correspond à une hausse et une couleur rouge s'il correspond à une baisse.

On obtient alors le code complet suivant pour notre fichier *main.dart* :

```
import 'dart:async';
import 'dart:convert';
import 'dart:math';
import 'package:http/http.dart' as http;
import 'package:flutter/material.dart';

void main() async {
  // Amélioration à faire : utiliser une progress view :)
  List prices = await getPrices();

  runApp(new MaterialApp(
    home: new Center(
      child: new CryptoListWidget(prices),
    ),
  ));
}

Future<List> getPrices() async {
  String apiUrl = 'https://api.coinmarketcap.com/v1/ticker/?limit=50';
  http.Response response = await http.get(apiUrl);
  return json.decode(response.body);
}

class CryptoListWidget extends StatelessWidget {
  final Random _random = new Random();
  final List<MaterialColor> _colors = [Colors.green, Colors.blue, Colors.red,
      Colors.orange, Colors.brown, Colors.yellow, Colors.cyan, Colors.purple];
  final List _prices;

  CryptoListWidget(this._prices);

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      body: _buildBody(),
      backgroundColor: Color(0xFF43a047),
      floatingActionButton: new FloatingActionButton(onPressed: () {
        // Do something when FAB is pressed
      }, child: new Icon(Icons.refresh),
        backgroundColor: Colors.lightGreen
      ),
    );
  }

  Widget _buildBody() {
    return new Container(
      margin: const EdgeInsets.fromLTRB(8.0, 56.0, 8.0, 0.0),
      child: new Column(
        children: <Widget>[_getAppTitleWidget(), _getListViewWidget()],
      ),
    );
  }

  Widget _getAppTitleWidget() {
    return new Text(
      'Crypto Prices',
      style: new TextStyle(
          color: Colors.white, fontWeight: FontWeight.bold, fontSize: 24.0),
    );
  }
```

```
Widget _getListViewWidget() {
 return new Flexible(
   child: new ListView.builder(
     itemCount: _prices.length,
     itemBuilder: (context, index) {
      final Map price = _prices[index];
      final MaterialColor color = _colors[_random.nextInt(_colors.length)];
      return _getListItemWidget(price, color);
     }));
}

CircleAvatar _getLeadingWidget(String currencyName, MaterialColor color) {
 return new CircleAvatar(
  backgroundColor: color,
  child: new Text(currencyName[0],
    style: new TextStyle(color: Colors.white, fontWeight: FontWeight.bold)),
 );
}

Text _getTitleWidget(String currencyName) {
 return new Text(
  currencyName,
  style: new TextStyle(fontWeight: FontWeight.bold),
 );
}

RichText _getSubtitleText(String priceUsd, String percentChange1h) {
 TextSpan priceTextWidget = new TextSpan(text: "\$$priceUsd\n", style:
 new TextStyle(color: Colors.black),);
 String percentChangeText = "1 hour: $percentChange1h%";
 TextSpan percentChangeTextWidget;
 if(double.parse(percentChange1h) > 0) {
  percentChangeTextWidget = new TextSpan(text: percentChangeText,
   style: new TextStyle(color: Colors.green),);
 }
 else {
  percentChangeTextWidget = new TextSpan(text: percentChangeText,
   style: new TextStyle(color: Colors.red),);
 }

 return new RichText(text: new TextSpan(
   children: [
    priceTextWidget,
    percentChangeTextWidget
   ]
 ),);
}

ListTile _getListTile(Map currency, MaterialColor color) {
 return new ListTile(
  leading: _getLeadingWidget(currency['name'], color),
  title: _getTitleWidget(currency['name']),
  subtitle: _getSubtitleText(
    currency['price_usd'], currency['percent_change_1h']),
  isThreeLine: true,
 );
}

Container _getListItemWidget(Map currency, MaterialColor color) {
 return new Container(
  margin: const EdgeInsets.only(top: 5.0),
  child: new Card(
   child: _getListTile(currency, color),
  ),
 );
}
}
```

6 *Application mobile multiplateforme du suivi du cours des cryptomonnaies*



Test de notre application

Le développement de notre application mobile multiplateforme de suivi du cours des cryptomonnaies terminé, il est temps de la lancer sur Android et sur iOS afin de valider le travail réalisé (figure 6).

Conclusion

Cet article aura mis en exergue la facilité avec laquelle Flutter permet de créer des applications mobiles multiplateformes. Le travail réalisé par Google ces dernières années a été considérable et aura permis d'amener Flutter à un niveau de performance qui n'a rien à envier aux applications natives sur Android et iOS. Preuve de l'engagement de Google pour Flutter, il se murmure même que le framework de Mountain View sera la méthode principale mise à disposition des développeurs pour créer des applications pour le futur système d'exploitation Google Fuchsia. Les développeurs mobiles ont donc tout intérêt à s'y intéresser de près et prendre véritablement en considération Flutter pour le développement de leurs futures applications mobiles. •

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

niveau 200

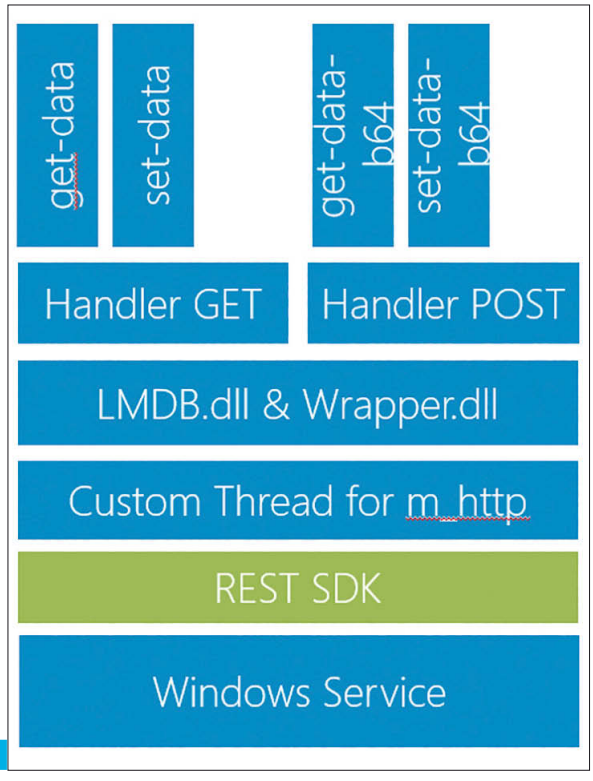
C++ et Containers Docker sur Azure

Pour un Gold Partner ayant des bureaux à Paris, j'ai réalisé une vitrine technologique Azure mettant en œuvre les technologies Microsoft .NET, C#, C++, LMDB, Win32 et Docker, puis la déclinaison du run d'une instance Docker sous Azure.

L'histoire commence avec la récupération de sources C++ sous brevet, le code d'une base de données non terminée… Ce projet était intitulé la base de données la plus rapide du monde. Il était question de moteur de cube OLAP et de reporting BI pour concurrencer SQL Server SSAS et Tableau Software… Après investigation dans le source code pendant un mois, je constate que c'est un beau projet mais qu'il n'est pas terminé et que cela va être compliqué de réaliser la partie manquante car le code est assez complexe et surtout dans quelles conditions et pour quel résultat ? En accord avec mon patron, on décide de rechercher des composants réutilisables. Le moteur de stockage attire mon attention. Or il se trouve que c'est une librairie open-source nommée OpenLDAP-LMDB qui existe sous Linux ; c'est un filesystem en mémoire… LMDB a été couvert dans le numéro de Mars 2019 de Programmez. Cherchez le paquet LMDBNet sur nuget : https://www.nuget.org/packages/LMDBNet

Architecture Logicielle 1

Voici le soft qui va être construit ; c'est un service Windows qui embarque un serveur web autonome sur lequel on a greffé un Web Service REST API JSON, le tout construit en C++ avec le Microsoft C++ REST SDK. Le Web service se publie sur le port 7001. Il faut être administrateur pour pouvoir créer le port. C'est une restriction WinInet.



1

Il s'agit d'un serveur web embarqué qui expose un web service REST API qui permet de gérer un ou plusieurs cache NoSQL LMDB. Le projet est un Windows Service x64 (LMDBService.exe) qui charge plusieurs dll :

- cpprest141d_2_10.dll
- LMDBWindowsDllD64.dll
- LMDBWrapperD64.dll
- MySharedStuffD64.dll
- Msvcp140d.dll
- Vcruntime140d.dll
- Ucrtbased.dll

Vous aurez remarqué au passage que les modules sont en debug… C'est pour faciliter le debugging.

Run en local

Le service Windows se lance via net start ou en mode console… Il consomme 2 MB de mémoire et le CPU est toujours à 0 %. C'est l'avantage du C++.

Console – lancement du démon

On teste le serveur web embarqué depuis Chrome.

Mise en œuvre de Docker

Avant tout, il faut installer Docker sous Windows. Ce n'est pas très compliqué. On lance le setup et c'est terminé. Le premier problème est de savoir comment créer une « application » pour Docker. Et là, c'est un moment de solitude car la documentation Docker est très mal faite, c'est une honte. Heureusement, je mets la main sur un ouvrage intitulé « Docker on Windows ». Une chance. Le livre explique que Docker est un système de virtualisation dans lequel une image vierge du système d'exploitation se voit greffer des composants via un fichier de boot (le dockerfile) et peut les exécuter comme une VM et dès que le container s'arrête, on perd tout. On peut recommencer à l'infini…. De manière isolée.

La base : le DockerFile

Un DockerFile est un fichier de boot pour l'image Docker. On va lui indiquer la base de l'operating system sur laquelle on boot et les différents paramétrages applicatifs à y ajouter. Avant de monter une instance de container Docker sous Azure, il faut d'abord maîtriser son fonctionnement en local avec un Docker en local, par exemple sous Windows 10. Voici mon dockerfile qui utilise une image Win-

dows Sever 2016 LTSC (Long time support). Le DockerFile est le suivant :

```
FROM microsoft/iis:windowsservercore-ltsc2016
#RUN netsh advfirewall set allprofiles state off
COPY *.* c:/
RUN sc sdset SCMANAGER D:(A;;CCLCRPRC;;;AU)(A;;CCLCRPWPRC;;;SY)(A;;KA;;;BA)S:(AU;
FA;KA;;;WD)(AU;OIIOFA;GA;;;WD)
RUN sc create LMDBService start=auto binpath="C:\LMDBService.exe"
#EXPOSE 80
EXPOSE 7001
RUN md c:\temp
```

On part d'une image d'OS fournie par Microsoft qui contient IIS…
Les étapes sont les suivantes :

- Copie des binaires sur l'image ;
- Mise à jour des droits admins sur le SCManager ;
- Création du service ;
- Ouverture des ports 80 et 7001.

Run en local sous Docker

Ensuite, il faut builder l'image Docker :

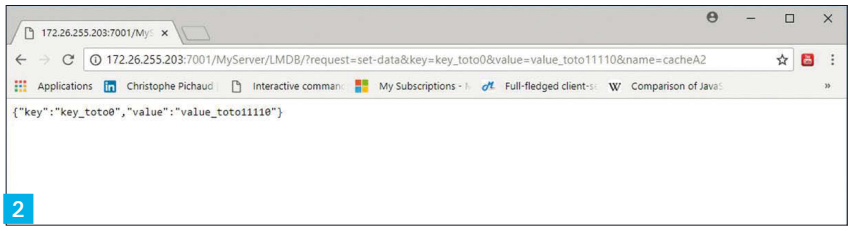
```
docker image build –tag mydocker/myserver d:\dev\docker
```

Ensuite, on peut lancer le container en démon en ouvrant le port 7001 créé par le service :

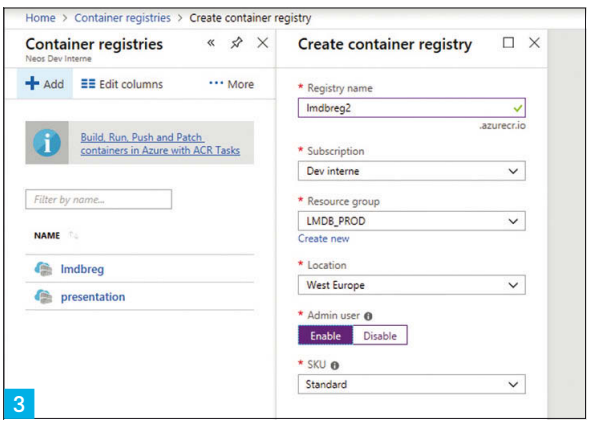
```
docker run -d -p 7001:7001 -it mydocker/myserver
```

Pour pouvoir tester le container, il nous faut son adresse IP avec son ID en utilisant :

```
docker ps –a
```



2



3

On prend le premier item de la liste. On lance la ligne suivante avec son id à la fin :

```
docker inspect -f "{{ .NetworkSettings.Networks.nat.IPAddress }}" 543e57f54047
```

La réponse s'affiche => 172.26.255.203
Il est maintenant possible de tester le container. 2

Pour arrêter le container, il faut faire :

```
docker -stop 543e57f54047
```

Comme vous pouvez le constater, Docker c'est quasiment indolore et transparent. Docker permet de faire fonctionner tout ce qui est non graphique. Les services de communications, les daemons, les services Windows sont des candidats idéaux pour fonctionner sous Docker. Exemple : faire tourner un SQL Server de développement. C'est un service Windows qui utilise le port 1433. IIS qui est un site web qui utilise le port 80.

Run dans Azure

Là, les choses se compliquent un peu… D'abord, pour utiliser Azure, il faut une souscription. Ensuite il faut créer un registry. En effet, un container ne peut être instancié que s'il existe un registry ou est déposée l'image Docker.
Première étape, il faut créer un repository dans Azure Container Registry. Une fois créée, cette ressource donne un user/pwd. Sélectionnez l'item Access Keys et récupérez le mot de passe dans le clipboard : Connectez-vous à Azure Container Registry via son user/pwd depuis la console docker:

```
docker login lmdbreg2.azurecr.io -u lmdbreg2 -p V5nUpEG7Hu/V4KZ8kGy6hBT4r70tleFI
```

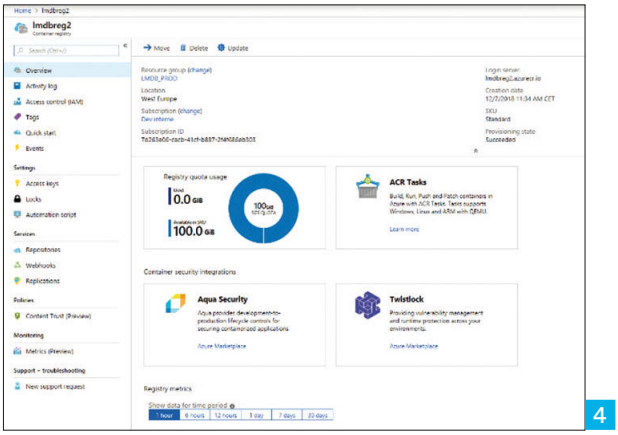
Ensuite, il faut tagguer le registry pour l'image Docker :

```
docker tag myserver lmdbreg2.azurecr.io/lmdb
```

Ensuite, il est possible d'uploader l'image docker vers ACR.

```
docker push lmdbws.azurecr.io/lmdb
```

La dernière commande envoie l'image dans ACR.



4

Créer une instance du container

Sélectionnez Container Instances et positionnez les informations de registry et les credentials. Cliquez sur OK. Selectionnez Open additional port et tapez 7001. Cliquez sur OK.
Cliquez sur OK. Le container est déployé sur Azure ACI en 3 ou 4 minutes. Une fois déployé, on va sur le portail pour y récupérer son adresse IP en haut à droite. 3 4 5 6 7 8

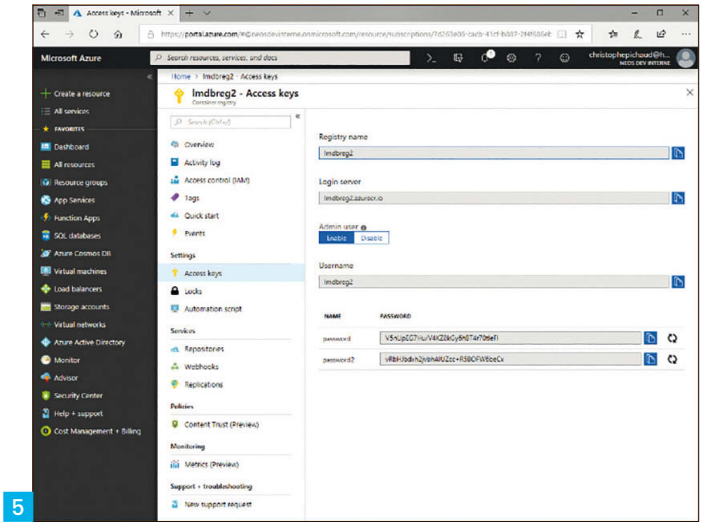
Mon service Windows contient un ping interne qui retourne son état. Faisons un test de ce ping logiciel :
http://137.117.141.0:7001/MyServer/LMDB/?request=ping 9

Faisons aussi un About logiciel :
http://137.117.141.0:7001/MyServer/LMDB/?request=about 10

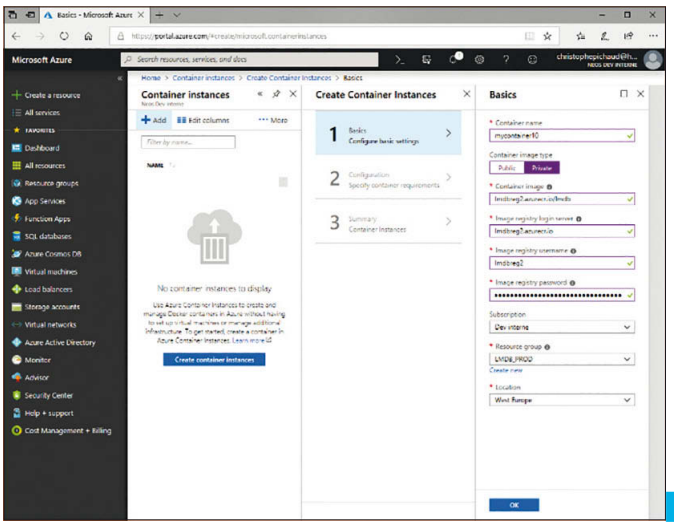
Dans une version évoluée du service Windows, je crée un fichier de logs dans c:\temp\logs. Je définis mon dockerfile avec une image qui contient IIS. Puis je crée un vdir sur c:\temp\logs.
Ainsi, avec le port 80 depuis mon browser, je peux lire mes logs :
http://137.117.141.0/Logs/lmdb.txt

Problème réseau

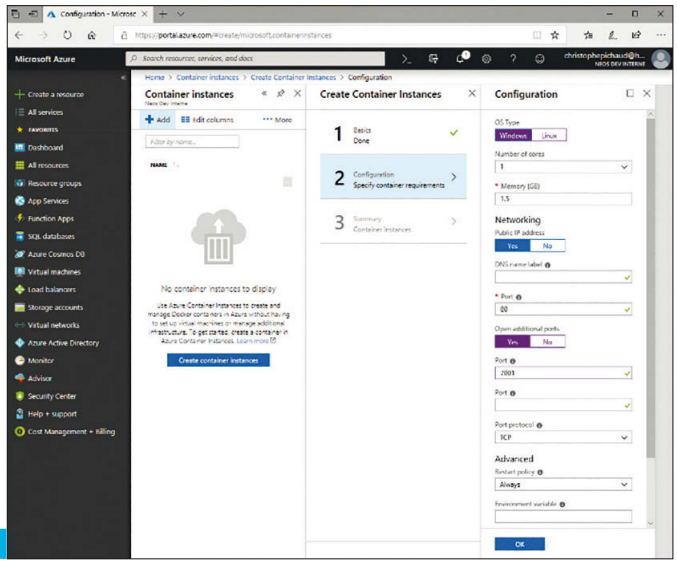
Lors de mes premiers tests sur Azure, j'ai rencontré des difficultés à établir la connexion entre le browser et mon service Web. Rien ne marchait. J'ai trouvé une information sur le Web qui disait d'énumérer les NICs (les cartes réseaux). Et en effet, il y a plusieurs possibilités. Soit créer les services sur l'ensemble des cartes réseaux, soit sur… la dernière ! Voici le code qui itère sur les cartes et garde en mémoire la dernière adresse IP :



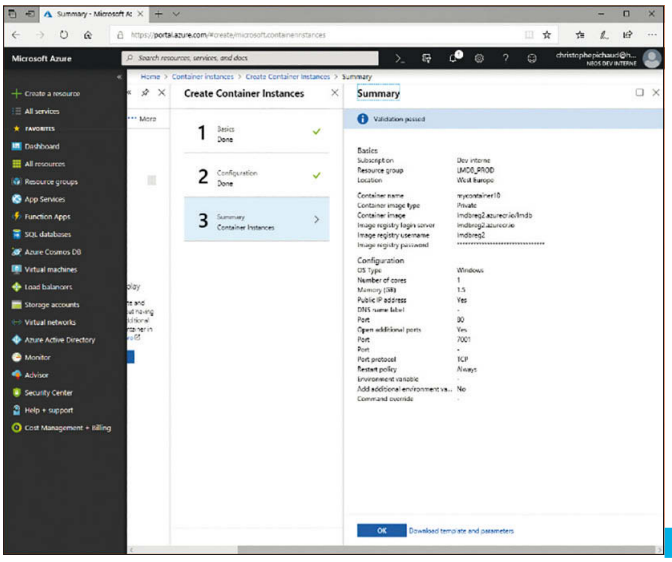
5



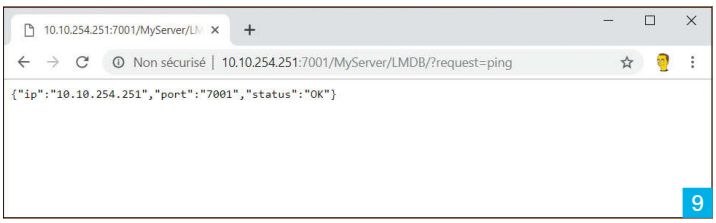
6



7



8



9

10

```
std::wstring ServerHelper::GetIP()
{
    //return L"127.0.0.1";
    //return L"0.0.0.0";

    // Init WinSock
    WSADATA wsa_Data;
    int wsa_ReturnCode = WSAStartup(MAKEWORD(2, 2), &wsa_Data);

    // Get the local hostname
    char szHostName[255];
    gethostname(szHostName, 255);

    struct hostent *remoteHost;
    struct in_addr addr;
    std::string ip;

    remoteHost = gethostbyname(szHostName);

    if (remoteHost->h_addrtype == AF_INET)
    {
        int i = 0;
        while (remoteHost->h_addr_list[i] != 0)
        {
            addr.s_addr = *(u_long *)remoteHost->h_addr_list[i++];
            printf("\tIPv4 Address #%d: %s\n", i, inet_ntoa(addr));
            ip = inet_ntoa(addr);
        }
    }

    WSACleanup();
    std::wstring ipw(ip.begin(), ip.end());
    return ipw;
}
```

Cette routine est utilisée dans la procédure d'initialisation du service qui crée le serveur web et le service REST sur le port 7001 :

```
DWORD AutomateThread(LPVOID pParam)
{
    try
    {
        g_Logger.WriteLog(_T("Running in console mode... Entering while()..."));
        std::wstring port = Constants::MasterNodePort;
        std::wstring host = ServerHelper::GetHostName();
        std::wstring ip = ServerHelper::GetIP();
        std::wstring url = ServerHelper::BuildURL(ip, port);
        http::uri uri = http::uri(url);
        std::wstring address = uri.to_string();
        std::wstring name = _T("Master");

        std::vector<std::wstring> v = ServerHelper::GetIPs();
        for (std::wstring ipE : v)
        {
            g_Logger.WriteLog(ipE);
        }
        g_Logger.WriteLog(_T("Enum IP addresses ok"));

        //
        // Create the server instance
        //

        TCHAR sz[255];
        _stprintf_s(sz, _T("IP : %s"), ip.c_str());
        g_Logger.WriteLog(sz);

        g_Logger.WriteLog(_T("Creating server ip/port..."));
        TheServer client(address);
        g_Logger.WriteLog(_T("Creating server ip/port ok"));
        client._server = ip;
        client._port = port;
        client._name = name;
        client.Init();

        g_Logger.WriteLog(_T("Waiting..."));
        while (TRUE)
        {
            if (_Module.m_bStop)
            {
                std::wcout << _T("Running in console mode.") << std::endl;
                goto stop_service;
                break;
            }

            Sleep(5000);
            g_Logger.WriteLog(_T("MainThread Sleep..."));

        } // Main loop
```

Observations sur Docker

L'avantage de faire du containeur avec du code C++ est que le code étant rapide. On met rarement le CPU en chauffe, la consommation de RAM est minimum, donc on ne paie presque rien. Les machines qui font tourner les containers Docker sont des monstres et l'exemple de mon code C++ fait que je suis à 0% de RAM et 0% de CPU…. Bref, pour faire tourner du bon code C++ de communication avec différentes versions de librairies, c'est une bonne option.

Conclusion

L'utilisation d'une application dans Docker n'est pas très compliquée à mettre en place. Il faut d'abord, créer un DockerFile avec une image d'OS et y copier ses binaires. C'est tout ! Ensuite, on ouvre les ports de communication et hop, le tour est joué.

•

Denis Duplan
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.frauteur

La promesse des promises en JavaScript

Partie 2

L'implémentation de la spécification ECMAScript 2015 sur laquelle JavaScript est fondé a conduit à introduire un nouvel objet standard en JavaScript : Promise. Cet objet permet de faciliter et de sécuriser l'écriture d'un appel à une fonction asynchrone. Son apparition souligne l'importance prise par l'asynchronisme en JavaScript.

niveau 200

Chaîner les promises

Retour à l'API Promise. Un autre point essentiel est que `.then ()` et `.catch ()` retournent une promise. Cela peut surprendre, car ces méthodes prennent pour paramètre(s) un fulfiller et/ou un rejecter dont la signature n'est pas du tout celle d'un executor. Pour rappel :

- un executor prend pour paramètres des fonctions `resolve ()` et `reject ()` ;
- un fulfiller en paramètre une valeur communiquée à `resolve ()` par l'executor ;
- un rejecter prend en paramètre une valeur communiquée à `reject ()` par l'executor.

D'où la question : à partir de quel executor `.then ()` et `.catch ()` créent-elles une promise ?

Pour le comprendre, il faut repartir du comportement attendu de `.then ()` . Comme déjà expliqué, le système des promises doit permettre d'échapper au « callback hell ». Cela signifie que ce système doit offrir une syntaxe simple pour spécifier une callback lors de l'appel à une fonction asynchrone – ce que `.then ()` permet, comme on vient de le voir. Toutefois, cela signifie aussi que ce système doit offrir une syntaxe simple pour spécifier la callback de cette callback si jamais la première callback est aussi une fonction asynchrone, et ainsi de suite. Pour cela, ne serait-il pas intéressant de permettre une telle écriture ? :

```
asyncGetSomeData (10).then (asyncComputeSomething).then (syncDisplaySomeResult);
```

De fait, c'est exactement ce que le système des promises permet, comme l'a montré l'exemple déjà donné en évoquant la promesse des promises, au début de cet article. Il est possible d'enchaîner des appels à des fonctions asynchrones – ou synchrones car qui peut le plus, peut le moins ; or une fonction synchrone, n'est-ce pas une fonction asynchrone qui retourne son résultat immédiatement ? Ce mécanisme est plus rigoureusement appelé composition, car le chaînage peut se lire formellement à l'envers :

```
syncDisplaySomeResult (asyncComputeSomething (asyncGetSomeData (10)))
```

Partant, il est attendu que `.then ()` se comporte de différentes manières selon ce que retourne le fulfiller qui lui est passé en paramètre :

- Si un premier fulfiller ne retourne rien, `.then ()` permet simplement d'appeler un second fulfiller après :

```
function f (resolve, reject) {
    window.setTimeout (() => resolve (666), 2000);
}
var p = new Promise (f);
p.then (result => {}).then (result => console.log (result));
undefined
```

- Si un premier fulfiller retourne une valeur, `.then ()` permet de la relayer en paramètre à un second fulfiller appelé après :

```
function f (resolve, reject) {
    window.setTimeout (() => resolve (666), 2000);
}
var p = new Promise (f);
p.then (result => (result + 1)).then (result => console.log (result));
667
```

- Si un premier fulfiller retourne une promise – ce qui sera déterminé en cherchant à savoir si ce qui est retourné est thenable – ce dont il est question plus loin -, cette promise ne sera résolue qu'après la première – logique, car le fulfiller qui retourne la seconde promise n'est appelé que lorsque la première promise est résolue -, et `.then ()` permet alors d'ajouter un fulfiller à cette seconde promise :

```
function f0 (resolve, reject) {
    window.setTimeout (() => resolve (666), 2000);
}
function f1 (resolve, reject) {
    window.setTimeout (() => resolve (13), 2000);
}
var p = new Promise (f0);
p.then (result => { console.log (result); return (new Promise (f1)); }).then (result => console
.log (result));
666 // 2 secondes après le début du programme
13 // 4 secondes après le début du programme (ie : 2 secondes après le résultat précédent)
```

Ces trois comportements peuvent être indifféremment assurés en partant du principe que « qui peut le plus, peut le moins » : dans tous les cas, `.then ()` doit retourner une promise. Simplement, la promise sera créée par le système des promises – ce qui n'implique pas la fourniture d'un executor quand le fulfiller ne retourne pas une promise, mais rien ou une valeur -, sur le résultat retourné par le fulfiller ou le rejecter :

rien	Promise fulfilled dont le résultat est undefined
valeur	Promise fulfilled dont le résultat est la valeur
promise pending	Promise qui sera résolue dont le résultat sera celui de la promise
promise fulfilled	Promise fulfilled dont le résultat est celui de la promise
promise rejected	Promise rejected dont la raison est celle de la promise
(exception)	Promise rejected dont la raison est l'erreur soulevée

La promise retournée est bien une nouvelle promise, même si le fulfiller ou le rejecter a retourné une promise, et dans quelque état que soit cette dernière. Par exemple, fulfilled :

```
var p;
var p0 = new Promise ((resolve, reject) => resolve (666));
var p1 = p0.then (result => { p1 = new Promise ((resolve, reject) => resolve (result)); return (p); });
// Affiche false
console.log (p === p1);
```

Il est donc clair que `p.then ().then ()` ne se lit pas du tout comme `p.then ()` suivi de `p.then ()` !

Appeler du code dans tous les cas

En plus des fulfillers et des rejecters que `.then ()` et `.catch ()` permettent d'ajouter à une promise, la méthode `.finally ()` permet de lui ajouter une fonction qui sera appelée, sans aucun paramètre, dans tous les cas après les fulfillers ou les rejecters, sur le modèle de l'instruction `finally ()` en matière de gestion d'exception :

```
var mustSuccess = true;
var p = new Promise ((resolve, reject) => mustSuccess ? resolve () : reject ());
p.then (result => console.log ("Success"), reason => console.log ("Failure"));
// "Finally" toujours affiché, que mustSuccess soit true ou false
p.finally (() => console.log ("Finally"));
```

L'idée est de permettre de spécifier une fonction à exécuter systématiquement, plutôt que de contraindre le développeur à la spécifier deux fois comme ce serait le cas ainsi… :

```
p.then (() => console.log ("Finally"), () => console.log ("Finally"));
```

…ou ainsi :

```
p.finally (() => console.log ("Finally"));
p.catch (() => console.log ("Finally"));
```

Comme `.then ()` et `.catch ()`, `.finally ()` retourne une promise.

Résoudre plusieurs promises « parallèlement »

L'objet Promise dispose pareillement d'une méthode `.reject ()` qui retourne une promise rejected pour la raison fournie :

```
var p = Promise.rejected (13);
```

L'objet Promise dispose aussi de méthodes `.all ()` et `.race ()` pour demander la résolution de plusieurs promises :

- `.all ()` retourne une promise qui est résolue une fois que toutes les promises fournies en paramètres sont résolues, et dont le résultat est alors un tableau des résultats de ces dernières, ou rejetée aussitôt qu'une de ces promises est rejetée, et dont la raison est alors celle de la promise rejetée :

```
function write (text, delay) {
    return (new Promise ((resolve, reject) => {
        window.setTimeout (() => {
            console.log (`Returning "${text}"`);
            resolve (text);
        }, delay);
    }));
}
Promise.all ([write ("Hello", 1000), write ("World", 2000)]).then (result => console.log (`.all
 (): ${result}`));
Returning "Hello"
Returning "world"
.all(): ["Hello", "world"]
```

- `.race ()` retourne un promise qui est résolue aussitôt qu'une des promises fournies en paramètres est résolue, et dont le résultat est alors celui de cette promise, ou rejetée aussitôt qu'une de ces promises est rejetée, et dont la raison est alors celle de la promise rejetée :

```
Promise.race ([write ("Hello", 1000), write ("World", 2000)]).then (result => console.log
 (`.race (): ${result}`));
Returning "Hello"
.race (): "Hello"
Returning "world"
```

Une promise n'est pas une instance de Promise

Le test suivant échoue, car les deux pages n'ont pas même objet `Promise` :

```
<!-- index.html -->
<html>
<body>
<iframe src="iframe.html"></iframe>
<script>
test = (p) => console.log (p instanceof Promise ? "ok" : "nok");
</script>
</body>
</html>

<!-- iframe.html -->
<html>
<body>
<script>
parent.test (new Promise ((resolve, reject) =>; resolve ()));
</script>
</body>
</html>
```

Pour déterminer si un objet est une promise, il ne faut pas chercher à déterminer si c'est une instance de Promise. La technique à employer est celle du « duck-typing » :
if it looks like a duck, and qwacks like a duck, it must be a duck.
Elle consiste à déterminer si l'objet est thenable, c'est-à-dire s'il dispose d'une méthode `.then ()`.

Une écriture facilitée avec async et await

Depuis ECMAScript 2017, JavaScript s'est enrichi de deux instructions supplémentaires : `async` et `await`. Ces instructions permettent de faciliter plus encore l'écriture du code asynchrone, en masquant en grande partie le recours à l'objet `Promise`.
Un exemple permet de comprendre de quoi il en retourne. Soit une API qui expose une fonction retournant une promise :

```
function asyncGetSomeData (n) {
    return (new Promise ((resolve, reject) => {
        console.log ("Getting some data...");
        window.setTimeout (() => {
            var i, numbers;
            numbers = new Array ();
            for (i = 0; i != n; i ++)
```

```
                numbers.push (i);
            resolve (numbers);
        }, 2000);
    }));
}

function asyncComputeSomething (numbers) {
    return (new Promise ((resolve, reject) => {
        console.log ("Computing something...");
        window.setTimeout (() => {
            var i, s;
            s = 0;
            for (i = 0; i != numbers.length; i ++)
                s += numbers[i];
            resolve (s);
        }, 2000);
    }));
}

function syncDisplaySomeResult (sum) {
    console.log (`And the result is: ${sum}`);
}
```

L'API s'utilise normalement ainsi :

```
function doSomething (n) {
    return (asyncGetSomeData (n).then (asyncComputeSomething).then (syncDisplaySomeResult));
}
console.log ("Start");
doSomething (10).then ((result) => console.log ("API used"));
console.log ("End");
Start
Getting some data...
End
Computing something...
And the result is: 45
API used
```

Grâce à `async` et `await`, il est désormais possible d'utiliser l'API d'une manière qui s'apparente plus à l'utilisation de fonctions synchrones :

```
async function doSomething (n) {
    var data = await asyncGetSomeData (n);
    var sum = await asyncComputeSomething (data);
    syncDisplaySomeResult (sum);
}
console.log ("Start");
doSomething ().then ((result) => console.log ("API used"));
console.log ("End");
```

`async` entraîne la création d'un objet `AsyncFunction`. L'appel à une telle fonction retourne une promise sur cette fonction. Cette promise est donc fulfilled quand la fonction retourne une valeur, et rejected quand la fonction soulève une exception.

`await` ne peut s'utiliser que dans une fonction `async`, sur une promise. La fonction rend la main au programme, mais en interne, son exécution est suspendue jusqu'à ce que la promise soit résolue. Lorsque la promise est fulfilled ou rejected, le résultat, ou la raison, est alors retourné et l'exécution de la fonction reprend – après la fin du programme, « run-to-completion » oblige.

Bref, avec `async` et `await`, il est possible de se dispenser de faire une référence apparente au système des promises.

Conclusion

Tout cela ayant été expliqué, il est possible de revenir sur la définition d'une promise donnée dans la spécification ECMAScript 2015 : *A Promise is an object that is used as a placeholder for the eventual results of a deferred (and possibly asynchronous) computation.*

Effectivement, une promise est un objet qui est retourné par une fonction asynchrone au programme qui l'appelle, et qui permet à ce programme d'accéder au résultat retourné par cette fonction une fois qu'il sera disponible, via un ou plusieurs fulfillers que le programme fournit à la promise via sa méthode `.then ()`. Ce qui complique la compréhension du système des promises à partir de la définition, c'est que cette dernière ne donne pas à voir la fonction asynchrone en question. Quand on écrit… :

```
var p = new Promise (f);
```

…ce n'est pas `f ()` qui est la fonction asynchrone ! La fonction asynchrone, c'est celle quand le contexte duquel ce code se trouve, ici la fonction `asyncF ()` :

```
function asyncF () {
    return (new Promise ((resolve, reject) => {
        /*
        Ici le code exécuté de manière asynchrone, qui doit se terminer par... :
            resolve (result);
        ...s'il réussit et par... :
            reject (reason);
        s'il échoue.
        */
    });
}
```

Or le problème, c'est qu'à force d'avoir pris l'habitude de lire la référence de l'API sur MDN plutôt que la spécification ECMAScript du fait que cette dernière est totalement illisible, le développeur se jette sur l'objet `Promise` comme sur le reste, sans prendre assez de recul pour comprendre que ce n'est que la partie émergée d'un iceberg. Et pour comprendre l'iceberg, il faut repartir à la base : comprendre comment fonctionne JavaScript, et non plus se contenter de programmer en JavaScript sans se poser de question.

Bref, il faut fournir l'effort de comprendre la machine virtuelle sur laquelle s'exécute un programme JavaScript, tout comme on a fourni l'effort de comprendre la machine physique sur laquelle s'exécute un programme C. Cette machine virtuelle, c'est un ensemble constitué du runtime JS, et du navigateur qui l'utilise.

Bibliographie sélective

You dont' know JS: Async & Performance par Kyle Simpson
https://github.com/getify/You-Dont-Know-JS/tree/master/async%20%26%20performance

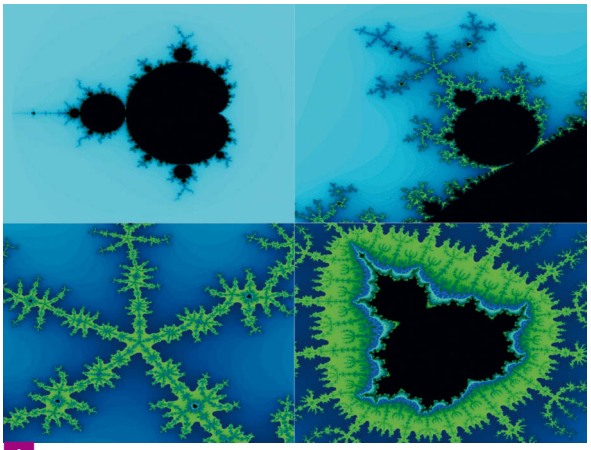
Exploring ES6 par Axel Rauschmayer
http://exploringjs.com/es6/index.html

Par ailleurs, cette vidéo sur l'event loop est très intéressante :
In The Loop par Jake Archibald lors de la JSConf.Asia 2018 :
https://www.youtube.com/watch?v=cCOL7MC4Pl0

Denis Duplan
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Mandelbrot en WebAssembly

Partie 1



1 *Représentations d'un ensemble de Mandelbrot.*

Ainsi donc, grâce à WebAssembly, il serait possible de faire tourner des programmes fulgurants dans une page Web. Mazette ! Pourquoi ne pas tester la proposition en réalisant un petit programme qui requiert traditionnellement une certaine puissance de calcul, comme une représentation de l'ensemble de Mandelbrot ? (Figure 1)

niveau 200

Ce serait aussi l'opportunité de discuter un peu des performances des opérations graphiques nécessitant de travailler au pixel pour produire une image dans une page Web. Oh! Nous ne sortirons pas l'artillerie lourde, à savoir WebGL, qui a été longuement présenté dans plusieurs articles. Nous nous concentrerons simplement sur le contexte 2D d'un objet `<canvas>`, en dessinant au pixel.

Vous pouvez télécharger tous les exemples sur le site de l'auteur.

Ce qu'il faut savoir sur WebAssembly

Puisqu'il s'agit de faire découvrir WebAssembly à travers un exemple, il convient d'en préciser les notions de WebAssembly qui vont nous servir, et celles-là seulement. Pour des explications plus générales, je renvoie à cet article précédemment publié.

WebAssembly, c'est quoi ? S'agissant de quelque-chose qui a été défini ici pour répondre à plusieurs besoins, il n'est pas facile de répondre facilement à cette question. Le mieux est de dire que c'est une technologie, qui permet d'exécuter du code de manière très rapide mais aussi très sécurisée dans une page Web – ce qui implique une indépendance au regard de la plateforme.

Cette technologie comprend plusieurs choses :

- Tout d'abord, un format binaire, que les auteurs ont fait le choix de rendre intelligible en permettant de l'exprimer sous forme d'un langage, une forme d'assembleur que nous appellerons Wasm pour faire la distinction avec la technologie. Ce format binaire est celui que doit prendre un programme WebAssembly – code et données -, ou module, pour pouvoir être exécuté.
- Ensuite, un moteur qui permet d'exécuter un module sous forme binaire. Ce moteur se saisit du binaire qui lui est communiqué, le valide au regard de tout un ensemble de règles – par exemple, la définition d'une fonction ne doit pas mentionner plus d'un résultat – et exécute le binaire dans un environnement très sécurisé, car cette exécution ne peut conduire à déborder de la mémoire attribuée au données pour empiéter sur celle où se trouve le code.

Pour produire un module sous forme binaire, il est possible de s'y prendre de deux manières :

- la première est de compiler un source écrit dans un langage haut-niveau – par exemple C ou C++ ;
- la seconde est d'assembler un source écrit dans le langage bas-niveau – le Wasm.

Dans cet article, nous adopterons la seconde solution – en utilisant le verbe « assembler », comme ce doit être le cas. Toutefois, il doit être bien clair que ce n'est pas ce que les concepteurs de WebAssembly ont eu en tête. Leur solution, c'est la première. Par ailleurs, si nous cherchons à réaliser un module exécuté dans une page Web, il doit être tout aussi clair que la nature même de la technologie doit permettre de l'utiliser dans d'autres contextes, comme précisé ici, tels que Node.js.

Rappelons que la spécification de WebAssembly est illisible pour le commun des mortels. Fort heureusement, Dan Gohman, qui a participé à l'élaboration de la technologie, a produit l'excellent WebAssembly Reference Manual, sorte de spécification officielle, mais lisible pour sa part. Pour ce qui concerne l'utilisation de WebAssembly dans une page Web, ce qui sort du champ de la spécification, il est possible de se référer à diverses documentations officielles, mais les meilleures sont celles de MDN, tout particulièrement les didacticiels.

Charger et interagir avec le module

Pour le développeur d'applications front-end sur le Web que nous allons incarner maintenant, WebAssembly se présente sous le jour d'une nouvelle API officiellement présentée ici, mais documentée de manière plus intelligible ici sur MDN. Cette API permet de charger le fichier du module mis sous forme binaire, de l'instancier, et d'échanger avec depuis un programme JavaScript – accéder à des variables, appeler des fonctions.

En l'espèce, le programme JavaScript – ci-après le programme – fera le minimum : demander au module Wasm – ci-après, le module – de calculer les couleurs des pixels de l'image constituant la représentation de l'ensemble de Mandelbrot, sur la base de paramètres décrivant cet ensemble et l'image. Autrement dit, le programme appellera une fonction exposée par le module et lui transmettra une référence sur un espace mémoire que le module remplira, et que le programme se débrouillera ensuite pour afficher. Comment charger et instancier le module ? Contentons-nous de rappeler la solution qui, en l'absence de définition officielle d'un type MIME adéquat,

fonctionne à date avec Firefox. Elle mobilise les API Fetch et WebAssembly de JavaScript :

```
fetch ("mandelbrot.wasm").then (
    response => response.arrayBuffer ().then (
        arrayBuffer => WebAssembly.instantiate (arrayBuffer, importObject).then (
        response => {
            // Utiliser response.instance
        },
        error => { console.log (error.message) }
)));
```

Le point important est la manière dont le programme et le module communiquent. En premier lieu, c'est le programme qui communique un objet contenant une référence sur l'espace mémoire à peupler de pixels. La manœuvre se déroule au moment de l'instanciation du module, via un objet ici nommé `importObject` transmis en paramètre à `WebAssembly.instantiate ()` :

```
var importObject = {
    someImports: {
        memory: new WebAssembly.Memory ({ initial: Math.ceil (WIDTH * HEIGHT * 4 / 65536) })
    }
}
```

Les imports doivent être structurés sur deux niveaux, ce qui explique pourquoi `importObject` contient un autre objet, pour l'occasion nommé imports, qui contient à son tour l'objet memory qu'il s'agit de partager avec le module.

Cet objet `WebAssembly.Memory` correspond à l'espace mémoire où le module doit stocker les pixels. Comme le programme utilise un canvas pour afficher l'image, un pixel doit être codé sur 4 octets – transparence, rouge, vert et bleu. Et comme la taille d'une mémoire est donnée en pages, et qu'une page correspond à 64 Ko, le calcul est vite fait pour déterminer le nombre de pages requises pour une image de `WIDTH` x `HEIGHT` pixels.

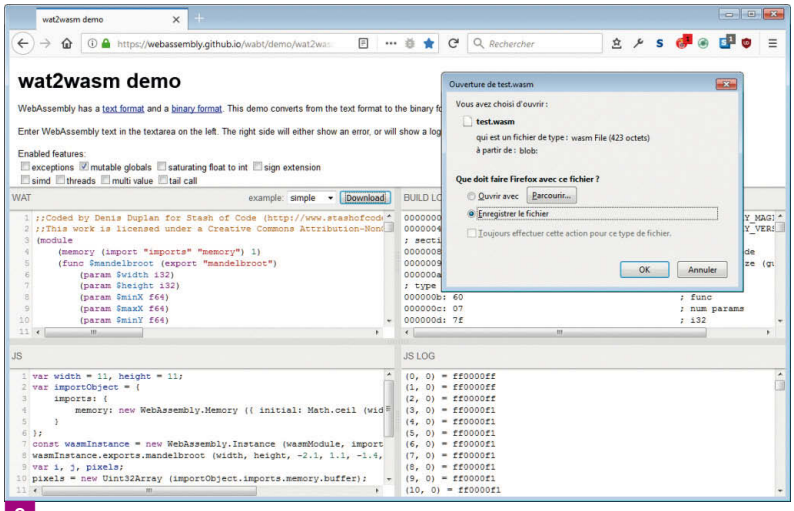
Une fois que le module est chargé, la séquence des promesses se résout par l'exécution du code utile du programme, en premier lieu l'appel à une fonction exportée par le module sous le nom de `mandelbrot`, qui calcule les couleurs des pixels de l'ensemble de Mandelbrot en fonction des paramètres qui lui parviennent. L'utilisation de `response.instance` mentionnée plus tôt prend donc avant tout la forme suivante :

```
response.instance.exports.mandelbrot (/* paramètres... */);
```

Ecrire et assembler le module

Un module est composé de différentes sections. En l'espèce, le module doit contenir une section pour définir une mémoire correspondant à celle importée depuis le programme, et une section pour définir une fonction `$mandelbrot ()` exportée, que le programme connaîtra sous le nom de `mandelbrot ()`.

```
(module
    (memory (import "imports" "memory") 1)
    (func $mandelbrot (export "mandelbrot")
        ;; Calculer les pixels et les stocker dans la mémoire
    )
)
```



2 *wat2wasm, pour créer un fichier binaire .wat.*

Noter que la taille initiale de la mémoire doit être définie de nouveau. Pourquoi avoir indiqué seulement une page, et non le nombre minimal de pages calculé dans le programme lors de la création de l'objet `WebAssembly.Memory` ?

La section du *WebAssembly Reference Manual* n'est pas claire sur ce point, mais la documentation officielle ne l'est pas non plus. Une seule chose est certaine : lorsqu'une mémoire est importée, la taille initiale spécifiée dans le programme doit être supérieure ou égale à celle spécifiée dans le module.

A l'inverse, que se passe-t-il si la taille initiale spécifiée dans le module est inférieure à celle spécifiée dans le programme, comme c'est le cas ici ? Pas d'explications. Un test montre que spécifier une seule page dans le module n'entrave pas les accès en écriture au-delà de cette page, tant que c'est dans la limite de la taille initiale spécifiée dans le programme. Dans ces conditions, autant se contenter de spécifier une page dans le module… Si quelqu'un trouve des précisions, qu'il me les rapporte…

Le module se présente alors sous la forme d'un fichier texte, au format WebAssembly Text (`.wat`). Il doit être assemblé pour produire un fichier binaire (`.wat`), que le programme pourra charger. La solution la plus simple pour cela consiste à utiliser l'outil wat2wasm du WebAssembly Binary Toolkit. Mais plutôt que de perdre du temps à l'installer, il suffit d'utiliser la version en ligne dont l'URL est mentionnée dans les références à la fin de cet article.

La manœuvre consiste simplement à copier-coller le texte du module dans la fenêtre en haut à gauche, puis à cliquer sur le bouton Download pour télécharger le fichier résultant, pour l'occasion nommé `mandelbrot.wasm` (Figure 2).

Tout n'est pas instruction… mais quoi ?

Le cadre étant posé, il est maintenant possible de se focaliser sur le code utile, à savoir le code du module qui doit générer les pixels, et le code du programme qui doit les afficher. Mais avant d'y venir, il faut exposer quelques principes de la programmation en Wasm à laquelle le présent article doit permettre de s'initier. Précisons qu'il ne faut pas confondre les mots-clés utilisés pour décrire un module, tels que memory, et les instructions proprement dites du langage, tels que `i32.const`. Toutefois, opérer cette distinction n'est pas toujours évident. En particulier, ça ne l'est pas pour `func`, qui n'est pas une ins-

truction mais un mot-clé qui entraîne formellement la création de plusieurs sections. En effet, le code suivant… :

```
(func)
```

…est assemblé sous la forme binaire suivante :

```
0000000: 0061 736d ; WASM_BINARY_MAGIC
0000004: 0100 0000 ; WASM_BINARY_VERSION
; section "Type" (1)
0000008: 01 ; section code
0000009: 00 ; section size (guess)
000000a: 01 ; num types
; type 0
000000b: 60 ; func
000000c: 00 ; num params
000000d: 00 ; num results
0000009: 04 ; FIXUP section size
; section "Function" (3)
000000e: 03 ; section code
000000f: 00 ; section size (guess)
0000010: 01 ; num functions
0000011: 00 ; function 0 signature index
000000f: 02 ; FIXUP section size
; section "Code" (10)
0000012: 0a ; section code
0000013: 00 ; section size (guess)
0000014: 01 ; num functions
; function body 0
0000015: 00 ; func body size (guess)
0000016: 00 ; local decl count
0000017: 0b ; end
0000015: 02 ; FIXUP func body size
0000013: 04 ; FIXUP section size
; section "name"
0000018: 00 ; section code
0000019: 00 ; section size (guess)
000001a: 04 ; string length
000001b: 6e61 6d65 name ; custom section name
000001f: 02 ; local name type
0000020: 00 ; subsection size (guess)
0000021: 01 ; num functions
0000022: 00 ; function index
0000023: 00 ; num locals
0000020: 03 ; FIXUP subsection size
0000019: 0a ; FIXUP section size
```

Comme il est possible de le constater, ce qui peut apparaître comme la définition d'une seule section dans le source du module se traduit sous la forme de plusieurs sections dans le binaire : « Function » pour la fonction, « Type » pour son type – une fonction sans argument qui ne retourne rien -, « Code » pour son corps.Et qui utilise le code suivant… :

```
(func)
(func (param i32))
```

…peut constater que cela n'entraîne pas la création de nouvelles sections dans le binaire, mais la création de « sous-sections » dans ces dernières, notamment pour distinguer le type de la première fonction de celui de la seconde dans la section « Type », et pour distinguer leurs codes respectifs dans la section « Function ». `func` n'est donc pas une instruction ? Oui et non. Car comme cela sera expliqué plus loin, le mot-clé entraîne la définition d'une position dans le code à laquelle il est possible de sauter grâce à diverses instructions de contrôle. En ceci, `func` se comporte comme `br`, qui est à proprement parler une instruction…
Dans un module, le code est toujours logé dans des fonctions – en l'occurrence, il n'y en aura qu'une, qui prendra de nombreux paramètres sans retourner de résultat :

```
(module
    (func $mandelbrot (export "mandelbrot")
        (param $width i32)
        (param $height i32)
        ;; Et autres paramètres...
        ;; Calculer les pixels et les stocker dans la mémoire
    )
)
```

A quoi ressemble le code ? Comme précisé ici dans la spécification, programmer en Wasm, c'est programmer une machine à pile :
WebAssembly code consists of sequences of instructions. Its computational model is based on a stack machine in that instructions manipulate values on an implicit operand stack, consuming (popping) argument values and producing or returning (pushing) result values.
Autrement dit, il n'y a pas de registres, comme par exemple en assembleur MC68000 :

```
move.w #100,d0   ;;Stocker 100 dans D0
move.w #10,d1    ;;Stocker 10 dans D1
sub.w d1,d0      ;;Calculer 100 - 10 et stocker 90 dans D0
```

En Wasm, tous les opérandes d'une instruction comme `i32.sub` doivent être empilés avant que l'instruction ne soit utilisée. L'instruction dépilera les opérandes, calculera la différence et empilera le résultat :

```
i32.const 100   ;;Empiler 100
i32.const 10    ;;Empiler 10
i32.sub         ;;Dépiler 100 et 10, calculer 100 - 10, empiler 90
```

Une notation à base de S-expressions permet de condenser l'écriture, mais aussi de la faciliter en renversant l'ordre dans laquelle une opération est constituée. Ainsi, le code précédent peut aussi être écrit ainsi :

```
(i32.sub (i32.const 100) (i32.const 10))
```

Pour une liste exhaustive des instructions de Wasm, le mieux est de se référer à la partie du *WebAssembly Reference Manual* consacrée aux instructions.
Le code de notre fonction `$mandelbrot ()` n'utilise qu'un nombre limité d'instructions. Elles manipulent deux des quatre types de valeurs possibles en Wasm : des flottants 64 bits (`f64`) et des entiers 32 bits (`i32`). La plupart de ces instructions sont assez intuitives. La liste suivante est un peu lancinante à lire, mais sa lecture permet de bien visualiser l'omniprésence des accès à la pile :

i32.const 666	Empile 666 sous forme d'un entier 32 bits
f64.const 0.123	Empile 0.123 sous forme d'un flottant 64 bits
f64.convert_u/i32	Dépile la valeur A dernièrement empilée (A doit être un entier 32 bits), convertit A en flottant 64 bits, empile le résultat sous forme d'un flottant 64 bits
i32.truc_u/f64	Dépile la valeur A dernièrement empilée (A doit être un flottant 64 bits), convertit sa partie entière en entier 32 bits, empile le résultat sous forme d'un entier 32 bits
f64.add	Dépile la valeur B dernièrement empilée, puis la valeur A précédemment empilée (A et B doivent être des flottants 64 bits), additionne B à A, empile le résultat sous forme d'un flottant 64 bits
i32.add	Comme `f64.add`, mais avec des entiers 32 bits
f64.sub	Dépile la valeur B dernièrement empilée, puis la valeur A précédemment empilée (A et B doivent être des flottants 64 bits), soustrait B de A, empile le résultat sous forme d'un flottant 64 bits
i32.sub	Comme `f64.sub`, mais avec des entiers 32 bits
f64.mul	Dépile la valeur B dernièrement empilée, puis la valeur A précédemment empilée (A et B doivent être des flottants 64 bits), multiplie A par B, empile le résultat sous forme d'un flottant 64 bits
i32.mul	Comme `f64.mul`, mais avec des entiers 32 bits
f64.div	Dépile la valeur B dernièrement empilée, puis la valeur A précédemment empilée (A et B doivent être des flottants 64 bits), divise A par B, empile le résultat sous forme d'un flottant 64 bits
i32.or	Dépile la valeur B dernièrement empilée, puis la valeur A précédemment empilée (A et B doivent être des entiers 32 bits), combine A avec B par OU logique bit à bit, empile le résultat sous forme d'un entier 32 bits
i32.eqz	Dépile la valeur A dernièrement empilée (A doit être un entier 32 bits), teste si A est égal à l'entier 32 bits 0, empile le résultat (0 pour faux, 1 pour vrai) sous forme d'un entier 32 bits
i32.ne	Dépile la valeur B dernièrement empilée, puis la valeur A précédemment empilée (A et B doivent être des entiers 32 bits), teste si B est différent de A, empile le résultat sous forme d'un entier 32 bits (0 pour faux, 1 pour vrai)
i32.store	Dépile la valeur B dernièrement empilée, puis la valeur A précédemment empilée (A et B doivent être des entiers 32 bits), écrit les 4 octets qui composent B à l'offset A (donné en octets !) dans l'unique mémoire autorisée
set_local $x	Dépile la valeur A dernièrement empilée (A doit être du type de $x, déclarée par exemple une variable entière 32 bits déclarée à l'aide de `(local $x i32)`), stocke A dans la variable locale `$x` en little endian
get_local	Empile la valeur de la variable `$x`

Toutefois, le code utilise aussi les instructions `block`, `loop`, `br`, et `br_if`, dont le fonctionnement est nettement moins intuitif.

Comprendre les instructions de contrôle en Wasm

Les instructions de contrôle sont peu nombreuses, mais leur usage est particulièrement délicat. Comme précédemment indiqué, il ne s'agira que d'évoquer celles qui sont utilisées dans notre module, à savoir : `block`, `loop`, `br`, et `br_if`. De toute manière, les autres sont bien plus faciles à comprendre.

Par ailleurs, nous n'allons pas du tout rentrer dans les détails. La manière dont ces différentes instructions fonctionnent, et tout particulièrement comment elles jouent sur la pile, est un sujet assez complexe, qui sera traité dans un prochain article. Par exemple, il s'agira de comprendre pourquoi le code suivant peut être valide… :

```
(func (result i32)
    i32.const 7
    i32.const 13
    br 0
    i32.const 666
)
```

…alors que celui-là ne l'est pas :

```
(func (result i32)
    i32.const 7
    br 0
    i32.const 13
    i32.const 666
)
```

Dans le cadre du présent article – et cela doit sonner comme un avertissement -, il ne s'agira que d'explorer des usages parfaitement balisés des instructions de contrôle mentionnées.

Comme le WebAssembly Reference Manual permet relativement bien de le comprendre, `block` et `loop`, mais aussi `func` qui constitue un cas particulier, servent à délimiter des parties du code, dites régions.

`func` doit toujours figurer dans une S-expression, ce qui permet d'en signaler la fin. Pour leur part, `block` et `loop` doivent se terminer par un `end`. Par exemple :

```
(func
    (local $i i32)
    block
        i32.const 7
        set_local $i
    end
)
```

Toutefois, il est ici encore possible d'utiliser une S-expression pour simplifier l'écriture. En l'occurrence, cela permet de ne pas mentionner `end` :

```
(func
    (local $i i32)
    (block
        i32.const 7
        set_local $i
    )
)
```

Cet exemple offre l'opportunité de lever une ambiguïté qui peut faire perdre beaucoup de temps. Le code suivant produit le même résultat :

```
(func
    (local $i i32)
    (loop
        i32.const 7
        set_local $i
    )
)
```

Pourquoi ? Parce que contre toute attente, `loop` n'effectue pas de boucle – merci aux génies qui ont choisi de nommer l'instruction ainsi ! Dès lors, quelle différence entre `block`, et `loop` ?
Pour le comprendre, il faut savoir qu'en plus de définir une région, ces instructions définissent une position dans la région – au niveau de quelle instruction dans la région, nous le verrons plus loin. Quand une position est définie, machine à pile oblige, elle est empilée. L'empilement des positions contribue à structurer le code sous la forme d'une imbrication de régions.
Depuis une région, il est possible de sauter à une autre région à l'aide de `br` ou `br_if`. A cette occasion, il faut mentionner un numéro qui, partant de 0 pour désigner la région courante, est augmenté de 1 à chaque région englobante traversée : c'est la profondeur d'imbrication (nest depth), qui s'exprime donc relativement. Par exemple, si trois régions A, B et C sont successivement imbriquées :
• dans le contexte de A, la seule région accessible est 0 ;
• dans le contexte de B, les régions accessibles sont 0 (B) et 1 (A) ;
• dans le contexte de C, les régions accessibles sont 0 (C), 1 (B) et 2 (A).
Noter que le saut vers une région s'effectue alors à la position définie pour cette région – nous verrons qu'elle dépend de l'instruction utilisée pour définir la région.
Cela implique qu'il n'est donc pas possible de sauter à tout instant de n'importe quelle région, à la position de n'importe quelle autre région. Depuis une région, il est seulement possible de sauter vers cette région, ou vers une région englobante. Cela se comprend, car pour accéder à la position d'une région lors d'un branchement, il faut dépiler celles qui ont été empilées à sa suite.
Pour aller au-delà, le plus simple est de mobiliser une notation qui n'est somme toute qu'une figuration de la réalité, mais qui est assez explicite pour permettre de saisir facilement cette dernière. Aussi, il faut considérer que `func`, définit une position à la fin de la région qu'elle définit. Introduisons un premier branchement inconditionnel par `br` :

```
(func            ;; Empiler (A)
   (local $i i32)
   (block        ;; Empiler (B)
      br 0       ;; Sauter en (B)
      i32.const 7
      set_local $i
      ;; (B)
   )
   ;; (A)
)
```

Comme il est possible de le constater, la notation consiste à faire figurer explicitement dans le code, sous forme de commentaire pour ne pas bloquer l'assemblage dans wat2wasm, les positions définies par les instructions de contrôle, en comptant `func` parmi ces dernières. Ainsi, `br 0` correspond à un branchement dans la région courante – celle de la fonction serait désignée par `1` – à la position implicitement définie à l'entrée de la région par `block`, qui se trouve tout à la fin. Noter que `func` définit une position qui, elle aussi, se trouve tout à la fin de sa région.
La différence entre `block` et `loop`, c'est que `loop` définit une position au début, et non à la fin, de sa région. Dans ces conditions, chacun comprendra que simplement remplacer `block` par `loop` dans le code précédent génère une boucle infinie :

```
(func            ;; Empiler (A)
   (local $i i32)
   (loop         ;; Empiler (B)
      ;; (B)
      br 0       ;; Sauter en (B)
      i32.const 7
      set_local $i
   )
   ;; (A)
)
```

S'il s'agit de réaliser une boucle avec un compteur, il est possible d'écrire :

```
(func                           ;; Empiler (A)
   (local $i i32)
   (set_local $i (i32.const 10))
   (loop                        ;; Empiler (B)
      ;; (B)
      (set_local $i (i32.sub (get_local $i) (i32.const 1)))
      (br_if 0 (i32.ne (get_local $i) (i32.const 0)))    ;; Sauter en (B)
   )
   ;; (A)
)
```

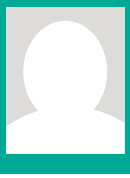
Toutefois, selon les conditions de sortie et la manière dont elles sont écrites, il peut être nécessaire d'introduire une instruction `block` définissant une région dans laquelle celle définie par `loop` sera imbriquée. Par exemple, une boucle décrémentant de 1 deux variables, et dont il s'agirait de sortir aussitôt que l'une d'entre elles atteint 0 :

```
(func                           ;; Empiler (A)
   (local $i i32)
   (local $j i32)
   (set_local $i (i32.const 5))
   (set_local $j (i32.const 10))
   (block                       ;; Empiler (B)
      (loop                     ;; Empiler (C)
         ;; (C)
         (set_local $i (i32.sub (get_local $i) (i32.const 1)))
         (set_local $j (i32.sub (get_local $j) (i32.const 1)))
         (br_if 1 (i32.eqz (get_local $i)))    ;; Sauter en (C)
         (br_if 1 (i32.eqz (get_local $j)))    ;; Sauter en (C)
         br 0                   ;; Sauter en (B)
      )
      ;; (B)
   )
   ;; (A)
)
```

Pourquoi cette imbrication ? Car la seule solution pour sortir de la boucle serait autrement un saut conditionnel à la région `1`, c'est-à-dire à la position se trouvant à la fin de la région définie par `func`. Dans conditions, les éventuelles instructions se trouvant entre la fin de la boucle et la fin de la fonction ne seraient pas exécutées, ce qui ne serait vraisemblablement pas un effet recherché :

Code complet sur programmez.com & github

La suite de cet article dans le prochain numéro



Franck Franchin
VoltaNode
franckf@voltanode.com

Python Quantique

Il y a une petite dizaine d'années, quand mes élèves ingénieurs me disaient tout le bien qu'ils pensaient du langage Python, je leur répondais, narquoisement, que j'avais été élevé aux vrais langages comme le C ou le Fortran. D'aucuns avaient pourtant essayé de me contaminer avec des horreurs comme le Lisp ou Java mais j'avais tenu bon à grands coups revanchards d'assembleur si nécessaire. Je n'avais trahi que pour le C++. J'avais même évité PHP, c'est dire. J'étais alors loin d'imaginer qu'en 2019, j'écrirais un article sur le Python… et encore moins, implémenté sur un ordinateur… quantique !

niveau 200

J'ai fondé et anime VoltaNode, une société franco-suisse de conseil en technologies et en innovation qui travaille principalement avec des investisseurs (VCs, family offices) et des grandes entreprises (directions de la stratégie ou du développement, incubateurs, etc.). Les grands sujets que nous abordons sont l'IoT, la cybersécurité, l'intelligence artificielle et bien évidemment l'informatique quantique.
Chez VoltaNode, nous essayons de sortir du *bullshit* traditionnel des sociétés de conseil : peu de blabla et encore moins de *slideware* éblouissant, mais, au contraire, du pragmatisme acquis sur le terrain, sur le « bench », auprès de startups, de chercheurs, d'universitaires et de laboratoires de renom.

Nous avons commencé à travailler sur le quantique il y a quasi dix ans, principalement sur les aspects liés à la cybersécurité (cryptographie et Internet quantique). Depuis ces 3 dernières années, tout s'est accéléré : communauté, simulateurs, outils, barrières technologiques qui sautent. Même s'il reste encore énormément de travail à réaliser et de sauts technologiques à faire, toutes les parties prenantes sont confiantes dans cette dynamique positive. Comme j'aime à dire : "no pain, no gain, high pain, high gain".
Nous avons créé deux formations sur l'informatique quantique :

- Un micro-séminaire dans la série de nos formats « 60 minutes pour comprendre » ;
- Un atelier pratique.

C'est dans le cadre de cet atelier pratique que nous avons été amenés à créer quelques scénarios de programmation quantique, dans un premier temps sur simulateur, puis dans un second temps… sur un vrai ordinateur quantique !

Le quantique c'est sympa

Le quantique, c'est un peu comme le sexe chez les ados. Beaucoup en parlent mais peu ont réellement pratiqué. Nous avons donc voulu mettre les mains dans le cambouis, enfin, dans le code, afin de comprendre réellement ce changement majeur de paradigme.
La suite de cet article est le récit synthétique de cette expérience, qui, j'espère, vous passionnera autant que moi. Je dois vous avouer que j'en suis sorti un peu « changé ».

Spoiler : *Vous allez vite le découvrir - donc je préfère vous le dévoiler tout de suite : je suis un piètre développeur en Python…*

Je pars du principe que tous les lecteurs connaissent Python et disposent d'un environnement de développement ad hoc (mon préféré : Visual Studio Code + Jupyter + Anaconda).
Commençons donc par quelques notions d'informatique quantique afin que vous puissiez suivre ce que nous allons faire.

Tout d'abord, pourquoi faire de l'informatique quantique ? Parce que c'est plus cool que d'instancier pour la centième fois une C5n sur AWS… Oui, mais pas seulement.

En quelques mots et avec une grosse simplification que les experts me pardonneront bien volontiers, l'informatique quantique permet de résoudre certains problèmes avec une complexité polynomiale et non exponentielle. Pour le commun des mortels, cela signifie que casser certains algorithmes de cryptographie ou calculer la structure d'une protéine complexe pourra, grâce à l'informatique quantique, prendre un temps à l'échelle humaine – quelques minutes à quelques mois et non plus à l'échelle de l'univers – quelques centaines à quelques milliards d'années. On imagine aisément les applications et les attentes de la communauté scientifique.

Un ordinateur quantique est un système (extrêmement complexe) qui contrôle des états quantiques, les fameux *qubits*. Le qubit, c'est la version quantique de nos fameux bits. A la différence de nos amis binaires, un qubit peut avoir la valeur 0, la valeur 1, ou… les deux en même temps (c'est ce qu'on appelle la **superposition**). Pour ne pas confondre avec les bits classiques, on utilise une notation particulière : |0> ou |1>.

Si on dispose d'un ordinateur avec 4 qubits, on peut ainsi avoir une superposition simultanée de 2^4 états logiques distincts (soit 16 états) de |0000> à |1111>. Oui, j'ai bien écrit *simultanée* ! Si on a la chance de disposer d'un ordinateur avec 32 qubits, on passe à 2^{32} états logiques simultanés, soit plus de 4 milliards d'états !

A titre de comparaison avec le monde réel que nous connaissons, imaginons que je dispose de 4 pièces de monnaie. Si je joue à pile ou face avec les quatre pièces en même temps, j'ai 2^4 possibilités, mais une fois les pièces lancées et retombées, j'ai un seul 'résultat' visible parmi les 2^4 possibilités. Chaque résultat a une probabilité de $1/2^4$ (si les pièces sont idéales et non truquées). Les 4 qubits d'un ordinateur quantique existent eux dans 16 états en même temps ce qui permet de faire des calculs de manière très efficace (ce n'est toutefois pas du vrai parallélisme).
Une autre notion importante de l'informatique quantique est **l'intrication** (*entanglement*). Deux qubits peuvent former un système

avec intrication, ce qui signifie que l'état du système peut être connu de manière indépendante de l'état des qubits qui le composent. On parle aussi de **corrélation** entre les deux qubits. Si on observe un des qubits, on peut savoir quel est l'état de l'autre qubit s'il était mesuré de la même manière. Même si l'autre qubit est à l'autre bout de la planète ou de l'univers. « *Par la Force des qubits intriqués, intrigué tu seras* ».
Afin de pouvoir réaliser des traitements sur les qubits, on dispose de sortes de portes ou d'opérateurs quantiques (*quantum gates*), qui permettent de construire une logique quantique, tout comme on a construit une logique binaire dans les ordinateurs classiques.

Il existe actuellement deux principaux paradigmes majeurs en informatique quantique :

- Quantum Annealing (QA), dont les promoteurs sont les sociétés D-Wave Systems, Google et la NASA. Cette architecture est assez particulière et limitée en termes d'algorithmes quantiques. En particulier, elle ne peut pas exécuter le fameux algorithme de Shor qui permet de factoriser un entier dans un temps polynômial et non exponentiel. Par contre, les systèmes de ce type comportent déjà plusieurs centaines de qubits.
- Gate Model (QM), est plus généraliste et indépendant d'un fabricant particulier. IBM et Rigetti Computing sont les promoteurs de cette architecture, proposent des outils et surtout un accès (limité) à leurs ordinateurs quantiques. C'est pour cela que nous avons retenu cette architecture. Pour l'instant, ces systèmes ne dépassent pas quelques dizaines de qubits.

Rigetti et IBM mettent à notre disposition deux *frameworks*, Forest SDK et Qiskit, respectivement. D-Wave Systems propose son framework Ocean d'outils en Python, qui est Open-Source. Microsoft met à disposition un framework nommé Q# disponible pour Windows et MacOS/Linux.

Cela peut paraître surprenant pour certains d'entre vous mais Python est réellement devenu le langage de référence en informatique quantique, tout comme il s'est imposé en intelligence artificielle. Il existe toutefois des implémentations en Perl, C et environnement CUDA.

L'offre Qiskit

J'ai un petit faible pour Qiskit d'IBM qui est disponible en Open-Source aussi.
Si vous voulez reproduire les exemples de cet article, vous devez installer Qiskit sur votre ordinateur. Les prérequis sont : Ubuntu 16.04 ou ultérieur, macOS 10.12.6 ou ultérieur, Windows 7 ou ultérieur. Python 3.5 ou ultérieur.

Qiskit offre quatre outils :

- Qiskit Terra : un outil de bas niveau (circuits de base et impulsions de contrôle) qui permet de travailler et d'optimiser la programmation en prenant en compte les particularités technologiques d'un processeur quantique. Il permet aussi de « compiler » et d'exécuter le code sur un vrai système quantique (on parle de *backends*). C'est un peu la fondation du framework Qiskit.
- Qiskit Aqua : une librairie qui permet de travailler à haut niveau (applicatif) et avec des algorithmes dans des domaines comme les finances, l'intelligence artificielle ou la chimie.
- Qiskit Aer : des simulateurs à plusieurs niveaux de complexité et de performance.
- Qiskit Ignis : cet outil est destiné à ceux qui veulent comprendre et maîtriser les problématiques liées au bruit dans les processeurs et ordinateurs quantiques. Pour experts.

Je ne saurais que trop vous recommander d'utiliser Anaconda pour disposer d'un environnement simple et sain. L'installation du framework se fait alors en quelques commandes (environnement quantum dans l'exemple ci-dessous, mais nous omettrons cette précision de l'environnement dans la suite de l'article) :

```
conda update -n base -c defaults conda
conda create -n quantum python=3
source activate quantum
conda activate quantum
pip install qiskit[visualization] qiskit-aqua
```

Maintenant que vous avez installé votre environnement de travail, roulements de tambour ! Nous allons demander à IBM de nous permettre d'accéder à ses simulateurs d'ordinateurs quantiques mais aussi à ses ordinateurs quantiques véritables, disponibles via le Cloud (IBM Quantum Cloud Services).
De manière assez classique, nous allons créer un compte et récupérer un API Token.
Une fois le token récupéré, vous devez le sauver dans un fichier nommé qiskitrc en exécutant le code suivant :

```
from qiskit import IBMQ
IBMQ.save_account('MY_API_TOKEN')
```

Où 'MY_API_TOKEN' est votre Token. Bref, tout ça est très classique et pas bien compliqué. Merci IBM.

Lorsque vous vous connectez sur votre compte, vous pouvez voir que vous avez accès (limité) à des ordinateurs quantiques d'IBM. Effet Whaouh garanti lors du prochain apéro Startups ! 1

Vous pouvez même vérifier via Python à quelles machines vous avez accès. Je vous conseille de le faire dès maintenant car cela va vous permettre de vérifier que tout votre environnement est fonctionnel (dont votre Token). Pour cela, utiliser le code suivant :

```
from qiskit import IBMQ
IBMQ.load_accounts()
print("Available backends:",IBMQ.backends())
```

Le résultat devrait ressembler à :

```
Available backends:
[<IBMQBackend('ibmqx4') from IBMQ()>, <IBMQBackend('ibmqx2') from IBMQ()>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ()>, <IBMQBackend('ibmq_qasm
_simulator') from IBMQ()>]
```

L'accès aux ordinateurs quantiques d'IBM s'effectue via une file d'attente. Il est possible de lancer une requête qui donne la machi-

ne la plus disponible selon le nombre de qubits qu'on souhaite. Par exemple, si notre programme a besoin de 3 qubits, le code est :

```
from qiskit import IBMQ
IBMQ.load_accounts()
from qiskit.providers.ibmq import least_busy
large_enough_devices = IBMQ.backends(filters=lambda x: x.configuration().n_qubits
 > 3 and not x.configuration().simulator)
backend = least_busy(large_enough_devices)
print("The best backend is " + backend.name())
```

Cela étant dit, IBM met aussi à notre disposition un simulateur/émulateur de 32 qubits qui s'utilise comme les vrais. Nous en reparlerons.

Maintenant que nous nous sentons tout puissants avec nos accès à de vrais ordinateurs quantiques, passons à l'écriture d'un vrai code. C'est le moment où je vais être obligé de calmer les ardeurs des plus passionnés d'entre vous : on ne va ni casser un RSA 1024 ni plier une protéine complexe. On va… juste jouer à pile ou face.

Vous allez me dire : « Quoi ? tout ça pour un simple pile ou face ? » Oui.. mais sur un vrai ordinateur quantique, quand même ! Plus sérieusement, nous nous heurtons à deux difficultés majeures. Premièrement, la connaissance nécessaire pour implémenter de vrais algorithmes quantiques dépasse largement ces quelques lignes, et sans vouloir froisser personne, dépasse aussi les connaissances du développeur geek ' de base' (*no offense!*). Deuxièmement, nous n'allons travailler qu'avec 3 qubits, ce qui limite pas mal notre champ d'évaluation et d'investigation.

Une pièce de monnaie pour jouer à pile ou face est un système à deux niveaux ou états : pile ou face. Cela ressemble pas mal à un qubit dont l'état quantique est une distribution probabiliste. En utilisant la notation propre à l'univers quantique (notation de Dirac), on peut écrire cet état quantique sous forme d'un vecteur colonne, appelé *ket*, avec $a_0, a_1 \in \mathbb{C}$ (bref, des nombres complexes).

$$|\psi\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

Comme nous savons qu'en probabilités, la somme des probabilités est toujours égale à un, nous pouvons aussi écrire que :

$$\sqrt{|a_0|^2 + |a_1|^2} = 1$$

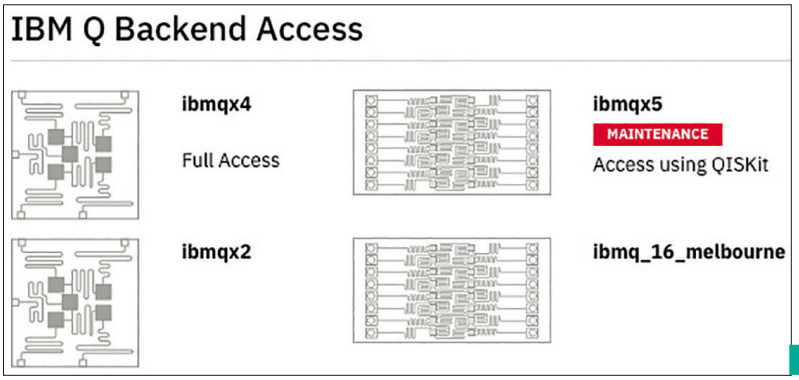
Pour ceux d'entre vous qui se rappellent un peu de leurs cours de mathématiques, on peut décomposer un vecteur dans une base canonique. La base canonique quantique serait donc :

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Qui sont deux qubits un peu particuliers. Et voici donc la décomposition :

$$|\psi\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = a_0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + a_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = a_0|0\rangle + a_1|1\rangle.$$

Nous aboutissons alors à une superposition, qui donne un résultat



1

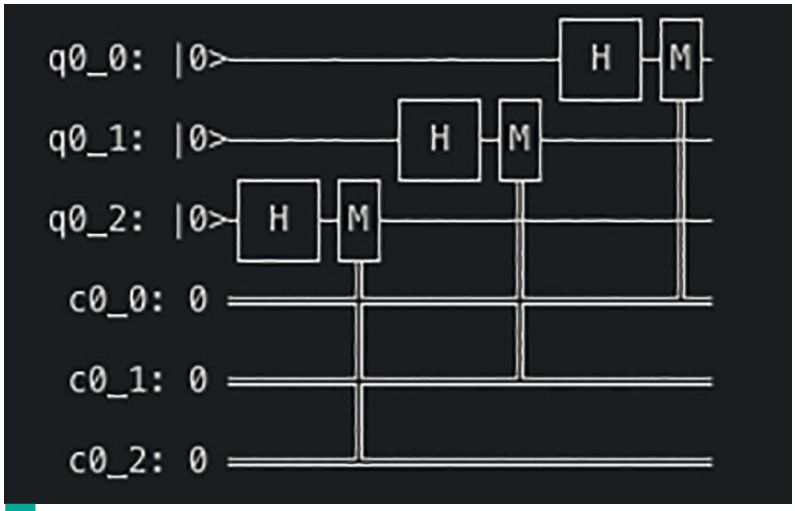
de 0 avec une probabilité de $|a_0|^2$ et un résultat de 1 avec une probabilité de $|a_1|^2$.

Nous allons utiliser cette analogie pour simuler le lancement de 3 pièces de monnaie sous la forme de 3 qubits et d'un circuit quantique assez simple qui mette en superposition les 3 qubits et qui permette de lire les résultats.
Qiskit Terra nous fournit 3 objets de base : QuantumCircuit, QuantumRegister et ClassicalRegister.

Nous avons besoin de 3 registres quantiques qubits, via QuantumRegister, et de 3 registres « miroirs » classiques, via ClassicalRegister, pour lire (mesurer) le résultat.
Nous allons aussi utiliser une porte quantique (*gate*) un peu particulière, dite de Hadamard, notée ***H***, pour mettre chacun des 3 qubits en état de superposition (cool !).
Par défaut chaque qubit est initialisé à $|0\rangle$.
Grâce à Qiskit, il est possible de visualiser le circuit avec QuantumCircuit.draw, ce qui s'avère bien pratique, surtout au début quand on ne maîtrise pas encore tout bien comme moi.
Le code est le suivant (sur simulateur pour l'instant).

```
from qiskit import ClassicalRegister, QuantumRegister, QuantumCircuit
from qiskit import execute
from qiskit import BasicAer
import numpy as np

backend = BasicAer.get_backend('qasm_simulator')
q = QuantumRegister(3)
c = ClassicalRegister(3)
circuit = QuantumCircuit(q, c)
circuit.h(q[0])
circuit.h(q[1])
circuit.h(q[2])
circuit.measure(q, c)
print(QuantumCircuit.draw(circuit, output='text'))
job = execute(circuit, backend, shots=100)
print(job.result().get_counts(circuit))
```

A l'exécution, on obtient le résultat suivant. Je laisse les gourous du traitement des listes sous Python peaufiner un affichage plus sympathique (histogramme ?). 2



2

```
{'100': 15, '110': 10, '101': 9, '000': 14, '011': 16, '111': 12, '001': 12, '010': 12}
```

Votre résultat est peut-être différent et c'est tout à fait normal. Exécutons une nouvelle fois le programme :

```
{'101': 9, '110': 16, '100': 14, '011': 10, '000': 17, '010': 15, '111': 11, '001': 8}
```

Nous avons bien une distribution probabiliste sur 100 lancés de 3 pièces en pile ou face. Mathématiquement, chaque combinaison devrait avoir la même probabilité mais il faudrait, comme dans la réalité, bien plus de 100 lancés pour y arriver. Passons maintenant à l'exécution du code sur un vrai ordinateur quantique :

```
from qiskit import ClassicalRegister, QuantumRegister, QuantumCircuit
from qiskit import execute
from qiskit import IBMQ
IBMQ.load_accounts()
from qiskit.providers.ibmq import least_busy
large_enough_devices = IBMQ.backends(filters=lambda x: x.configuration().n_qubits
> 3 and not x.configuration().simulator)
backend = least_busy(large_enough_devices)
print("The best backend is " + backend.name())
print("On lance le calcul sur un vrai ordinateur quantique ! ")
q = QuantumRegister(3)
c = ClassicalRegister(3)
circuit = QuantumCircuit(q, c)
circuit.h(q[0])
circuit.h(q[1])
circuit.h(q[2])
circuit.measure(q, c)
job = execute(circuit, backend, shots=100, max_credits=3)
print(job.result().get_counts(circuit))
```

Voici le résultat :

```
The best backend is ibmqx4
On lance le calcul sur un vrai ordinateur quantique !
{'011': 14, '101': 12, '000': 19, '100': 11, '010': 13, '111': 5, '110': 10, '001': 16}
```

Vous avez désormais fait tourner votre premier programme sur un vrai ordinateur quantique.

Pour ceux que cela intéresse, IBM propose aussi un simulateur HPC d'ordinateur quantique qui émule les vraies conditions (bruits, décohérence, etc…). Pour notre exemple, le code serait :

```
from qiskit import ClassicalRegister, QuantumRegister, QuantumCircuit
from qiskit import execute
from qiskit import IBMQ

IBMQ.load_accounts()

backend = IBMQ.get_backend('ibmq_qasm_simulator', hub=None)
q = QuantumRegister(3)
c = ClassicalRegister(3)
circuit = QuantumCircuit(q, c)
circuit.h(q[0])
circuit.h(q[1])
circuit.h(q[2])
circuit.measure(q, c)
print("On lance le calcul sur le simulateur HPC quantique !")
job = execute(circuit, backend=backend, shots=100, max_credits=3)
print(job.result().get_counts(circuit))
```

Alors, me direz-vous, informatique quantique, mensonge ou vérité ? Parce qu'il faut bien reconnaître qu'on ne va pas révolutionner le monde avec la simulation d'un jeu de pile ou face.

Conclusion

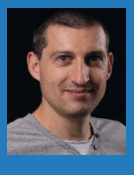
En l'état actuel, à l'aube des années 2020, l'industrie de l'informatique quantique est encore très loin de mettre sur le marché les ordinateurs disposant de milliers de qubits logiques dont on aurait besoin pour révolutionner l'informatique.

Comme toute révolution technologique, ce sont les industries à fort revenus ou valeur ajoutée qui vont être les premiers utilisateurs : les GAFAM et leurs équivalents chinois, les BATX, les secteurs de la finance, de la pharmacie, de la chimie. Ce n'est pas demain que tout un chacun aura un ordinateur quantique chez lui mais il en est de même avec les supercalculateurs de type HPC.

A l'heure où j'écris ces lignes, l'ordinateur quantique le plus puissant intègre 79 qubits (société IonQ) et dispose d'un nombre d'instructions limitées mais adressables en langages évolués comme C++ ou Python. On s'attend à des ordinateurs avec 1000 qubits en 2021.

Les experts s'accordent pour dire que le changement de paradigme sera une réalité lorsqu'on sera capable de réaliser un processeur quantique disposant de milliers, voire de millions de qubits, tout comme on sait le faire en informatique traditionnelle. La route est donc encore longue mais les progrès sont constants et les investissements importants (la Chine vient d'annoncer 10 milliards $).

Voilà qui termine cet article volontairement limité et court dont le seul but était de vous titiller les neurones. Si vous avez envie d'aller plus loin, mon ouvrage « Informatique Quantique : 60 minutes pour comprendre » est disponible sur Amazon.
N'hésitez pas à me contacter à franckf@voltanode.com. •



Emeric Martineau
Consultant DevOps à Zenika Nantes

zenika
<animés par la passion>

Débuter avec Redis

niveau 100

Redis est un stockage en mémoire de structures de données simples (chaînes de caractères, hash, listes, ensembles, bitmaps…). Il a la particularité de travailler seulement en mémoire, ce qui lui confère de très bonnes performances. Il peut toutefois faire persister ses données sur disque.

Il supporte aussi des fonctionnalités avancées comme :
- Transactions,
- Mécanismes de publication et d'inscription,
- Le scripting Lua,
- Gestion automatique du TTL sur les clefs,
- Haute disponibilité,
- Clustering.

Modes de fonctionnement

Fonctionnement en mémoire uniquement

Dans ce mode, Redis utilise seulement la mémoire. Si le serveur est arrêté, les données disparaissent.

Fonctionnement en mémoire avec persistance régulière sur le disque

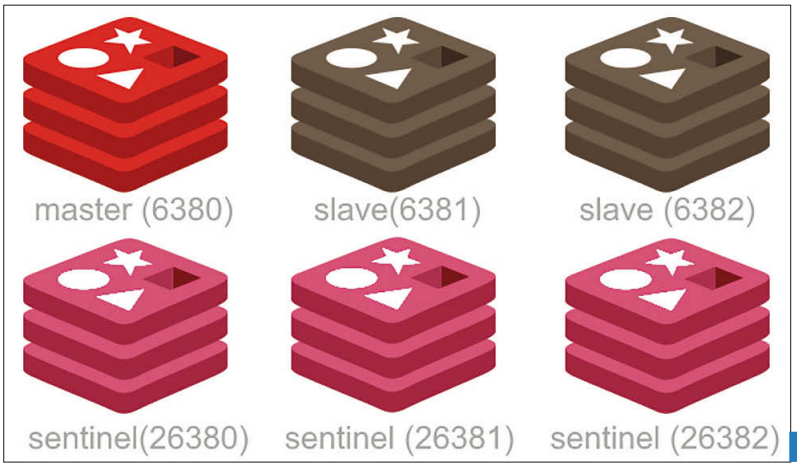
Dans ce cas, Redis qui fonctionne avec toutes ses données en mémoire, va les enregistrer régulièrement (suivant la configuration) sur le disque. Si le serveur est arrêté, les données ajoutées ou modifiées depuis la dernière persistance sur disque seront perdues. Lorsque Redis reçoit une nouvelle donnée à insérer, à modifier ou à supprimer, il incrémente un compteur. Suivant le nombre d'opérations d'écriture et le temps, Redis va faire persister sa mémoire sur disque (dans un fichier RDB).
C'est le rôle de la configuration suivante :

```
save 900 1
save 300 10
save 60 10000
```

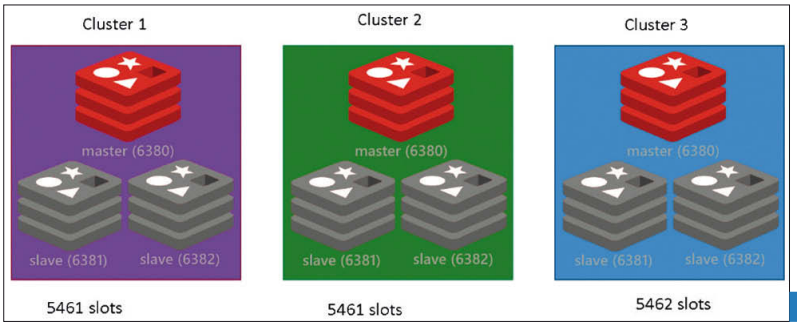
La première ligne indique de sauvegarder si au moins une clef a été modifiée dans les 15 dernières minutes. La seconde, de sauvegarder si au moins dix clefs ont été mises à jour (ajoutées, supprimées, modifiées) dans les 5 minutes. La dernière, de sauvegarder au bout d'une minute si 10000 clefs ont été mises à jour.

Fonctionnement en mémoire avec persistance régulière sur le disque et enregistrement des commandes

Afin de minimiser encore plus la perte de données, sans pour autant sacrifier les performances, Redis va enregistrer les commandes qu'il reçoit dans un fichier à part (AOF – Append Only File), en mode append.
Ainsi, si le serveur est arrêté, au redémarrage de Redis, ce dernier réexécutera les commandes afin de retrouver les données dans l'état au moment de l'arrêt.
Le fichier AOF sera régulièrement réinitialisé lorsque sa taille deviendra trop importante (plus de détails dans l'article Redis persistence demystified).



1



2

Les différentes architectures

Architecture standalone

Il s'agit d'un seul Redis. Si le serveur est coupé, plus d'accès aux données.

Architecture master/replica 1

Dans ce mode, l'application lit et écrit sur le master. Des sentinels (au moins 3), vérifient l'état du master. Si le master est défaillant, elles élisent un nouveau master. Le master se réplique régulièrement sur les slaves. Cela signifie que si le master est défaillant avant qu'il ait répliqué ses données, celles reçues depuis la dernière réplication seront perdues. Il est toutefois possible de forcer une réplication synchrone au risque de dégrader les performances.

L'architecture cluster 2

En mode cluster, Redis va répartir les données sur les différents noeuds master (sharding).
Cette répartition se fait via un hash CRC16 modulo 16384. 16384 est le nombre de slots disponibles sur le cluster (valeur fixe).

Pourquoi 16384 ? Salvatore Sanfilippo y répond dans le ticket GitHub 2576.
Si un noeud master devient défaillant, c'est un noeud slave qui lui est associé qui prend le relais (comme dans le cas du master/replica). À la différence de l'architecture master/replica, ce sont les noeuds (absence de sentinel) entre eux qui élisent le nouveau master. Ces noeuds communiquent entre eux par le port d'écoute de commande + 10000 (valeur fixe).

Passons à la pratique

Installation

Afin de faciliter l'exercice, nous allons utiliser l'image officielle Docker.

```
$ docker pull redis:5.0
```

Redis en standalone

Maintenant que nous avons l'image Docker, lancez le container :

```
$ docker container run –rm -it redis:5.0 /bin/bash
```

Pour l'exercice, nous allons lancer directement Redis avec les paramètres par défaut (sans fichier de configuration) :

```
root@c2b1c8a7a9df:/# cd /tmp/
root@c2b1c8a7a9df:/tmp# redis-server &
18:C 02 Jan 2019 13:27:09.215 # o000o000o000o Redis is starting o000o000o000o
18:C 02 Jan 2019 13:27:09.215 # Redis version=5.0.0, bits=64, commit=00000000, modified=0, pid=18, just started
…
18:M 02 Jan 2019 13:27:09.217 * DB loaded from disk: 0.000 seconds
18:M 02 Jan 2019 13:27:09.217 * Ready to accept connections
Le port par défaut est 6379.
```

Testons simplement la connexion et le bon fonctionnement de redis. On lance redis-cli sans paramètres (il prend la configuration par défaut) :

```
root@c2b1c8a7a9df:/tmp# redis-cli
127.0.0.1:6379>
```

et on crée une clef :

```
127.0.0.1:6379> set toto « coucou ca va »
OK
127.0.0.1:6379> keys *
1) « toto »
127.0.0.1:6379> exit
```

ensuite on arrête Redis :

```
root@c2b1c8a7a9df:/tmp# kill 18
18:signal-handler (1546436001) Received SIGTERM scheduling shutdown…
root@c2b1c8a7a9df:/tmp# 18:M 02 Jan 2019 13:33:21.739 # User requested shutdown…
18:M 02 Jan 2019 13:33:21.739 * Saving the final RDB snapshot before exiting.
18:M 02 Jan 2019 13:33:21.749 * DB saved on disk
18:M 02 Jan 2019 13:33:21.749 # Redis is now ready to exit, bye bye…
```

maintenant, on liste les fichiers dans le répertoire/tmp :

```
root@c2b1c8a7a9df:/tmp# ls
dump.rdb
```

On constate que nos données ont bien été sauvegardées dans un fichier. Mais uniquement car nous l'avons arrêté gracieusement avec le signal SIGTERM.
Redémarrons Redis, listons les clefs avec keys * . Insérons une nouvelle clef :

```
127.0.0.1:6379> keys *
1) « toto »
127.0.0.1:6379> set titi « BASIC is never die! »
OK
127.0.0.1:6379> keys *
1) « titi »
2) « toto »
127.0.0.1:6379>
```

Si on refait l'exercice avec une kill -s kill <pid>.
On relance le serveur Redis :

```
$ redis-server &
[1] 30
root@c2b1c8a7a9df:/tmp# 30:C 02 Jan 2019 13:47:00.716 # o000o000o000o Redis is starting o000o000o000o
…
30:M 02 Jan 2019 13:47:00.717 * Ready to accept connections
$ redis-cli
127.0.0.1:6379> keys *
1) « toto »
127.0.0.1:6379>
```

On constate que la nouvelle donnée a disparu.
Rendez-vous sur le Github Zenika pour continuer les exercices.

•

Jean-Christophe QUETIN

J'ai débuté sur MO5 avec le LOGO et sa petite tortue (un des seuls langages en français). Puis je suis passé au BASIC, plus adapté pour les jeux (le code était publié dans des magazines spécialisés). 30 ans plus tard j'ai retrouvé l'électronique et la programmation avec le Raspberry Pi et ses GPIO. J'ai eu l'occasion de travailler sur l'Arduino en collège et lorsque j'ai appris que les Éditions ENI cherchait un auteur, je me suis lancé. J'ai donc écris le livre : Arduino - Apprivoisez l'électronique et le codage (Editions ENI) qui est sorti en 2018. Et j'ai eu envie de continuer à écrire, j'ai donc créé mon blog début 2019.

arduiblog.com - twitter.com/jcquetin - www.linkedin.com/in/jean-christophe-quetin-29682a164/

Hacker une cafetière avec du Bluetooth

niveau 200

On peut facilement trouver une cafetière de type Senseo pour 50 €, neuve et entre 20 et 30 €, d'occasion. Elles sont faciles à utiliser, les dosettes ne sont pas très chères et elles permettent de faire du café à la demande. Nous allons voir comment la hacker ! Enjoy.

Cette cafetière est extrêmement simple, elle n'a que 3 boutons. Un bouton central qui permet de l'allumer et un bouton de chaque côté pour choisir la taille du café (simple ou double).

Mais avec le modèle de base, cela demande 3 étapes :

1) Mettre la tasse, l'eau, 1 ou 2 dosettes et allumer la cafetière en appuyant sur le bouton central.

Attendre presque 1 min 30 que l'eau soit suffisamment chaude.

2) Sélectionner le café désiré (simple tasse ou double tasse).

Attendre encore environ 1 min que le café coule dans la tasse.

3) Prendre le café (et le boire)

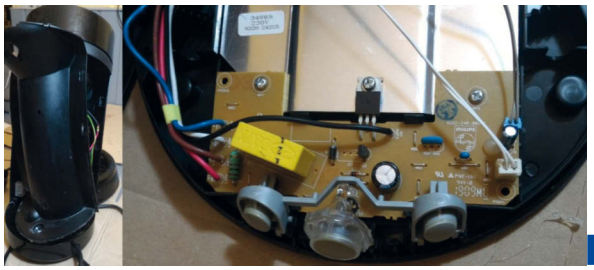
Ce serait quand même plus pratique si l'on pouvait supprimer l'étape n°2 et sélectionner son café dès le début (comme avec la Senseo Viva). Et ensuite on pourrait en profiter pour ajouter de nouvelles fonctions...

Ce tuto utilise une Senseo modèle 7820. Si vous possédez une autre cafetière, il faudra peut-être modifier un peu le code ou les branchements mais le principe reste le même.

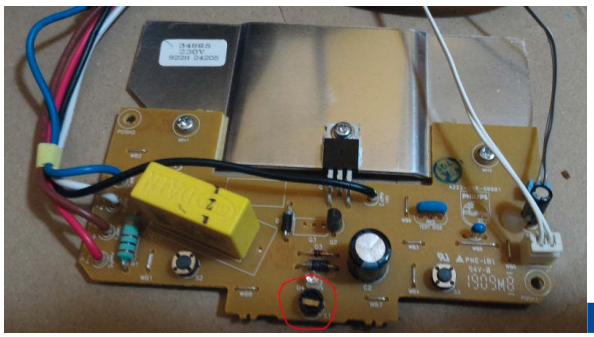
Préparation de la cafetière

Pour contrôler la cafetière, il faut simuler l'appui sur les boutons. La première étape est donc de démonter la cafetière, pour accéder à la carte de contrôle. Ensuite, il faudra repérer les contacts correspondant aux 3 boutons poussoirs et souder des fils qui seront reliés à des relais.

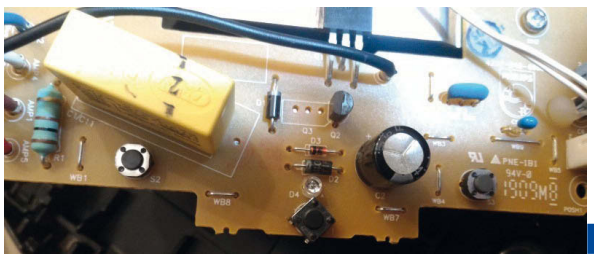
Il est évident qu'après cette manipulation, la garantie du constructeur ne pourra plus être invoquée en cas de panne de la cafetière. Je vous conseille donc d'utiliser une vieille cafetière.



1



2



3

Préparation de la cafetière

L'accès à la cafetière n'est pas extrêmement facile car il faut déjà démonter l'arrière, qui tient avec une vis torx et qui est clipsée, avant d'accéder aux 2 vis torx qui maintiennent la plaque du dessous, sur laquelle est fixée la carte de contrôle.

N'hésitez pas à consulter le guide du Repair Café, à cette adresse : https://smogey.org/wp-content/uploads/2017/09/Guide-de-réparation-Senseo-Version-4.1.1-fr-Bêta-du-6-décembre-2015-2.pdf 1

Je vous conseille de faire preuve d'un minimum de délicatesse, si vous ne voulez pas rencontrer le même problème que moi : en insérant un gros tournevis à l'intérieur de la cafetière pour déclipser l'avant qui faisait un peu de résistance, j'ai pris appui sur le bouton central, qui n'est apparemment pas prévu pour supporter ce type de traitement... 2

Heureusement, j'ai réussi sans problèmes, à le déssouder et le remplacer par un autre (carré). 3

Une fois la carte de contrôle extraite, soudez les fils sur les contacts des 3 boutons (au dos). Il suffit de seulement 4 fils car les 3 boutons sont reliés à la masse, mais si vous préférez, vous pouvez aussi utiliser 6 fils (2 x 3 boutons).

Quelques conseils : utilisez un fer à souder de bonne qualité, ne dénudez pas les fils sur une trop grande longueur et étamez-les avant la soudure. 4

Repérez bien les fils et remontez la cafetière. 5

Les fils seront ensuite branchés à un module 4 relais (Alimentation de la cafetière + 3 boutons). 6

Attention l'une des bornes est reliée à l'alimentation de la cafetière (230V).

Travaillez toujours hors tension et enfermez le module relais dans un boîtier isolé.

Vérifiez aussi que la puissance de votre cafetière (1450W, dans cet exemple avec la senseo 7820) ne dépasse pas la puissance maximum admise par vos relais (10A X 230V = 2300W). Si c'est le cas, utilisez un relais plus puissant pour gérer l'alimentation. 7

Vous pensez peut-être que l'utilisation de relais est un peu démesurée. En effet la carte de contrôle de la Senseo 7820 fonctionne en 5V (comme l'Arduino). Il serait donc possible d'utiliser de simples transistors ou mosfet (sauf pour l'alimentation).

Mais j'aimerais faire un tuto simple et universel, ou du moins qui puisse s'adapter facilement à d'autres modèles de cafetière. Les relais permettent de simuler des interrupteurs et ils acceptent jusqu'à 250V et 10A, autant dire qu'il y a de la marge (sauf si la puissance de votre cafetière dépasse 2300W). De plus, les relais sont beaucoup plus sûrs, car les circuits de la cafetière sont séparés de l'appareil qui la contrôlera, il y donc peu de risque d'endommager votre matériel.

Un Arduino avec 2 boutons et une LED

C'est la 1ère étape. L'Arduino est certainement l'une des solutions les plus simples pour contrôler les relais. Dans cet exemple, il suffit de 2 boutons pour gérer la cafetière. Le bouton n°1 pour une simple tasse et le bouton n°2 pour une double tasse. La LED n'est pas obligatoire mais elle permet d'indiquer que l'appui sur l'un des 2 boutons a bien été pris en compte, que le café est en cours de préparation et qu'il faut attendre avant de pouvoir en refaire un autre.

Matériel nécessaire

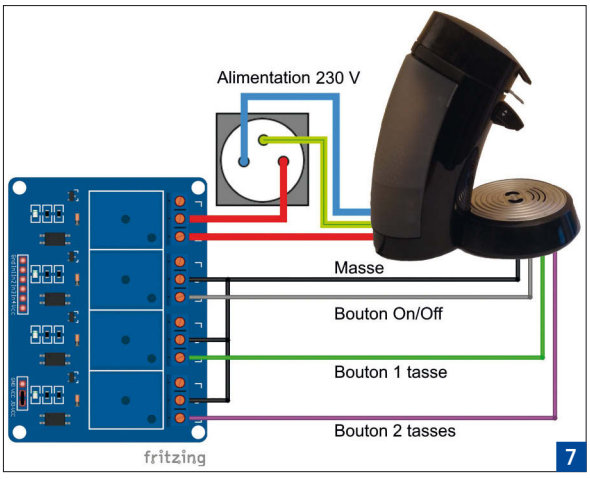
- La cafetière modifiée (avec 1 module d'au moins 4 relais)
- 1 Arduino Uno et son alimentation,
- 1 breadboard avec des câbles Dupont,
- 2 boutons poussoirs,
- 1 LED et 1 résistance d'environ 220 Ω (facultatives, car vous pouvez aussi regarder la LED interne de l'Arduino qui est reliée à la même borne). 8

Vous pouvez reproduire le schéma ci-dessous : 9

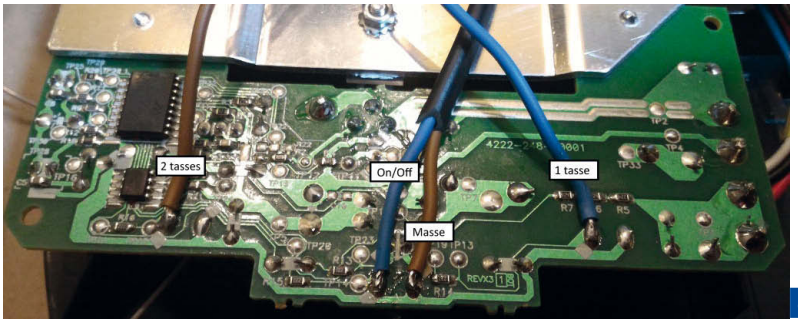
Et téléverser le code correspondant dans l'Arduino (en adaptant éventuellement le temps de chauffe à votre cafetière) :

Code complet sur programmez.com & github

Pour préparer un café, les 3 boutons en façade de la cafetière sont désormais inutiles puisqu'ils sont gérés par l'Arduino. Lorsque la

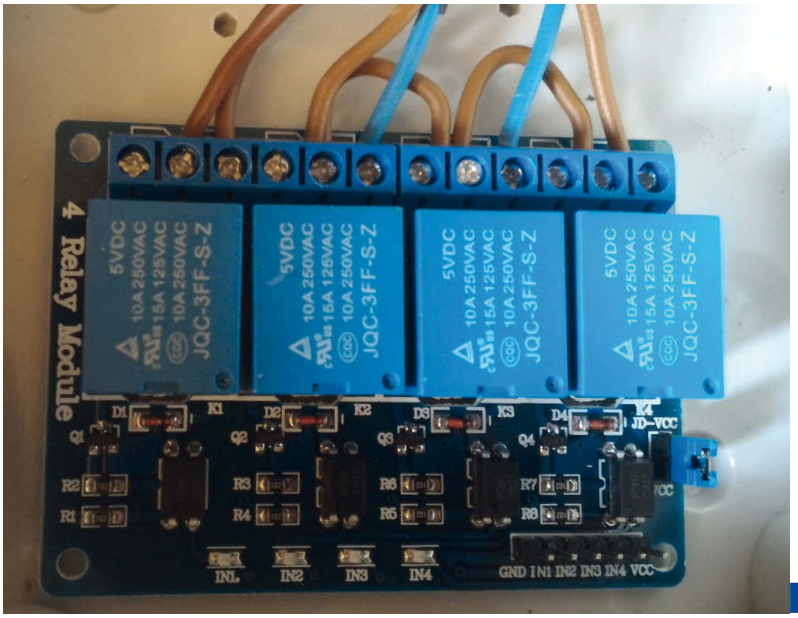


7

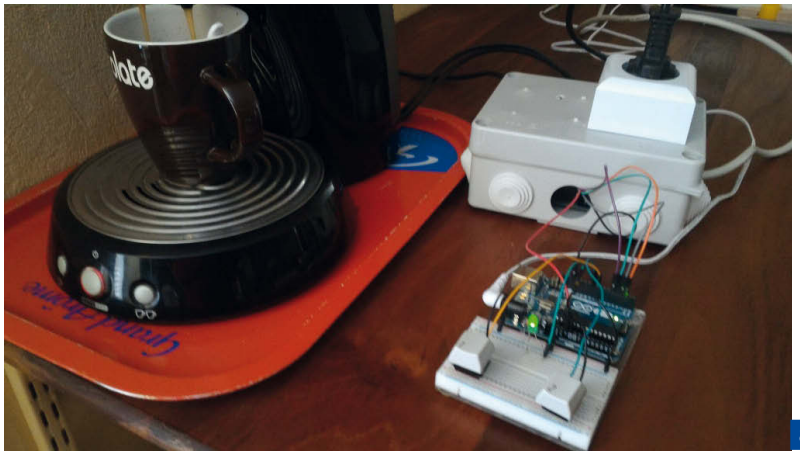


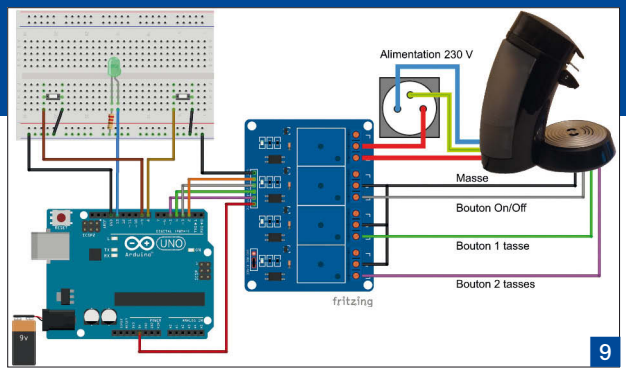
4


5

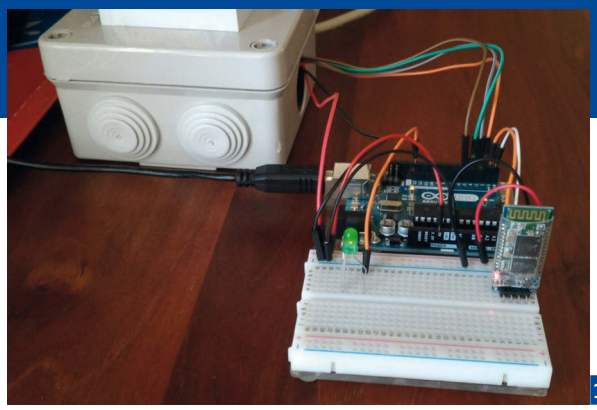


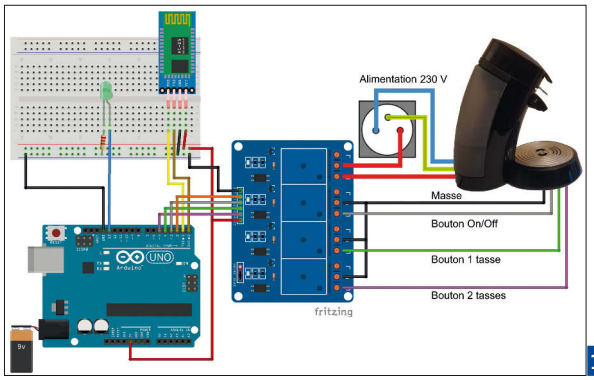
6


8



9


10



11

LED témoin est allumée en continu, il suffit de :
1) Mettre la tasse, l'eau, 1 ou 2 dosettes et d'appuyer sur un des 2 boutons de la breadboard (simple ou double tasse).
L'Arduino prend en charge l'alimentation et allume la cafetière, puis la LED clignote, l'eau chauffe, le café coule et l'eau contenue dans le réservoir se remet à chauffer.
2) Prendre le café (et le boire).
Lorsque la LED s'arrête de clignoter, vous pouvez préparer un autre café. Mais de toute façon, la cafetière s'éteindra au bout de 5 minutes d'inactivité.
Mais il faut quand même se lever pour appuyer sur le bouton de l'Arduino. Ce serait quand même plus pratique de déclencher la cafetière à distance.

Un Arduino en Bluetooth

Nous allons donc utiliser un smartphone (ou une tablette) Android pour contrôler la cafetière. Évidemment, il faudra toujours préparer la cafetière (avec l'eau et les dosettes) mais ensuite, vous pourrez tranquillement vous asseoir et activer plus tard la cafetière depuis votre fauteuil.

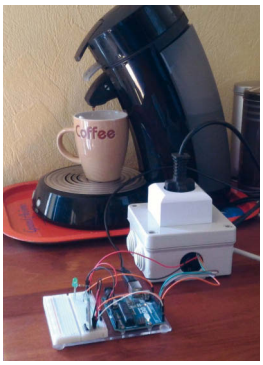
Matériel nécessaire

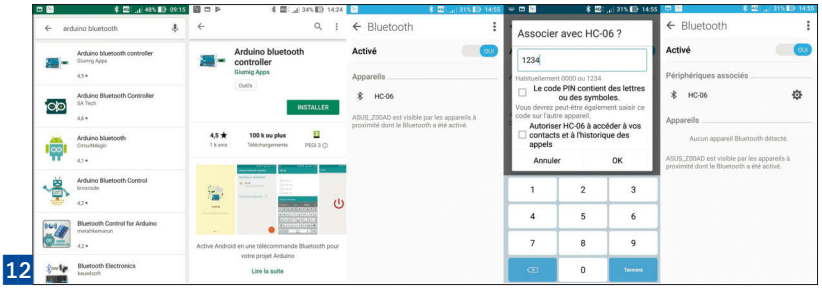
- La cafetière modifiée (avec 1 module d'au moins 4 relais)
- 1 smartphone Android
- 1 Arduino Uno et son alimentation,
- 1 breadboard avec des câbles Dupont,
- 1 module Bluetooth de type HC-05,
- 1 LED et 1 résistance d'environ 220 Ω (facultatives, car vous pouvez aussi regarder la LED interne de l'Arduino qui est reliée à la même borne). 10

Vous pouvez téléverser le code dans l'Arduino et reproduire le schéma suivant. 11
N'oubliez pas d'inverser les bornes TX et RX. C'est à dire, de brancher le TX de l'Arduino sur le RX du module et le RX de l'Arduino sur le TX du module (sinon cela ne fonctionnera pas).
Attention, le module Bluetooth utilise la liaison série. Il faut donc le débrancher pendant le téléversement du code.

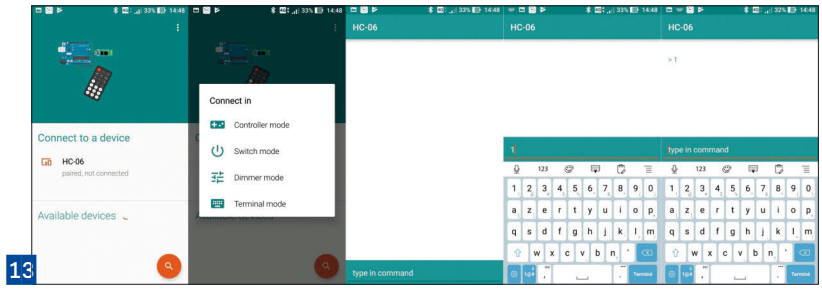
```
// Cafetiere_Bluetooth
const int ALIMENTATION = 2;
const int ONOFF = 3;
const int SIMPLE_TASSE = 4;
const int DOUBLE_TASSE = 5;
const int LED_TEMOIN = 13;
// Temps avant mise en veille de la cafetière (en minutes)
const int VEILLE = 5;
// Etat de la cafetière (0 éteinte, 1 allumée)
boolean allumageCafetiere = 0;
// Stockage du contenu du message
char message = 0;
// Moment de la derniere utilisation
unsigned long derniereUtilisation;

void setup() {
 Serial.begin(9600);
 // Initialisation des sorties
 pinMode(ALIMENTATION, OUTPUT);
 pinMode(ONOFF, OUTPUT);
 pinMode(SIMPLE_TASSE, OUTPUT);
 pinMode(DOUBLE_TASSE, OUTPUT);
 pinMode(LED_TEMOIN, OUTPUT);
 // Mise hors tension de la cafetière
 digitalWrite(ALIMENTATION, HIGH);
 // Les 3 boutons ne sont pas appuyés
 digitalWrite(ONOFF, HIGH);
 digitalWrite(SIMPLE_TASSE, HIGH);
 digitalWrite(DOUBLE_TASSE, HIGH);
 // La LED témoin est allumée
 digitalWrite(LED_TEMOIN, HIGH);
}
void loop() {
 // Réception du message en Bluetooth
 if (Serial.available()>0){
  message = Serial.read();
   switch (message) {
   // Simple tasse
   case '1':
   appuyer(SIMPLE_TASSE);
   break;
   // Double tasse
   case '2':
   appuyer(DOUBLE_TASSE);
   break;
  }
  message = 0;
 }
```

```
 // Si la cafetière est allumée et qu'elle n'a pas été utilisée depuis longtemps
 if ((millis() - dernierеUtilisation) >= (VEILLE*60000) && allumageCafetiere == 1){
  // Mise hors tension de la cafetière
  digitalWrite(ALIMENTATION, HIGH);
  // Indique que la cafetière est éteinte
  allumageCafetiere = 0;
 }
}

void appuyer(int bouton){
 digitalWrite(LED_TEMOIN, LOW);
 // Si la cafetière est éteinte
 if (allumageCafetiere == 0){
  // Mise sous tension de la cafetière
  digitalWrite(ALIMENTATION, LOW);
  delay(1000);
  // Appui sur le bouton ON/OFF pour allumer la cafetière
  digitalWrite(ONOFF, LOW);
  delay(500);
  // Relachement du bouton ON/OFF
  digitalWrite(ONOFF, HIGH);
  // Chauffage de l'eau dans la cuve
  // Ajustez ces valeurs en fonction de votre cafetière
  pause(90); // 90 secondes (1 min 30 sec)
  // Indique que la cafetière est allumée
  allumageCafetiere = 1;
 }
 // Appui sur le bouton
 digitalWrite(bouton, LOW);
 delay(500);
 // Relachement du bouton
 digitalWrite(bouton, HIGH);
 // Préparation du café et chauffage de l'eau
 // Ajustez ces valeurs en fonction de votre cafetière
 if (bouton == SIMPLE_TASSE){
  pause(70); // 70 secondes (1 min 10 sec)
 }
 else {
  pause(100); // 100 secondes (1 min 40 sec)
 }
 // Mise à jour du moment de la dernière utilisation
 dernierеUtilisation = millis();
 digitalWrite(LED_TEMOIN, HIGH);
}


void pause(int secondes){
 // Faire clignoter la LED témoin en attendant. le nb de secondes indiqué
 for (int i=0; i < secondes; i++){
  digitalWrite(LED_TEMOIN, HIGH);
  delay(500);
  digitalWrite(LED_TEMOIN, LOW);
  delay(500);
 }
}
```

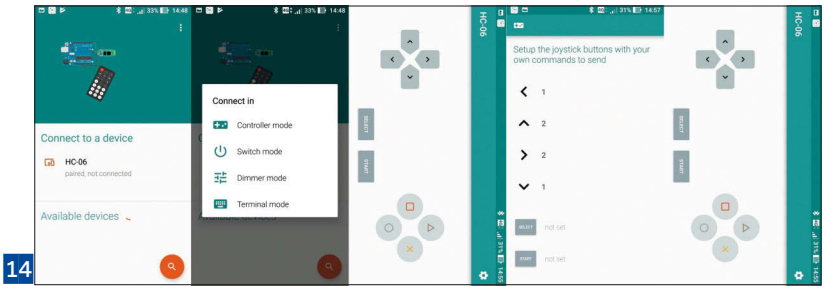




12



13



14

Contrôle de la cafetière

Pour contrôler la cafetière depuis un smartphone (ou une tablette) Android, vous devez télécharger une application qui se chargera d'envoyer les données en Bluetooth. Pour cela, rendez-vous sur le *Play Store* et recherchez les termes : "arduino bluetooth". Il existe de nombreuses applications qui peuvent convenir. Mais dans cet exemple, nous allons utiliser "Arduino Bluetooth Controller".

Après avoir installé l'application, vous allez dans les paramètres Bluetooth du téléphone. Si l'Arduino est allumé, vous devriez voir apparaître le module Bluetooth. Sélectionnez-le pour l'associer et tapez le code "*1234*". Lorsque le module apparaît dans la liste des périphériques associés, vous pouvez passer à l'étape suivante. 12

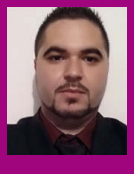
Le mode terminal

Pour le moment, la LED du module Bluetooth clignote. Lancez l'application, sélectionnez le module HC-06 et "*Terminal mode*". Le capteur ne clignote plus, il est prêt à recevoir les instructions du smartphone. Si vous souhaitez un café simple, tapez 1 et appuyez sur "*terminer*" (tapez 2 pour un café double). L'Arduino reçoit le message (en Bluetooth) et la cafetière prépare votre café. 13

Programmation des boutons

Il est possible de programmer les boutons pour qu'ils envoient "1" ou "2" à l'Arduino. Pour cela, sélectionnez "*Controller mode*" au lieu de "*Terminal mode*", appuyez sur le petit engrenage pour accéder aux paramètres. Et programmez les boutons en leur affectant les valeurs "1" ou "2". Ensuite, il suffit d'appuyer sur un bouton pour lancer la cafetière. 14

A vous de jouer maintenant.



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

Bonnes pratiques d'utilisation d'HttpClient et stratégies de résilience avec Polly dans .NET Core 2.2

Vous avez probablement déjà eu besoin d'accéder à des données distantes dans votre application .NET Core, notamment via des appels HTTP, avec HttpClient. Vous vous souvenez ? Très probablement, mais vous rappelez-vous si vous avez utilisé correctement HttpClient ? Je suis moins sûr de ça.

niveau 200

Certes, comme vous, j'ai commis des erreurs en les utilisant. Ces erreurs peuvent avoir causé des problèmes de performances. Dans cet article, je vais expliquer comment utiliser correctement **HttpClient**, mais aussi comment rendre votre application plus robuste en configurant une stratégie de nouvelles tentatives et d'interruption avec Polly.

Qu'est-ce que Polly ?

Polly est une bibliothèque .NET de résilience et de gestion des incidents transitoires qui permet aux développeurs d'exprimer des stratégies telles que de nouvelles tentatives, interruptions temporaires, Timeouts et d'autres encore de manière fluide et sécurisée. Polly cible .NET 4+ ainsi que .NET Standard 1.1 et 2.0. Polly est hautement recommandée par Microsoft, c'est une bibliothèque légère et sans dépendance qui peut fonctionner n'importe où .NET peut fonctionner. Polly a d'ailleurs éclipsé une autre librairie nommée TOPAZ.

- **Qu'est-ce qu'une stratégie de nouvelle tentative ?**
  De nombreuses erreurs sont transitoires et peuvent se corriger d'elles-mêmes après un court délai (timeout, service indisponible). De ce fait nous aimerions essayer à nouveau. Finalement une stratégie de nouvelle tentative permet de configurer des tentatives automatiques un certains nombre de fois dans un certain délai.
- **Qu'est qu'une stratégie d'interruption ?**
  Lorsqu'un système rencontre de graves difficultés, échouer rapidement est préférable à faire attendre les utilisateurs ou le système appelant. En effet, en protégeant un système défaillant contre une surcharge d'appels qui n'aboutiront pas peut l'aider à récupérer. Une stratégie d'interruption permet de configurer une coupure du circuit d'exécution pendant un certain temps, lorsque le nombre d'erreurs dépasse un certain seuil.

BONNES PRATIQUES D'UTILISATION d'HttpClient

Il y a plusieurs façons d'utiliser **HttpClient**:

- Utiliser directement **HttpClient** ;
- Utilisation d'**HttpClientFactory** avec **IHttpClientFactory** ;
- Utilisation d'**HttpClientFactory** avec des clients typés ;
- Utilisation d'**HttpClientFactory** avec des clients nommés.

Utiliser directement HttpClient

Je n'utilise jamais directement **HttpClient**, je ne le recommande pas, mais si vous voulez l'utiliser directement (et de la manière la mieux adaptée), je vous suggère fortement de suivre ces lignes directrices qui expliquent comment éviter les problèmes de performances. Maisavant cela, voici un exemple de code similaire à ce que vous avez pu faire jusqu'à maintenant et qui pose justement problème :

```
public async Task<List<Message>> GetMessagesAsync()
{
var uri = "http://www.contoso.com/messages";
    using (HttpClient client = new HttpClient())
    {
          HttpResponseMessage response = await client.GetAsync(uri);
        response.EnsureSuccessStatusCode();
        return await response.Content.ReadAsAsync<List<Message>>();
    }
}
```

Créer un **HttpClient** pour chaque demande n'est pas une bonne idée car il faudrait le réutiliser autant que possible. Mais nous devons le faire pour appeler des urls différentes. Alors que devons-nous faire ?
Eh bien observons attentivement un constructeur parmi les trois proposés d'**HttpClient** :

```
public HttpClient(HttpMessageHandler handler, bool disposeHandler);
```

Il possède deux paramètres importants, le premier est **HttpMessageHandler. E**t figurez-vous que c'est le gestionnaire de connexion ! Si vous on ne le supprime pas et qu'on le réutilise dans un nouvel **HttpClient** pour chaque requête, il résout notre problème de performances; en fait, c'est la multiple création de cet objet qui cause des problèmes ! le deuxième paramètre lui, sert à disposer ou non notre **HttpMessageHandler.** Vous aurez donc compris qu'il faille le mettre à **true.**
Réécrivons maintenant notre service en partageant notre **HttpMessageHandler** entre tous nos **HttpClient.** Considérons dans l'exemple suivant que tous nos **HttpClient** sont regroupés dans le même service. A noter que cet exemple va fonctionner correctement seulement si nous n'exécutons pas ces mêmes méthodes en parallèle avec la Parallel Task Library.
N'oublions pas également de disposer la réponse (implicitement avec **using** dans cet exemple).

```
public class MessagesServices
{
private HttpClientHandler _httpClientHandler;

    public MessagesServices()
    {
            _httpClientHandler = new HttpClientHandler();
    }

    public async Task<Message> GetMessagesAsync(int id)
    {
       var uri = $"http://www.contoso.com/message/{id}";
           using (var client = new HttpClient(_httpClientHandler, false))
        using (var response = await client.GetAsync(uri))
{
                response.EnsureSuccessStatusCode();
                return await response.Content.ReadAsAsync<Message>();
}
    }

    public async Task<List<Message>> GetMessagesAsync()
    {
        var uri = "http://www.contoso.com/messages";
        using (var client = new HttpClient(_httpClientHandler, false))
        using (var response = await client.GetAsync(uri))
{
                response.EnsureSuccessStatusCode();
                return await response.Content.ReadAsAsync<List<Message>>();
}
    }
}
```

Vous noterez que j'utilise la classe **HttpClientHandler**, qui hérite de la classe **HttpMessageHandler,** elle-même étant une classe abstraite. Voici donc comment utiliser correctement **HttpClient** quand vous l'utilisez directement. Nous pouvons nous épargner tout ceci en utilisant plutôt **IHttpClientFactory**, voyons comment cela fonctionne…

Utilisation d'HttpClientFactory avec IHttpClientFactory

Voici la description d'**IHttpClientFactory** fourni par Microsoft :

- Fournit un emplacement central pour le nommage et la configuration d'instance de **HttpClient** logiques. Par exemple, un client *github* peut être inscrit et configuré pour accéder à GitHub. Un client par défaut peut être inscrit à d'autres fins.
- Codifie le concept de middleware (intergiciel) sortant via la délégation de gestionnaires dans **HttpClient** et fournit des extensions permettant au middleware **Polly** d'en tirer parti.
- Gère le regroupement et la durée de vie des instances de **HttpClientMessageHandler** sous-jacentes pour éviter les problèmes de DNS courants qui se produisent lors de la gestion manuelle des durées de vie de **HttpClient**.
- Ajoute une expérience de journalisation configurable (via **ILogger**) pour toutes les requêtes envoyées via des clients créés par la fabrique.

Réécrivons maintenant notre **MessageService** :

```
public class MessagesServices
{
private IHttpClientFactory _httpClientFactory;

    public MessagesServices(IHttpClientFactory httpClientFactory)
    {
            _httpClientFactory = httpClientFactory;
    }

    public async Task<Message> GetMessagesAsync(int id)
    {
       var uri = $"http://www.contoso.com/message/{id}";
           using (var client = _httpClientFactory.CreateClient())
        using (var response = await client.GetAsync(uri))
{
                response.EnsureSuccessStatusCode();
                return await response.Content.ReadAsAsync<Message>();
}
    }

    public async Task<List<Message>> GetMessagesAsync()
    {
        var uri = "http://www.contoso.com/messages";
        using (var client = _httpClientFactory.CreateClient())
        using (var response = await client.GetAsync(uri))
{
                response.EnsureSuccessStatusCode();
                return await response.Content.ReadAsAsync<List<Message>>();
}
    }
}
```

Nous voilà maintenant avec une classe de clients http bien montés ! Maintenant il existe une autre manière d'utiliser **HttpClient** tout en bénéficiant des mêmes avantages concernant la performance, regardons en quoi consiste l'utilisation des clients typés et ce que cela apporte de plus que l'utilisation d'**IHttpClientFactory**

Utilisation d'HttpClientFactory avec les clients typés

Les clients typés fournissent un emplacement unique pour configurer et interagir avec un **HttpClient** particulier. Un autre avantage est qu'ils travaillent avec le système d'injection de dépendance et donc peuvent être injectés partout dans votre application. Derrière un client typé se trouve **HttpClientFactory** qui gère les instances **HttpClient** de la même manière qu'**IHttpClientFactory.**

Voyons comment notre implémentation évolue en conséquence :

Dans notre **Startup.cs,** configurons notre **HttpClient** en tant que service :

```
public void ConfigureServices(IServiceCollection services)
{
services.AddHttpClient<IMessagesServices, MessagesServices>(client =>
    {
            client.BaseAddress = new Uri("http://www.contoso.com");
    });
}
```

Comme vous le voyez, nous le paramétrons tel un service avec son interface, avec la méthode **AddHttpClient** puis nous fournissons des options, ici le **baseUrl**, cela nous évitera de l'écrire dans la classe concrète.
Maintenant jetons un œil à la classe de MessageServices :

```
public class MessagesServices : IMessagesServices
{
private HttpClient _client;
    public MessagesServices(HttpClient client)
    {
          _client = client;
    }

    public async Task<Message> GetMessagesAsync(int id)
    {
      var uri = $"/message/{id}";
       using (var response = await _client.GetAsync(uri))
{
            response.EnsureSuccessStatusCode();
            return await response.Content.ReadAsAsync<Message>();
}
    }

    public async Task<List<Message>> GetMessagesAsync()
    {
       var uri = "/messages";
       using (var response = await _client.GetAsync(uri))
{
            response.EnsureSuccessStatusCode();
            return await response.Content.ReadAsAsync<List<Message>>();
}
    }
}
```

Nous observons le retour d'**HttpClient** de manière concrète dans le constructeur, ne vous y méprenez pas il est géré par **HttpClientFactory** et instancié automatiquement, et bien disposé par le même **HttpClientFactory**.
Nous avons donc dans cette classe uniquement le nécessaire à l'implémentation du service, sa configuration se trouve externalisée dans le **Startup.cs**, ce qui est une bonne chose. Nous verrons plus bas que finalement c'est l'implémentation que nous choisirons pour implémenter des appels http résilients avec **Polly**, car la configuration suivra celle d'**HttpClientFactory** dans le **Startup.cs**.

Utilisation d'HttpClientFactory avec les clients nommés

Je ne suis pas particulièrement fan de cette implémentation d'**HttpClientFactory**. Elle n'a pas vraiment d'intérêt comparée à l'utilisation de clients typés. Je ne décrirai pas son fonctionnement dans cet article, cependant si vous souhaitez en savoir plus vous pouvez consulter la documentation de Microsoft ici : **https://docs.microsoft.com/fr-fr/aspnet/core/fundamentals/http-requests?view=aspnetcore-2.2**

STRATÉGIES DE RÉSILIENCE AVEC POLLY

Bon ! Nous avons choisi l'implémentation la plus performante et la plus flexible et paramétrable pour notre application ! Nous pouvons maintenant y appliquer notre stratégie de résilience.
Nous allons implémenter pour cela une stratégie de nouvelle tentative et une stratégie d'interruption comme je l'ai dit précédemment en introduction.
Pour pouvoir implémenter les exemples qui vont suivre il est nécessaire de télécharger le package NuGet suivant :

```
Polly.Extensions.Http
```

Dans les exemples qui vont suivre je vais employer les termes anglais *Retry Policy* pour parler de stratégie de nouvelle tentative et de *CircuitBreaker Policy* pour parler de stratégie d'interruption.
Enfin avant d'implémenter nos stratégies, nous allons avoir besoin de faire de la configuration et de créer une classe qui servira de classe de base pour chacune de nos stratégies. J'ai choisi deux interfaces plutôt que deux autres objets car je vais vous le montrer plus tard, mais je veux utiliser le plus possible l'abstraction de manière à maximiser la généricité et réutilisabilité de mes stratégies.
Ensuite je créerai une classe de base partagée par mes stratégies implémentant les règles communes à ces stratégies.

Création de la configuration des stratégies

Ma *Retry Policy* nécessite un paramètre de configuration nommé *RetryCount*. C'est le nombre de nouvelles tentatives après le premier échec qui seront exécutées avant que le *CircuitBreaker* prenne le relais.
Ma *CircuitBreaker Policy* va, elle, avoir besoin d'un paramètre nommé *BreakDuration* qui correspond à la durée d'interruption du système et également d'un *RetryCount* pour permettre le déclenchement de cette dernière après *RetryCount + 1* échecs de tentatives (1 pour la tentative initiale, *RetryCount* pour le nombre de nouvelles tentatives). Voici donc configuration fortement typée :

```
public interface ICircuitBreakerPolicyConfig
{
    int RetryCount { get; set; }
    int BreakDuration { get; set; }
}
public interface IRetryPolicyConfig
{
    int RetryCount { get; set; }
}
public class PolicyConfig : ICircuitBreakerPolicyConfig, IRetryPolicyConfig
{
    public int RetryCount { get; set; }
    public int BreakDuration { get; set; }
}
```

A ces paramètres nous allons donner des valeurs.
Dans mes propres implémentations, je choisis de mettre le plus sou-

vent 30 secondes pour la *BreakDuration* et 3 pour le nombre de nouvelles tentatives, ce qui fera en sorte que ma *CircuitBreaker Policy* se déclenchera lors de la 4ème tentative qui échouera.

```
"PolicyConfig": {
  "RetryCount": 3,
  "BreakDuration": 30
}
```

Implémentation d'une stratégie de bases communes

Ma stratégie de base est de type **PolicyBuilder**, je vais implémenter mes stratégies sous forme de méthodes d'extension en prenant ce **PolicyBuilder** comme base de construction pour fabriquer une **CircuitBreakerPolicy** et une **RetryPolicy**.

Cette **PolicyBuilder** définira simplement des règles communes à appliquer à nos stratégies. Dans notre cas, nous souhaitons appliquer nos règles sur les erreurs transitoires (5xx) et les erreurs de type timeout (408) avec la méthode **HandleTransientHttpError**(). Voici à quoi cela ressemble :

```
public static class HttpPolicyBuilders
{
    public static PolicyBuilder<HttpResponseMessage> GetDefaultBuilder()
    {
        return HttpPolicyExtensions.HandleTransientHttpError();
    }
}
```

Implémentation d'une stratégie de nouvelle tentative

Nous y voilà ! Voici comment nous utilisons la configuration **(IRetryPolicyConfig)** que nous avons définie précédemment pour élaborer notre stratégie. Nous avons également besoin d'un logger pour journaliser nos tentatives. Nous utiliserons **ILogger** pour enregistrer les tentatives avec le code http retourné, le nombre de tentatives et la durée avant la prochaine tentative. La méthode **OnHttpRetry** exécutera la journalisation. Nous devons calculer à chaque tentative la durée avant la tentative suivante : chaque tentative prend plus de temps car nous voulons maximiser la probabilité que la source distante récupère d'elle-même avec le temps, la méthode **ComputeDuration** calculera ensuite le nombre de tentatives en exposant de 2, plus une durée aléatoire comprise entre 0 et 100 millisecondes appelée *Jitter* afin de limiter les tentatives sur le serveur à l'identique pour des raisons de performances.

Voici l'implémentation de notre *RetryPolicy* :

```
public static class HttpRetryPolicies
 {
    public static RetryPolicy<HttpResponseMessage> GetHttpRetryPolicy(ILogger logger,
IRetryPolicyConfig retryPolicyConfig)
    {
        return HttpPolicyBuilders.GetDefaultBuilder()
                        .WaitAndRetryAsync(retryPolicyConfig.RetryCount,
                                ComputeDuration,
                                (result, timeSpan, retryCount, context) =>
                                {
                                 OnHttpRetry(result, timeSpan, retryCount, context, logger);
                                });
    }

    private static void OnHttpRetry(DelegateResult<HttpResponseMessage> result, TimeSpan time
Span, int retryCount, Polly.Context context, ILogger logger)
    {
        if (result.Result != null)
        {
            logger.LogWarning("Request failed with {StatusCode}. Waiting {timeSpan} before
next retry. Retry attempt {retryCount}", result.Result.StatusCode, timeSpan, retryCount);
        }
        else
        {
            logger.LogWarning("Request failed because network failure. Waiting {timeSpan} before
 next retry. Retry attempt {retryCount}", timeSpan, retryCount);
        }
    }

    private static TimeSpan ComputeDuration(int input)
    {
        return TimeSpan.FromSeconds(Math.Pow(2, input)) + TimeSpan.FromMilliseconds
(new Random().Next(0, 100));
    }
 }
```

Implémentation d'une stratégie d'interruption

Même chose ici en ce qui concerne la configuration et le logger. Créons notre *CircuitBreaker Policy* ainsi :

L'interruption se produira après (*RetryCount* + 1) tentatives infructueuses comme je l'ai indiqué précédemment.

OnHttpBreak sera déclenché lorsque l'interruption se produit et enregistrera un avertissement indiquant que l'appel http ne fonctionnera pas pendant une certaine durée équivalente à la valeur attribuée au paramètre *BreakDuration* et génèrera une défaillance d'application pour le contexte d'exécution en cours avec l'exception **BrokenCircuitException**. Nous implémentons également une autre méthode d'évènement nommée **OnHttpReset** qui enregistrera un message à la fin de l'interruption.

Voici l'implémentation :

```
public class HttpCircuitBreakerPolicies
 {
    public static CircuitBreakerPolicy<HttpResponseMessage> GetHttpCircuitBreakerPolicy
(ILogger logger, ICircuitBreakerPolicyConfig circuitBreakerPolicyConfig)
    {
        return HttpPolicyBuilders.GetDefaultBuilder()
                        .CircuitBreakerAsync(circuitBreakerPolicyConfig.RetryCount + 1,
                                TimeSpan.FromSeconds(circuitBreakerPolicyConfig.Break
Duration),
                                (result, breakDuration) =>
                                {
                            OnHttpBreak(result, breakDuration, circuitBreakerPolicyConfig.
RetryCount, logger);
```

```
                                },
                                () =>
                                {
                                    OnHttpReset(logger);
                                });
    }

    public static void OnHttpBreak(DelegateResult<HttpResponseMessage> result, TimeSpan
breakDuration, int retryCount, ILogger logger)
    {
        logger.LogWarning("Service shutdown during {breakDuration} after {DefaultRetryCount}
failed retries.", breakDuration, retryCount);
        throw new BrokenCircuitException("Service inoperative. Please try again later");
    }

    public static void OnHttpReset(ILogger logger)
    {
        logger.LogInformation("Service restarted.");
    }
}
```

Implémentation des extensions d'IHttpClientBuilder

Nous avons bientôt fini !

Nous allons maintenant créer des extensions sur l'interface **IHttpClientBuilder** pour pouvoir attacher nos stratégies à nos clients http typés. **IHttpClientBuilder** est un builder permettant de configurer les **HttpClient** gérés par l'**HttpClientFactory.**

Dans ces extensions, nous avons besoin d'un nom de section de configuration *(PolicySectionName)* pour localiser notre configuration dans le fichier de configuration **appsettings.json**, **ILoggerFactory** pour créer des enregistreurs pour chaque classe de stratégie et **IConfiguration** pour accéder à la configuration.

AddPolicyHandlers ajoute notre stratégie de nouvelle tentative et notre stratégie d'interruption et transmet la configuration respective à chaque stratégie.

Implémentation :

```
public static class HttpClientBuilderExtensions
{
    public static IHttpClientBuilder AddPolicyHandlers(this IHttpClientBuilder httpClientBuilder,
string policySectionName, ILoggerFactory loggerFactory, IConfiguration configuration)
    {
        var retryLogger = loggerFactory.CreateLogger("PollyHttpRetryPoliciesLogger");
        var circuitBreakerLogger = loggerFactory.CreateLogger("PollyHttpCircuitBreakerPolicies
Logger");

        var policyConfig = new PolicyConfig();
        configuration.Bind(policySectionName, policyConfig);

        var circuitBreakerPolicyConfig = (ICircuitBreakerPolicyConfig)policyConfig;
        var retryPolicyConfig = (IRetryPolicyConfig)policyConfig;

        return httpClientBuilder.AddRetryPolicyHandler(retryLogger, retryPolicyConfig)
                .AddCircuitBreakerHandler(circuitBreakerLogger, circuitBreakerPolicyConfig);
    }

    public static IHttpClientBuilder AddRetryPolicyHandler(this IHttpClientBuilder httpClientBuilder,
ILogger logger, IRetryPolicyConfig retryPolicyConfig)
    {
        return httpClientBuilder.AddPolicyHandler(HttpRetryPolicies.GetHttpRetryPolicy(logger,
retryPolicyConfig));
    }

    public static IHttpClientBuilder AddCircuitBreakerHandler(this IHttpClientBuilder httpClient
Builder, ILogger logger, ICircuitBreakerPolicyConfig circuitBreakerPolicyConfig)
    {
        return httpClientBuilder.AddPolicyHandler(HttpCircuitBreakerPolicies.GetHttpCircuit
BreakerPolicy(logger, circuitBreakerPolicyConfig));
    }
}
```

Appliquons maintenant nos stratégies au client précédemment décrit dans le **Startup.cs** avec l'implémentation complète de la classe **Startup** :

```
public class Startup
{
  public Startup(IConfiguration configuration, ILoggerFactory loggerFactory)
  {
    Configuration = configuration;
    LoggerFactory = loggerFactory;
  }

  public IConfiguration Configuration { get; }

  public ILoggerFactory LoggerFactory { get; }

  public void ConfigureServices(IServiceCollection services)
  {
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);

    public void ConfigureServices(IServiceCollection services)
   {
   services.AddHttpClient<IMessagesServices, MessagesServices>(client =>
          {
              client.BaseAddress = new Uri("http://www.contoso.com");
          }).AddPolicyHandlers("PolicyConfig", LoggerFactory, Configuration);
   }
  }

  public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory
loggerFactory)
  {
    app.UseMvc();
  }
}
```

Ceci clôt cet article décrivant les bonnes pratiques de développement autour d'**HttpClient** dans une application *.NET Core 2.2*. Vous savez maintenant comment éviter les problèmes de performances et augmenter la robustesse de votre application. J'espère que cet article vous sera utile dans vos futurs développements •

Vincent Rivière
Développeur à l'Université Paris 1 Panthéon-Sorbonne
http://vincent.riviere.free.fr/
https://www.youtube.com/c/Vretrocomputing

Programmation système sur Atari ST

Partie 2

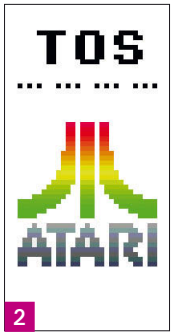
Le mois dernier, je vous ai présenté l'Atari ST et ses successeurs, de 1985 à nos jours. Aujourd'hui, je vais détailler les différentes couches du système d'exploitation qui équipe ces machines, ainsi que la manière d'y accéder depuis vos programmes. 1

1

niveau 200

Le système d'exploitation de l'Atari ST s'appelle **TOS** (The Operating System). Il vous a été présenté en détail lors du Meetup #2 de *Programmez!* en septembre dernier. Vous pouvez retrouver le diaporama en ligne [1]. Il existe plusieurs variantes du TOS :

- Le TOS original d'Atari, de la version 1 du ST à la version 4 du Falcon ; 2
- Les TOS patchés, pour les cartes accélératrices et les clones ;
- Geneva et MagiC, des systèmes d'exploitation compatibles ;
- EmuTOS, une réimplémentation libre du TOS ;
- FreeMiNT (et son ancêtre MiNT), un noyau multitâche préemptif.



2

Documentation

La documentation de référence en français est « **Le livre du développeur sur Atari ST** » de chez *Micro Application*. Hélas, ce livre est assez ancien : il est sorti en 1989 et ne couvre les fonctions du TOS que jusqu'à la version 1.2 du STf. Il existe aussi un tome 2 qui illustre la programmation système avec de nombreux exemples, et qui introduit les nouveautés du STe. Heureusement, la documentation en anglais est bien plus fournie. Le livre le plus complet est sans conteste « **The Atari Compendium** » sorti en 1992 chez *Software Development Systems*. Il couvre l'ensemble des machines jusqu'au Falcon avec son TOS 4, et même les fonctions étendues de MiNT. Mais de nos jours, c'est le site web « **tos.hyp** » [2] qui fait référence, car il est très complet et activement maintenu. Par ailleurs, je vous recommande d'aller jeter un coup d'œil au site « **Atari Document Archive** » [3] qui regroupe une masse assez considérable d'anciennes documentations.

Processeur

Tous les Atari ST sont équipés d'un processeur **Motorola 68000** ou d'un de ses successeurs. Ce processeur est relativement puissant (pour l'époque) et très agréable à programmer. Il fonctionne en interne sur 32 bits, aussi bien avec les données qu'avec les adresses. Toutefois, en externe, le bus d'adresses est limité à 24 bits. Cela autorise un espace d'adressage de 16 Mo, ce qui est très confortable. Plus tard, le TT et le Falcon seront équipés d'un 68030 permettant un adressage externe sur l'intégralité des 32 bits. Le 68000 dispose de 2 niveaux de privilèges :

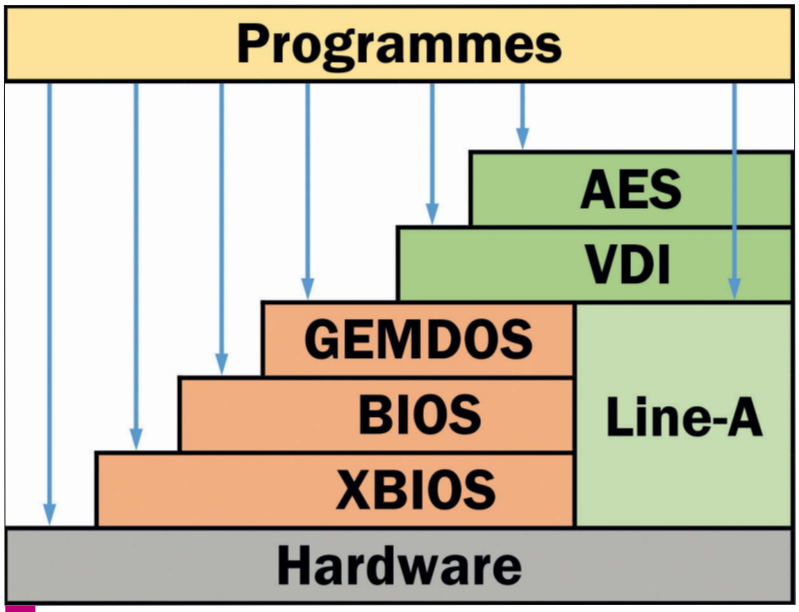
- Le *mode superviseur*, utilisé par le système d'exploitation, permet d'accéder à toutes les fonctionnalités du processeur et de la machine ;
- Le *mode utilisateur*, utilisé par les programmes, est plus restreint. Cela offre un certain niveau de protection. J'y reviendrai.

Couches du TOS

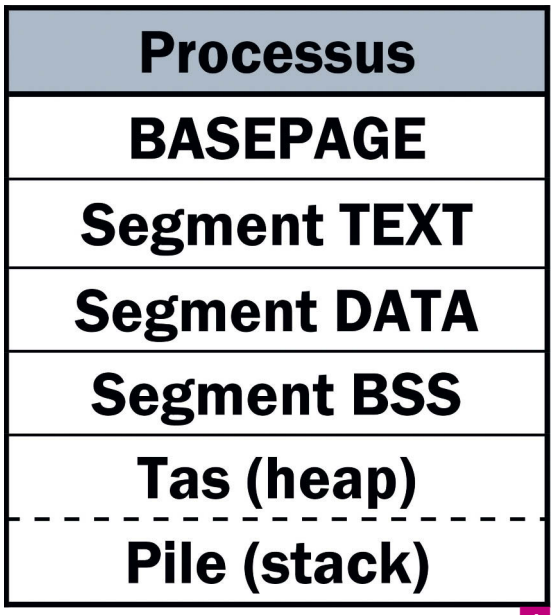
Le TOS est composé de plusieurs couches empilées les unes sur les autres. Chaque couche fait appel aux couches plus basses, et les programmes peuvent faire appel à toutes les couches 3. De bas en haut, on trouve :

- La couche matérielle (**hardware**). Bien qu'elle ne fasse pas partie du TOS à strictement parler, c'est sur elle que tout repose ;
- Le **XBIOS** (eXtended BIOS) : c'est la couche la plus basse du TOS. Elle contient des fonctions pour accéder directement au matériel : lecteur de disquettes, écran, clavier, port série…
- Le **BIOS** (Basic Input/Output System) est une couche d'abstraction du matériel sous-jacent. Elle gère des périphériques virtuels de type *bloc* (disquettes, partitions de disque dur…) et *caractère* (entrée clavier, sortie console, port parallèle…) ;
- Le **GEMDOS** (GEM Disk Operating System) est la couche de plus haut niveau du TOS (sans compter les couches graphiques). Elle gère la mémoire, les fichiers et les programmes. Le GEMDOS est très similaire à la couche MS-DOS des PC, avec les mêmes numéros de fonctions et les noms de fichiers en 8.3. Par la suite, MiNT est venu enrichir le GEMDOS avec des fonctions inspirées d'UNIX ;
- La **Line-A** : cette couche se situe à peu près au même niveau que le BIOS et contient des primitives graphiques pour ST. Mais Atari a déclaré très tôt que la Line-A était obsolète, et a cessé de la faire évoluer à partir du TT ;

[1] https://fr.slideshare.net/fredericsagez/atari-st-history-of-the-os
[2] http://toshyp.atari.org/
[3] https://www.dev-docs.org/



3 *Les couches du TOS*



4

- La **VDI** (Virtual Device Interface) est une couche d'abstraction des périphériques. Elle gère principalement des primitives graphiques pour l'écran, mais aussi pour les imprimantes grâce à l'extension GDOS. La VDI gère également les périphériques d'entrée utilisateur tels que le clavier et la souris ;
- L'**AES** (Application Environment Services) gère l'interface utilisateur : fenêtres, menus, boîtes de dialogue, événements… C'est cette couche qui donne aux applications Atari leur aspect si caractéristique ;
- Le bureau (**desktop**) est l'application fournie en ROM qui permet de lancer les programmes et de gérer les fichiers.

Ces 3 dernières couches (VDI, AES et bureau) forment l'environnement graphique **GEM** (Graphics Environment Manager).

Vous venez d'avoir un bref aperçu des couches système de l'Atari ST. Elles sont présentes dans toutes les versions du TOS et des systèmes compatibles. Nous allons voir comment y faire appel depuis nos programmes. Mais tout d'abord, il faut définir… ce qu'est un programme pour TOS.

Programmes

Un programme Atari contient principalement du langage machine pour 68000. Les fichiers exécutables se reconnaissent à leur extension, et sont de 3 types :

- **TOS** (ou TTP) : programme en mode texte.
- **PRG** (ou APP, ou GTP) : programme GEM en mode graphique (ou pas). C'est l'extension la plus répandue.
- **ACC** (accessoire) : programme GEM chargé au démarrage et qui s'exécute en multitâche coopératif avec le bureau ou le programme principal. Cela reste un cas particulier.

Tous ces exécutables ont la même structure. Ils contiennent :

- Un **en-tête** de 28 octets ;
- Le **segment TEXT** : programme en langage machine et données en lecture seule ;
- Le **segment DATA** : données en lecture/écriture ;
- Le **segment BSS** : variables globales initialisées à zéro. Ce segment n'occupe aucun espace dans les fichiers exécutables, seule sa taille est indiquée dans l'en-tête ;
- Une **table des symboles**, en option. Elle permet aux débogueurs d'afficher les labels du source à la place des adresses ;
- Et enfin, une **table de relocation**. Cette dernière permet au système de charger les programmes n'importe où dans la mémoire, et de corriger les adresses absolues en conséquence.

Lorsqu'on lance un programme, par exemple en double-cliquant sur son icône, le GEMDOS le charge en RAM et l'exécute. Plus précisément, voici ce qui se passe :

- Le GEMDOS alloue le plus gros bloc de mémoire libre, que l'on appelle ensuite **TPA** (Transient Program Area). C'est l'adresse de ce bloc qui sert d'identifiant de processus ;
- Au début de cette zone, il initialise une **BASEPAGE** (page de base) d'une taille de 512 octets. Elle contient les données du processus, telles que la taille des segments, leur adresse en RAM, la ligne de commandes…
- Juste après la BASEPAGE, le GEMDOS charge les données des segments **TEXT** et **DATA**, puis réserve l'espace nécessaire pour le segment **BSS** ;
- Ensuite, il utilise la table de **relocation** pour corriger les adresses absolues des segments TEXT et DATA afin qu'elles pointent vers les adresses réelles en RAM ;
- Puis le GEMDOS initialise la **pile** du nouveau processus (registre SP, aussi appelé A7) tout à la fin de la TPA. La zone située entre la fin du segment BSS et la pile s'appelle le « **tas** » (heap) ;
- Ceci étant fait, le GEMDOS pousse l'**adresse de la BASEPAGE** sur la pile, suivie d'une adresse de retour factice (pointeur NULL) ;
- Et enfin, il passe le 68000 en mode utilisateur, et commence l'**exécution** du programme au début du segment TEXT.
- Le programme démarre, et peut au besoin retrouver l'adresse de sa BASEPAGE en utilisant l'expression **4(sp)**.

C'est ainsi que sont chargés tous les programmes sur Atari. 4

La suite de cet article dans le prochain numéro

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 : L. Fagnon, A. Faure, M. Beugnet, N. De Loof, C. Pichaud, F. Franchin, E. Martineau, J-C Quetin, D. Duplan, A. Giretti, C. Michel, V. Rivière, A. Weinbach, T. Tiry
Couverture : © Girolamo Sferrazza Papa - **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, mai 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