JAVA | PYTHON | PAIEMENT | QUARKUS | SERVERLESS | RASPBERRY PI4

programmez.com

Programmez!

Le magazine des développeurs

232 SEPTEMBRE 2019

Interfaces en Python avec WxPython

GraalVM
La nouvelle JVM

Trop d'écoles d'informatique tuent-elles le métier de développeur ?

IA : créer et coder
une (petite) voiture autonome

Accessibilité
Comprendre WCAG

M 04319 - 232 - F: 6,50 € - RD

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

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

Olymp

La formation nouvelle génération

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

Edité par

IdeaStudio


© Valérie Tunnel

EDITO

S22ep1 : Danger, Will Robinson

Avec les 50 ans de la mission Apollo 11, nous nous sommes replongés dans cette aventure spatiale où il a fallu inventer, imaginer et coder. Au début de cette aventure, le micro-ordinateur n'existait pas, ni le microprocesseur et encore moins le disque dur, la clé USB, etc.

Prenez Voyager 1 et 2 lancés en 1977, ils possédaient 3 ordinateurs embarqués (Computer Command System ou CCS, Flight Data Subsystem ou FDS et Attitude et Articulation Control System ou AACS). Les ordinateurs fonctionnent ensemble, chacun ayant leurs tâches et opérations dédiées. Le CCS avait été le premier système à double redondance du fameux Jet Propulsion Laboratory de la NASA : tous les éléments étaient doublés. Il faut dire que le CCS est au cœur des sondes : il gère toutes les opérations et les communications avec les autres ordinateurs, il charge les programmes en mémoire, il vérifie la bonne santé de la sonde.

Côté FDS, son rôle n'est pas moins crucial : gérer les données récoltées, les stocker, collecter toutes les données de la télémétrie. Le FDS est sans doute le premier ordinateur embarqué dans un appareil spatial à avoir un processeur à mémoire volatile. Une coupure et toutes les données stockées sont perdues. Vous sentez la pression sur les concepteurs ?

Les ordinateurs de chaque sonde (qui sont identiques) peuvent exécuter 81 000 instructions par seconde. Inutile de dire qu'aujourd'hui, un Smartphone exécute plusieurs milliards d'instructions par seconde ! 69,63 Ko de mémoire vive sont embarqués. La puissance desservie dans chaque sonde pour fonctionner est de 23 watts. Une Nvidea 1080 TI tire jusqu'à 240 watts, soit 10 sondes Voyager. Il faut +20 h pour recevoir ou envoyer des données / instructions. Chaque jour, l'équipe communique…

Les logiciels développés utilisaient Fortran 5 puis la version 77. Ils furent réécrits en C. Les codes bas niveaux sont particulièrement importants, car ce sont eux qui font bouger la sonde et assurent les communications entre les engins spatiaux et la Terre.

42 ans après les lancements, les sondes traversent toujours l'espace. Voyager 1 est à environ 22 milliards de km de la Terre. Tous les équipements non vitaux, tels que les caméras, ont été éteints pour économiser l'énergie. Aucune obsolescence programmée.

Une telle durée de mission, unique dans l'histoire spatiale, pose un autre défi : la conservation et la transmission de la connaissance. Les ingénieurs et développeurs des sondes sont soit à la retraite, soit morts. Mais il faut toujours maintenir, savoir comprendre les codes et les réécrire. De temps en temps, les codes réécrits en langages modernes sont envoyés. Il y a quelques années, la NASA cherchait des développeurs Fortran et en langage assembleur pour ces sondes ☺

Bonne rentrée !

François Tonic
ftonic@programmez.com

SOMMAIRE

Les dossiers du mois

A découvrir !

Menu code du mois

Nos classiques

Dans le prochain numéro !
Programmez! #233, dès le 27 septembre 2019

Sécurité & hacking
Python : coder un Mastermind avec Tkinter
Les nouveautés de Swift 5.1

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	DOE/SC/Oak Ridge National Laboratory United States	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM	2,414,592	148,600.0	200,794.9	10,096
2	DOE/NNSA/LLNL United States	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM / NVIDIA / Mellanox	1,572,480	94,640.0	125,712.0	7,438
3	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
4	National Super Computer Center in Guangzhou China	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 NUDT	4,981,760	61,444.5	100,678.7	18,482
5	Texas Advanced Computing Center/Univ. of Texas United States	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR Dell EMC	448,448	23,516.4	38,745.9	
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 Cray Inc.	387,872	21,230.0	27,154.3	2,384
7	DOE/NNSA/LANL/SNL United States	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect Cray Inc.	979,072	20,158.7	41,461.2	7,578
8	National Institute of Advanced Industrial Science and Technology (AIST) Japan	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR Fujitsu	391,680	19,880.0	32,576.6	1,649
9	Leibniz Rechenzentrum Germany	**SuperMUC-NG** - ThinkSystem SD650, Xeon Platinum 8174 24C 3.1GHz, Intel Omni-Path Lenovo	305,856	19,476.6	26,873.9	
10	DOE/NNSA/LLNL United States	**Lassen** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Tesla V100 IBM / NVIDIA / Mellanox	288,288	18,200.0	23,047.2	
11	Total Exploration Production France	**PANGEA III** - IBM Power System AC922, IBM POWER9 18C 3.45GHz, Dual-rail Mellanox EDR Infiniband, NVIDIA Volta GV100 IBM	291,024	17,860.0	25,025.8	1,367

1

Epitech : une fuite de données pour les vacances

L'école d'informatique a été victime d'une fuite de données internes : un internaute a diffusé des documents provenant de l'intranet de l'école sur Twitter, Github et d'autres plateformes. Parmi les données concernées, on retrouvait des CV, des notes et des documents pédagogiques concernant les élèves, le personnel et les partenaires de l'école. Début août, la direction de l'école a été contrainte de repousser son départ en vacances le temps de prévenir les étudiants, la CNIL et de gérer la crise.

L'ICO : la loi c'est eux !

À quelques jours d'intervalle, la CNIL britannique, l'ICO, a menacé de sanctionner lourdement deux sociétés pour non-respect du RGPD. La compagnie aérienne britannique, British Airways, et la chaîne d'hôtellerie internationale, Mariott, risquent tous deux plus de 100 millions de Livres (£) d'amende. La raison ? Une fuite massive des données clients. Un désagrément intolérable pour l'ICO qui prendra des mesures adéquates. Côté des accusés, chacun négocie pour diminuer la sanction financière.

Cloudwatt : fin de service annoncée

7 ans après son lancement, le service de cloud Cloudwatt fermera ses portes à compter du 1er février 2020. Orange, qui avait été l'un des porteurs du projet, a averti les clients de l'offre en leur indiquant que les données hébergées sur le service seront effacées à compter de la date fatidique. Champion du cloud souverain aux côtés de Numergy, Cloudwatt avait bénéficié à son lancement de fonds publics via la Caisse des Dépôts et consignations, mais l'initiative n'avait jamais réussi à émerger face aux acteurs internationaux. Orange, devenu actionnaire majoritaire en 2015, arrête les frais.

Facebook donne naissance à Libra

Il aura fallu un an de travail pour la fine fleur des ingénieurs Facebook, mais la cryptomonnaie "personnelle" du réseau social américain est prête. Dotée d'une nature indexée, d'une capacité de réserve et d'une technologie blockchain privée, Libra est une cryptomonnaie atypique. Avec sa toute nouvelle monnaie virtuelle, Facebook compte bien bousculer l'équilibre en proposant ses services à Whatsapp, et Instagram. Une chose est sûre, les banques traditionnelles ont du souci à se faire...

Superordinateur : Total, nouveau champion français 1

Total a dévoilé son nouveau superordinateur Pangea III. Avec la bagatelle de 76 pétaoctets de capacité de stockage et une puissance de calcul pouvant s'élever à 31,7 pétaflops. La machine peut prétendre au titre de superordinateur français le plus puissant, volant ce titre au Tera-1000-2 du CEA. Dans la course mondiale des supercalculateurs, Pangea III ne décroche que la 11e place au classement TOP500 : la Chine et les États unis occupent depuis longtemps la tête du classement.

Sexo : les airpods brouillent-ils l'écoute?

Les sondages se multiplient et leur rigueur est parfois contestable, mais certains résultats font sourire. Dans un sondage mené par la société de vente de billets Tickpick sur le sexe et la musique, les auteurs de l'étude ont posé la question qui nous brûle les lèvres : combien d'utilisateurs d'Airpod les conservent lors d'un rapport sexuel ? 17% des possesseurs d'écouteurs Apple interrogés ont répondu par l'affirmative. Loin de nous l'idée de les blâmer, chacun fait ce qui lui plaît.

Le bitcoin remonte la pente

Après plusieurs mois (très) difficiles, le bitcoin est repassé cet été au-dessus de la barre des 10 000 dollars. La monnaie virtuelle connaissait depuis le début de l'année 2018 une croissance soutenue, passant de 3000 à 6000 dollars en quelques mois. La fluctuation constante du bitcoin reste un mystère, pour certains les mouvements financiers chinois pourraient en être la cause. À son plus haut pic, le Bitcoin avait atteint 17 500$. 10 ans après ses premiers pas, la monnaie virtuelle de Satoshi Nakamoto n'est pas près de disparaître.

Bye Bye Jony

Designer emblématique d'Apple, Jonathan Ive a récemment quitté la firme californienne après 27 ans de bons et loyaux services. Celui qu'on surnommait Jony a joué un rôle essentiel dans le design de l'iPhone, de l'iPad ou encore de l'iMac et ses formes arrondies. Mais pas question de prendre sa retraite ! Jony sera en 2020 à la tête de son propre studio de design. Baptisé LoveFrom, le cabinet de création restera étroitement lié à Apple dans les années à venir. Finalement ce n'est qu'un au revoir.

MEETUP PROGRAMMEZ! #6

SPÉCIAL

programmation quantique

Tout savoir ou presque sur l'informatique quantique

1er octobre à partir de 18h

Où

VillagebyCA Paris
55 rue de La Boétie
75008 Paris
Dans l'auditorium

Métros / RER :
Métro 9 & 13 : station Miromesnil
Métro 9 : station Saint-Philippe du Roule
Bus 28, 32, 80 : arrêt Saint-Philippe du Roule
Bus 52, 93 : arrêt La Boetie / Percier

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

Meetups organisés par [Programmez!] Le magazine des développeurs

Septembre

11 & 12 : Cloud Foundry Summit / La Haye

Cette année la communauté Cloud Foundry s'arrêtera à La Haye. C'est le moment où les représentations de la fondation échangent et présentent les futures évolutions de la plateforme. La Fondation dévoilera son étude sur l'utilisation des technologies Cloud et les nouveautés au sein de son écosystème. Il y aura un dîner avec les utilisateurs et la possibilité de rencontrer Abby Kearns, Executive Directeur et Chip Childers, CTO.
Site : https://www.cloudfoundry.org/event_subpages/cfeu2019-schedule/

11-14 : mini Maker Faire / Meaux

Une mini Maker Faire se tiendra au centre commercial Aushopping à Chauconin-Neufmontiers

13 : JUG Summer Camp / La Rochelle

Pour bien démarrer la nouvelle saison : http://www.jugsummercamp.org/edition/10

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'Holacratie … 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

17 : Orléans Tech Conf

L'association Orléans Tech propose de nombreux évènements Tech & Networking au Lab'O : une journée entière de conférences techniques variées (dev back & front, IoT, DevOps, Data, Cloud, …).
Site : https://orleans-tech-conf.com/

20 : We love speed / Lille

Créé en 2018 par et pour la communauté, We Love Speed est né de l'envie de partager le plus largement possible les connaissances et expériences en matière de webperf. Professionnels du web, de l'e-commerce et experts de la webperf, cet évènement est fait pour vous !
Site : https://www.welovespeed.com/2019/

21-22 : Fabrique / Rennes

Le monde maker et du DIY seront à l'honneur à Rennes. site : http://www.fabriquerennes.fr

A partir du 30 septembre : .Net Challenge 2019

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

SPÉCIAL MEETUP

Paris JUG : migration vers Java 13 / 10 septembre
Bordeaux JUG : Quarkus / 5 septembre
PUG Paris : Kotlin / 12 septembre
Machine Learning Rennes : datascience en contexte chirurgical / 16 septembre
Agile Rennes : Agile Game Breizh / 21 septembre
GDG Toulouse : Kotlin / 19 septembre
GDG Cloud & IoT Lyon : sujet sur Cloud Run / 4 septembre
GDG Cloud & IoT Lyon : sujet non connu / 14 novembre
Python Afpy Lyon : soirées prévues les 25/9, 23/10, 27/11 et 18/12
Le réacteur Paris : formation dév web & mobile / 25 septembre
Deep Learning pour informaticiens Rennes :
les packages, outils, logiciels Python du Deep Learning / 9 septembre

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.
Site : https://devfesttoulouse.fr/fr/

10-12 : Paris Web / Paris

Paris Web existe depuis 2006. Toutes les personnes qui ont assisté aux conférences et ateliers ces douze dernières années peuvent témoigner combien ça leur a apporté. Mais si vous n'avez jamais participé à Paris Web, peut-être avez-vous besoin d'être convaincu ou de convaincre vos supérieurs de vous y envoyer.

17 : DevOps REX / Paris

La conférence 100 % retour d'expérience.
Site : http://www.devopsrex.fr

21-22 : DevFest / Nantes

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

21-23 : VoxxedDays Microservices / Paris

Conférence 100 % microservices. De nombreux ateliers et des sessions techniques vous permettrons de comprendre les dernières évolutions, l'intérêt du microservice et son usage aujourd'hui.
Informations : https://voxxeddays.com/microservices/

22-26 : mini Maker Faire / Louvroil

Une mini Maker Faire se tiendra au centre commercial Aushopping à Louvroil

24-25 : Forum PHP

Pour en savoir plus : https://event.afup.org

29-31 : Scala IO / Lyon

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

DevCon #8

Technologies vocales avec Amazon Alexa

24 OCTOBRE 2019 À 42

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

2 plénières
4 sessions
+ pizza beer party

NE RATEZ PAS CETTE CONFÉRENCE DÉVELOPPEUR

Une conférence Programmez! avec la participation d'Amazon Alexa

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

Novembre

4-8 : Devoxx Belgium / Belgique

L'évènement développeur belge de l'année : 200 speakers, +200 sessions, + 3200 personnes ! Pour en savoir plus : https://devoxx.be

5 : JFIE 2019 / Paris

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

6 : DevFest / Strasbourg

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

13-14 : Microsoft Ignite Tour

Pas de Microsoft Experiences cette année mais une étape de Microsoft Ignite. Cet évènement, un des plus importants de l'éditeur aux Etats-Unis, se veut dédier aux profils techniques et aux développeurs. C'est l'occasion pour rencontrer les experts venant du campus de Seattle.
Site : https://www.microsoft.com/en-us/ignite-the-tour/
Microsoft Experiences pourrait revenir courant 2020.

21 : Codeurs en Seine

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

2020

Janvier

22-25 : SnowCamp / Grenoble

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

Février

3 : dotSwift / Paris

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

FOSDEM 2020

L'évènement incontournable des développeurs open source. C'est l'occasion de découvrir des projets et surtout d'autres développeurs. On échange, on discute, on oppose les arguments. Un lieu passionné et passionnant. L'édition 2020 n'est pas encore annoncée.
Site : https://fosdem.org/2019/

Avril

15-17 : Devoxx France / Paris

LE rendez-vous des communautés Java et de technologies. Ce sera la 9e édition !

Girls Can Code : des week-ends pour s'initier au code

Un Girls Can Code! week-end est une initiation de deux jours à la programmation informatique pour collégiennes et lycéennes. On y découvre tranquillement le langage Python, facile à prendre en main, mais très utilisé notamment par YouTube ou encore la NASA !
Les prochains dates :
- Montpellier - Google Atelier Numérique, le 18 et 19 octobre 2019
- Nancy - Google Atelier Numérique, le 25 et 26 octobre 2019

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

Grand concours Amazon Alexa !

Relève le défi et développe des Skills. Tu trouveras des exemples de Skills sur amazon.fr.

1. tu développes des Skills comprenant le support d'APL.
2. tu fais certifier ta Skill qui sera publiée sur amazon.fr.
3. tu nous envoies l'URL amazon.fr de ta Skill (ex : amazon.fr/gp/product/B07RP281WP?ref=skillrw_dsk_staffpicks-FR-35_gw_0), lors de ton inscription au concours.

Si tu n'as pas encore développé de Skills, connecte toi au site et ouvre ton compte développeur Alexa :
developer.amazon.com/fr/alexa-skills-kit/training/building-a-skill

Tu peux apprendre à développer une Skill avec Programmez! #231.

Seules les Skills supportant APL (Alexa Presentation Language) combinant la voix et les écrans sont acceptées

Les gagnant(e)s recevront de très beaux prix :

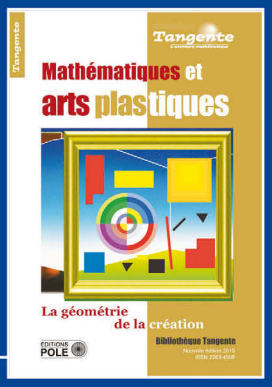
1er = Ring + echo show (valeur : 330€)
2e = Bose casque (valeur : 270€)
3e = Lenovo (valeur : 189€)

Les 3 gagnants seront sélectionnés par un jury Programmez et ZDNet.
Les vainqueurs seront annoncés durant la DevCon #8 spéciale Alexa du 24 octobre à l'école 42.

A toi de coder !

Inscription sur : **https://www.programmez.com/content/concours-amazon-alexa**

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



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.

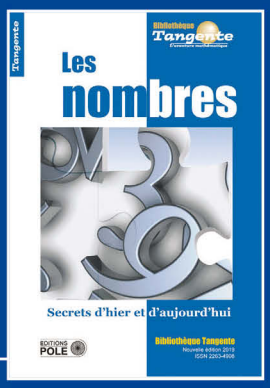
Mathématiques et philosophie

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

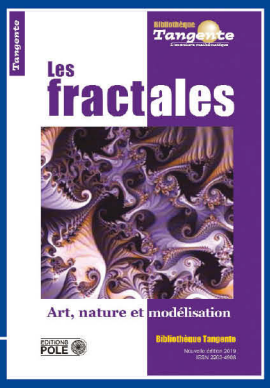
Entiers positifs ou négatifs, zéro, nombres fractionnaires, irrationnels, algébriques ou transcendants, imaginaires... Ils ont mis des siècles à s'imposer, à cohabiter. Découvrez leurs secrets.



Les nombres et leurs secrets

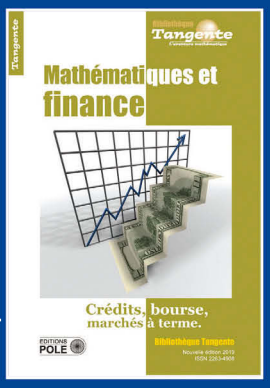
Bibliothèque Tangente
L'aventure mathématique

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 Mandelbrot, de ces magnifiques objets géométriques.



Mathématiques et finance

Disponibles chez votre libraire, ou sur Internet à l'adresse

www.infinimath.com/librairie

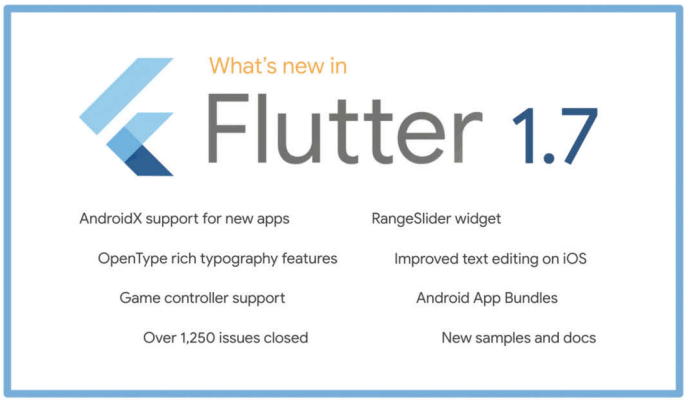
Roadmap 2019-2020 des langages & des IDE

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

DÉJÀ DISPONIBLE

Flutter 1.7

Date de sortie : disponible
Les principales nouveautés attendues / annoncées : cette nouvelle version du framework Google apporte de nombreuses améliorations et nouveautés telles que le widget RangeSlider, l'amélioration de l'édition de textes sur iOS, bundles app Android, nouveaux exemples de codes, support d'OpenType, support d'AndroidX. AndroidX est une nouvelle librairie aidant les apps Android à être à jour sur les composants en s'assurant de la rétrocompatibilité.



SWIFT 5.1

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

Kotlin 1.3.40

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

React 16.9

Date de sortie : août
Les principales nouveautés attendues / annoncées : la 16.9 est en développement depuis plusieurs mois. Durant l'été, les développeurs ont été appelés à tester les pré-versions. Parmi les nouveautés, on notera la dépréciation de UNSAFE_*, pas mal de corrections de bugs.
Pour suivre les sorties : https://github.com/facebook/react/releases

FIN 2019

Ruby on Rails 6

Date de sortie : été / automne
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 l'Action Cable testing et les tests parallèles. Pour la migration, il faudra partir d'une version 5.2 pour faire la transition. La RC2 a été publiée fin juillet avec plus de 172 changements par rapport à la RC1.

TypeScript 3.6

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

Java 13

Date de sortie : septembre - octobre
Les principales nouveautés attendues / annoncées : cette version apportera plusieurs nouveautés et améliorations. Parmi les annonces faites :

- Shenandoah : ramasse-miettes à faible temps de pause ;
- API de constantes JVM ;
- AArch64 : sert à supprimer toutes les sources liées aux arm64port. Le but est de faciliter le portage ARM ;
- archives CDS par défaut ;
- améliorations sur la GC G1.

Goland 1.13

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

C# 8.0

Date de sortie : été / automne
Les principales nouveautés attendues / annoncées : cette nouvelle évolution du langage phare .Net doit arriver avec .Net Core 3.0. Parmi les nouveautés attendues :
- Les types de références nullables doivent en finir avec les exceptions null. Pour cela, elles vous empêchent

de mettre null dans des types de référence ordinaire comme string. Par défaut, ces types seront non nullables ! Cela pourrait avoir un impact sur les codes existants ;
- Les flux asynchrones ;
- Types range et index ;
- Expressions Switch.

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

Python 3.8

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

Symfony 4.4

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

PHP 7.4

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

Ruby 2.7

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

COURANT 2020

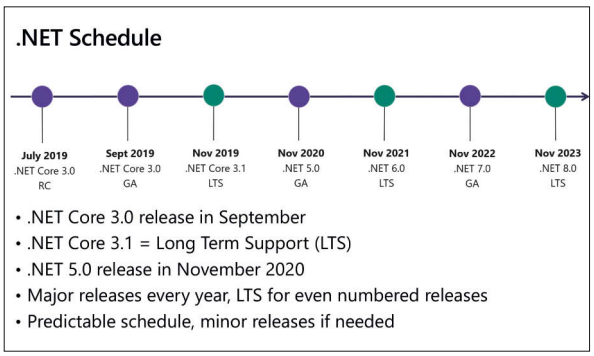
C++20

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

.Net 5

Date de sortie : novembre 2020.
Les principales nouveautés attendues / annoncées : l'annonce en a été faite à la dernière conférence BUILD. Il s'agit de .Net Core vNext, donc au-delà de .Net Core 3.0. Il s'agit de réunifier les noms. L'ambition est d'être disponible sur Windows, Linux, macOS, iOS, Android, tvOS, webassembly, etc.

Au-delà, Microsoft a annoncé une ambitieuse roadmap :



LTS

On parle souvent de LTS et de non-LTS. LTS signifie support long terme. C'est-à-dire que cette version est supportée officiellement durant x années et recevra les mises à jour et les patchs de sécurité nécessaires. Une version non-LTS a une durée de vie très courte, quelques mois. Seules les versions issues d'une forte communauté, d'une fondation, d'un éditeur sont LTS. Chacun fait un peu ce qu'il veut sur le support. Exemple : Angular est sur un cycle de 6 mois pour les versions majeures. Le support se fait sur une version.

VERSIONING

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

Raspberry Pi 4 : une belle évolution

François Tonic

On n'attendait pas la Pi 4 avant plusieurs mois même si des rumeurs commençaient à circuler ici et là. Après une Pi 3 décevante et des alternatives de plus en plus intéressantes, la fondation Raspberry se devait de réagir en proposant une carte avec de véritables évolutions.

Commençons par le positif. Reconnaissons que la Pi 4 constitue une belle avancée par rapport à la concurrence et la Pi 3.

	Pi 4	Pi 3	ODROID-N2
CPU	ARM Cortex A72 1,5 GHz 4 cœurs	ARM Cortex A53 1,4 GHz 4 cœurs	Cortex A73 1,8 GHz 4 cœurs + Cortex A53
GPU	VideoCore VI	VideoCore IV	Mali G52
décodage vidéo	H.265, 4Kp60, H.264 1080p60	H.264 & MPEG-4 1080p30	4Kp60
Mémoire vive	1, 2 ou 4 Go	1 Go	2 ou 4
Stockage	SD	SD	SD, eMMC
Sortie vidéo	2x micro HDMI supportant 4Kp60	HDMI 1.4 supportant 1080p60	HDMI 2.0
Ethernet	1 Gb	1 Gb (sur USB)	1 Gb
Réseau sans fil	WiFi + Bluetooth 5 + BLE	WiFi + Bluetooth 4.2 + BLE	-
USB	2xUSB 3 + 2xUSB 2	4xUSB 2	4xUSB 3
GPIO	40 broches	40 broches	25 broches
Alimentation	USB-C / PoE	USB / PoE (via hat)	dédiée
OS par défaut	Raspbian	Raspbian	Linux
Prix	35, 45, 55 $ (selon la RAM)	35 $	79 $

Si on compare, basiquement, la Pi 4 avec la Pi 3, en vert dans le tableau, il n'y a aucun doute : la Pi 4 améliore la plupart des éléments. Le plus important concerne le GPU, l'USB 3 qui arrive enfin, une mémoire vive capable d'exploiter les processeurs 64 bits, un vrai Ethernet 1 Gb. Le CPU connaît une mise à jour, mais plus discrète même si on y gagne, notamment pour les opérations lourdes. Sur la mémoire vive, on a le choix entre 1, 2 et 4 Go. Dommage ne pas avoir supprimé la version 1 Go pour garder uniquement deux choix : 2 ou 4 Go et démarrer à 35 $ pour le tarif. Deux nouveautés importantes sont à noter :

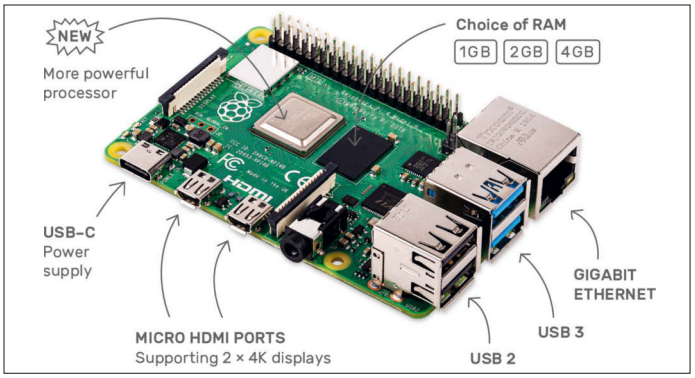
- L'apparition de l'USB 3 ! Oui, c'est une bonne nouvelle. Cela va booster un peu les performances des périphériques. Pi rapproche enfin son retard sur la concurrence
- L'Ethernet devient véritablement 1 Gb. Sur la Pi 3, il utilisait le bus USB ce qui ralentissait considérablement les débits

Toutes les améliorations (CPU + GPU + RAM) vont permettre de mieux supporter les vidéos haute définition de type 4K et donner plus de fluidité avec les jeux.

Un drôle de HDMI

Une nouveauté a beaucoup fait parler d'elle : les deux connecteurs micro-HDMI pour connecter 2 écrans. Honnêtement, nous sommes assez dubitatifs sur l'intérêt :

1. Le format adapté est vieillissant et il oblige à avoir un adapteur micro-HDMI. Mais il n'était pas possible de mettre du mini HDMI, car le connecteur prend plus de place
2. Deux écrans pourquoi pas, mais attention à la qualité de l'affichage en 4k, car cela oblige à stresser les ressources.



Nous aurions préféré un seul connecteur, mais dans un format classique.

Sur la partie alimentation, désormais, la Pi 4 utilise le format USB-C. Vous devrez donc avoir un bon adaptateur 5V / 3A (ou plus). On peut utiliser le PoE (Power over Ethernet) qui est géré par défaut, mais il faut toujours une PoE Hat.

Si le format de la carte reste quasiment identique, à quelques mm près, l'agencement des composants a changé. Vous devrez donc racheter, ou imprimer, de nouveaux boîtiers. Sur la partie GPIO, hat, pas de changement.

Sur la partie stockage, la Pi 4 reste toujours à la traîne. On doit se contenter des cartes mémoires SD, pendant que des cartes alternatives proposent un eMMC offrant des performances honorables. Le stockage est pour nous la faiblesse de la Pi. Pensez à utiliser la dernière version de Raspbian.

Performances

Nous avons procédé à quelques tests pour charger la carte. Le premier exemple est OpenArena avec une résolution de 800x600 :

- Pi 3 : à peine 5 fps
- Pi 4 : jusqu'à 40, en usage entre 25 et 35

OpenArena est véritablement fluide sur la Pi 4. La mise à niveau CPU + GPU fait merveille. Sur la Pi 3, l'intérêt du jeu est limité.

Autre test avec GLXGears pour la partie OpenGL. Là c'est un peu l'inverse. Sur la Pi 3, nous monterons jusqu'à 203 fps, mais seulement 59 en Pi 4. Problème d'optimisation du bench.

Côté sysbench, en stressant à 20000, avec 4 threads, on affiche :

	Pi 4	Pi 3
Temps total	99,3723 s	159,1891 s
Temps total pour exécution	397,4417	636,6817

La Pi 4 est incontestablement plus véloce, l'évolution CPU est donc la bienvenue, la gain n'est pas négligeable. Après, attention, pas de miracle contre des CPU plus puissants. La Pi 4 est un excellent PC d'entrée de gamme, pour un barebone sans prétention, avec un peu de jeu (OpenArena est sanglant).

Oui, on achète !

Autant la Pi 3 nous avait déçue, autant la Pi 4 nous convainc avec de vraies nouveautés et une mise à jour globale de la carte. Malgré quelques réserves sur l'intérêt de la version 1 Go et du lent et vieux port SD, pour le reste, pas grand-chose à reprocher. Le double port micro-HDMI est un choix étonnant. C'est une belle version qui permet à la Pi de se rapprocher d'une bonne partie des cartes concurrentes telles que les ODROID qui proposent de très belles choses (voir tableau et les éléments en rouge). Pour tenir les tarifs, la fondation a fait des choix.

Les +

- Jusqu'à 4 Go de RAM
- Ethernet 1 Gb non bridé
- USB 3
- Bluetooth 5
- USB-C pour l'alimentation
- CPU

Les –

- Changement de boîtier obligatoire
- micro-HDMI
- Intérêt du double HDMI
- Port SD
- Version 1 Go peu utile

François Tonic

Ecoles informatiques et formations vont-elles tuer le développeur ?

Partie 1

On ne compte plus les nouvelles écoles et formations en informatique, aux métiers du numérique et bien entendu, pour devenir développeur. On en parle partout : on manque de développeurs, on manque de compétences en informatique. Nous avons déjà eu l'occasion de revenir sur cette vision de marché qui mise plus sur la quantité que sur la qualité. On promet aux nouveaux venus un job et une employabilité immédiate.

Avertissement : nous avons posé les mêmes questions aux formations et aux écoles. Nous donnons dans cet article notre sentiment sur la situation et comment nous observons le marché. Les langages cités sont des exemples

En préparant ce dossier, plusieurs questions sont rapidement apparues :

- Quelle utilité d'ouvrir toujours plus d'écoles et de formations informatiques ?
- Comment vivent ces écoles ? Vont-elles toutes (sur)vivre dans la durée ?
- Comment vont-elles recruter des candidates/candidats (jeunes, réorientation professionnelle, etc.) ?
- À quoi peut-on prétendre à la fin de la formation ?

« Il y a une différence entre apprendre à coder et être développeur. On apprend à coder, on apprend un langage pour vivre. En 6 mois, la personne va apprendre React, si on lui demande de se mettre à Python, elle sera perdue et ne saura pas faire. » attaque d'emblée Nicolas Bouteillier (développeur).

Cette réflexion était-elle aussi abrupte qu'il y paraît ? Quand on prend une formation de 3, 6 ou 8 mois, généralement, on se concentre sur quelques technologies/langages et on apprend à coder dessus même si on n'a aucune base en programmation. C'est la promesse de nombreux cursus courts.

Nicolas Sicard (directeur des études à l'EFREI) analyse ainsi la situation : cette multiplication est la traduction d'un besoin, d'une demande des entreprises et des personnes. Signe aussi de la bonne santé du numérique et de l'informatique en général. Naturellement, les choses se stabiliseront même si aujourd'hui, on a l'impression que ça part dans tous les sens. Les besoins (des entreprises) sont forts.

Mais est-ce pour autant que l'on devient réellement développeuse/développeur ? Pour Nicolas Bouteillier, il faut distinguer deux choses : apprendre à coder et être développeur. Le développeur possède, normalement, les bases de la programmation, il a appris C++(ou 1 autre langage), l'algorithme, utilise les bons outils. Il possède une base théorique en plus de la pratique. Un développeur, en connaissant les fondamentaux, saura être réactif pour découvrir une nouvelle techno, un nouveau langage. Dans le cas d'une formation courte, pas sûre que la personne apprendra ces bases. Nous ne disons pas qu'il sera un mauvais codeur, mais il aura sans doute une base technique moins forte sauf à l'étoffer personnellement.

Le fait de se focaliser sur des technologies ayant une forte demande n'est pas à blâmer : le marché dirige et on répond à la demande. Mais il ne faut pas enfermer les personnes dans ces technologies, car par définition, elles vont changer dans les 3-5 ans. Le monde informatique évolue : un langage, une techno que tout le monde cherche ne le sera pas en 2021 ou 2022. Se pose alors la question de la pérennité des développeurs formés.

Développeur : un profil à 2 vitesses ?

Le métier de développeur se transforme-t-il en métier à plusieurs vitesses ? C'est d'ores et déjà une réalité. Nous avons d'un côté les développeurs passionnés, avec une solide base technique, qui ont l'art du code, l'envie de toujours aller plus loin et de l'autre, une variété de profils. On peut citer : on fait le job, et même bien, durant les heures de travail, mais c'est tout, juste un travail alimentaire pour d'autres (je travaille et j'ai le salaire à la fin du mois), un domaine qui recrute où je me forme rapidement sans rien y connaître et plutôt par opportunisme. Le développeur n'échappe donc pas à ce que l'on trouve dans tous les autres métiers : ceux qui aiment le métier et les autres aux motivations très diverses.

Pour une entreprise, une ESN, etc., il faut pallier les manques de compétences à court terme. Comme il y a une forte demande de recrutement, on forme. CQFD. Un problème peut se poser, outre la compétence réelle : le turn-over. Si le développeur est pérenne dans l'entreprise, il apporte sa compétence, son savoir-faire. Pour une entreprise, ce sont des connaissances importantes. Un turn-over élevé signifie perdre de la connaissance et le risque est d'autant plus vrai avec les développeurs ayant plusieurs années dans l'entreprise. En ESN/intégrateur, la question se pose autrement, car les développeurs/consultants sont en mission pour x mois et le turn-over y est historiquement élevé.

D'autre part, ce sont aussi de grands consommateurs de ressources. De nombreux développeurs arrivant sur le marché y trouveront leur premier poste.

> « *La pression est forte pour pouvoir recruter d'où la multiplication de formations courtes de -1 an.* »

« Je le constate en Master, les étudiants ont le temps d'apprendre les bases. Je vois clairement les profils passionnés et pour celles et ceux pour qui c'est juste un job. Pour moi, il n'est pas possible qu'en 6 mois, on soit performant et efficace. On forme du dév pour pisser du code. Même si aujourd'hui, le dév fait moins de code qu'avant. Il existe de très bonnes formations et de très bons formateurs. S'il le peut, au formateur de faire apprendre les bases, de donner la passion du code. » commente Nicolas Bouteillier.

La nécessité de sensibiliser ces développeurs à la veille technologie, de se former, est crucial. Mais

nous avons déjà constaté que cette partie du métier de dév est parfois peu considérée ou tout simplement, totalement absente. Mais un dév qui ne se forme pas est un dév qui deviendra obsolète. « *Dans 2 ans, on passera peut-être à autre chose que Angular ou JS. Soit le dév se forme, soit il quitte le job* » enfonce Nicolas Bouteillier.

Certains secteurs sont exigeants sur les compétences, notamment les secteurs bancaires et des jeux. Comme nous l'a précisé Aude Barral de Codingame, parallèlement au marché tendu du recrutement, des formations « alternatives » sont apparues. Durant des dizaines d'années, on avait le modèle classique de recrutement : ingénieur, 1-2 ans d'expérience. Aujourd'hui, il faut élargir les critères et savoir recruter d'une autre manière (organiser des challenges de programmation, évaluer les compétences, etc.) et ne pas se focaliser (que) sur le diplôme qui reste un garde-fou pour beaucoup d'entreprises. Mais il faut aussi éduquer les entreprises et démystifier le diplôme et le profil de développeur. Car il est possible de trouver des profils potentiels en dehors du parcours classique. Le recrutement par la compétence est donc un élément crucial en faisant passer des tests de compétences techniques par exemple. D'autre part, il ne faut pas hésiter, à favoriser la formation interne, la progression, voire, la mobilité géographique et, quand c'est possible, le télétravail.

Comme nous l'a précisé Aude, il arrive que l'on fasse miroiter beaucoup de promesses : 100 % d'embauche à fin (de la formation), des salaires élevés. Mais attention au retour à la réalité.

> « *Ce n'est pas en 6 mois que l'on maîtrise un métier* »
> (Nicolas Bouteillier)

Il faut avoir des candidats !

On peut regarder deux éléments :

- La multiplication des formations et cursus dédiés provoque une concurrence
- Le coût de la formation

« *Oui il faut arriver à attirer. Nous cherchons à recruter des étudiants passionnés et motivés. Je pense que nous n'attirons pas tous les mêmes (profils). Nos étudiants sont plus généralistes et notre parcours l'est aussi.* » évoque Nicolas Sicard (EFREI).

Le coût n'est pas à négliger. Par exemple, pour une formation de 18 semaines (avec 1 mois de stage), le coût s'élève à 5 000 €. Ce montant n'est pas neutre et il faut pouvoir le financer. D'autres proposent des formations de 10 semaines ou 25 semaines (selon temps plein ou partiel) pour 5 900 €. Pour les formations longues, les cursus classiques, les tarifs peuvent atteindre 8 000 €/an. Quand le cursus s'étale sur 2, 3, voire 5 ans, le total peut faire peur. Bien entendu, il existe des solutions de financements : l'alternance, les prêts, les CIF, Pôle Emploi, etc. Cependant, il faut avoir conscience du coût et des capacités de financement que vous pouvez avoir.

Pour toutes ces formations, l'équation n'est pas forcément simple. Car il faut proposer une gamme de formations (courtes, longues, alternance, différents profils, etc.) et avoir des candidats. C'est un peu le nerf de la guerre : les candidats. Or, avec la concurrence de plus en plus vive, il faut pouvoir attirer en nombre suffisant. C'est aussi pour répondre aux attentes différentes que l'offre s'étoffe : 100 % alternance, cours en ligne, formation courte de quelques mois, cursus classique (de 2 à 5 ans), spécialisation dans le web et mobile, etc.

Une partie des candidats ne souhaite pas une formation longue. Ils veulent plutôt être rapidement en entreprise. On l'a constaté pour les réorientations professionnelles, les jeunes, ou des plus âgés, sans diplôme ou avec.

Le fait que l'informatique recrute incite des milliers de personnes à s'y diriger, car il y a la possibilité d'un job. Il faut avoir conscience que tout le monde n'est pas adapté à nos métiers du code ni plus globalement à l'IT. Aux formations/écoles de mettre en garde. Nous avons des jobs exigeants où le travail personnel joue un grand rôle.

Pour les écoles, même si la concurrence est vive, nous ne voyons pas de changements notables. Au lancement de 42, il y avait eu beaucoup de remous et de critiques, mais finalement, le paysage n'a pas fondamentalement changé sauf à adapter la pédagogie et mettre en avant l'approche projet et l'alternance.

> « *Job passionné vs job alimentaire* »

Regardez aussi ce qui est délivré à la sortie de la formation. Le diplôme ou la certification ne fait pas de bons développeurs, mais en France, le diplôme reste souvent la première étape pour avoir un poste. « Le diplôme reste une assurance qualité. » précise Nicolas Sicard. Le diplôme est un cadre, il faut regarder la qualité de la formation. Les écoles ayant un grade d'ingénieur sont auditées et répondent à un cahier des charges.

Quand un(e) étudiant(e) paie plusieurs milliers € par an et qu'une école cherche à faire du chiffre, existe-t-il une tentation d'acheter le diplôme ou de délivrer des diplômes avec un niveau technique plus faible ? Question délicate, mais légitime. Pour une école le risque est réel de voir ses diplômés « black-listés » et de facto, l'école elle-même.

Une différence Paris – province ?

L'offre est plus nombreuse en île de France qu'en province. C'est une réalité. Cependant, nuançons, car de nombreuses écoles ouvrent : que ce soit des créations locales ou des émanations de groupes. Par exemple, l'EPSI possède 10 campus dans toute la France, l'Epitech en possède 13.

Selon Aude Barral, la différence Paris – Province est toujours là, sur les salaires, le différentiel peut atteindre 20 à 30 %. Cet écart n'existe pas, ou très peu, dans les métropoles actives (Grenoble, Montpellier, Bordeaux, Nantes, Lyon, Toulouse, etc.). Certaines régions sont plus dynamiques que d'autres, notamment au niveau emploi. Pour de l'alternance, le tissu économique est primordial.

Si certaines régions/métropoles possèdent un solide attrait, d'autres peinent à intéresser. À plusieurs reprises, nous avons demandé à de jeunes développeurs si se déplacer à Clermont-Ferrand pourrait les intéresser : l'enthousiasme n'a pas été au rendez-vous. Au mieux : quelques jours par semaine pour une durée de quelques mois. Au pire : non, pas question ! Dommage car la région cherche à dynamiser les technologies. Et elle ne manque pas d'intérêt.

Trop d'écoles et de formations ?

Ce serait une conclusion facile. Cette multitude profite d'une demande qui explose. Est-ce simplement cyclique ou durable ? Nous ne le savons pas. Le marché se retourna. Et là, la pérennité sera mise à l'épreuve. •

Suite du dossier dans le numéro 233.

SOFT SKILLS : le truc à ne pas oublier

Au-delà des diplômes et des compétences techniques, on parle beaucoup de soft skills. On peut traduire cela par compétences douces, qualités humaines/personnelles, un certain savoir-faire, compétences humaines. Cela va au-delà des centres d'intérêt. Problème : c'est un terme fourre-tout.

En vrac, on peut y trouver : la communauté, le leadership (un grand mot qui veut tout dire et ne rien dire), l'empathie, l'esprit d'équipe, la curiosité, la résolution de problème, gérer la pression, la flexibilité/adaptabilité, le volontarisme, créer de l'initiative, etc.

Questions-réponses avec Djamchid Dalili (3W Academy)

Djamchid Dalili est le fondateur de la 3W Academy, école dédiée au web et au mobile. Il était précédemment directeur général de G7 Taxi Service. Il était aussi en poste chez Alcan et Arcelor.

Depuis quelques années, les écoles (cursus complet / alternances), les centres de formations sur les métiers de l'informatique et notamment le développement se multiplient. Avec la promesse de trouver un emploi derrière. Quel est votre sentiment sur cette situation ?
On estimait en 2012, année de la création du bootcamp 3W Academy, qu'il manquait 50 000 développeurs. Sept ans plus tard, malgré la multiplication des filières qui ont copié la 3W Academy ou ont proposé des offres différentes, on estime qu'il manque 100 000 développeurs. La 3W Academy a formé 4000 développeurs à ce jour et continue d'en former 1000 par an. Cela ne suffit pas à couvrir la demande. La raison est évidente, nous assistons à un bouleversement complet des process de consommation, de vente, de production et de gestion. Tout s'achète, se vend et s'organise via le web. Et derrière chaque service qui est proposé sur le web ou migré vers le web, il y a un développeur. La demande est forte, et ce sont des compétences qui ne s'enseignaient pas il y a dix ans. Il y a donc une pénurie de ressources. Et cette pénurie est durable.

Finalement, trop d'écoles / formations ne tuent-elles pas l'école informatique avec une offre pléthorique et le risque d'être peu visible ? Et surtout, avec cette concurrence, comment faites-vous pour recruter car sans étudiants, pas d'école ?
La 3W Academy a créé le premier bootcamp français de code. Nous sommes les inventeurs et les initiateurs du système. Du coup Google est notre ami : nous recevons 60 candidatures spontanées par jour. Vous avez raison, beaucoup d'écoles ont été créées, mais très peu ont réussi à survivre, beaucoup vivotent. La 3WA est le leader du marché, nous n'avons aucun problème pour avoir des candidats. Et nous accélérons le pas. Vous avez peut-être vu dans la presse que nous avons acheté GT'M Ingénierie, leader de la formation Java. GT'M forme 300 développeurs Java par an, 300 développeurs qui sont pré-embauchés par les grandes entreprises clientes de GT'M. Nous complétons ainsi l'offre : PHP et Java, PME et Grandes Entreprises, Bac+2 et Bac+5, sur toute la France.

Combien coûte le cursus à (école) ? Qui paie quoi ?
J'ai créé la 3W Academy pour que la formation soit accessible au plus grand nombre de talents. Le prix a été calculé en fonction de ce critère. Nos étudiants payent 3400€ pour une formation de trois mois à plein temps en petits groupes de 15 personnes encadrés par un formateur du métier.

Avez-vous le sentiment que l'on mise plus sur la quantité que la qualité ? Est-ce valable dans toute la France ou selon les régions ?
On a besoin de former toute une génération, oui cela fait de la quantité. On a besoin de mettre le pied à l'étrier à un maximum de personnes. La qualité vient ensuite avec l'expérience en entreprise et les formations complémentaires. Ce n'est que de cette façon que nous aurons la qualité. C'est pourquoi nous avons introduit depuis deux ans déjà des formations de niveau Bac+4, accessibles aux personnes qui ont fait le bootcamp junior qui est de niveau Bac+2.

Ne fait-on pas (trop) rêver le jeune, l'étudiant ou la personne qui se réoriente ? Oui l'informatique embauche mais pas tous les profils ni toutes les compétences ? Un choc culturel pour certain(e)s ?
Tout ce que vous dites est exact. Il faut rêver pour entreprendre, il faut accepter de "pivoter" si le rêve ne se concrétise pas comme espéré. Il faut savoir échouer. Le garde-fou, c'est de ne pas gâcher de talent. Cet état d'esprit, rêver-savoir appuyer sur l'accélérateur-savoir pivoter-savoir échouer-savoir rebondir, c'est l'état d'esprit des startups et des entrepreneurs qui réussissent. Pour devenir développeur, il faut aussi épouser cette culture. Un diplômé d'école de commerce qui a fait un bootcamp de code et qui travaille ensuite en web marketing, ce n'est pas un codeur raté, c'est un jeune qui a pivoté deux fois et qui réussit mieux que s'il était resté sur le chemin que ses parents lui avaient tracé il y a 20 ans.

A la 3W Academy, quels sont les focus que vous mettez en avant ? Quelle est la part du travail personnel ? Comment motiver pour que les étudiants soient curieux car nos métiers changent vite ?
A la 3W Academy, la classe est inversée. Le prof n'est pas là pour faire un cours magistral, youtube le fait mieux. L'étudiant n'est pas là pour écouter passivement, il peut le faire devant youtube! L'étudiant est là pour coder, le prof est là pour coacher-déboguer-synthétiser. A la sortie de la formation, l'employeur s'attend à ce que le développeur soit autonome et efficace. Il faut donc savoir chercher l'information, en se posant la bonne question. Stackoverflow est notre ami, apprenons à lui poser les bonnes questions.

A la sortie de votre formation, à quoi peut prétendre la personne ? quelles sont ses compétences ?
A la sortie du bootcamp junior, on doit savoir coder des applications simples en toute autonomie, pour être une force productive en entreprise et pas une charge. Le seul objectif est d'accumuler de l'expérience pour être tout à fait à l'aise au bout d'un an. Au bout d'un an d'expérience, on réfléchit à l'étape suivante, faire une formation complémentaire, faire une auto-formation, s'attaquer à des projets plus complexes, respirer !

N'est-ce pas une certaine « fuite en avant » avec les recherches parfois extravagantes des recruteurs et entreprises qui cherchent les développeurs à 42 bras sachant tout faire pour un salaire inférieur à la moyenne ?
Vous avez bien décrit la caricature de l'annonce mal rédigée qui ne sépare pas l'essentiel (technos, environnement humain) de l'accessoire (litanie de nice to have). •

Questions-réponses avec Ridouan Abagri (Collège de Paris)

Ridouan Abagri a fondé l'école Digital College. Il est désormais directeur de l'innovation du Collège de Paris. L'école inclut aussi un Fab Lab et un studio vidéo sur le campus de la Grande Arche.

Depuis quelques années, les écoles (cursus complet / alternance), les centres de formations sur les métiers de l'informatique et notamment le développement se multiplient. Avec la promesse de trouver un emploi derrière. Quel est votre sentiment sur cette situation ?

Nous vivons une période de plein emploi dans le domaine de l'informatique, où les techniques et comportements changent rapidement. Pour autant, les savoir-faire techniques antérieurs sont toujours nécessaires et d'actualité. Ainsi, le socle technique à acquérir se voit élargi, il est donc nécessaire d'apporter au secteur des compétences à la fois plus profondes et plus diversifiées. Aujourd'hui, si effectivement de nombreuses écoles se créent avec des champs d'actions hétéroclites, on n'a pas encore assez d'établissements pour accompagner le boom de la demande. On est encore loin de se trouver sur un marché régulé.

Finalement, trop d'écoles / formations ne tuent-elles pas l'école informatique avec une offre pléthorique et le risque d'être peu visible ? Et surtout, avec cette concurrence, comment faites-vous pour recruter car sans étudiants, pas d'école ?

Nous sommes face à un phénomène économique classique d'offre et de demande. Auparavant ce marché très technique se limitait à la caste des écoles d'ingénieurs et écartait des talents potentiels. Il créait une forme de complexe chez les étudiants, l'élitisme étant toujours source de doute, voire de peur.

Plutôt que de le tuer, ces nouvelles écoles à format hybride enrichissent le marché en réduisant la pénurie de profils, sans pour autant la combler. On progresse ainsi dans la réponse à la demande mais sans la satisfaire pleinement, c'est pourquoi ces nouveaux établissements ne représentent pas une menace mais une opportunité ; la concurrence fait du bien à un marché, elle permet d'affiner l'offre en qualité et en diversité.

Dans notre cas, nous avons choisi d'approfondir cette notion clé de qualité d'enseignement, aussi le bouche-à-oreille est l'un de nos principaux canaux. De la même façon, nos cursus étant ouverts aux contrats en alternance, nous possédons un réseau d'entreprises partenaires très qualifié, qui attire énormément ces profils techniques.

Avez-vous le sentiment que l'on mise plus sur la quantité que la qualité ? Est-ce valable dans toute la France ou selon les régions ?

Il est vrai que les écoles d'ingénieur vont généralement plus miser sur la qualité, et les écoles type école de commerce sur la quantité. Chaque type d'établissement a un modèle économique différent et va apporter une offre pédagogique à plusieurs niveaux. Dans tous les cas la qualité reste évidemment l'objectif à atteindre afin d'apporter des compétences justes par rapport à la demande du marché de l'emploi. Dans le même temps, la quantité est un défi à relever pour tous, également pour répondre à cette demande. Le besoin est tel chez les entreprises qu'on est obligés de miser sur la quantité d'une façon ou d'une autre.

Chez Digital College nous nous efforçons de miser sur les deux. Nous pensons que la qualité est tout simplement un standard dans le domaine informatique où on a besoin de précision et de technicité.

Ces questions vont effectivement être perçues différemment selon les régions : l'Île-de-France reste le pôle économique majeur des nouvelles écoles. En région, on va pouvoir avoir plus d'écoles d'ingénieur, dont la notoriété fait qu'elles n'ont pas besoin d'être forcément en région urbaine pour attirer des candidats.

Ne fait-on pas (trop) rêver le jeune, l'étudiant ou la personne qui se réoriente ? Oui l'informatique embauche mais pas tous les profils ni toutes les compétences ? Un choc culturel pour certain(e)s ?

Notre métier n'est pas de faire rêver mais de former. Néanmoins cette forme de rêve qu'ont les étudiants est presque une obligation : cela encourage leurs ambitions personnelles, les motive à aller en cours et est donc essentielle au bon déroulement de leur carrière académique. Aujourd'hui on embauche vraiment sur toutes les problématiques et toutes les compétences, mais il est vrai qu'on va rechercher une association de compétences plutôt qu'une compétence précise. On va notamment rechercher des profils avec un aspect managérial, le mode projet obligeant les jeunes informaticiens à devenir de futurs chefs de projets ; or, actuellement, on constate que le milieu de l'informatique est celui dans lequel on enregistre les managers les moins performants.

Dans votre école, quels sont les focus que vous mettez en avant ? Quelle est la part du travail personnel ? Comment les motiver pour que les étudiants soient curieux car nos métiers changent vite ?

On peut dire qu'on se positionne à la fois dans le passé, le présent et le futur : le passé car on va toujours prendre en compte des compétences informatiques de base type Java, etc. qui existent toujours et font partie du socle nécessaire à l'étudiant. Dans le présent parce qu'on doit répondre au besoin immédiat du marché de l'emploi, et dans le futur parce qu'une problématique contemporaine va toujours nous rattraper. Nous développons donc des programmes sur des fonctions qui se développent tout juste ou qui vont se développer. Notre but est de former des polymorphes au profil intergénérationnel.

Dans ce cadre, le travail personnel est essentiel. Nous allons faire avec l'étudiant le plus gros du travail, mais le développement de ses softs skills est un enjeu majeur : il doit évoluer en tant que personne, développer des capacités relationnelles, sa curiosité, être en veille constante, bref, se construire en tant que professionnel.

La motivation pour arriver à ce développement passe chez nous par une vraie sensibilisation, via des séminaires techniques, des tables rondes, des rencontres de professionnels… Nous sommes avant tout dans l'expérientiel, pour un secteur où rien n'est jamais acquis et où il faut sans cesse se renouveler.

À la sortie de votre formation, à quoi peut prétendre la personne ? Quelles sont ses compétences ?

A la sortie de notre formation, les diplômés peuvent prétendre à avoir le choix, tout simplement. Il y a de la place pour se développer en France, mais nos profils sont très attirés par l'étranger également. L'école informatique made in France est très réputée à l'international et on constate que, dans le berceau du secteur à la Silicon Valley, beaucoup de top managers sont Français dans les plus grandes startups et PME. L'école française a un très bel avenir dans ce milieu !

En termes de fonctions, nos diplômés vont d'emblée pouvoir occuper des postes managériaux avec des rémunérations très attractives. Dans certains domaines comme la cybersécurité, les profils juniors peuvent prétendre à une rémunération senior par exemple ; ce sont des profils très recherchés et c'est l'une des grandes problématiques sur lesquelles nous travaillons, avec la blockchain, un autre marché en pleine expansion. Enfin nos diplômés sont également tout à fait à même d'exercer des postes plus techniques, il y a des besoins à tous les niveaux et de nombreuses portes leur sont ouvertes ! •

Marc Hage Chahine
Key Member centre d'expertise de test d'Altran (ITQ). Créateur et animateur du blog La taverne du testeur
*Auteur du livre "**Tout sur le test logiciel**"*

La chasse aux champignons

J'entends souvent que le test c'est compliqué, qu'il est difficile d'avoir une vision d'ensemble ou tout simplement de le présenter avec un exemple compréhensible pour un enfant de 6 ans.

niveau 100

En fait, comprendre les principes des activités de base du test c'est très simple. On peut d'ailleurs remarquer que ces activités se rapprochent fortement d'une activité que nous avons quasiment presque tous pratiquée : la chasse aux champignons !

Penchons-nous sur la chasse aux champignons.

Le but d'une chasse aux champignons est de **trouver des champignons**. Néanmoins ce n'est pas si facile. Si l'on veut arriver à un bon résultat il faut se préparer à cette chasse, bien la mener, et, bien sûr, s'en servir pour les prochaines chasses.

Pour bien illustrer l'ensemble des activités relatives à une chasse aux champignons, je vous propose de passer par un processus.

La première chose à faire est de **comprendre le contexte**. Dans quelle forêt ou quelle parcelle allons-nous rechercher nos champignons ? Combien serons-nous de chasseurs ? Combien de temps durera cette chasse ? Comment nous-y rendrons-nous ? En quelle saison somme-nous ? Quelle a été et sera la météo avant notre chasse ? Il va de soi qu'en fonction de la réponse à ces question la chasse sera organisée différemment !

Après la compréhension de ce contexte, nous avons la **préparation de la chasse**. Comment se regrouper avant la chasse ? Qui ramène quoi ? Un repas doit-il être prévu ? Quels champignons sont-ils recherchés ? Où devons-nous chercher en priorité ? Quelle est la taille de la parcelle sur laquelle nous chasserons ?

Vient alors **la chasse** en elle-même. On met en place la stratégie définie lors de la préparation. On s'adapte aussi en fonction de son évolution en concentrant plus nos efforts là où on trouve des champignons et en diminuant les recherches dans les coins pauvres en champignons.

Vient alors le **Bilan**. Ce bilan intervient à la fin de la chasse, et pour être honnête, c'est ce qui intéresse tout le monde ! Combien de champignons avons-nous récolté ? Les champignons récoltés étaient-ils bien les champignons cherchés ? La stratégie de recherche était-elle la bonne ?

Les étapes d'une campagne de test, c'est pareil !

Partons du postulat que les champignons sont en fait les bugs.

Pour avoir des activités de test performantes il va falloir **comprendre le contexte** de notre application. Ici la question n'est plus quelle forêt mais plutôt quelle application ? Quelle sont les faiblesses de cette dernière ? Quel est le public visé ? Y-a-t-il des normes à suivre ?

Lorsque le contexte est compris il faut alors **préparer sa ou ses campagnes de test**. Dans ce cadre on définit comment sera testée l'application. En fonction des risques identifiés, les tests sont conçus et/ou sélectionnés avant d'être priorisés. On définit également l'ensemble des environnements et données de test afin de permettre aux testeurs de travailler dans des conditions optimales.

Vient alors la **campagne de test**. Lors de cette campagne on exécute les tests définis par la stratégie mais en fonction des événements et des bugs trouvés il est possible de réorienter nos recherches de bugs et donc la stratégie. Il n'est pas rare de devoir sacrifier certains tests peu prioritaires suite à un problème de délai ou à un risque qui a été sous-estimé et qui doit être couvert.

Arrive enfin le **bilan**. Tout comme pour les champignons, c'est ce bilan qui intéresse le plus grand nombre de personnes, et plus particulièrement les personnes n'ayant pas participé à la chasse. Lors de ce bilan on donne généralement un Go/No Go définissant si le niveau de qualité minimum souhaité pour l'application est atteint. De même, fait une rétrospective afin de savoir ce qui pourrait être amélioré et si notre stratégie (plan de test) a été efficace.

L'analogie chasse aux champignons et activités de test, c'est aussi une meilleure compréhension des principes de base du test !

Il existe 7 principes fondateurs du test dont 6 sont couverts par cette analogie qui permet de mieux les comprendre.

Ces principes sont :

Les tests montrent la présence de défauts : si on trouve des champignons sur une parcelle de forêt c'est qu'il y avait des champignons. Si on n'en trouve pas cela ne veut pas dire qu'il n'y en avait pas. C'est la même chose pour les tests ! En effet, les scénarios de test ne peuvent découvrir de bug que sur le parcours applicatif qu'ils traversent. Ils ne donnent donc pas d'information sur les sections non parcourues.

Le paradoxe des pesticides : si on passe toujours au même endroit en cherchant des champignons, au bout d'un moment il n'y en aura plus dans ces zones alors que des champignons auront pu pousser à d'autres endroits de la parcelle. Cela se voit d'ailleurs aisément sur les gros sentiers qui ne comptent que très peu de champignons à leur abord. C'est la même chose pour les tests. Si l'on fait toujours les mêmes tests on

Campagne de chasse aux champignons

Contexte	Préparation	Chasse	Bilan
Quelle forêt/parcelle ? Combien de chasseurs ? Quelle est la saison ? Quelle est la météo des jours précédents ?	Comment se regrouper ? Qui ramène quoi ? Où chercher en priorité ?	Appliquer la stratégie Adapter la stratégie aux champignons trouvés	Combien de champignons trouvé ? La stratégie était-elle efficace ?

Campagne de test

Contexte	Préparation	Chasse	Bilan
Quelle application? Combien de testeurs? Quels outils pour les tests? Quelles modifications ? Quelles nouvelles fonctionnalités?	Combien de temps? Comment optimiser ce temps? Quels sont les bugs fréquents? Complexité de l'application?	Exécution de la stratégie (tests + priorisation) Adaptation possible en cours de campagne	Quel niveau de qualité de l'application? Combien de bugs trouvés? Quel type de bugs? Stratégie de test efficace?

ne trouvera au final plus aucun bug alors que ces derniers pourront proliférer sur des sections non couvertes par les tests.

Les tests exhaustifs sont impossibles : il est impossible de couvrir chaque cm^2 d'une parcelle ! De même, il est impossible de tout tester (ce qui explique en partie le premier principe énoncé plus haut). Ce principe est très simple à appréhender dès lors où l'on travaille sur des applications dont le nombre d'étapes consécutives possible est infini (ex : application mail).

Les tests dépendent du contexte : en fonction de la forêt et de la saison, les champignons recherchés seront différents, les techniques de recherches également. Sur un projet de test, c'est pareil. Les stratégies et besoins en test sont différents en fonction du fonctionnel, des technologies utilisées et de tout autre élément du contexte (ex : niveau de qualité à atteindre).

Il faut tester tôt : plus tôt on découvre un champignon, moins gros est le champignon. Si on considère que le poids (ou la taille) d'un champignon correspond au coût de correction du bug, on arrive à ce qui a permis l'émergence du « Shift left » ; plus tôt on détecte une anomalie, moins elle coûte cher à corriger.

Les défauts (bugs) sont regroupés : lorsque l'on tombe sur un champignon, on peut être sûr d'en trouver des identiques à proximité. Pour les bugs, c'est pareil ! Lorsque l'on en trouve un, il y a de fortes chances que d'autres bugs similaires soient présents sur la fonctionnalité en question.

Et on peut même aller encore plus loin !

Il y a des **champignons vénéneux** : ces champignons peuvent être assimilés à des faux bugs, remontés par des tests pas assez bien conçus.

Il y a des **champignons que nous ne souhaitons pas ramasser** : ces champignons peuvent être assimilés à des bugs que l'on ne souhaite pas corriger, et ce, pour de multiples raisons comme le coût trop important, l'adoption du bug par les utilisateurs…

La chasse aux champignons nécessite **des outils** comme un panier, un bâton, des chaussures de randonnée : pour le test c'est pareil ! Le panier peut même être comparé à un gestionnaire d'anomalie

L'**expérience** d'un chasseur de champignons détermine en partie sa faculté à trouver des défauts : tout comme l'expérience et la connaissance d'un produit permet à un testeur d'être plus efficace (et pas uniquement à l'exécution).

Le **poids d'un champignon** peut être assimilé au coût de la correction.

L'**espèce du champignon** peut quant à elle être assimilée à un type de bug. Par exemple, les Bolets peuvent être considérés comme des bugs fonctionnels et les pieds de moutons comme des bugs liés à la performance.

Adapter ses recherches est essentiel, particulièrement lorsque l'on passe régulièrement sur une parcelle. On a ici l'intérêt des tests exploratoires qui sont de plus en plus préconisés dans les méthodes incrémentales et donc agiles.

La **stratégie de recherche** pour une parcelle ou une forêt peut être comparée au plan de test.

Chaque **forêt** peut être considérée comme une application. Chaque forêt est différente et à ses spécificités. En fonction de ces spécificités les champignons recherchés seront différents et le nombre de personnes nécessaires pour la parcourir également.

Chaque **parcelle** de forêt est comme une grosse fonctionnalité de l'application. C'est le même principe que pour les forêts tout en sachant que le type de champignon trouvé sur différentes parcelles d'une même forêt est souvent proche.

Les **arbres, les plantes et les pierres,** présents dans la parcelle ont un impact direct sur les types de champignons qui poussent. Ces éléments peuvent être considérés comme l'environnement technique

La **faune** fait évoluer la parcelle (par exemple, les sangliers remuent la terre, les écureuils ramassent les glands…). Ces animaux peuvent être considérés comme des partenaires avec qui l'application communique mais n'étant pas sous le contrôle du logiciel.

La **pluie** peut quant à elle être comparée aux changements apportés à l'application et donc les mise en production/mise en service car la pluie permet de faire pousser les plantes et les champignons et donc de faire évoluer la forêt qui est l'analogie de l'application.

Les promeneurs peuvent quant à eux être considérés comme des utilisateurs. Promeneurs qui, comme les utilisateurs peuvent trouver des champignons et décider ou non de les ramasser (le cas de ramassage peut être considéré comme la remontée du bug à l'équipe de support).

Pour résumer on peut avoir ce tableau de correspondance :

Chasse aux champignons	Activités de test
Champignon	Bug
Poids du champignon	Coût de la correction
Espèce de champignon	Type de bug (fonctionnel, performance…)
Champignon non comestible	« Faux » bug
Champignon détecté et non ramassé	Bug à ne pas corriger
Stratégie de recherche	Plan de test
La forêt	L'application
La parcelle	Une fonctionnalité de l'application
La flore (arbres, plantes)	L'environnement technique
La faune (animaux)	Les partenaires
La pluie	Les mises en production
Les promeneurs	Les utilisateurs
L'expérience du chasseur	L'expérience du testeur
Le bilan de la chasse	Le bilan des tests La rétrospective pour l'amélioration continue

Paiement : ça bouge vite, très vite !

Le paiement a beaucoup évolué ces 30 dernières années et a suivi une profonde mutation depuis l'explosion du web. Nous sommes passés d'un monde uniquement bancaire à une multitude de solutions et de prestataires. Internet est passé par là ainsi que le monde startup. Paypal a été un des pionniers du paiement en ligne.

La rédaction

Petit retour sur PayPal

Elon Musk crée d'abord X.com en 1999. Il propose des services de paiements : verser, payer, retirer de l'argent. Une sorte de banque en ligne. Le succès est rapide. Un an auparavant, des étudiants créent Confinity, un système de paiement purement électronique. La société lance rapidement PayPal. Après des relations difficiles et de longues négociations, X.com et PayPal fusionnent au printemps 2000. Musk lève 100 millions $ pour financer son développement et peaufiner le modèle économique : proposer à tout le monde un système de paiement électronique sécurisé, rapide et fiable. Le succès est rapide.

Finalement Musk revend PayPal à eBay pour 1,5 milliard ! La FinTech était née ainsi que la révolution du paiement.

Aujourd'hui, on entend partout parler de FinTech, les startups spécialisées dans la finance. En juin dernier, les acteurs de la FinTech française avaient levé +350 millions pour pouvoir financer les développements et leur présence sur un marché extrêmement concurrentiel. En comparaison, l'Anglais OakNorth a levé 440 millions $, CheckOut 230 millions ! La FinTech française a émergé avec 2-3 ans de retard par rapport aux autres grands pays, mais les niveaux d'investissement n'ont rien de comparable.

Ces levées montrent le potentiel du marché mondial. On veut disrupter le marché financier et les moyens de paiements. Les solutions de type Apple Pay/Google Pay ont ouvert un nouveau marché. Et ce n'est pas fini.

Finalement, on a besoin de quoi ? 1/2

Comme nous l'a présenté Stripe (acteur du paiement 100 % cloud), basiquement, il y a 3 couches clés dans le paiement :

- KYC (Know your customer ou ce que l'on sait de notre client) : qu'est-ce que l'on connaît du client ? Lutte contre le blanchiment. À qui va réellement l'argent ?
- Transaction : on branche les câbles et on effectue la transaction de paiement.
- Ops : le suivi des opérations, appliquer la réglementation, les taxes, etc.

Le flux du paiement est finalement assez simple : nous avons le client qui achète sur un site/une application. Le site/l'app implémente une ou plusieurs solutions de paiement (PayPal, CB, etc.). Cette implémentation permet de connecter l'interface de paiement du site/app au backoffice pour faire la validation et l'envoi du paiement (= on envoie/réceptionne l'argent sur un compte).

On parle aussi de push et de pull. Par définition, le push est un paiement que l'on va recevoir (chèque, virement, liquide), le pull est le fait de retirer l'argent depuis le compte client (paiements récurrents, prélèvements automatiques, etc.). À cela se rajoute le chargeback. Cette fonction permet le remboursement du client. En principe, le chargeback ne transite pas par le système de paiement et le développeur ne s'occupe de rien (si la fonction est bien faite). En effet, c'est le service ou la banque qui effectue l'opération.

Si nous prenons le cas de PayPal, le chargeback est défini ainsi : l'acheteur peut simplement demander le remboursement de la transaction et informe la société de la CB d'annuler et de rembourser le paiement. PayPal gèle les fonds correspondants du compte vendeur.

Le chargeback doit respecter certaines règles :

- L'acheteur ne reçoit pas sa commande ;
- La commande est reçue endommagée ou comporte des défauts ;
- CB non reconnue ;
- L'acheteur paie des charges non prévues ou excessives ;
- Achat non autorisé par l'acheteur.

En chargeback, la transaction (=paiement) est enclenchée.

Finalement, on a besoin de quoi ? 2/2

Le développeur va implémenter les API/SDK/services nécessaires pour exposer le ou les services de paiement. Par exemple, sur programmez.com, nous avons deux services de paiement en ligne : PayPal et CB. Chaque service possède son propre flux et son API. Ce qui complexifie la gestion au quotidien. À cela, se rajoutent, en gestion manuelle, les virements SEPA et les chèques.

Au-delà des mécanismes que l'on va implémenter, la partie visible pour le client étant le bouton paiement et le formulaire de saisie (+ les facteurs d'authentification si nécessaire), mais il y a tout un backoffice qui doit prendre en compte :

- Saisie et vérification des données de paiement : numéro CB + cryptogramme, compte PayPal, etc.
- Validation du paiement à x facteurs : c'est notamment le cas avec la CB avec par exemple 3D Secure et un code de validation envoyé par SMS ou via un générateur de token externe ;
- Le paiement validé va suivre le flux transactionnel allant du site/application vers l'établissement financier/service de paiement avec, bien entendu, authentification de la plateforme (par exemple mon site e-commerce) et connexion au compte. Si tout est OK, la transaction se réalise avec le transfert de l'argent.

Les logs sont très importants, car parfois, une transaction n'aboutit pas. Ce problème peut intervenir à différents moments :

- Annulation par le client ;
- Mauvaise authentification (abandon où le client rejoue le paiement), échec authentification, rejet du paiement ;
- Problème de transaction non visible.

Le dernier point est important. Car, parfois, la transaction est valide côté client, valide côté plateforme (donc mon site), mais quelque chose ne va pas en backoffice. Cela peut être un rejet a posteriori, un paiement mis en attente pour x raisons, échec de transaction suite à une panne API/service bancaire. Il peut arriver que des pannes impactent le fonctionnement de l'API que l'on implémente, mais ce n'est pas toujours visible et surtout cela apparaîtra après la validation du paiement.

Les logs vont donc être cruciaux pour voir ce qu'il se passe et comprendre pourquoi une transaction n'aboutit pas. Aussi automatisé soit le paiement, vous devez toujours garder un œil sur les logs et les transactions pour voir si tout se passe bien.

STRIPE : L'AMBITION DE SIMPLIFIER LES CHOSES

Devant la multiplication des solutions de paiements et les multiples législations, les monnaies et taxes, le développeur est parfois perdu quand il doit adapter les mécanismes de paiement à l'étranger. Pour simplifier les choses, il est nécessaire d'utiliser une plateforme qui masquera une partie de cette plomberie.

Stripe veut aider les développeurs à aller plus vite et à industrialiser tout ce qui était possible, comme la gestion des documents légaux. Un gros travail est réalisé pour intégrer les spécificités de chaque partenaire de paiement. Stripe propose à la fois des API et SDK pour les principaux langages, mais aussi des tableaux de bord pour suivre l'ensemble des flux. Et depuis quelques mois, on peut utiliser SQL directement dans les tableaux de bord. À cela se rajoute la communauté qui n'hésite pas à développer des modules et plug-ins tiers pour faciliter l'intégration de Stripe.

Si les chèques et les virements SEPA sont supportés, côté PayPal c'est plus difficile, car le service ouvre peu les API aux agrégateurs comme Stripe. Comme le précise Stripe, le développeur n'a pas à être un expert en paiement. Ce n'est pas son job, son travail est d'intégrer les mécanismes pour la transaction.

Côté tarif, Stripe simplifie :

- 1,4 % + 0,25 € pour des paiements par CB. Par défaut, l'ensemble des fonctions est disponible ;
- Pour les volumes importants de transactions : la plateforme est à la carte avec des coûts personnalisés ;
- Pour le module billing (facturation & abonnement) : l'outil est gratuit jusqu'à 1 million €.

Christophe Vergne , Benjamin Gosset, Adil Outlioua.
Capgemini
Capgemini Financial Services accompagne les grands acteurs du paiement en France et à l'étranger dans leur transformation digitale. Capgemini Financial Services publie annuellement sur l'évolution de l'industrie un World Payments Report disponible gratuitement sur www.worldpaymentsreport.com

Quoi de neuf dans les paiements ?

Le paiement, un monde d'innovation permanente en accélération … mais pas que…

Paylib, Applepay, Lydia, Leetchi, … le monde du paiement ne cesse d'innover, que ce soit dans les façons d'accéder au service de paiement, en direct ou via des terminaux ou, plus récemment avec de nouveaux instruments de paiement. C'est en étant au plus proche des clients et au travers d'innovations « customer oriented » que se trouve la capacité pour ces fournisseurs de services de se démarquer de la concurrence.

Si les commerçants, et en particuliers les e-commerçants, ont été confrontés rapidement à la demande de leurs clients de pouvoir payer avec tous types d'instruments de paiement partout et à tout moment, les trésoriers d'entreprises sont également demandeurs de solutions leur permettant de pouvoir se connecter à échelle mondiale à toutes leurs banques plutôt que de disposer d'une solution propre à chaque pays ou chaque banque.

Nous voyons rapidement que la chaîne de valeur paiement se découpe en deux avec d'un côté l'échange des opérations avec les clients (virements, prélèvements, cartes, paiements domestiques et internationaux) sujet à des innovations permanentes (« les trains »), et, de l'autre, leur traitement des transactions en tant que tel. Une activité industrielle et non différenciante qui traite les volumes de transactions permettant de rester dans la course performance/prix (« les rails »).

Dès lors une multitude d'acteurs s'engouffre dans ce marché et vient concurrencer les acteurs traditionnels (banques) autours de 4 axes :

- L'émission des opérations ou comment rendre la création des ordres de paiement transparente dans le parcours d'achat du client ou du corporate, où qu'il soit ;
- L'offre d'acceptation de paiement pour les commerçants couvrant tous les moyens de paiements et permettant au commerçant de se consacrer à son métier ;
- La gestion de la sécurité et de la fraude par des acteurs spécialisés sur base analytique et contextuelle ;
- Des services à valeur ajoutée extraits de l'analyse des données de paiements, pour développer les ventes ou les revenus indirects.

Cette fragmentation est à la fois un défi pour les acteurs traditionnels qui se retrouvent désintermédiés, mais également une opportunité s'ils parviennent à créer et proposer des écosystèmes permettant de fournir à leurs clients des services toujours plus personnalisés, innovants et disponibles plus rapidement.

Pour permettre à ces écosystèmes de se développer, il faut favoriser les échanges de données et de flux financiers sans compromettre la protection des données clients (RGPD) et la sécurité des échanges financiers (DSP2 et Instant Payments).

L'Instant Payment, un nouvel instrument « universel »

L'Instant Payment (Paiement Instantané) est un virement interbancaire de compte à compte, pan-européen en cours de déploiement dans plus de 30 pays.

Il réutilise les standards techniques du virement SEPA (norme ISO 20022) et s'échange en interbancaire en 24/7, 365 jours par an, en moins de 10 secondes, avec toutes les garanties de sécurité fournies par la Banque Centrale Européenne (BCE) et le réseau des autres banques centrales.

L'Instant Payment vient ainsi s'ajouter aux divers instruments de paiement déjà existants, voire même les concurrencer avec comme idées sous-jacentes d'accroître l'innovation et de faire baisser les prix pour les consommateurs.

Ainsi les banques et les autres fournisseurs de services de paiement vont pouvoir développer de nouvelles offres (« les trains » au-dessus de nouveaux « rails ») pour faire du virement de particulier à particulier un moyen de paiement dans toutes les situations possibles (achat d'une voiture d'occasion, paiement de la facture du restaurant, don à un proche, etc.) et permettre ainsi au bénéficiaire d'être crédité immédiatement tout en étant notifié par SMS. Le bénéficiaire peut réutiliser l'argent sans délai, même week-ends et jours fériés. Les entreprises pourront également recourir à ce nouveau moyen de paiement pour optimiser leur trésorerie et payer au dernier moment des notes de frais, des factures en retard ou toute somme attendue urgemment par leur bénéficiaire : remboursement de sinistre d'assurance, allocation, …

Ce paiement instantané est interopérable à l'échelle de la zone euro. Vous pourrez ainsi payer un bénéficiaire de Francfort comme de Marseille en 10 secondes et en toute sécurité. Dans le futur il est même envisageable d'interconnecter les systèmes équivalents dans d'autres régions du globe et dans d'autres devises... mais nous n'y sommes pas encore.

Pour l'instant ce service est offert par les grands établissements français (groupes BPCE – Banque Populaire Caisse d'Epargne, Arkea, Crédit Mutuel, Société Générale, BNP Paribas, Crédit Agricole) qui couvrent plus de 80% du marché en volume de transactions.

Des API normalisées pour permettre aux fournisseurs de services de paiements de dialoguer avec les banques

La définition de standards de communication partagés, sécurisés et respectueux de la règlementation relative à la protection des données privées est une condition sine qua none à la création de nouveaux écosystèmes.

En dépit de multiples tentatives (ISO, R3 sur blockchain, BIAN sur l'architecture bancaire, FIDO pour l'authentification en ligne, …), ces initiatives ont toujours manqué d'adoption globale et peinent à s'imposer sur le marché. Le régulateur Européen, en association avec la Commission, les représentants des banques et des utilisateurs et l'Autorité bancaire pour les standards techniques ont publié une directive Européenne d'application pour septembre 2019, la seconde Directive des Services de Paiements (DSP2)

La DSP2 prévoit notamment que les écosystèmes des acteurs du marché communiquent entre eux au travers d'API (Application Programming Interface), interfaces sécurisées d'échange de données reposant en grande partie sur les standards du web (HTTPs, GET/POST, WoA-Web oriented Architecture, …). Le régulateur prévoit en outre que sur des cas d'usages simples comme la collecte de données relative aux comptes bancaires ou l'initiation de paiements, les banques ne puissent s'opposer aux instructions données par leurs clients, à condition bien évidemment que ceux-ci soient authentifiés au préalable (authentification forte, voir plus loin). Les principaux acteurs du marché ont donc normalisé la description de ces interfaces, à l'échelle française (STET) puis Européenne (Berlin group), de sorte à faire de ces API un instrument universel d'échange de données entre acteurs (nouveaux et historiques), mais également à l'intérieur de leurs propres écosystèmes permettant ainsi une meilleure urbanisation de leur système d'information (architecture REST).

Le paiement de demain est donc une industrie ouverte et collaborative qui s'appuie sur des standards partagés et de l'échange de données en temps réel via des services API.

La data : au centre des nouveaux modèles de business

Aujourd'hui, ce n'est plus la capacité à traiter des opérations en grand nombre qui fait la force des prestataires de service de paiement. Il s'agit désormais d'une commodité, peu différenciante d'un prestataire à l'autre et sur laquelle la guerre des prix fait rage, quand ceux-ci ne sont pas fournis gratuitement aux particuliers.

La force d'un prestataire réside dorénavant dans sa capacité à comprendre et anticiper les besoins de ses clients, quels qu'ils soient, et à y répondre par de nouvelles offres proposées directement ou au travers de ses partenaires. Cette connaissance des besoins clients, dans un monde digital, passe par l'acquisition et le stockage massifs des données clients et dans la capacité de ces prestataires à les exploiter.

La grande leçon de Google et des GAFA en général, transposée désormais aux paiements, c'est la transformation du business model où la transaction est offerte/gratuite (recherche Google, paiement, …) mais l'information et les données qu'elle génère devient exploitable permettant ainsi de payer indirectement ce service (publicité, marketing).

Cette transformation contraint aujourd'hui les grands établissements bancaires à ouvrir leur SI et développer la production/consommation d'API avec de nouveaux partenaires mais également à re-urbaniser leur infrastructure de sorte à développer l'analyse des données clients et opérer des traitements en temps réel, tout le temps – 24/7/365.

Le tout bien entendu en toute sécurité et conformément à la règlementation.

Condition impérative de succès : la sécurité pour le consommateur

La sécurité reste un enjeu majeur pour le secteur des paiements. Afin de préserver la confiance de leurs clients, les Payment Service Provider (PSP) ne cessent de mettre en œuvre tous les moyens nécessaires pour garantir la sécurité des transactions. Ils effectuent d'importants investissements pour respecter les nouvelles réglementations européennes (PSD2, …) qui portent notamment sur le renforcement du processus d'authentification autour des paiements particulièrement sur internet ou via mobile.

L'authentification forte est un exemple très concret de solution mise en place par les PSP pour répondre à ce besoin. Les clients sont authentifiés de façon certaine sur la base de 2 des 3 facteurs : qui il est, ce qu'il sait, ce qu'il possède. A décliner selon le cas d'usage et la stratégie de la banque en un bouquet de solutions d'identification « forte ».

Avec le développement des services en ligne et la diversification des canaux d'accès, le risque de fraude accroît. Malgré la mise en place de procédures strictes comme la séparation entre le front end, composé principalement d'interfaces clients web, et le back end, infrastructure permettant le traitement des transactions, l'utilisateur final en bout de chaîne reste néanmoins vulnérable et la cible principale des cybercriminels. Des attaques de type « man in the middle » ou « man in the browser » font très souvent surface. Ces techniques permettent ainsi de récupérer les données bancaires liées à l'utilisateur et donc d'usurper son identité. Face à cette problématique, une réponse a été apportée : la tokenisation des flux.

La tokenisation consiste à substituer les données bancaires sensibles (numéro de cartes, …) par des données périssables à utilisation unique appelés « tokens ». Ces derniers sont non réversibles, sans valeur intrinsèque et inutilisables dans le cas d'une compromission, ce qui permet la désensibilisation des données critiques. Du point de vue commerçant, c'est également un moyen très efficace pour réduire le périmètre PCI-DSS (norme de sécurité de l'industrie des cartes de paiement) puisque les données bancaires sensibles ne sont plus stockées sur leur système d'information. Ce processus bénéficie donc à l'ensemble des acteurs de la chaîne de paiement, ce qui explique l'intérêt majeur qui lui est apporté. •

Olivier Bouzereau

Florian Pradines

Eviter le casse-tête du paiement sécurisé : l'exemple de Skeerel Pay

Une procédure d'authentification via un tiers de confiance peut simplifier la tâche d'intégration du commerce en ligne, garantir les paiements à distance et rassurer les internautes.

niveau 100

Quelles que soient sa taille et sa renommée, toute boutique en ligne se heurte à la question de la confiance de l'internaute sur le point de valider son panier d'achat. L'abandon reste fréquent ; il est motivé par de multiples raisons possibles : une connexion depuis un cyber-café, un transfert sans confidentialité, le stockage de données à l'étranger, une clause juridique suspecte ou une facturation aléatoire dissuadent de nombreux clients distants. Pire, un compte créé avec un mot de passe ré-utilisé sera sûrement piraté, la fuite de données privées n'épargnant aucun fournisseur de services en ligne.

Proposer le paiement en un clic

Une alternative française résout ce casse-tête pour l'internaute, pour le cyber-marchand et le développeur de la boutique en ligne. La solution Skeerel Pay, brevetée, simplifie le design, les tests et la validation des codes de paiement à implémenter. L'empreinte digitale captée sur smartphone (iOS ou Android) ou une signature électronique assurent l'authentification de l'internaute distant. Le parcours d'achat est raccourci et la commande confirmée sans crainte de fuite de données. En effet, aucun mot de passe ni aucun numéro de carte bancaire ne sont transmis lors de ce mode d'achat en ligne.

Pour enregistrer une commande et procéder à son paiement à distance, des procédures d'identification et d'authentification demeurent indispensables. L'intervention d'un tiers de confiance permet de limiter le partage de secrets. Inutile de transmettre à chaque cybermarchand toutes les inscriptions de sa carte bancaire. L'internaute apporte une simple preuve d'identité, et le tiers de confiance se charge du reste ; il devient le garant du paiement reçu, de la somme captée dans la devise attendue par le marchand.

En résumé, ce mécanisme substitue au secret bancaire une caractéristique personnelle de l'acheteur distant, son empreinte digitale ou un certificat numérique par exemple. L'acheteur confirme simplement qu'il est bien celui qu'il prétend, au moment de régler ses achats en ligne.

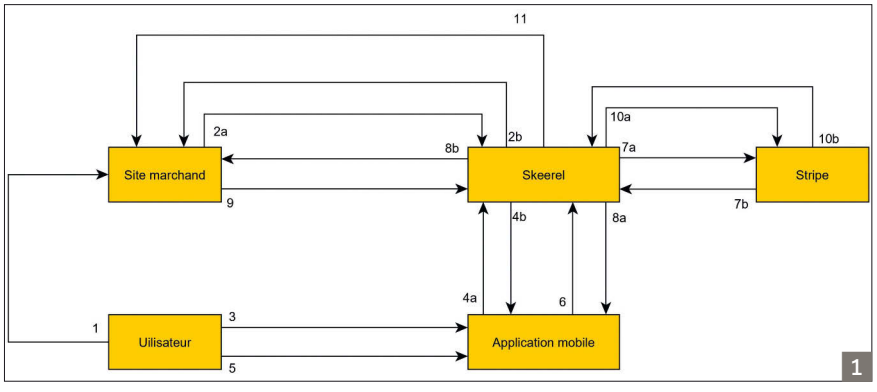
Les interactions entre les parties prenantes sont représentées dans le schéma ci-dessus : 1

- Depuis la page panier, l'utilisateur clique sur le bouton Skeerel Pay (paiement par Skeerel)
- Le code JavaScript de la page panier établit une communication avec Skeerel Pay qui initialise une session, puis un QR code unique est généré
- L'utilisateur flashe ce QR code depuis l'application mobile Skeerel Pay
- L'application se connecte à la session courante
- L'utilisateur s'authentifie et confirme le paiement
- L'application communique à Skeerel Pay la notification de demande de paiement
- Skeerel Pay demande au processeur de paiement de sécuriser les fonds
- Skeerel Pay notifie l'application et le site marchand du succès de la sécurisation des fonds
- Après redirection vers la page de vérification du site marchand, celui-ci accuse réception auprès de Skeerel
- Skeerel Pay demande au processeur de paiement de capturer les fonds
- Skeerel Pay répond au site marchand en lui notifiant le succès de la capture des fonds

L'achat depuis un ordinateur

Suivons le parcours de l'internaute sur un site marchand de vêtements. Il sélectionne le t-shirt de son choix, précise sa taille, puis ajoute une casquette ou un éventail USB, chaque nouvel article remplissant son panier virtuel. Lorsqu'il a fini ses emplettes, il se rend sur la page du panier où il choisit la solution Skeerel intégrée au site marchand. 2

Conformément à son choix, il découvre un QR code largement affiché dans un cadre à l'écran. C'est en flashant ce code depuis son smartphone qu'il établit le lien avec la session courante, grâce à l'application Skeerel Pay installée préalablement (sur Android ou iOS). 3



1

2

L'internaute flashe le code affiché par le cybermarchand avec l'application Skeerel Pay de son smartphone. Puis il se connecte à son profil, sans mot de passe, via son empreinte digitale captée par le combiné ou au travers d'une signature cryptographique, ou encore via son compte Google ou Facebook. Il peut également agir en tant qu'invité, et payer avec son compte Apple Pay ou Google Pay. 4

L'internaute a pu enregistrer plusieurs moyens de paiement sur son profil. Il doit maintenant préciser le compte à débiter du total des achats qu'il vient d'effectuer. 5

Cette sélection faite, il voit le récapitulatif de sa commande. C'est alors le bon moment de vérifier l'adresse email où il recevra sa confirmation d'achat, la date de livraison prévue, l'adresse de livraison et l'adresse de facturation préenregistrée, ainsi que le total facturé bien sûr. 6

À ce stade, l'acheteur distant peut encore modifier la méthode de livraison et préférer un point relais selon son agenda et la date prévue de livraison. 7

Lorsque l'internaute clique sur le bouton Payer, sa transaction est immédiatement traitée et validée. C'est-à-dire que le compte retenu est débité. Puis, le marchand reçoit une preuve de paiement pour la somme totale et la devise prévues. 8

L'achat depuis un terminal mobile

Les étapes du paiement depuis une tablette ou un téléphone mobile sont semblables à celles de l'achat depuis un ordinateur. Seule différence, au lieu de flasher un code, un bouton apparaît qui permet de télécharger et de lancer l'application. Le mobinaute n'a plus qu'à lire et confirmer les informations inscrites à l'écran pour valider son achat, l'adresse de livraison et le mode de paiement retenu.

Un e-commerce déculpabilisé

Pour le gestionnaire de la boutique en ligne, ce mécanisme procure un meilleur taux de conversion grâce au transfert automatique des informations de l'internaute, au paiement rapide et sécurisé. Le taux de fraude diminue et la cyberboutique devient conforme aux nouvelles normes 3DSecure 2, DSP2 et SCA (Strong Customer Authentication), un scoring intelligent déterminant le risque de fraude.

Du côté du développeur ou du webmaster, l'intégration des composants logiciels est accélérée et la sécurité accrue. L'installation s'effectue en quelques clics lorsqu'un module est disponible pour la plateforme de e-commerce en place. Dans le cas contraire, le développeur ne part pas de zéro, mais s'appuie sur une bibliothèque retirant toute la complexité du design.

L'intégration en PHP en trois étapes

L'installation de la bibliothèque s'effectue en une seule ligne de commande :

composer require arcansecurity/skeerel-php 2.2.0

Alternativement, on peut l'installer via le fichier *composer.json* :

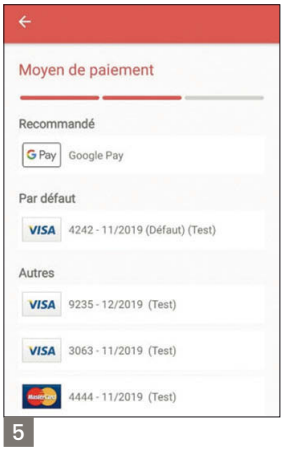
```
{
  "require": {
    "arcansecurity/skeerel-php": "2.2.0"
  }
}
```

v rel i: re étape c tramer da transantion ´ : s da paèe panier

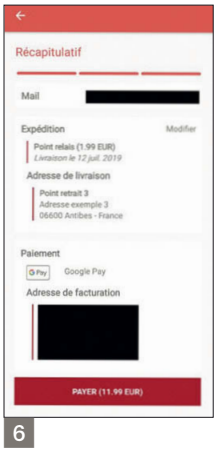
Lorsque l'utilisateur affiche la page panier, la première étape consiste à générer un jeton de session afin d'éviter des attaques par rejeu ou « Cross-site request forgery ». Cela s'effectue automatiquement avec la bibliothèque Skeerel ou, au choix, manuellement :

```
\Skeerel\Skeerel::generateSessionStateParameter();
```

4

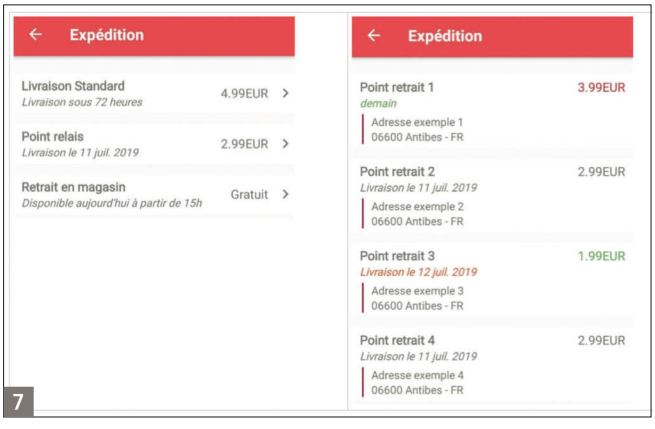


5

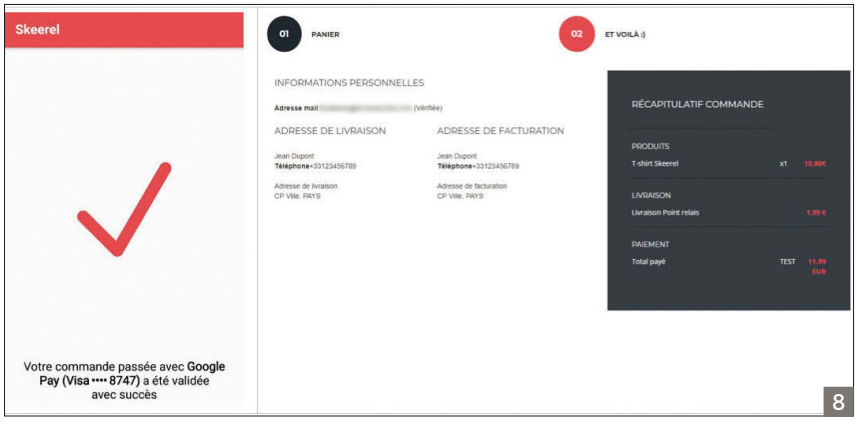


6

3



7



8

Ensuite, il faut ajouter le bouton de paiement par « Skeerel Pay » sur le site marchand. Cela se fait à l'aide d'un petit script en langage JavaScript. Le bouton apparaît à l'endroit où est inséré le script :

```
<script type="text/javascript" src="https://api.skeerel.com/assets/v2/javascript/api.min.js"
    id="skeerel-api-script"
    data-website-id="IDENTIFIANT_DU_SITE_MARCHAND"
    data-state="<?php echo \Skeerel\Util\Session::get(\Skeerel\Skeerel::DEFAULT_COOKIE_NAME); ?>"
    data-redirect-url="URL de redirection lorsque le processus d'achat est terminé"
    data-need-shipping-address="" // Si vous souhaitez récupérer une adresse de livraison
    data-need-billing-address="" // Si vous souhaitez récupérer une adresse de facturation
    data-delivery-methods-url="https://site.com/delivery_methods.php?user=__USER__&zip_code=__ZIP_CODE__&city=__CITY_
    data-checkout=""

    data-amount="1000" // Somme à payer
    data-currency="eur" // Devise de paiement    ></script>
```

Comme on peut le voir, ce code offre plusieurs options pour configurer le comportement de l'application et laisser l'acheteur maître de ses choix.

Deuxième étape : récupérer des informations de livraison

Lorsque l'utilisateur doit être livré, plusieurs modes de livraison sont disponibles : à domicile, via un point relais, en magasin, et ainsi de suite. L'option « data-delivery-methods-url » du code ci-dessus permet à l'application Skeerel Pay de récupérer les modes de livraison disponibles. Pour davantage de personnalisation, le développeur pourra retenir d'autres variables, par exemple les dimensions ou le poids du colis. Il est également possible de définir des variables qui seront remplacées automatiquement par les informations de l'utilisateur.

__USER__ sera remplacé par l'identifiant Skeerel de l'utilisateur ce qui s'avère utile, en programme premium, pour savoir si la livraison doit-être gratuite par exemple.

__ZIP_CODE__ ; __CITY__ ; __COUNTRY__ afficheront respectivement le code postal, la ville et le pays de l'utilisateur.

Là encore, la bibliothèque Skeerel Pay simplifie le transfert des modes de livraison :

```
// Livraison standard à domicile
$deliveryMethodStandard = new DeliveryMethod();
$deliveryMethodStandard->setId("standard");
$deliveryMethodStandard->setType(Type::HOME());
$deliveryMethodStandard->setPrimary(true);
$deliveryMethodStandard->setName("Livraison Standard");
$deliveryMethodStandard->setDeliveryTextContent("Livraison le 12 juillet");
$deliveryMethodStandard->setPrice(499);

// Livraison en point relais
$deliveryMethodRelay = new DeliveryMethod();
$deliveryMethodRelay->setId("my_relay");
$deliveryMethodRelay->setType(Type::RELAY());
$deliveryMethodRelay->setName("Point relais");
$deliveryMethodRelay->setDeliveryTextContent("Disponible demain");
$deliveryMethodRelay->setPrice(299);

// Points relais
$pickUpPoint1 = new PickUpPoint();
$pickUpPoint1->setId("1");
$pickUpPoint1->setName("Point relais 1");
$pickUpPoint1->setAddress("Adresse 1");
$pickUpPoint1->setZipCode("Code postal");
$pickUpPoint1->setCity("Ville");
$pickUpPoint1->setCountry("Pays");

$pickUpPointsRelay = new PickUpPoints();
$pickUpPointsRelay->add($pickUpPoint1);

$deliveryMethodRelay->setPickUpPoints($pickUpPointsRelay);

// Récupération directement en magasin
$deliveryMethodCollect = new DeliveryMethod();
$deliveryMethodCollect->setId("store_collect");
$deliveryMethodCollect->setType(Type::COLLECT());
$deliveryMethodCollect->setName("Clic & collect");
$deliveryMethodCollect->setDeliveryTextContent("Disponible dans 2 heures");
$deliveryMethodCollect->setPrice(0);

// Points de collecte
$collectPoint1 = new PickUpPoint();
$collectPoint1->setId("1");
$collectPoint1->setName("Magasin 1");
$collectPoint1->setAddress("Adresse 1");
$collectPoint1->setZipCode($zip);
$collectPoint1->setCity($city);
$collectPoint1->setCountry($country);

$pickUpPointsCollect = new PickUpPoints();
$pickUpPointsCollect->add($collectPoint1);

$deliveryMethodCollect->setPickUpPoints($pickUpPointsCollect);

// Renvoi des méthodes de livraison
$deliveryMethods = new DeliveryMethods();
$deliveryMethods->add($deliveryMethodStandard);
$deliveryMethods->add($deliveryMethodRelay);
$deliveryMethods->add($deliveryMethodCollect);

echo $deliveryMethods->toJson();
```

Troisième étape : récupérer des informations de paiement

Lorsque l'internaute a validé son paiement, il est redirigé vers la page « data-redirect-url » définie dans le script de configuration. À cette url est ajouté un paramètre « token » permettant, en association à l'identifiant et à la clé secrète du site internet, de récupérer les informations de l'utilisateur et du paiement.

Parmi les informations récupérées :

- Les informations du profil utilisateur
- Les informations du paiement
- L'adresse de facturation
- Les informations de livraison

Voici un exemple de réponse complète : 9

```
{
  "status": "ok",
  "data": {
    "user": {
      "uid": "add15a68-e826-438b-a1f5-663ac7fdd797",
      "first_name": "John",
      "last_name": "Doe",
      "mail": "mail@domain.com",
      "mail_verified": true
    },
    "payment": {
      "id": "32b2fe1a-d987-487b-9fa1-e10964212e76",
      "live": true,
      "amount": 5000,
      "currency": "eur",
      "captured": true,
      "billing_address": {
        "name": "John Doe",
        "address": "1735 Joes Road",
        "address_line_2": "Building 2",
        "address_line3": "Second floor",
        "zip_code": 47161,
        "city": "NEW SALISBURY",
        "country_code": "US",
        "phone_number": +11234567890,
        "status": "LLC",
        "company_name": "MyCompany",
        "vat": "string",
      },
    "delivery": {
      "method_id": "my_method_id",
      "shipping_address": {
        "name": "John Doe",
        "address": "1735 Joes Road",
        "address_line_2": "Building 2",
        "address_line3": "Second floor",
        "zip_code": 47161,
        "city": "NEW SALISBURY",
        "country_code": "US",
        "phone_number": +11234567890,
        "status": "LLC",
        "company_name": "MyCompany",
        "vat": "string",
      },
      "pick_up_point_id": "my_pick_up_point_id"
    }
  }
}
```

9

La bibliothèque permet de gérer toutes ces informations sans effort, comme le montre le traitement conditionnel suivant:

```
if (\Skeerel\Skeerel::verifyAndRemoveSessionStateParameter($_GET['state']
    $skeerel = new \Skeerel\Skeerel('YOUR_WEBSITE_ID', 'YOUR_WEBSITE_SECR
    $data = $skeerel->getData($_GET['token']);
}
```

Notez la première étape de vérification et de suppression du jeton de session, capitale pour éviter les attaques par rejeu ou de type « Cross-site request forgery ».

Les actions permises via l'API Skeerel Pay

- Permettre aux utilisateurs de payer sur le site marchand
- Récupérer les informations d'un paiement a posteriori

```
$skeerel = new \Skeerel\Skeerel('YOUR_WEBSITE_ID', 'YOUR_WEBSITE_SECRET');
$payment = $skeerel->getPayment("ce365eec-c287-43b6-ad38-25d8d9029fd1");
```

- Lister les paiements effectués via Skeerel Pay

```
$skeerel = new \Skeerel\Skeerel('YOUR_WEBSITE_ID', 'YOUR_WEBSITE_SECRET');
$payment = $skeerel->listPayments(); //10 derniers paiements
$payment = $skeerel->listPayments(true); //Seulement les paiement en production
$payment = $skeerel->listPayments(true, 15, 20); //lister 20 paiements en production à partir du 15ème paiement
```

- Rembourser partiellement ou en totalité un paiement effectué via Skeerel Pay

```
$skeerel = new \Skeerel\Skeerel('YOUR_WEBSITE_ID', 'YOUR_WEBSITE_SECRET');
$skeerel->refundPayment("32b2fe1a-d987-487b-9fa1-e10964212e76"); //remboursement total
$skeerel->refundPayment("32b2fe1a-d987-487b-9fa1-e10964212e76", 100); // remboursement partiel
```

Un modèle économique par commission

Skeerel Pay n'exige aucun frais de mise en service et ne prélève aucun frais lorsqu'une transaction n'aboutit pas. L'éditeur ArcanSecurity capte uniquement un pourcentage sur chaque transaction réussie. Il développe à présent les modules de paiement pour les plateformes e-commerce les plus répandues. Son modèle économique repose sur la confiance du cybermarchand et sur celle de l'acheteur distant. Ce dernier peut toutefois décider de n'enregistrer aucune carte bancaire sur son application et régler l'ensemble de ses achats via Google Pay ou Apple Pay. •

Nicolas Bouteillier
$Nicolas = ['Développeur', 'technicien', 'artisan', 'entrepreneur', 'geek']; Nicolas est développeur depuis 2002, CTO @Akopso, il a accompagné en mode lean startup plusieurs démarrages de projets, gérant de sa SARL, autoentrepreneur, salarié, c'est un touche à tout riche d'expériences variées qui a accosté sur les rives de l'agilité sans jamais quitter le monde entrepreneurial. Son fil conducteur est «Kaizendo», la voie de l'amélioration continue.
Blog: https://www.kaizendo.fr
Pour les pro du tourisme: https://www.akopso.com - https://fr.linkedin.com/in/kaizendo - https://twitter.com/NicolasKaizen

Passons à la pratique !

Pour illustrer ce dossier sur le paiement en ligne, j'ai choisi «Lyra» qui propose entre autres la solution «PayZen» bien connue de tous les e-commerçants clients de banques françaises. Cette solution est celle proposée notamment par la Banque Populaire sous la marque «SystemPay». L'autre solution largement distribuée en marque blanche par les banques françaises est SIPS-ATOS.

niveau 200

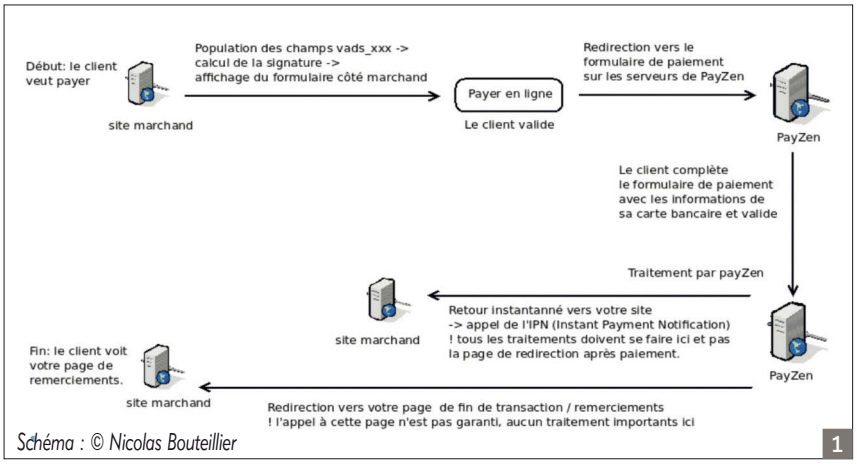


Schéma : © Nicolas Bouteillier 1

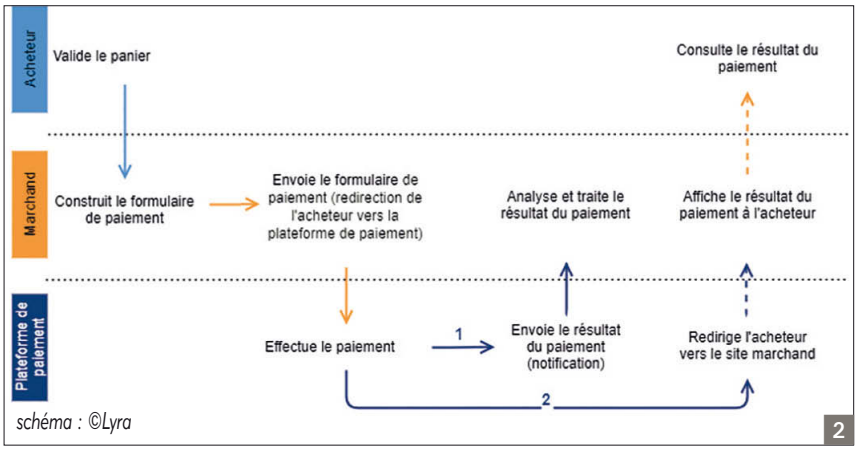


schéma : ©Lyra 2

ATTENTION : ***ce code n'est pas à utiliser en production, il faut ajouter des vérifications sur les valeurs de configuration, sur les valeurs transmises par les formulaires, utiliser des protections contre les attaques CSRF … Notre but est d'illustrer les principes d'une transaction avec du code fonctionnel et concret.***

Types de paiements

Il existe plusieurs manières d'implémenter le paiement en ligne, ils sont détaillés dans la doc de «PayZen» :

- Paiement embarqué / Javascript ;
- Paiement par iFrame ;
- Webservices REST de paiement ;
- Paiement par page de redirection.

Nous allons voir la quatrième, à mon avis la plus répandue pour les petits et moyens projets. De plus les acheteurs y sont habitués et sa mise en œuvre est relativement simple.
Principe de fonctionnement simplifié : 1 2

Un peu de configuration

Pour notre exemple, la configuration est faite dans un fichier au format « INI » qui est ensuite lu avec la fonction :

```
parse_ini_file("config.ini");
```

Vous pouvez évidemment utiliser le format qui vous convient le mieux, pour ma part, j'ai une préférence pour le YAML.

```
; 1 : Permet d'afficher les champs qui seront envoyés à la plateforme.
; 0 : Redirection automatique vers la page de paiement.
debug = 1

; Identifiant Boutique à récupérer dans le Back office de la solution de paiement.
vads_site_id = 11111111

; Certificat à récupérer dans le Back office de la solution de paiement.
; Attention ce certificat est différent en fonction de vads_ctx_mode, TEST ou PRODUCTION.
; Le certificat n'est pas envoyé à la plateforme de paiement mais intervient dans le calcul de la signature.
TEST_key = 2222222222222222
PROD_key = 3333333333333333

; mode: TEST ou PRODUCTION
vads_ctx_mode = TEST

; Obligatoire, doit être = V2.
vads_version = V2

; Langue dans laquelle s'affiche la page de paiement : fr | en
vads_language = fr

; Code devise. 978 pour EURO.
vads_currency = 978

; Obligatoire, doit être = PAYMENT.
vads_page_action = PAYMENT
; Obligatoire
vads_payment_config = SINGLE

; Obligatoire
; INTERACTIVE : saisie des informations de la carte réalisée sur la plateforme de paiement.
```

```
; IFRAME : saisie des informations de la carte sur une page de paiement simplifiée et allégée que
 le marchand peut imbriquer dans la page web de son choix.
vads_action_mode = INTERACTIVE

; Url de retour à la boutique. Lorsque le client clique sur "retourner à la boutique"
; Peut être configurée dans le Back office marchand
vads_url_return = https://mon-site.tld/paiement/thank_you_so_much.php

; Facultatif, vous permet de récupérer les informations de la transaction à des fins d'informations
 client : GET | POST | NONE/EMPTY/UNSET
vads_return_mode = POST

; Durée avant un retour automatique à la boutique pour un paiement accepté : 0 < X < 600
 secondes
vads_redirect_success_timeout = 1

; Message sur la page de paiement avant le retour automatique à la boutique dans le cas d'un
 paiement accepté.
vads_redirect_success_message = Redirection vers la boutique dans quelques instants

; Durée avant un retour automatique à la boutique pour un paiement échoué : 0 < X < 600
 secondes
vads_redirect_error_timeout = 1


; Message sur la page de paiement avant le retour automatique à la boutique dans le cas d'un
 paiement échoué.
vads_redirect_error_message = Redirection vers la boutique dans quelques instants
; Obligatoire. Elle est calculée.
;signature =

; Obligatoire, date de la transaction exprimée sous la forme AAAAMMJJHHMMSS
; sur le fuseau UTC=0. Cette valeur sera calculée.
;vads_trans_date =

; Obligatoire, identifiant de la transaction. Cet identifiant doit être :
; - unique sur une même journée.
; - sa longueur est obligatoirement de 6 caractères.
; - Sa valeur est doit être comprise entre 000000 et 899999.
; Calculé comme vous voulez en respectant les contraintes ci-dessus
;vads_trans_id =
```

Enfin du code !

Parler c'est bien, mais coder c'est mieux.
Vous devez afficher les informations sur le paiement à votre client et vous assurer qu'elles soient valides, vous avez besoin de :

- Langue (lang)
- Numéro client (vads_cust_id)
- Nom (vads_cust_name)
- Adresse (vads_cust_address)
- Code postal (vads_cust_zip)
- Ville (vads_cust_city)
- Pays (vads_cust_country)
- Téléphone (vads_cust_phone)
- Email (vads_cust_email)
- Montant du paiement (vads_amount)

D'autres champs sont disponibles => RTFM.
Ci-dessous le code HTML/CSS (très simpliste). Chaque information est affichée dans un champ, ce qui va vous permettre de faire différents tests en changeant facilement les valeurs. Cette étape correspond à la validation du panier.

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8">
    <style>
      label {line-height: 2em;}
      .container {
        width: 400px;
        height:200px;
        margin:0 auto;
        padding-top: 50px;
      }
      .submit {
        line-height: 2em;
        margin-top:10px;
        border: 1px solid green;
        background-color: green;
        width: 100%;
        color: #ffffff;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <form method="POST" action="form_payment.php">
        <input type="hidden" name="lang" value="fr">

        <label for="vads_order_id">Identifiant marchand de la commande (vads
_order_id)</label>
        <input type="text" name="vads_order_id" value="123456" size=20>
        <br>

        <h4>Informations client:</h4>
        <label for="vads_cust_id">Numéro client (vads_cust_id)</label>
        <input type="text" name="vads_cust_id" value="2380" size=20>
        <br>

        <label for="vads_cust_name">Nom (vads_cust_name)</label>
        <input type="text" name="vads_cust_name" value="Gandalf Le Blanc " size=20>
        <br>

        <label for="vads_cust_address">Adresse (vads_cust_address)</label>
        <input type="text" name="vads_cust_address" value="Middleearth" size=20>
        <br>

        <label for="vads_cust_zip">Code postal (vads_cust_zip)</label>
        <input type="text" name="vads_cust_zip" value="5555" size=20>
        <br>

        <label for="vads_cust_city">Ville (vads_cust_city)</label>
```

```
      <input type="text" name="vads_cust_city" value="Fondcombe" size=20>
      <br>

      <label for="vads_cust_country">Pays (vads_cust_country)</label>
      <input type="text" name="vads_cust_country" value="FR" size=20>
      <br>

      <label for="vads_cust_phone">Téléphone (vads_cust_phone)</label>
      <input type="text" name="vads_cust_phone" value="0606060606" size=20>
      <br>

      <label for="vads_cust_email">Email (vads_cust_email)</label>
      <input type="text" name="vads_cust_email" size=20>
      <br>

      <label for="vads_amount">Montant du paiement (vads_amount)</label>
      <input type="text" name="vads_amount" value="1000" size=20>
      <br>

      <button class="submit" type="submit" class="validationButton">
        Confirmer les informations et payer ma commande
      </button>
    </form>
  </div>
 </body>
</html>
```

Création du formulaire de paiement et redirection vers «PayZen»

Lorsque votre client a validé ses données avec le montant du paiement qu'il s'apprête à effectuer (son panier), vous allez utiliser ces informations et les valeurs des champs de votre configuration pour générer toutes les données nécessaires à la solution de paiement.
Vous allez construire un formulaire qui sera automatiquement posté par un petit bout de Javascript pour éviter une action à l'utilisateur.
Si vous avez configuré «debug = 1», l'envoi automatique est désactivé et les valeurs passées sont affichées.
Le formulaire précédent appelle la page « form_payment.php».
Les fonctions nécessaires sont séparées dans un fichier «functions.php» qui est inclus dans les différentes pages de traitement.

>>> Fichier functions.php

Code complet sur programmez.com & github

IPN (Instant Payment Notification) et page de retour

Ces deux URL doivent être accessibles depuis les serveur de «PayZen».
Comme exemple je vous propose un code simple qui vérifie la signature et qui affiche l'ensemble des données reçues en POST.
Important : tous les traitements doivent être faits via l'URL IPN, car le serveur l'appelle systématiquement, c'est avec ce script que vous devez traiter les données du paiement comme le statut, notifier vos utilisateurs, gérer vos stocks …
L'URL «vads_url_success» ne sera appelée que si l'utilisateur clique sur « Retourner à la boutique» ou si il est redirigé automatiquement après son achat.
Elle va vous permettre de remercier votre client, lui proposer le suivi de sa commande, une promo pour son prochain achat, le rassurer, bref, terminer la relation client sur votre site après son passage chez «PayZen» pour le paiement.

```
<?php
include 'functions.php';
echo '
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Merci pour votre achat</title>
 </head>
 <body>
  <div>
  ';
  $confTxt = parse_ini_file('config.ini');
  if ('TEST' == $conf_txt['vads_ctx_mode']) {
    $key = $conf_txt['TEST_key'];
  } else {
    die('Que pour les tests on a dit !');
  }
  $control = CheckSignature($_POST, $key);
  if ('true' != $control) {
    // Gérer l'erreur, afficher une 404, prévenir un admin par mail => stopper l'exécution ici ...
  }
  // Seulement quelques exemples de statuts possibles
  if (isset($_REQUEST['vads_trans_status'])) {
    switch ($_REQUEST['vads_trans_status']) {
      case 'ABANDONED':
        echo 'Paiement abandonné';
        break;
      case 'AUTHORISED':
        echo 'Paiement accepté';
        break;
      case 'REFUSED':
        echo 'Paiement refusé';
        break;
      case 'CANCELLED':
        echo 'Paiement annulé';
        break;
      default:
        // Gérer l'erreur, code inconnu = comportement anormal, stopper l'exécution
        break;
    }
  } else {
    // Pas de statut dans le retour : gérer l'erreur, afficher une 404, prévenir un admin par mail
=> stopper l'exécution ici ...
  }
  // pendant le dév vous pouvez tout afficher
  foreach ($_POST as $nom => $valeur) {
    if ('vads_' == substr($nom, 0, 5)) {
      echo "$nom = $valeur <br/>";
    }
```

```
  }
 echo '</div>
 </body>
</html>
';
```

Comment intégrer à un CMS

«PayZen», comme d'autres, fournissent gratuitement des modules de paiement pour bon nombre de CMS, open source et propriétaires, ce qui vous permet de facilement intégrer leur solution à votre projet : https://payzen.io/fr-FR/module-de-paiement-gratuit/ 3

Sécurité, code et paranoïa

Une dernière fois avant la fin, ce code est proposé à titre d'exemple, pour que vous puissiez vous faire une idée de la manière d'implémenter une solution de paiement comme «PayZen», mais il ne doit pas être utilisé tel quel en production.

Vous devez systématiquement valider et vérifier les entrées des formulaires, vérifier la cohérence des données, leur type (string, int …), les dates (passé, présent …), utiliser les techniques de protections de vos formulaires comme les jetons CSRF.

N'oubliez pas que Murphy règne en maître sur nos destins et qu'un jour ou l'autre si vous laissez la moindre chance que quelque chose se passe mal, ça arrivera !

Plus sérieusement, l'expérience d'achat est une étape cruciale pour votre business, votre client doit être rassuré, chouchouté, guidé, vous devez lui expliquer ce qui se passe, quand et comment ça se passe, rien n'est plus frustrant qu'un code d'erreur obscur ou une simple page blanche, et lorsqu'il est question de paiement ça devient vite angoissant.

Un acheteur qui a vécu une mauvaise expérience d'achat aura plus de mal à repasser commande, c'est potentiellement un client perdu.

3

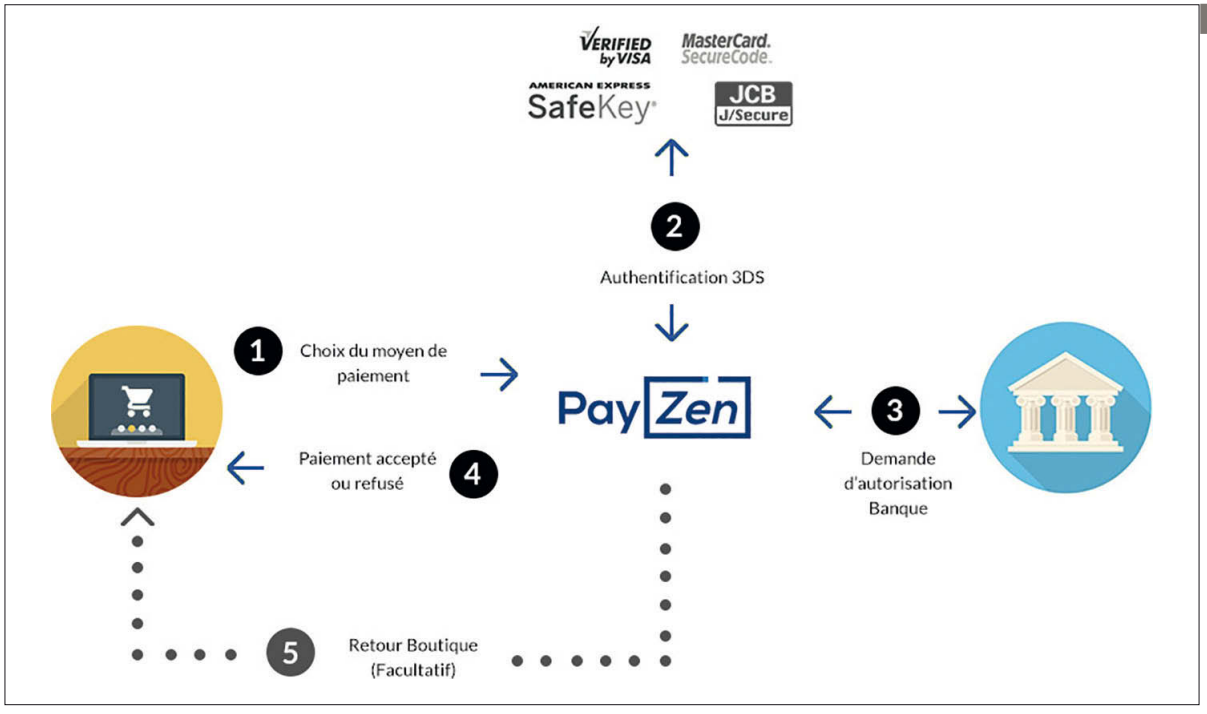
Pour aller plus loin

Je ne vous ai montré ici que quelques-unes des possibilités de paiement, que quelques-uns des champs et des différentes configurations possibles. Vous pouvez par exemple avoir une URL de retour spécifique si la transaction est « success », « refused », « abandonned» ce qui vous permet de bien adapter votre discours client. 4

Le paiement en plusieurs fois, la sécurité «3D secure», les paiements différés… Vous devez prendre le temps de bien lire la documentation et savoir ce qu'il est possible de faire avant de vous lancer, vous gagnerez du temps pendant votre implémentation.

Voici les liens pour vous aider à aller plus loin :
https://lyra.com
https://payzen.eu/
https://payzen.io/fr-FR/form-payment/quick-start-guide/sitemap.html

Le code source sur lequel je me suis basé pour ces exemples :
https://payzen.io/fr-FR/code/



4 *Exemple de fonctionnement d'un service de paiement avec PayZen*

Philippe BOULANGER
Manager des expertises C/C++ et Python
www.invivoo.com

PYTHON ET LE GUI : wxPython

Partie 1

Beaucoup de personnes partent du principe que Python n'est qu'un langage de script. Elles le cantonnent souvent à l'automatisation des tâches. Grâce à des frameworks, on peut aussi s'en servir pour faire des clients lourds. Le module tkInter permet de faire des boîtes de dialogue assez facilement. Il a longtemps été utilisé par les ingénieurs systèmes pour développer des mini-applications et faciliter leur travail. Il existe des portages de GTK, Qt et wxWidgets pour Python qui permettent de faire de belles applications. Spyder, par exemple, est un IDE pour Python écrit en Python avec du Qt. Dans cet article nous allons travailler avec wxPython, portage de wxWidgets ; une bibliothèque que j'ai utilisée à de nombreuses reprises pour faire un démonstrateur ou pour faire des outils internes pour des clients.

niveau 100/200

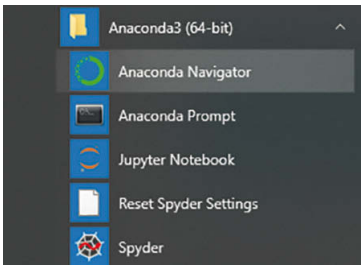
Installation

La bibliothèque wxPython est disponible sur le site www.wxPython.org Vous y trouverez toutes les informations utiles pour l'installation ainsi que la documentation en ligne. 1

Sous Windows et Mac, l'installation peut se faire via l'utilitaire Pip grâce à la commande suivante :

```
pip install -U wxPython
```

Sous Windows, si vous utilisez Anaconda, vous pouvez utiliser l'outil de gestion de package fourni par cette distribution : Anaconda Navigator :



Suivant la façon dont est installé votre PC, il sera peut-être nécessaire d'exécuter Anaconda Navigator en mode Administrateur afin de permettre la mise à jour.

Pour Linux, compte tenu des nombreuses distributions, l'installation peut s'avérer plus complexe :

```
pip install -U \
  -f https://extras.wxpython.org/wxPython4/extras/linux/gtk3/ubuntu-16.04 \
  wxPython
```

Il faut alors bien choisir sa version en fonction de la distribution cible.

Les extras ?

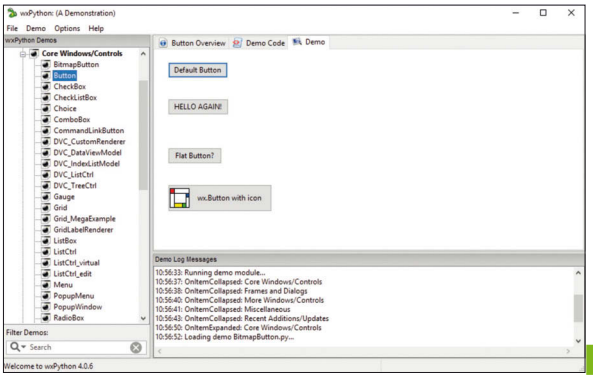
À chaque nouvelle version de wxPython, vous trouverez un lien pour accéder aux extras. 2

L'un des gros intérêts de wxPython est l'application « wxPython-demo » présente dans ces extras. Voici un aperçu de l'application : 3

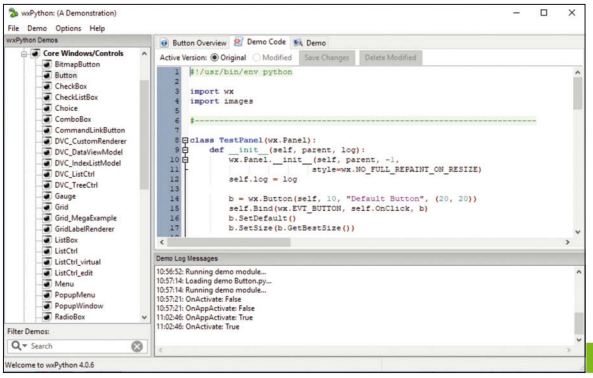
Cette application est écrite en Python avec wxPython. Elle permet de naviguer dans toutes les fonctionnalités de la bibliothèque, de voir des exemples des contrôles graphiques et d'accéder au code. 4

1

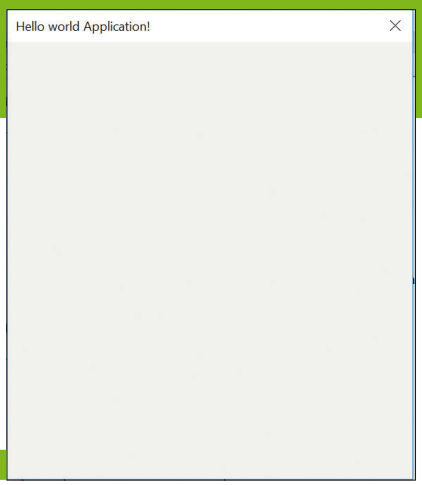
2



3



4



5

Cela permet de développer plus rapidement son application sans avoir besoin de lire complètement la documentation.
Extras contient aussi la documentation offline pour ceux qui travaillent souvent en déplacement sans avoir une connexion internet.

Outils ?

Créer des interfaces utilisateurs peut être compliqué. Il existe des outils pour faciliter leur création. Pour wxPython, vous pouvez utiliser wxFormBuilder ou wxGlade qui sont, tous deux, gratuits et téléchargeables sur le net.

« HELLO WORLD »

Avant de se lancer dans l'écriture d'une application plus complexe, nous allons commencer par créer une application 'boîte de dialogue' qui affiche le message « Hello World ». Puis nous y rajouterons un bouton, ce qui nous permettra de comprendre le positionnement des contrôles.

Au commencement, l'application

Avant toute chose, il faut créer une instance de wx.App qui fournira les fonctionnalités de gestion des évènements : sans cette instance, rien ne fonctionnera.

```
import wx

class MyApplication( wx.App ):

  def OnInit( self ):
    print( "MyApplication.OnInit" )
    self.SetAppName( "HelloWordApp" )
    return True


if __name__ == '__main__':
  app = MyApplication()
  app.MainLoop()
```

Pour l'instant, ce programme s'exécute mais n'affiche rien. La fonction *OnInit* de notre classe est appelée pour initialiser le GUI après le constructeur : elle doit retourner ***True*** pour que l'application puisse s'initialiser proprement.

Créer une boîte de dialogue

Une fois l'application créée, il faut afficher une fenêtre. Le type de fenêtre le plus simple que l'on puisse utiliser est la boîte de dialogue wx.Dialog : nous allons créer une classe qui en dérive.

6

```
class MyDialog( wx.Dialog ):
  def __init__( self,
          parent,
          id,
          title,
          size  = wx.DefaultSize,
          pos   = wx.DefaultPosition,
          style = wx.DEFAULT_DIALOG_STYLE,
          name  = 'My dialog' ):
    wx.Dialog.__init__( self )
    self.Create( parent, id, title, pos, size, style, name )
```

'parent' désigne l'objet parent de la boîte de dialogue : la première fenêtre n'a pas de parent, on lui passe ***None***. 'title' contient le nom qui sera affiché dans la barre de titre. 'id' est un identifiant unique qui désigne une fenêtre, cela permet d'aiguiller les évènements aux bons composants. Si on définit 'id' à -1, on demande à wxPython de générer automatiquement un identifiant.
Pour permettre à l'application d'afficher la boîte de dialogue il faut modifier la fonction OnInit :

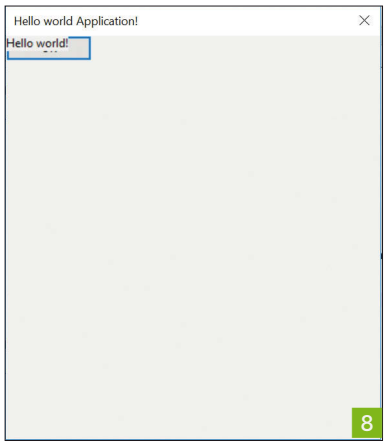
```
class MyApplication( wx.App ):

  def OnInit( self ):
    # initialize
    print( "MyApplication.OnInit" )
    self.SetAppName( "HelloWordApp" )

    # create the dialog box
    dlg = MyDialog( None, -1, "Hello world Application!" )
    print( "before ShowModal" )
    dlg.ShowModal()
    print( "after ShowModal" )
    dlg.Destroy()
    return True
```

La première étape consiste à créer l'objet MyDialog et de stocker l'instance dans 'dlg'. Cela ne suffit pas à déclencher l'affichage, il faut appeler la fonction ShowModal. Tant que la boîte n'est pas fermée, la ligne après n'est pas exécutée : le ShowModal est une action bloquante. Une fois la boîte fermée, il faut penser à supprimer les objets systèmes liés en appelant la fonction Destroy.
Lorsque l'on exécute le programme on obtient : 5

Et mon « Hello World »

Maintenant que ma boîte de dialogue s'affiche, il nous faut rajouter un champ texte avec « Hello World » à l'intérieur. En regardant dans l'application wxPython-demo, on trouve ceci : 6
wx.StaticText est un contrôle qui nous permet d'afficher un texte non-modifiable. Il nous faut ajouter un appel dans la fonction '__init__' de MyDialog :

```
class MyDialog( wx.Dialog ):
```




```
def __init__( self,
        parent,
        id,
        title,
        size  = wx.DefaultSize,
        pos   = wx.DefaultPosition,
        style = wx.DEFAULT_DIALOG_STYLE,
        name  = 'My dialog' ):
    wx.Dialog.__init__( self )
    self.Create( parent, id, title, pos, size, style, name )
    wx.StaticText( self, -1, 'Hello world !' )
```

Ce qui nous permet d'obtenir : 7

Rajoutons un bouton OK et un bouton Cancel

En recherchant dans wxPython-demo le bon contrôle, nous trouverons wx.Button. Et son utilisation est assez simple :

```
class MyDialog( wx.Dialog ):
  def __init__( self,
          parent,
          id,
          title,
          size  = wx.DefaultSize,
          pos   = wx.DefaultPosition,
          style = wx.DEFAULT_DIALOG_STYLE,
          name  = 'My dialog' ):
    wx.Dialog.__init__( self )
    self.Create( parent, id, title, pos, size, style, name )
    wx.StaticText( self, -1, 'Hello world !' )
    okButton     = wx.Button( self, -1, 'OK' )
    cancelButton = wx.Button( self, -1, 'Cancel' )
```

A l'affichage on obtient : 8

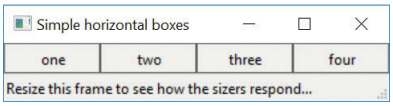
Tous les contrôles s'affichent au même endroit. Outre le fait que l'interface est illisible, il est impossible d'appuyer sur les boutons.

Positionnement des contrôles et gestion des évènements

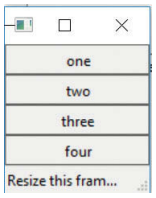
Pour positionner les contrôles les uns par rapport aux autres, wxPython propose les 'sizer' :

- wx.Sizer : une classe de base
- wx.BoxSizer :

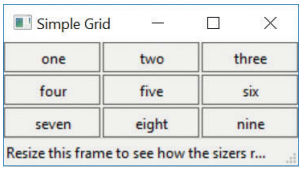
c'est une 'boîte' qui permet de ranger les contrôles les uns après les autres soit horizontalement :



soit verticalement :



wx.GridSizer : c'est une grille de contrôles



Mettons en œuvre les sizers :

```
import wx

class MyDialog( wx.Dialog ):
  def __init__( self,
          parent,
          id,
          title,
          size  = wx.DefaultSize,
          pos   = wx.DefaultPosition,
          style = wx.DEFAULT_DIALOG_STYLE,
          name  = 'My dialog' ):
    wx.Dialog.__init__( self )
    self.Create( parent, id, title, pos, size, style, name )
    staticText   = wx.StaticText( self, -1, "Hello world!" )
    okButton     = wx.Button( self, wx.ID_OK, "OK" )
    cancelButton = wx.Button( self, wx.ID_CANCEL, "Cancel" )

    self.Bind( wx.EVT_BUTTON, self.OnOK, okButton )
    self.Bind( wx.EVT_BUTTON, self.OnCancel, cancelButton )

    topSizer = wx.BoxSizer( wx.VERTICAL )
    topSizer.Add( staticText,  0, wx.EXPAND )
    hSizer = wx.BoxSizer( wx.HORIZONTAL )
    hSizer.Add( okButton,    0, wx.EXPAND )
    hSizer.Add( cancelButton, 0, wx.EXPAND )
    topSizer.Add( hSizer, 0, wx.EXPAND )
    self.SetSizer( topSizer )
    topSizer.Fit( self )


  def OnOK( self, event ):
    print( "OnOK" )
    self.EndModal( wx.ID_OK )
```

```
  def OnCancel( self, event ):
    print( "OnCancel" )
    self.EndModal( wx.ID_CANCEL )


class MyApplication( wx.App ):

  def OnInit( self ):
    # initialize
    print( "MyApplication.OnInit" )
    self.SetAppName( "HelloWordApp" )

    # create dialog box
    dlg = MyDialog( None, -1, "Hello world Application!" )
    print( "before ShowModal" )
    res = dlg.ShowModal()
    print( "after ShowModal" )
    if res == wx.ID_OK:
      print( "exit OK" )
    elif res == wx.ID_CANCEL:
      print( "exit CANCEL" )
    else:
      print( "exit %d" % res )
    dlg.Destroy()
    return True


if __name__ == '__main__':
  app = MyApplication()
  app.MainLoop()
```

'self.Bind' permet de rediriger les évènements. Dans le cas des boutons, il y a 3 paramètres :

- wx.EVT_BUTTON : l'identifiant de l'évènement « appui sur un bouton » ;
- la fonction qui doit recevoir l'évènement ;
- l'objet wx.Button dont on veut capter l'appui.

Les handlers d'évènements OnOK et OnCancel servent à clore la boîte de dialogue. Pour ce faire il faut appeler la fonction self.EndModal en lui passant l'identifiant de l'évènement.

L'ajout de contrôles dans les sizers se fait via la fonction 'Add' ; les paramètres d'appel sont :

- le contrôle à ajouter ;
- 'proportion' : un entier qui définit la proportion ;
- 'flag' : un masque d'options ;
- 'border' : l'épaisseur de la bordure ;
- 'userData' : une donnée utilisateur que l'on peut attacher.

Le sizer le plus externe (celui qui contient tous les contrôles) doit être ajouté à la fenêtre via la fonction 'SetSizer'. Pour ajuster la taille de chaque contrôle ainsi que celle de la fenêtre, nous appelons la fonction 'Fit' sur le sizer le plus externe en lui passant en paramètre l'objet de la fenêtre.

PASSONS AUX CHOSES SERIEUSES

Après cette mise en bouche avec une application relativement simple, nous allons créer une application qui permet de visualiser des images. Mon objectif est d'avoir un explorateur de fichiers qui me permette de parcourir les fichiers du disque dur. Si je double-clique sur un fichier « image » celle-ci s'affichera dans une fenêtre et nous pouvons avoir plusieurs images ouvertes en même temps.

Il se trouve que wxPython propose une classe wx.Image qui supporte plusieurs formats de fichiers bitmaps (BMP, PNG, JPEG, GIF, ICO, TGA, TIFF, etc…). Par contre, pour afficher une image nous utiliserons la classe wx.StaticBitmap qui est un contrôle.

Afin de se rapprocher d'un design plus professionnel, nous ajouterons une barre d'outils ainsi qu'une barre de menu. Cela nous permettra de comprendre leurs créations ainsi que la gestion des évènements associés.

```
# -*- coding: utf-8 -*-
import os
import wx
import wx.adv

ID_Menu_New  = 5000
ID_Menu_Open = 5001
ID_Menu_Exit = 5002

wildcard = "Bitmap files (*.bmp)|*.bmp|"          \
      "JPEG files (*.jpg, *jpeg)|*.jpg, *.jpeg|" \
      "PNG files (*.png)|*.png|"           \
      "GIF files (*.gif)|*.gif|"           \
      "Icon files (*.ico)|*.ico|"          \
      "Targa files (*.tga)|*.tga|"         \
      "TIFF files (*.tif,*tiff)|*.tif,*tiff|"  \
      "All files (*.*)|*.*"



class MyParentFrame( wx.MDIParentFrame ):

  def __init__( self ):
    wx.MDIParentFrame.__init__( self,
                  None,
                  -1,
                  "MDI Parent",
                  size = (600,400),
                  style = wx.DEFAULT_FRAME_STYLE | wx.HSCROLL | wx.VSCROLL )
    self.create_menu_bar()
    self.create_toolbar()



  def create_menu_bar( self ):
    # create the "File" menu
    menuFile = wx.Menu()
    menuFile.Append( ID_Menu_New,  "&New Window" )
    menuFile.Append( ID_Menu_Open, "&Open file" )
    menuFile.AppendSeparator()
    menuFile.Append( ID_Menu_Exit, "E&xit" )

    # create the menu bar
    menubar = wx.MenuBar()
    menubar.Append( menuFile, "&File" )
```

```
    self.SetMenuBar( menubar )

    # bind the events
    self.Bind( wx.EVT_MENU, self.OnNewWindow, id = ID_Menu_New )
    self.Bind( wx.EVT_MENU, self.OnOpenFile,  id = ID_Menu_Open )
    self.Bind( wx.EVT_MENU, self.OnExit,     id = ID_Menu_Exit )



def create_toolbar( self ):
    # create the toolbar
    tsize = ( 32, 32 )
    tb   = self.CreateToolBar( True )
    tb.SetToolBitmapSize( tsize )

    # new window
    new_bmp =  wx.ArtProvider.GetBitmap( wx.ART_NEW, wx.ART_TOOLBAR, tsize )
    tb.AddTool( ID_Menu_New,
            "New",
            new_bmp,
            wx.NullBitmap,
            wx.ITEM_NORMAL,
            "New",
            "Long help for 'New'",
            None )

    # open file
    open_bmp = wx.ArtProvider.GetBitmap( wx.ART_FILE_OPEN, wx.ART_TOOLBAR, tsize )
    tb.AddTool( ID_Menu_Open,
            "Open",
            open_bmp,
            wx.NullBitmap,
            wx.ITEM_NORMAL,
            "Open",
            "Long help for 'Open'",
            None )

    # display the toolbar
    tb.Realize()

    # bind the events
    self.Bind( wx.EVT_TOOL, self.OnNewWindow, id = ID_Menu_New )
    self.Bind( wx.EVT_TOOL, self.OnOpenFile,  id = ID_Menu_Open )



def OnNewWindow( self, event ):
    win   = wx.MDIChildFrame( self, -1, "Child Window" )
    canvas = wx.ScrolledWindow( win )
    win.Show( True )



def OnOpenFile( self, event ):
    # choose the file
    dlg = wx.FileDialog( self,
                message    = "Choose a file",
                defaultDir  = os.getcwd(),
                defaultFile = "",
                wildcard    = wildcard,
                style      = wx.FD_OPEN | wx.FD_MULTIPLE | wx.FD_CHANGE_DIR |
                        wx.FD_FILE_MUST_EXIST | wx.FD_PREVIEW )
    if dlg.ShowModal() == wx.ID_OK:
        for path in dlg.GetPaths():
            self.read_file( path )
    dlg.Destroy()



def OnExit( self, event ):
    self.Close( True )



def read_file( self, filename ):
    # read image if possible
    try:
        image = wx.Image( filename, wx.BITMAP_TYPE_ANY )
    except:
        return

    # create the window
    win    = wx.MDIChildFrame( self, -1, filename )
    canvas  = wx.ScrolledWindow( win )
    sizer   = wx.BoxSizer( wx.HORIZONTAL )
    statBmp = wx.StaticBitmap( canvas,
                        wx.ID_ANY,
                        image.ConvertToBitmap() )
    sizer.Add( statBmp, 1, wx.EXPAND )
    canvas.SetSizer( sizer )
    sizer.Fit( canvas )
    win.Show( True )



class MyApp( wx.App ):
    def OnInit( self ):
        frame = MyParentFrame()
        frame.Show( True )
        self.SetTopWindow( frame )
        return True



if __name__ == '__main__':
    app = MyApp( False )
    app.MainLoop()
```

CONCLUSION

wxPython est une bibliothèque qui vous permettra de développer rapidement des applications avec un look&feel professionnel. Elle est très intéressante tant pour faire un prototype/démonstrateur que pour faire une application interne. wxPython-demo est un formidable outil qui permet de rapidement trouver les contrôles qui correspondent à notre besoin ainsi que le code qui permet de les construire. •

Pierre-Yves PAMART
Scientific Computing Expert
& Software Gardener
@ Safran Aircraft Engines

SAFRAN

Adoptez une structure de dépôt standardisée pour votre projet Python !

Chers Pythonistas, ne vous est jamais-t-il arrivé de vous demander s'il existait des bonnes pratiques vis-à-vis du placement de vos fichiers et dossiers contenant du code, de la documentation, des tests ou encore de ce fameux fichier README et d'autres fichiers divers et variés ? Cet article vise à faire le point en proposant des bonnes pratiques commentées sur la structure d'un dépôt Python classique. La génération automatique fera l'objet d'un prochain article. Il ne vous sera pas nécessaire d'avoir de prérequis particulier.

niveau 200

Les bonnes pratiques de développement, que ce soit en Python ou pour tout autre langage, concernent également ce qui n'est pas à proprement dit du code source. C'est le cas de la structure du dépôt, c'est-à-dire l'organisation de ses fichiers et dossiers constitutifs. Quelles que soient la taille et l'ambition d'un projet, respecter des bonnes pratiques de structure est indispensable. En effet, elle doit être vue comme une composante de la documentation, qui, standardisée aidera considérablement les nouveaux venus sur votre projet. Tout un chacun pourra alors être rapidement opérationnel car il sera aisé de savoir où placer les choses et de retrouver les informations. Avant de passer au vif du sujet, précisons que ce qui est vu par la suite est valable quel que soit le projet : petit, moyen ou gros. Même si une logique est imposée par certains frameworks, comme *Django* par exemple, elle reste tout à fait compatible avec ce qui suit. En général c'est la structuration du côté du code source, voire de l'utilisation de ressources, qui est imposée par ces frameworks. Le contenu de cet article est une compilation des bonnes pratiques Python reconnues, certains passages sur l'organisation des dépôts sont empruntés au guide Python(1) de Kenneth Reitz(2) & Real Python(3).

Préliminaires

Glossaire

Afin de clarifier la suite des propos, comprenez un module comme un fichier Python contenant du code source, et un package comme du dossier regroupant un ou plusieurs modules. Un dossier pour être transformé en package contiendra un fichier, même vide, nommé __init__.py.

Enfin, un dépôt est ici l'ensemble des fichiers et dossiers de votre projet Python, c'est-à-dire le code source et toutes autres ressources tels que par exemple la documentation ou les tests. Le dépôt est ce que vous devez versionner.

Encodage des fichiers

Les fichiers contenant du texte lisible par l'homme sont en réalité stockés sous forme de fichiers binaires composés d'octets. Alors comment l'ordinateur fait-il le lien entre octets avec des chiffres et des lettres ? Simplement grâce à une table de correspondance dite d'*encodage des caractères*. Il en existe de nombreuses, qui sont généralement nées de l'histoire de l'informatique et des liens envers les différentes langues.

Toutefois, en termes de développement informatique la bonne pratique est d'utiliser un standard international, en particulier le *Universal character set Transformation Format* sur 8 bits : **UTF-8**.

L'ISO l'a développé comme répertoire universel de caractères codés sous la norme internationale ISO/CEI 10646, et est aujourd'hui compatible avec le standard Unicode. UTF-8 reste compatible avec la norme ASCII limitée à l'anglais de base.

Une erreur d'encodage se manifeste généralement avec des artefacts. Par exemple, le pangramme de Gilles Esposito-Farèse encodé en UTF-8 : "*Dès Noël où un zéphyr haï me vêt de glaçons würmiens je dîne d'exquis rôtis de bœuf au kir à l'aÿ d'âge mûr & cætera !*" ouvert avec l'encodage ASCII s'affiche : "*DÃ¨s NoÃ«l oÃ¹ un zÃ©phyr haÃ¯ me vÃªt de glaÃ§ons wÃ¼rmiens je dÃ®ne dâ€™exquis rÃ tis de bÃ"uf au kir Ã lâ€™aÃ¿ dâ€™Ã¢ge mÃ»r & cÃ¦tera !*" !

Avec des tables qui ne gèrent pas tous les caractères, vous pouvez même créer de superbes bugs ! En effet il est possible d'avoir deux lettres différentes mais qui ont la même représentation visuelle : le même caractère pour l'homme mais deux caractères différents pour la machine. Donc imaginez un tel caractère dans un nom de variable !

Donc utilisez UTF-8 pour tous vos fichiers sauf si vous êtes contraint pour de bonnes raisons. Les environnements de développement modernes utilisent d'ailleurs cet encodage par défaut. Sachez que plus de 93.6% des sites web utilisent l'UTF-8(4).

Langue

Dans quelle langue nommer les fichiers et dossiers ? L'anglais. La langue de Shakespeare apporte de nombreux avantages :

- C'est la langue internationale du développement informatique ;
- Il est généralement facile de trouver un nom en anglais plus court qu'en français ;
- Pour nous français, cela nous oblige à reformuler et à clarifier ;
- Les acronymes et abréviations sont généralement plus explicites et triviaux.

La structure de votre dépôt Python

Tout projet doit avoir comme première préoccupation celle d'être facilement partageable, réutilisable et maintenable. C'est

(1) https://docs.python-guide.org/writing/structure/

(2) http://kennethreitz.org

(3) https://realpython.com/

(4) https://w3techs.com/technologies/cross/character_encoding/ranking

pourquoi un projet Python doit être conçu comme un package qui pourra être installé dans une installation Python à l'aide d'un gestionnaire de packages tels que pip ou conda. Et ceci, même s'il existe des scripts ou un fichier principal d'entrée pour lancer une application.

Lorsque les ambitions sont grandes ou encore que les thèmes abordés sont nombreux, il est recommandé de créer autant de dépôts Python que nécessaire en sachant qu'il devront être consistants et autonomes. Par exemple, dans le cadre d'un logiciel avec interface graphique, la bonne pratique est de créer un dépôt pour le coeur de calcul qui sera réutilisable dans d'autres projets, si ce dernier nécessite une montée de niveau d'abstraction pour simplifier son utilisation ou automatiser des tâches alors cela se fera dans un deuxième dépôt. Enfin l'interface graphique fera l'objet d'un dernier dépôt. A chaque dépôt ses fonctionnalités offertes et son niveau d'abstraction. Avec des périmètres fonctionnels judicieux de vos dépôts, les gestions de versions et de configurations en seront alors simplifiées et le travail en équipe plus simple à piloter. Si ce n'est pas le cas, n'hésitez pas à faire un refactoring du côté de l'organisation et dépendances de dépôts.

Le postulat pour la suite est que votre projet est versionné avec un outil tel que Git, et qu'il existe un dépôt d'intégration centralisé sur un serveur géré par un logiciel tel que GitHub ou GitLab.

En arrivant sur un tel serveur, que voit un utilisateur potentiel ou un contributeur qui arrive sur la page d'un dépôt ? En premier lieu le nom du projet, puis sa description et enfin un tas de fichiers et dossiers. C'est seulement quand ils font défiler la page qu'ils voient le fichier README de votre projet mis en forme.

Un amas massif de fichiers en désordre ne fera certainement pas une bonne première impression. Vous allez travailler de nombreuses heures sur votre dépôt alors soignez son organisation et facilitez-vous la vie.

Convention de nommage des packages, modules et dépôts

Le nom du package redistribuable

Le package dit *redistribuable* est le coeur de votre dépôt : il contient tout le code source de l'application. C'est-à-dire qu'il est la *librairie* ou *bibliothèque logicielle* qui sera exploitée par votre gestionnaire de package préféré. Le nommage de ce package est important car il participe à l'identité de votre projet. La PEP-0008(5), *Style Guide for Python Code*, nous aide en indiquant que les noms donnés aux packages et modules « *doivent être courts et en minuscules. Les underscores peuvent être utilisés dans le nom si cela améliore la lisibilité mais restent déconseillés* ». Ce nom sera celui utilisé pour les imports, donc rendez-le le plus intuitif et non ambigü possible, et mieux, entre 6 et 8 lettres pour le confort de lecture et écriture. Pour illustrer ceci prenons les exemples suivants : numpy = Numeric Python, scipy = Scientific Python, pandas = Panel Data, etc.

Le nom du dépôt

Quels noms donner aux dépôts ? La bonne pratique courante est celle appliquée pour les noms de packages vus précédemment.

Dans le cas d'un projet avec multiples dépôts, le nom sera composé d'un nom commun suivi d'un tiret puis d'un nom de dépôt. Si l'un des dépôts est considéré comme étant le maître, alors il prendra uniquement celui commun. Par exemples : flask (maître), flask-alchemy, flask-test, django (maître), django-formtools, django-filebrowser-django13 …

Structure standard d'un dépôt

La communauté Python a adopté la recommandation de structuration de dépôt de Kenneth Reitz(6), et en a fait de facto un standard pour les projet conventionnels Python, c'est-à-dire tous les projets dont l'organisation des fichiers et dossiers est libre de contraintes imposées par un framework ou adhérences avec un autre logiciel. Vous pouvez la retrouver sous GitHub(7) pour un package nommé pour l'exercice *sample*.

Toutefois cette structuration doit être légèrement complétée pour garantir un minimum de cohérence et vous aider à partir sur de bonne bases. C'est cette structure qui est détaillée dans la suite et est illustrée sur la figure 1.

(5) https://www.python.org/dev/peps/pep-0008/

(6) http://www.kennethreitz.org/essays/repository-structure-and-python

(7) https://github.com/kennethreitz/samplemod

Le coeur du dépôt : le code source

Le package maître

Emplacement	./package_name
Objectif	Contenir le code source Python du package

Comme explicité plus haut, votre dépôt Python doit être conçu en vue de fournir un package redistribuable qui pourra être installé par votre gestionnaire de packages favori. C'est pourquoi tout le code source qui sera partagé, point central du dépôt, doit être contenu dans un dossier, ici `./package_name`. Ce dossier est un package Python du fait de la présence du fichier `./package_name/__init__.py`. Ce dernier fichier est important puisqu'il va gérer correctement la gestion et la présentation des fonctions, classes, modules et sous-packages à l'utilisateur sans se soucier de la structure interne. Ceci fera d'ailleurs l'objet d'un prochain article.

Si votre projet consiste en un seul module Python, il est tout de même préférable de le placer sous un package maître pour anticiper l'évolution du projet et tirer de la puissance des fichiers `./package_name/__init__.py` et `./package_name/__main__.py`.

La convention de nommage ayant été abordée précédemment, bannissez les noms comme "`src`", "`python`" ou tout autre nom déjà réservé, c'est-à-dire des mot clés / instructions Python ou de librairies déjà existantes (pour connaître l'existant utilisez le moteur de recherche de pypi(8)).

Rendez votre package exécutable avec `__main__.py`

Emplacement	./package_name/__main__.py (facultatif)
Objectif	Rendre un package exécutable

Votre projet est peut-être plus qu'un package, et nécessite un point d'entrée pour lancer une application ou offrir des services ?

Python est fourni avec des fonctionnalités fantastiques mais trop souvent ignorées, et c'est le cas du fichier `__main__.py` que l'on place à la racine du package maître. Ce fichier permet de rendre un package exécutable, c'est-à-dire que la commande :

```
$ python -m package_name
```

va chercher un fichier nommé `__main__.py` dans le package *package_name* et l'exécu-

(8) https://pypi.org/

ter. Son exécution implique l'import au préalable du package *package_name*, et donc l'exécution de `./package_name/__init__.py` en premier. Dans le cas d'un import classique de *package_name*, `__main__.py` ne sera pas exécuté.

Placer vos ressources au bon endroit

Emplacement	./package_name/resources/ (facultatif)
Objectif	Centraliser les ressources utilisées par le code source

Il arrive souvent que le code source doivent utiliser et embarquer des ressources pour fonctionner : icônes pour les interfaces graphiques, templates, tables de data, etc. Deux possibilités s'offrent à vous : soit les mettre à proximité des modules les exploitant, soit dans un dossier explicite `./package_name/resources/`. Ce dernier choix aura pour avantage de faciliter le packaging en vue de produire une librairie partageable.

README !

Emplacement	./README.[txt|md|rst]
Objectif	Informations sur le dépôt et le projet.

Le fichier `./README` apporte les informations sur votre dépôt et votre projet. Le *GNU Coding Standard*(9) propose d'ailleurs un aperçu de ce que devrait contenir un tel fichier. L'extension classique est le `.txt` qui aujourd'hui tend à disparaître au profit du Markdown(10), un langage de balisage léger, d'extension `.md`. L'avantage est que les gestionnaires de dépôts modernes tels que GitHub ou GitLab sont capables de faire un rendu de ce fichier, amplifiant ainsi son usage notamment en offrant la possibilité d'insérer des médias.

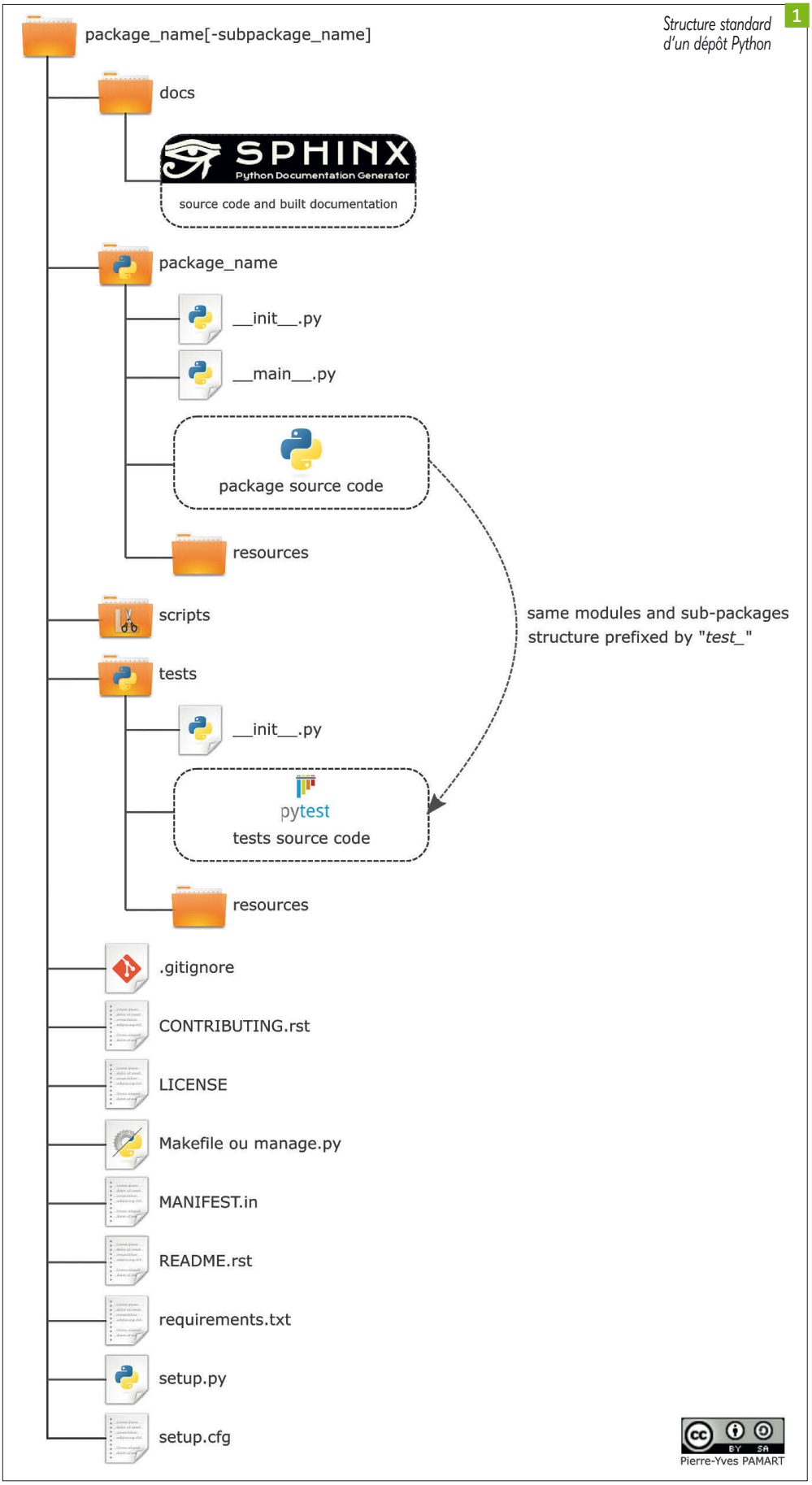
Les documentations de projets développés avec le langage Python sont typiquement générées à l'aide de sphinx(11) qui se base sur le langage à balise reStructuredText(12) plus complet que le Markdown. C'est pourquoi il est possible d'écrire le fichier `./README` en reStructuredText. Privilégiez ce format en veillant à ce que votre gestionnaire de dépôts soit compatible, sinon rabattez-vous sur le Markdown.

(9) https://www.gnu.org/prep/standards/html_node/Releases.html#Releases

(10) https://fr.wikipedia.org/wiki/Markdown

(11) http://www.sphinx-doc.org/en/master/

(12) http://docutils.sourceforge.net/rst.html



Structure standard d'un dépôt Python 1

Pourquoi mettre le nom de ce fichier en majuscule ?
Toutes les lettres majuscules se démarquent et rendent le fichier facilement visible. De plus, les noms de fichier commençant par une lettre majuscule seront listés avant les noms minuscules dans le tri ASCIIbetical (LC_COLLATE = C), ce qui permet de rendre le fichier visible dès le premier coup d'oeil. Et le fichier `./README` est bel et bien celui que l'on souhaite, lu en premier par un nouveau venu sur le projet.

La documentation

Emplacement	./docs/
Objectif	Documentation de référence du package.

Comme évoqué dans l'explication du fichier README, les documentations de projets développés avec le langage Python sont typiquement générées à l'aide de sphinx(13). Et tout ce qui concerne la documentation, que ce soient les fichiers sources au format reStucturedText `.rst`, les ressources, ou encore les artefacts de génération de la documentation, doit se trouver dans le dossier ./docs/. Après avoir installé *sphinx* dans votre environnement (virtuel ! c'est mieux !) à l'aide de la commande pip :

```
$ pip install sphinx
```

ou conda :

```
$ conda install sphinx,
```

l'initialisation du contenu de ce dossier se fera par la commande :

```
$ sphinx-quickstart
```

Une interface en ligne de commande vous posera quelques questions afin de préconfigurer le tout.
A noter que *sphinx* nécessite un fichier `.rst` comme point d'entrée, et la bonne pratique est de le nommer `index.rst`. Celui-ci est généralement dédié à la page d'accueil ou d'introduction qui redirige le lecteur vers les différents chapitres constitutifs de la documentation.

Suite de tests

Emplacement	./tests/
Objectif	Tests unitaires et d'intégrations.

Aujourd'hui, qui ne développe plus sans être adepte de la méthodologie du *Test-Driven Development(14)* ? Si ce n'est pas encore le cas, vous allez aimer cela. Bref, les tests représentent une partie importante de vos projets. En Python, il est coutume d'utiliser les frameworks de tests *unitests(15)* ou *pytest(16)*. Ceux-ci impliquent la création de fichiers de tests que l'on placera dans `./tests`. Il sera préféré de reproduire la structure du package afin de faciliter la correspondance entre le code source du package et celui de ses tests associés. Ce qui aura pour effet de bord, de simplifier la configuration du `./setup.py`. De plus, le fait de mettre les tests dans un dossier spécifique et non avec le code source du package est en soi un test : celui des imports.
Le dossier `./tests/resources/` vise à contenir des données qui pourront être exploitées par les tests tels que des jeux de données.

Licence

Emplacement	./LICENSE
Objectif	Se couvrir juridiquement.

Souvent négligé, ce fichier devrait être considéré comme obligatoire que ce soit pour un projet open source ou propriétaire. Les revendications de copyright et le texte complet de la licence doivent être placés dans ce fichier. Dans le cas d'un projet open source, le site **choosealicense.com** peut être d'une précieuse aide. Et dans l'autre cas rapprochez-vous de juristes pour vous aider à produire un fichier de licence qui protégera la propriété de votre entreprise ou client.

Packaging

Setup.py et setup.cfg

Emplacement	./setup.py + ./setup.cfg
Objectif	Gestion de la distribution et de la création de package redistribuable.

La production d'un package redistribuable de votre travail en Python est généralement assuré par *setuptools*(17) qui vous apporte des outils pour créer (build package) et distribuer (intégration dans un gestionnaire de package) de manière aisée, notamment en gérant les dépendances à d'autres librairies. Son utilisation est pilotée par un script Python nommé `./setup.py` placé à la racine du dépôt apportant la configuration nécessaire à setuptools. Il est parfois utile de recourir à un fichier supplémentaire contenant des options par défaut pour les commandes de `./setup.py` qui sont placées dans le fichier `./setup.cfg` et écrites au format ini.
La commande :

```
$ python setup.py install
```

suffira pour installer le package dans la distribution Python ou l'environnement virtuel activé. Et une commande plus complexe comme par exemple :

```
$ python setup.py sdist bdist_wheel
```

(les packages setuptools et wheel devront être installés) vous permettra de produire un package redistribuable binaire sous la forme d'une wheel introduite par la PEP-0247(18). Pour plus de renseignements consultez le site de la Python Packaging Authority(19) et la documentation fournie sur le packaging(20).

MANIFEST.in

Emplacement	./MANIFEST.in (facultatif)
Objectif	Spécifier les fichier supplémentaires à redistribuer

La génération d'un package redistribuable à l'aide du seul fichier `./setup.py` suffit, mais parfois vous souhaiterez embarquer des fichiers supplémentaires à distribuer. La façon classique de le faire est d'écrire un manifeste modèle, appelé `./MANIFEST.in` par défaut. Ce manifeste modèle est juste une liste d'instructions pour générer votre fichier manifeste final, MANIFEST, qui est la liste exacte des fichiers à inclure dans votre distribution source. La commande :

```
$ python setup.py sdist
```

traite ce modèle et génère un manifeste à partir de ces instructions et de ce qu'elle trouve dans le système de fichiers. Vous trouverez toutes les informations nécessaires à son utilisation dans la documentation de distutils(21).

(13) http://www.sphinx-doc.org/en/master/

(14) *Robert C. Martin, « Professionalism and Test-Driven Development », IEEE Software, vol. 24, no 3, mai 2007, p. 32–36 (ISSN 0740-7459, DOI 10.1109/ms.2007.85)*

(15) https://docs.python.org/3.7/library/unittest.html

(16) https://docs.pytest.org/en/latest/

(17) https://github.com/pypa/setuptools, https://setuptools.readthedocs.io/en/latest/setuptools.html

(18) https://www.python.org/dev/peps/pep-0427/

(19) https://www.pypa.io

(20) https://packaging.python.org

(21) https://docs.python.org/fr/3/distutils/sourcedist.html

Reconstruire l'environnement de développement

Emplacement	./requirement.txt
Objectif	Définir les dépendances du dépôt

L'installation de packages se réalise généralement avec l'outil pip(22) livré avec Python. Le fichier `./requirement.txt` spécifie les dépendances requises pour contribuer au projet : le code source du dépôt, les tests, les builds et la génération de la documentation. Ce fichier est simplement généré avec la commande :

```
$ pip freeze > requirements.txt.
```

Même si vous n'avez pas de dépendances, il est peut être considéré comme bonne pratique de placer un fichier `./requirement.txt` vide en termes de dépendance et avec un commentaire expliquant qu'il est volontairement vide, afin de prévenir le contributeur qu'il n'y a pas de dépendance de manière explicite.

La tendance, voire la convergence à court terme, est d'utiliser l'outil *pipenv*(23) qui combine à la fois pip et le gestionnaire d'environnements virtuels virtualenv(24). Entre autre il vise également à apporter une solution à certains écueils(25) liés à la gestion du fichier `./requirement.txt` . Pipenv génère et utilise deux fichiers `./Pipfile` and `./Pipfile.lock` . Alors que faire aujourd'hui ? Usez et abusez de pipenv, mais continuez à générer(26) le `./requirement.txt` à l'aide de pipenv pour ceux n'ayant pas encore adopté son usage.

Automatisez vos tâches !

Emplacement	./Makefile ou ./manage.py (facultatif)
Objectif	Tâches de gestion courantes.

Tout développeur qui se respecte doit travailler en sorte de ne plus retravailler plus tard ! Donc automatisez vos tâches à l'aide d'un Makefile. Cela peut également être vu comme une manière de capitaliser du savoir-faire. Sous GNU/Linux, GNU make(27) est un outil utile pour définir des tâches génériques pour votre projet. Le listing 1 donne un exemple de Makefile avec deux commandes. La commande :

```
$ make init
```

réalisera l'installation des dépendances, et la commande :

```
$ make tests
```

de lancer les tests.

```
init:
  pip install -r requirements.txt

tests:
  py.test tests
```

Listing 1 : exemple de Makefile

Si vous ne souhaitez ou ne pouvez pas utiliser make, la création d'un CLI (Command Line Interface) est encouragée. Pour cela il suffira de créer un fichier `./manage.py` (toute ressemblance au framework Django(28) est fortuite) qui pourra exploiter l'excellent package click(29).

Le dossier scripts

Emplacement	./scripts/ (facultatif)
Objectif	Centraliser les scripts de tâches de gestion courantes et les scripts Python accompagnant le package.

Souvent l'automatisation de tâches au travers du `./Makefile` ou `./manage.py` nécessite un peu plus de code qui peut être judicieusement placé dans un module à part entière. Celui-ci sera alors placé dans le dossier `./scripts/` .

De même, de nombreux packages Python incluent des outils de ligne de commande. Ceci est utile pour distribuer les outils de support associés à un package ou simplement tirer parti de l'infrastructure setuptools / PyPI pour distribuer un outil de ligne de commande utilisant Python. Ces scripts seront également placés dans le dossier `./scripts/` , et pourront être gérés et distribués à l'aide *setuptools* correctement configurés(30).

.gitignore

Emplacement	./.gitignore
Objectif	Versionner ce qui doit être versionné avec Git.

Le classique fichier `./.gitignore` (31) est placé à la racine du dépôt afin de spécifier ce qui ne doit pas être versionné avec Git. 99% du travail sera fait à l'aide du fichier *.gitignore* proposé(32) par GitHub pour le langage Python.

Comment contribuer à votre projet ?

Emplacement	./CONTRIBUTING.rst
Objectif	Versionner ce qui doit être versionné avec Git.

Quoi de mieux que d'accueillir et de guider un nouveau contributeur ? Le fichier `./CONTRIBUTING.rst` est fait pour cela. Soignez vos contributeurs, vous en aurez besoin.

Conclusion

Ce premier article d'une série dédiée aux bonnes pratiques Python, vous à montré comment partir sur de bonnes bases avec une structure de base standardisée et adoptée par de nombreux projets Python. Il s'agira de commencer votre projet Python en créant cette structure de base puis de faire votre premier 'commit'. Ensuite, vous pourrez commencer à créer un environnement virtuel et à coder.

Il existe de nombreux compléments à cette structure tels que les fichiers de configurations de *tox(33)* ou des d'outils d'intégration continue, un HISTORY.rst, etc. : utilisez-les mais gardez à l'esprit d'être explicite et de prendre soin d'un potentiel contributeur.

Les structures imposées par des frameworks tel que *Django(34)* peuvent proposer des alternatives mais restent en accord avec ce qui a été présenté.

Enfin, il n'est jamais trop tard pour faire un refactoring et revenir vers une structure standard, cela aura les avantages de réduire la dette technique de votre projet et de simplifier la prise en mains de votre dépôt par autrui. •

(22) https://pip.pypa.io/en/stable/

(23) https://github.com/pypa/pipenv

(24) https://virtualenv.pypa.io

(25) https://www.kennethreitz.org/essays/a-better-pip-workflow

(26) https://docs.pipenv.org/en/latest/advanced/#generating-a-requirements-txt

(27) https://www.gnu.org/software/make/manual/make.html

(28) https://www.djangoproject.com/

(29) https://click.palletsprojects.com/en/7.x/

(30) https://python-packaging.readthedocs.io/en/latest/command-line-scripts.html

(31) https://git-scm.com/docs/gitignore

(32) https://github.com/github/gitignore/blob/master/Python.gitignore

(33) https://tox.readthedocs.io/en/latest/

(34) https://www.djangoproject.com/

Arnaud Thiefaine
Software Crafter
Arolla
@ArnaudThiefaine

arolla

Man versus Legacy : Gilded Rose

Partie 1

Les techniques proposées dans cet article permettent de réécrire du code legacy de façon efficace et sans risque, à travers la réalisation d'un kata de code. Le contenu présenté est applicable à n'importe quel langage, même si les exemples de code sont en Java. Les outils décrits permettent de s'en sortir plus confortablement, mais ne sont pas indispensables, le plus important étant de disposer d'un IDE moderne (notamment pour faciliter le refactoring et pour la couverture de code intégrée). Le contenu s'adresse aux développeurs connaissant les bases de la programmation et désireux de transformer un code ignoble vers du code propre, fiable et testé.

niveau 100/200

Intervenir sur des bases de code pénibles constitue une des réalités ingrates du métier de développeur. Ce type de code est fréquemment qualifié du terme politiquement correct de legacy. A l'origine, on désignait ainsi du code très ancien, et peu voire pas documenté. Le consensus actuel se réfère à du code démuni de tests, peu importe qu'il date ou qu'il sorte encore fumant du clavier. Lorsqu'on est confronté à un tel code, on sait qu'on va passer un moment pénible, avec les difficultés suivantes :

- Le code en lui-même est difficile à comprendre ;
- L'absence de tests :
  - Rend le besoin auquel le code répond plus difficile à cerner,
  - Empêche de travailler en sécurité, nous exposant à des régressions.

On peut néanmoins se faciliter la tâche (ou a minima la rendre moins éprouvante) au moyen de certains outils et techniques. Pour cela, nous nous appuierons, à travers une série d'articles, sur des *kata* de code conçus spécifiquement pour aborder ces problématiques.

1

```
public void updateQuality() {
    for (int i = 0; i < items.length; i++) {
        if (!items[i].name.equals("Aged Brie")
                && !items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
            if (items[i].quality > 0) {
                if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                    items[i].quality = items[i].quality - 1;
                }
            }
        } else {
            if (items[i].quality < 50) {
                items[i].quality = items[i].quality + 1;

                if (items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                    if (items[i].sellIn < 11) {
                        if (items[i].quality < 50) {
                            items[i].quality = items[i].quality + 1;
                        }
                    }

                    if (items[i].sellIn < 6) {
                        if (items[i].quality < 50) {
                            items[i].quality = items[i].quality + 1;
                        }
                    }
                }
            }
        }

        if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
            items[i].sellIn = items[i].sellIn - 1;
        }

        if (items[i].sellIn < 0) {
            if (!items[i].name.equals("Aged Brie")) {
                if (!items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                    if (items[i].quality > 0) {
                        if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                            items[i].quality = items[i].quality - 1;
                        }
                    }
                } else {
                    items[i].quality = items[i].quality - items[i].quality;
                }
            } else {
                if (items[i].quality < 50) {
                    items[i].quality = items[i].quality + 1;
                }
            }
        }
    }
}
```

C'EST QUOI UN KATA ?

Un *kata* de code consiste en un exercice de programmation permettant aux développeurs de se perfectionner par la pratique et la répétition.
Les *kata* de refactoring ont la particularité de proposer une base de code existante, l'objectif étant de retravailler le code. Le code est rendu délibérément déplaisant et incompréhensible.
Le premier kata de refactoring que nous allons aborder est le *kata* Gilded Rose.

Le kata Gilded Rose

Dans le kata Gilded Rose, le code est rendu obscur par une complexité cyclomatique importante (beaucoup de chemins possibles), et les tests se distinguent par leur absence. Toute ressemblance avec du code réellement en production serait bien évidemment purement fortuite.

T'façon ce qui compte, c'est les valeurs

Intéressons-nous maintenant au problème posé par Gilded Rose (trouvable sur le https://github.com/emilybache/GildedRose-Refactoring-Kata), décrit dans le https://github.com/emilybache/GildedRose-Refactoring-Kata/blob/master/GildedRoseRequirements.txt.

On nous parle d'une petite auberge dans une cité prospère, qui propose des articles à la vente. Seulement, leur qualité se dégrade au fur et à mesure que leur date limite de vente approche. Pour se faciliter la gestion d'inventaire, l'aubergiste, Allison, a demandé à Leeroy, un aventurier, de développer une application pour ça.

Ce dernier étant parti pour de nouvelles aventures, elle nous demande d'intervenir pour supporter une nouvelle catégorie d'articles (de type "Conjured"), dont la qualité se dégrade deux fois plus rapidement que d'ordinaire.

Chaque article est caractérisé par un *nom*, une *qualité* qui indique sa valeur intrinsèque, et une *date limite* désignant le nombre de jours avant lequel l'article doit être vendu. Dès qu'on discute des règles de gestion, les choses se corsent. On se retrouve face à de nombreux cas particuliers, et on a du mal à distinguer le général du spécifique. A l'issue de la discussion on ne se sent pas très avancé. Heureusement, on peut essayer de s'appuyer sur le code pour avancer dans la compréhension du problème. Ou pas : 1

A la lecture du code, on ne retrouve pas vraiment les règles métiers énoncées par l'aubergiste. L'ensemble est peu clair, avec plusieurs niveaux de branchements imbriqués et de nombreux effets de bord (en violant allègrement plusieurs principes et bonnes pratiques du clean code). Les choses se présentent donc plutôt mal …

Remarque : *le code présenté ici correspond à la version Java du kata. Si on est plus familier avec d'autres langages, ce n'est pas un problème : le kata est disponible dans de nombreux langages.*

Man versus wild legacy

La première difficulté consiste à savoir par où commencer. Étudions les options qui s'offrent à nous.

Attaquer bille en tête

L'évolution demandée ne semblant pas très compliquée, on pourrait l'implémenter directement dans le code.
Cette méthode est très aléatoire tant on ne dispose d'aucune garantie de répondre au besoin tout en préservant le comportement existant. Cette approche de *tête brûlée* est donc à proscrire …

Prendre le temps de comprendre

A l'inverse, on peut se poser, se munir de papier et essayer de comprendre le code. Seulement, ça peut être très chronophage et mener à des interprétations erronées du comportement de l'application.

Ecrire des tests

Modifier le code tel quel reviendrait à bâtir sur des ruines. Nous allons plutôt choisir de prendre soin de nous et de mettre en place des tests. Ils nous permettront d'apporter des changements en limitant les risques, tout en améliorant notre compréhension du code. C'est donc le premier objectif vers lequel on doit tendre.

Hé ! Pas si vite …

Avant cela, il faut s'assurer que le code est dans un état qui permet d'écrire des tests. On doit donc être capable d'exécuter le code. Pour cela, on doit disposer de tous les éléments nécessaires (code et dépendances), et configurer le projet correctement.
Le *kata* Gilded Rose se présentant sous la forme d'un projet Maven, il est facile de l'importer et de le compiler dans son IDE préféré. On trouve même une classe exécutable (avec une méthode *main()*) qui appelle quelques fonctions du code et affiche le résultat dans la sortie standard.
On peut jeter un œil sur le code de production également. Le *kata* fournit deux classes :

- Item : décrit les articles mis en dépôt à l'auberge, caractérisés par un nom, une qualité et une date limite ;
- GildedRose : gère une collection d'articles et possède une unique méthode, *updateQuality()* pour gérer le cycle de vie des articles ; c'est au niveau de cette méthode que nous sommes supposés intervenir ;

Ecriture des tests

Le but de nos tests est de garantir l'absence de régression, tout en couvrant le plus possible de code.

Le premier test

Dans une situation classique, nous partons des besoins, et construisons les tests de façon incrémentale (en *baby steps*), en suivant l'approche TDD. On écrit un premier test, assez simple, puis une fois qu'il fonctionne, on écrit un deuxième, un peu plus complexe, et ainsi de suite ...
Mais dans la situation présente, nous avons déjà affaire à un existant.
Nous disposons certes d'un document d'expression des besoins. Mais, nous l'avons constaté, la discussion avec l'aubergiste était compliquée, les concepts métier sont mélangés, on distingue mal le général du spécifique. On pourrait retourner voir Allison, mais la journée avançant (et les boissons partagées avec les clients aussi), on ne peut compter sur une description plus claire des besoins.
De façon nettement plus problématique, rien ne garantit que l'implémentation actuelle soit conforme aux besoins. De façon assez naturelle, l'implémentation s'écarte assez rapidement de la documentation, et au final, c'est dans le code que réside la vérité. D'un certain point de vue, le code est la documentation la plus importante d'un système d'information (https://martinfowler.com/bliki/CodeAsDocumentation.html).
Partir du code constitue une solution viable dans certaines situations de refactoring. Il faut néanmoins que l'ensemble ne soit pas trop complexe, permettant de voir un point de départ assez rapidement. Avec Gilded Rose, c'est loin d'être immédiat.

Le métier, c'est pas si important … au début

La vraie difficulté provient du fait que, de façon assez intuitive, on cherche à se raccrocher au métier et à le comprendre pour être guidé. La complexité intrinsèque de Gilded Rose rend cette approche contre-productive.
Pour s'en sortir, il est nécessaire d'oublier, pour un temps au moins, le métier. Nous allons chercher à couvrir le maximum de code en un minimum d'efforts, de façon brute. Nous allons modifier la perspective des tests : on ne cherche plus à vérifier ce que le code fait, mais plutôt qu'il fait exactement la même chose qu'avant.
Pour cela, nous allons construire un *golden master*, remplissant le rôle d'un harnais de sécurité.

Confection d'un Golden Master

Principes

Un *golden master* peut se résumer comme l'ensemble des comportements qui caractérisent l'existant (on parle aussi de tests de caractérisation).
La première étape consiste à produire les résultats de tests de référence pour le code existant. On pourra ensuite les comparer aux résultats de ces mêmes tests pour le code en cours de refactoring.
Le comportement du code à tester étant plutôt obscur, le scénario des tests ne peut ni se baser sur les fonctionnalités, ni vérifier des assertions liées au métier (comme on le ferait habituellement avec du TDD ou du BDD).
L'idée est plutôt de produire un ensemble d'*inputs* de façon brute, afin de couvrir un maximum de chemins possibles dans le code, et d'en collecter les *outputs*.
De tels tests auront donc une valeur documentaire quasi-nulle, leur intérêt étant avant tout d'assurer la non-régression. L'idéal est de

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

[Programmez!] Le magazine des développeurs

Nos classiques

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

** Tarifs France métropolitaine*

Abonnement numérique

PDF 1 an - 11 numéros	35€
Option : accès aux archives	10€

Souscription uniquement sur
www.programmez.com

Offres 2019

1 an — 49€
11 numéros
+ 1 vidéo ENI au choix :

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

2 ans — 79€
22 numéros
+ 1 vidéo ENI au choix :

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

eni

** Offre limitée à la France métropolitaine*

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

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

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

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

☐ Vidéo : Symfony 3
☐ Vidéo : Raspberry Pi

PROG 232
Valable jusqu'au 27 septembre 2019

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

email indispensable pour l'envoi d'informations relatives à votre abonnement

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !
☐ Je souhaite régler à réception de facture

*** Tarifs France métropolitaine**

NOUVEAU ! OFFRES 2019

1 an
11 numéros

+ Histoire de la micro-informatique 1973 à 2007

+ clé USB Programmez!

69€*

2 ans
22 numéros

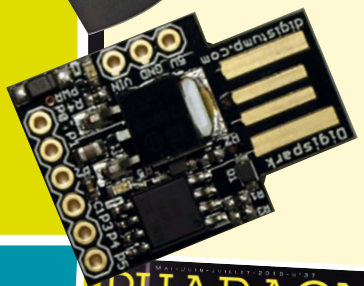
+ Histoire de la micro-informatique 1973 à 2007

+ clé USB Programmez!

+ pack Maker 2 cartes

Attention : quantité limitée

99€*



1 an
11 numéros

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

69€*

2 ans
22 numéros

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

99€*

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

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

les considérer de façon éphémère, le temps d'assurer la réécriture du code (on peut voir ça comme un échafaudage qui nous permettrait de nous en sortir le temps de tout mettre au propre).

Construction

Il existe différentes façons d'effectuer des tests en *golden master*.
La première d'entre elles consiste à générer des données de référence (textes, images, données binaires) correspondant à ce qui est habituellement produit par le code *legacy*. Ces données seront ensuite comparées avec la sortie du code réécrit.
Par exemple on comparera un fichier de texte de référence avec le texte produit dans la sortie standard. On peut y voir un parallèle avec le *approval testing*, l'idée étant de comparer des échantillons textuels correspondant à des objets complexes plutôt que d'écrire des multitudes d'assertions.
Les échantillons textuels peuvent être fournis directement dans le code des tests, si la volumétrie le permet. Dans le cas de données massives, on sera forcé de les fournir dans des fichiers, avec tout ce que ça implique pour outiller les tests.
Pour Gilded Rose, nous allons essayer de fournir les données directement dans le code du test.

Application à Gilded Rose

Les inputs

La définition des *inputs* peut constituer une difficulté en soi : il faut s'assurer de proposer des données en entrée suffisamment représentatives de ce qui peut être accepté (ou non, pour les cas d'erreurs) par l'application. Les entrées doivent être assez exhaustives pour déclencher une couverture de code suffisante.
Par chance, le *kata* Gilded Rose fournit une collection d'Item prête à l'emploi : 2
Il suffit d'initialiser un objet GildedRose avec cette collection d'articles dans les tests, et le tour est joué.

```
Item[] items = new Item[] {
        new Item("+5 Dexterity Vest", 10, 20), //
        new Item("Aged Brie", 2, 0), //
        new Item("Elixir of the Mongoose", 5, 7), //
        new Item("Sulfuras, Hand of Ragnaros", 0, 80), //
        new Item("Sulfuras, Hand of Ragnaros", -1, 80),
        new Item("Backstage passes to a TAFKAL80ETC concert", 15, 20),
        new Item("Backstage passes to a TAFKAL80ETC concert", 10, 49),
        new Item("Backstage passes to a TAFKAL80ETC concert", 5, 49),
        // this conjured item does not work properly yet
        new Item("Conjured Mana Cake", 3, 6) };
```

2

```
@Test
public void should_update_quality_as_expected () {
    GildedRose inn = new GildedRose();
    assertThat(inn.items).extracting( fieldOrProperty: "name").containsExactly("");
}
```

```
java.lang.AssertionError: [Extracted: name]
Actual and expected should have same size but actual size was:
  <6>
while expected size was:
  <1>
Actual was:
  <["+5 Dexterity Vest",
    "Aged Brie",
    "Elixir of the Mongoose",
    "Sulfuras, Hand of Ragnaros",
    "Backstage passes to a TAFKAL80ETC concert",
    "Conjured Mana Cake"]>
Expected was:
  <[""]>
```

3

Et là, on pourrait légitimement se demander si le *kata* n'essaye pas un peu de nous enfumer. Un tout petit peu en fait. Dans une situation réelle, les entrées de test ont peu de chances d'être servies sur un plateau.
C'est une situation dont il faut tenir compte. Ici malheureusement, il faut expérimenter, tatonner et construire le jeu de tests de façon laborieuse. D'autres *kata* de refactoring (comme le TripService de Sandro Mancuso) mettent en avant des techniques pour se faciliter la vie, mais ce n'est pas l'objet du *kata* auquel nous nous intéressons en ce moment.

La méthode

Nous allons nous appuyer sur le point de vue que le code de production a systématiquement raison. Pour construire nos tests, nous allons donc récupérer les sorties obtenues et nous en servir comme valeurs de référence : dans les assertions, elles définiront les valeurs attendues. On met une valeur bidon (*null* ou collection vide) en attendu, et on laisse le test échouer. Le framework de test, dans sa grande bonté, va nous indiquer la valeur obtenue. Il suffit de la recopier en tant que valeur attendue dans l'assertion, et le tour est joué.
On procède alors de façon incrémentale, à chaque fois que les tests sont au vert, on complexifie un peu plus les actions pour le faire échouer, on recopie la valeur obtenue pour le faire fonctionner, et on recommence.
Pour GildedRose, on vérifiera pour chacun de nos tests l'état de chaque article déposé à l'auberge. On commence simplement, en vérifiant déjà l'état à l'initialisation. Ce premier test peut paraître digne du Captain Obvious, il permet malgré tout de s'assurer que la comparaison se fait correctement dans les assertions.
On échoue avec un tir à blanc : 3
On fait passer le test : 4
Ensuite on va vérifier le comportement de la fonctionnalité qui nous intéresse, à savoir la méthode *updateQuality()*. En l'exécutant une première fois. Puis 2 fois d'affilée (les résultats sont différents). Et ainsi de suite.
L'approche est assez naïve, mais elle nous permet de démarrer sans trop se faire de noeuds au cerveau. La difficulté va être de savoir quand s'arrêter.

Refactorez couverts

Pour se fixer une limite dans l'effort, on va s'appuyer sur la couverture de code. Le principe est simple : on utilise un outil automatique qui, lors de l'exécution des tests, va marquer le code de production de la façon suivante :

- les lignes de code exécutées au moins une fois par les tests apparaissent en vert ;
- les autres lignes apparaissent en rouge.

Le résultat obtenu avec un seule exécution de la méthode *updateQuality()* est le suivant : 5
On peut imaginer que la couverture va augmenter avec le nombre

```
@Test
public void should_update_quality_as_expected () {
    GildedRose inn = new GildedRose();
    assertThat(inn.items).extracting( fieldOrProperty: "name").containsExactly("+5 Dexterity Vest",
            "Aged Brie",
            "Elixir of the Mongoose",
            "Sulfuras, Hand of Ragnaros",
            "Backstage passes to a TAFKAL80ETC concert",
            "Conjured Mana Cake");
}
```

4

d'exécutions successives de *updateQuality()*. Une fois toutes les lignes couvertes, on pourra se dire qu'on a exécuté le code suffisamment de fois.
Avant de trouver le nombre correct d'exécutions, il peut être fastidieux de modifier les valeurs attendues à chaque essai.
On n'est pas obligé de s'infliger ça, en profitant d'une propriété importante du *code coverage* : les lignes parcourues sont marquées quel que soit le résultat des assertions (succès ou échec). On va donc d'abord chercher à couvrir totalement le code, sans se préoccuper des assertions, pour trouver le nombre suffisant d'exécutions. Ceci atteint, on pourra se concentrer sur les assertions.

Remarque : *cette propriété de la couverture de code est toutefois à double-tranchant, dans la mesure où il n'y a aucune garantie sur la pertinence des assertions. Nous reviendrons sur ce point un peu plus loin.*

Il se trouve que 16 exécutions de *updateQuality()* suffisent à atteindre une couverture complète. 6

Remarque : *on cherche à atteindre 100% de couverture lorsque c'est possible sans nécessiter des efforts surhumains, sinon on cherche juste à atteindre le meilleur score. De la même manière, on se concentre seulement sur le périmètre de code à réécrire, il n'est pas nécessaire de couvrir tout le projet. L'idée est de rester économe voire fainéant dans la démarche.*

Moins et moins ça fait plus

Nos tests sont-ils suffisamment fiables ? Actuellement, seul l'état final (au bout des n itérations) de l'application est vérifié. Ce n'est pas le cas des états intermédiaires.
Pour bien faire, il faudrait tester l'état après chaque itération.

```
public void updateQuality() {
    for (int i = 0; i < items.length; i++) {
        if (!items[i].name.equals("Aged Brie")
                && !items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
            if (items[i].quality > 0) {
                if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                    items[i].quality = items[i].quality - 1;
                }
            }
        } else {
            if (items[i].quality < 50) {
                items[i].quality = items[i].quality + 1;

                if (items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                    if (items[i].sellIn < 11) {
                        if (items[i].quality < 50) {
                            items[i].quality = items[i].quality + 1;
                        }
                    }

                    if (items[i].sellIn < 6) {
                        if (items[i].quality < 50) {
                            items[i].quality = items[i].quality + 1;
                        }
                    }
                }
            }

            if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                items[i].sellIn = items[i].sellIn - 1;
            }

            if (items[i].sellIn < 0) {
                if (!items[i].name.equals("Aged Brie")) {
                    if (!items[i].name.equals("Backstage passes to a TAFKAL80ETC concert")) {
                        if (items[i].quality > 0) {
                            if (!items[i].name.equals("Sulfuras, Hand of Ragnaros")) {
                                items[i].quality = items[i].quality - 1;
                            }
                        }
                    } else {
                        items[i].quality = items[i].quality - items[i].quality;
                    }
                } else {
                    if (items[i].quality < 50) {
                        items[i].quality = items[i].quality + 1;
                    }
                }
            }
        }
    }
```

5

```
@Test
public void should_update_quality_as_expected () {
    GildedRose inn = new GildedRose();
    for (int i = 0; i < 16; i++) {
        inn.updateQuality();
    }
    assertThat(inn.items).extracting( fieldOrProperty: "name").containsExactly("+5 Dexterity Vest",
            "Aged Brie",
            "Elixir of the Mongoose",
            "Sulfuras, Hand of Ragnaros",
            "Backstage passes to a TAFKAL80ETC concert",
            "Conjured Mana Cake");
    assertThat(inn.items).extracting( fieldOrProperty: "quality").containsExactly(0, 30, 0, 80, 0, 0);
    assertThat(inn.items).extracting( fieldOrProperty: "sellIn").containsExactly(-6, -14, -11, 0, -1, -13);
}
```

6

Nous ne sommes en effet pas à l'abri de provoquer des régressions locales, qui au niveau global s'annuleraient, et passeraient ainsi inaperçues.
Ceci nous impose d'écrire les assertions pour chaque itération. Il faudrait définir dans le code de test les valeurs de chaque état. Ça risque de prendre des plombes …
Notre approche naïve montre ici ses limites, nous obligeant à réfléchir un peu. Faisons preuve d'audace en changeant de point de vue.
On s'est concentré sur des valeurs spécifiques. Mais dans l'absolu, on s'en moque. D'autant qu'elles n'ont aucun rôle documentaire.
Revenons sur ce qui est important : s'assurer que la nouvelle version du code a un comportement identique à la version legacy. Il suffit simplement de s'assurer que les valeurs produites sont identiques entre les deux versions.
Pour cela, on duplique l'implémentation legacy, pour se retrouver avec deux implémentations :

- GildedRose : cette implémentation va être réécrite ;
- LegacyGildedRose : c'est notre implémentation de référence, elle ne doit **jamais** être modifiée (remarque : on peut même déplacer cette implémentation dans l'espace de code dédié aux tests).

Ce n'est pas très intuitif, et pourtant ça fonctionne. Dans le test, il ne reste plus qu'à exécuter les implémentations en parallèle et à comparer les valeurs produites à chaque étape : 7
Il faut s'assurer que chaque implémentation travaille sur des instances différentes d'Items. Pour cela, on fait une copie en profondeur : 8
Nous disposons enfin de notre *golden master* de test.
Au départ, tous les tests doivent être verts, étant donné que les deux implémentations sont identiques. Si ce n'est pas le cas, c'est qu'on a un problème dans l'écriture des tests (une erreur classique est de comparer les références des objets Item dans les assertions plutôt que leur contenu).

Golden master will remember that

Pour construire notre *golden master*, nous nous sommes finalement éloignés de l'approche classique en Text-based approval-testing. Cette approche, bien qu'intuitive, peut se montrer laborieuse avec

```
@Test
public void should_refactored_behave_exactly_as_legacy() {
    // GIVEN
    GildedRose inn = new GildedRose(items);
    LegacyGildedRose legacyInn = new LegacyGildedRose(legacyItems);

    for (int i = 0; i < 16; i++) {
        //WHEN
        inn.updateQuality();
        legacyInn.updateQuality();

        //THEN
        assertThat(inn.items).extracting( fieldOrProperty: "name").containsExactly(
                Arrays.stream(legacyInn.items).map(item -> item.name).toArray());
        assertThat(inn.items).extracting( fieldOrProperty: "quality").containsExactly(
                Arrays.stream(legacyInn.items).map(item -> item.quality).toArray());
        assertThat(inn.items).extracting( fieldOrProperty: "sellIn").containsExactly(
                Arrays.stream(legacyInn.items).map(item -> item.sellIn).toArray());
    }
}
```

7

```
Item[] items = new Item[]{
        new Item( name: "+5 Dexterity Vest",  sellIn: 10,  quality: 20), //
        new Item( name: "Aged Brie",  sellIn: 2,  quality: 0), //
        new Item( name: "Elixir of the Mongoose",  sellIn: 5,  quality: 7), //
        new Item( name: "Sulfuras, Hand of Ragnaros",  sellIn: 0,  quality: 80), //
        new Item( name: "Sulfuras, Hand of Ragnaros",  sellIn: -1,  quality: 80),
        new Item( name: "Backstage passes to a TAFKAL80ETC concert",  sellIn: 15,  quality: 20),
        new Item( name: "Backstage passes to a TAFKAL80ETC concert",  sellIn: 10,  quality: 49),
        new Item( name: "Backstage passes to a TAFKAL80ETC concert",  sellIn: 5,  quality: 49),
        // this conjured item does not work properly yet
        new Item( name: "Conjured Mana Cake",  sellIn: 3,  quality: 6)};

Item[] legacyItems = Arrays.stream(items)
        .map(item -> new Item(item.name, item.sellIn, item.quality))
        .toArray(Item[]::new);
```

8

des volumétries importantes de données. En optant pour une exécution en parallèle, notre vie est devenue moins compliquée. Mais en faisant ça, nous avons modifié l'essence de notre *golden master* : ce n'est plus l'ensemble de résultats, comme dans l'approche classique, mais directement la version legacy du code. La conséquence directe est qu'à l'avenir, nous aurons quelques difficultés à éliminer complètement cette version legacy du projet : nous sommes forcés de la conserver pour exécuter nos tests. Mais nous aurons l'occasion de revenir sur ce point plus tard.

Il ne peut plus rien nous arriver de mal ?

Les tests passent et le code est couvert à 100%. Est-ce suffisant pour se lancer dans le refactoring en toute sécurité ?

Le succès des tests nous garantit que l'exécution du code de production n'a rencontré aucune erreur ou invalidé aucune assertion. Le degré de confiance dépend alors de la pertinence et de l'exhaustivité des assertions.

Les 100% de couverture nous assurent que toutes les lignes du code de production ont été exécutées lors des tests, guère plus. Si dans le test actuel on commente les assertions, il sera quand même vert, avec une couverture à 100%, alors qu'en fait rien n'est testé.

De plus, en couverture de code, seul le parcours d'une ligne est vérifié, mais pas son contenu ou son comportement.

Ce constat n'est pas très rassurant, et nous amène à nous dire que peu importe les efforts fournis, on ne disposera d'aucune garantie de sécurité absolue. Mais est-ce si grave ?

C'est déjà moins grave que la situation de départ : absence de tests et code incompréhensible. Maintenant qu'on a des tests, on peut commencer à faire quelque chose. C'est mieux que de réécrire sans aucun test, sans filet. Il faut juste être prêt à accepter une part de risque résiduel.

Ensuite, il existe des techniques additionnelles de test permettant de réduire encore davantage ce risque, comme le *mutation testing*.

Pour faire simple, le *mutation testing* consiste à modifier dynamiquement le code de production lors de l'exécution des tests, et s'assurer que ces modifications font échouer au moins un test. Dans le cas contraire, on est généralement face à une des causes suivantes :

- les assertions ne sont pas suffisamment exhaustives ;
- la ligne de code qui a été mutée ne sert pas à grand-chose.

Il se trouve que sur le kata Gilded Rose, le mutation testing ne montre pas de problèmes particuliers. Le kata a en effet été calibré pour qu'une couverture de code à 100% soit suffisante pour refactorer en sécurité. Nous ne rentrerons donc pas plus dans les détails du mutation testing dans l'immédiat.

Et après ?

Maintenant qu'on dispose d'un filet de sécurité, on peut refactorer le code en toute quiétude. Cette phase de réécriture du code fera l'objet d'un second article. GitHub : https://github.com/athiefaine/gilded-rose-kata

Rétrospective

Faisons le bilan de ce que nous a appris cette première partie de kata :

- ne pas se résigner ni se désespérer devant un code pourri, on peut s'en sortir en affrontant un minimum de difficultés ;
- prendre de la hauteur pour tenter des approches contre-intuitives :
  - construire un *golden master*, autrement dit, écrire des tests de caractérisation basés autour des résultats de référence, plutôt que des tests basés sur la compréhension de l'application,
  - dupliquer les implémentations pour les comparer lors des tests.
- faire confiance à la couverture de tests, mais pas aveuglément, en prenant conscience de ses limites ;
- faire attention à la pertinence des assertions ;
- s'appuyer sur les outils et méthodologies, pour libérer le maximum de temps de cerveau.

Remerciements

Je tiens à remercier les personnes suivantes pour le temps passé à relire mon texte, et les différents conseils, idées et encouragements qu'ils ont pu me prodiguer : Dorra Bartaguiz, Alexia Hoang, Benjamin Hugot, Arnauld Loyer, Cyrille Martraire, Laury Maurice, Hadrien Mens-Pellen, Nicolas Morin, et Yvan Phélizot. •

Tous les liens :

https://github.com/emilybache/GildedRose-Refactoring-Kata
https://github.com/emilybache/GildedRose-Refactoring-Kata/blob/master/GildedRoseRequirements.txt
https://martinfowler.com/bliki/CodeAsDocumentation.html
https://approvaltests.com
https://github.com/emilybache/GildedRose-Refactoring-Kata/blob/master/Java/src/test/java/com/gildedrose/TexttestFixture.java
https://github.com/sandromancuso/trip-service-kata

Loïc Lefèvre
Développeur depuis l'âge de 8 ans, je multiplie mes expériences professionnelles depuis 20 ans : startups, grands comptes de la finance et actuellement chez Oracle. FullStack developer, architecte cloud et DBA applicatif, j'adore les challenges liés au développement d'applications modernes ! **@Loic_Lefevre**

GraalVM™, la machine virtuelle du futur : performante, sécurisée, polyglotte et intégrable

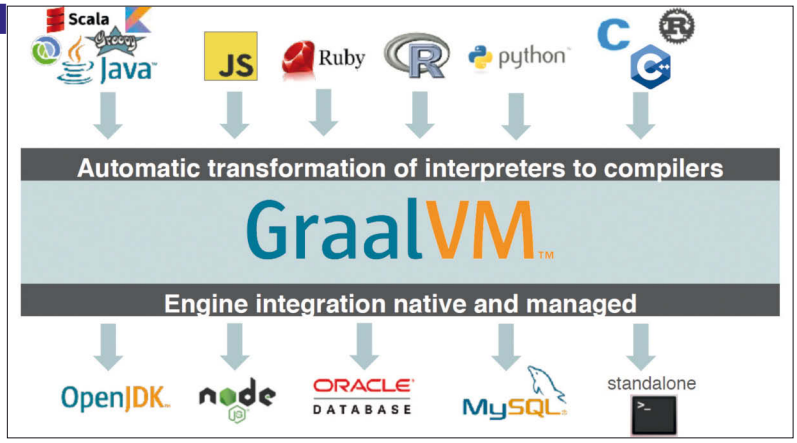
Issue du département de R&D Oracle Labs, GraalVM est une machine virtuelle universelle pour exécuter des applications écrites en JavaScript, Python, Ruby, R, Java, Scala, Groovy, Kotlin, Clojure, Rust, C et C++. S'appuyant sur la JVM SE, elle intègre le compilateur Graal et avec lui de toutes nouvelles optimisations. Polyglotte, GraalVM supprime les barrières entre ces différents langages et permet leur interopérabilité dans un environnement d'exécution partagé et, pour la première fois dans l'industrie, très performant. GraalVM peut fonctionner de manière autonome ou dans le contexte d'OpenJDK, de Node.js, de la base de données Multimodel Oracle ou encore la base de données open source la plus utilisée MySQL.

niveau 100

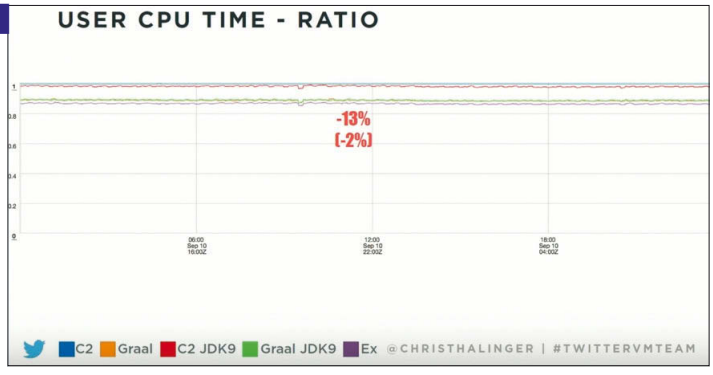
Historique

C'est en 2011 que le projet Graal(VM) naît du partenariat d'un groupe de chercheurs et d'Oracle Labs (département de R&D). Son ancêtre la MaxineVM avait pour but d'améliorer la productivité dans le domaine de la recherche et du développement de machines virtuelles Java ; écrites en C/C++ habituellement comme HotSpot ; sujet très complexe et très coûteux donc. Reprenant certaines des approches populaires de Jikes, la MaxineVM est la première machine virtuelle Java modulaire écrite en Java (machine virtuelle métacirculaire) et bénéficiant donc d'une plateforme de développement moderne et très riche. Les résultats de ce projet ont été concluants : performances et productivité étaient au rendez-vous.
En parallèle, Oracle Labs a également créé le projet TruffleVM, une machine virtuelle polyglotte. Partant du principe que les développeurs utilisent plusieurs langages de programmation pour adresser au mieux les cas d'usage rencontrés, migrer sur de longues périodes d'un langage à un autre ou encore réutiliser des bibliothèques existantes (et ne pas réinventer la roue constamment…). TruffleVM répond ainsi aux difficultés liées à la programmation polyglotte telles que des interfaces complexes, une souplesse insuffisante et des performances très dégradées notamment dues aux échanges de structures de données, recopiées entre les différents environnements d'exécution de chaque langage.
Ainsi GraalVM est la combinaison de ces deux projets de recherche majeurs : MaxineVM et TruffleVM. 1

1



2



Et pour les plus curieux d'entre vous qui se demandent si GraalVM est prête pour la production, sachez que Twitter utilise déjà la GraalVM Enterprise Edition depuis presque un an maintenant avec 13% de gains sur la consommation CPU se traduisant par 13% d'économies sur les ressources hardware notamment ce qui à l'échelle de Twitter est conséquent ! 2

Les différentes versions

La GraalVM est disponible depuis avril 2018 pour l'édition open source communautaire (CE) et depuis avril 2019 pour la version d'entreprise (EE) :

- Community Edition (CE) : open source et gratuite - https://github.com/oracle/graal/releases
- Enterprise Edition (EE) : ajoute des fonctionnalités plus poussées (performance sécurité, scalabilité), le support est payant mais il est possible de la tester gratuitement - https://www.oracle.com/technetwork/graalvm/downloads/

A noter que la version EE est disponible sans surcoût en souscrivant au cloud public Oracle (support inclus donc).

Compilateur Graal

La GraalVM utilise le tout nouveau compilateur, Graal. Celui-ci est écrit en Java et est disponible sous forme d'une bibliothèque native libgraal. Plus avancé que les compilateurs de la JVM HotSpot, il compile :

- « juste à temps » (Just-In-Time ou JIT) mais aussi « à l'avance »

(Ahead-Of-Time ou AOT)

- plusieurs langages à l'aide de l'intégration des frameworks Truffle et Sulong (LLVM)
- pour différentes architectures (Linux x64, Mac et Windows)

La JEP 243 (Java-Level JVM Compiler Interface ou JVMCI) permet en effet à HotSpot de se servir du compilateur Graal JIT.

Ce compilateur dynamique de dernière génération apporte de nouvelles optimisations lors de la traduction du bytecode Java en code machine natif. L' « Escape Analysis », apparue avec le JDK 6, détermine le scope d'utilisation et la durée de vie des objets alloués. Cette technique permet au compilateur d'effectuer de nombreuses optimisations : allocation d'objets sur la pile au lieu du tas, suppression de la synchronisation (et des verrous associés) si un objet ne quitte jamais un thread ou encore remplacer l'allocation d'un objet par des variables locales pour tous ses attributs. Ces optimisations participent donc à la réduction de la mémoire utilisée (suppression des allocations) et à la rapidité d'exécution (garbage collector moins sollicité, inlining plus agressif, moins de verrous à gérer…).

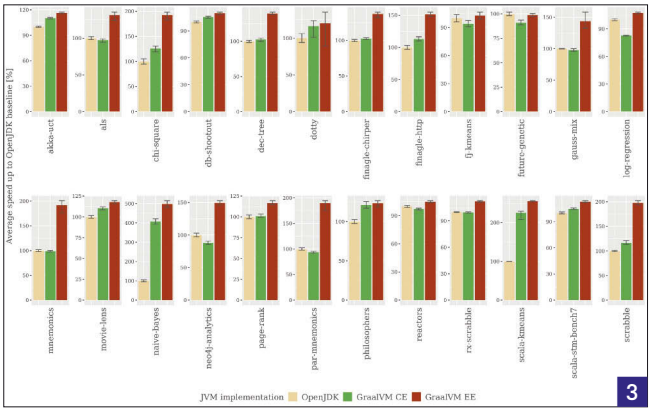
Malheureusement avec l'Escape Analysis, c'est tout ou rien. Dès qu'un objet « s'échappe » c'est-à-dire que son scope d'utilisation s'étend à une autre méthode voire à un autre thread, aucune optimisation n'est possible, y compris dans les scopes sous-jacents. Le compilateur Graal apporte le « Partial Escape Analysis ». Cette nouvelle approche permet de supprimer les allocations mémoires inutiles pour les branches d'exécution les plus courantes (application de l'Escape Analysis donc) mais conserve l'allocation usuelle et non optimale dans les branches d'exécution les moins utilisées. Ceci permet notamment d'appliquer en cascade l'Escape Analysis à d'autres appels de méthodes, d'autres threads d'exécutions…. qui auraient été simplement ignorés (tout ou rien).
Le compilateur Graal, c'est 62 phases d'optimisations dans sa version Enterprise et 35 dans sa version communautaire.

La nouvelle suite de benchmarks Renaissance (https://renaissance.dev/) a été créée tout spécialement pour la Graal VM CE et EE afin de valider les optimisations développées avec un maximum de cas d'usage modernes : Scala, Spark… contrairement aux benchmarks plus traditionnels comme SPECjvm2008 ou SPECjbb2015. Le graphique montre les gains de performance suivants par rapport à OpenJDK : 3

Un autre benchmark Java relatif aux Streams montre des gains de performance de 1,7 à 5 fois plus rapide : https://git.io/fj4uo 4

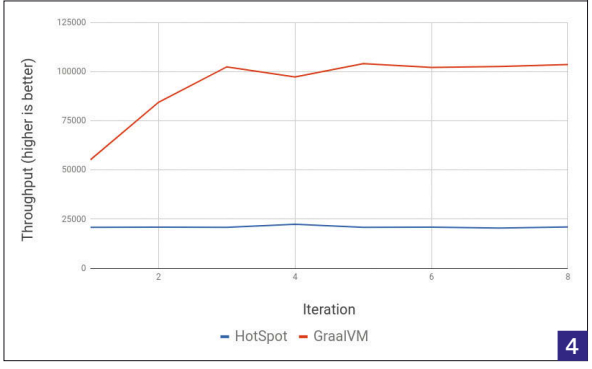
Et la raison essentielle de ce facteur 5 repose sur le « Partial Escape Analysis » présenté précédemment. Avec simplement l'Escape Analysis, l'instanciation de l'objet Person échappe à la méthode volleyballStars() et se retrouve utilisée par une méthode (voir schéma IGV, Ideal Graph Visualizer : https://lafo.ssw.uni-linz.ac.at/pub/idealgraphvisualizer) Invoke#ReferencePipeline$4$1.accept() (nœud n°1625) du point de vue du compilateur JIT traditionnel : 5

Graal va plus loin en appliquant l'inlining de la méthode accept() puis le Partial Escape Analysis : il supprime alors toute allocation d'objet Person du bytecode produit, car ce nouvel objet ne sert finalement qu'à récupérer la valeur p.height ou p.age + 1 dans les

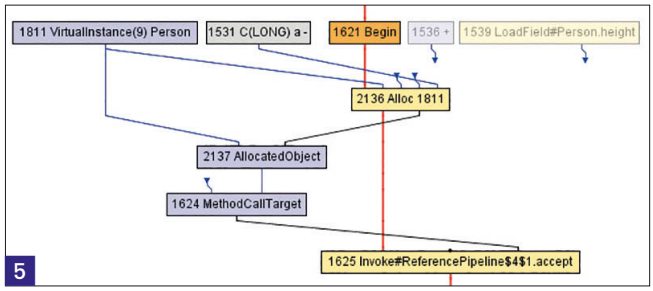


3

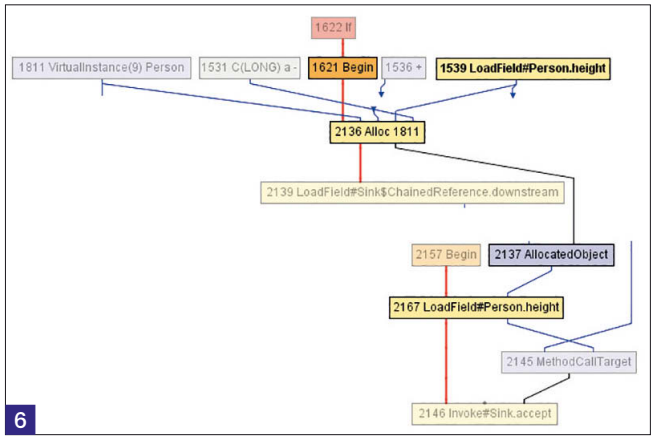
```
@Benchmark
public double volleyballStars() {
  return Arrays.stream(people)
    .map(p ->
    new Person(p.hair, p.age + 1, p.height))
    .filter(p -> p.height > 198)
    .filter(p -> p.age >= 18 && p.age <= 21)
    .mapToInt(p -> p.age)
    .average().getAsDouble();
}
```



4



5



6

méthodes filter() suivantes. La représentation intermédiaire (Intermediate Representation ou IR) de Graal ci-dessous montre le code « mort » supprimé (les nœuds moins contrastés) : 6

Dernière nouveauté dans ce domaine avec la version Enterprise 19.1.1 (sortie en juillet 2019), le compilateur JIT est 30% plus rapide et permet, pour les applications très gourmandes d'obtenir le meilleur code machine encore plus rapidement !

Image native

L'un des cas d'usage de la GraalVM actuellement très en vogue est également lié au compilateur Graal dans son mode de fonctionnement Ahead of Time (AOT) ou compilation statique. Ce mode de compilation est très utile pour les applications nécessitant une vitesse de démarrage instantanée et une empreinte mémoire faible telles que les microservices déployés avec Docker et Kubernetes ou encore les approches Serverless telles que le projet open source Fn (fnproject.io). 7

Ainsi, toute l'analyse du code et les optimisations sont prises en charge lors de la compilation qui dure (beaucoup) plus longtemps donc. Le compilateur génère ensuite directement du code natif.
Comme le temps de compilation est plus important, déporter cette phase dans votre pipeline d'intégration continue est une bonne idée. Un plugin Maven est d'ailleurs disponible pour ce scénario.
A l'exécution, il n'y a qu'un binaire qui contient l'application, les bibliothèques, le JDK et une machine virtuelle allégée, la Substrate VM pour la gestion de la mémoire, etc. ; la JVM n'est plus utilisée !
Côté limitations, l'analyse statique impose que tout le code accessible soit présent lors de la compilation : le chargement dynamique de classes est donc exclu tout comme l'instruction invokedynamic. L'initialisation des blocs statiques est également réalisée uniquement lors de la compilation donc quelques effets de bords doivent être anticipés. Enfin, l'introspection des objets (Reflection) est plus compliquée mais pas infaisable : un fichier de déclaration permet en effet d'indiquer lors de la compilation quelles classes seront chargées dynamiquement (Class.forName()).

La liste des limitations est disponible ici :
https://github.com/oracle/graal/blob/master/substratevm/LIMITATIONS.md
Les bénéfices par contre sont spectaculaires comme l'empreinte mémoire et la rapidité de démarrage des frameworks de microservices suivants : 8
Une diminution de la consommation mémoire d'un facteur 3 est généralement constatée. Pour le temps de démarrage, il s'agit de facteurs allant de 28 à 58 fois plus rapide ! Ces améliorations doivent notamment permettre des économies intéressantes lorsque la mémoire et le CPU sont des ressources payantes comme chez les clouds providers : Oracle, AWS ou Azure.

Oracle Labs a également réalisé d'autres comparatifs sur le benchmark ApacheBench où l'on voit clairement les améliorations apportées par une toute nouvelle fonctionnalité : 9

La version Enterprise 19.1.1 amène une fonctionnalité supplémentaire pour optimiser encore plus le code machine produit : l'optimisation guidée par profil (ou Profile-Guided Optimization, PGO).

```
+ javac Streams.java
+ $GRAALVM_HOME/bin/native-image Streams
+ ./streams 1000000 200
...
Iteration 20 finished in 1955 milliseconds with checksum 6e36c560485cdc01
```

Le principe consiste à générer une image native instrumentée qui va produire un profil d'exécution stocké dans le fichier default.iprof.

```
+ $GRAALVM_HOME/bin/native-image --pgo-instrument Streams
+ ./streams 1000 200
```

La regénération de l'image native en se servant de ce profil va permettre au compilateur Graal d'appliquer de nouvelles optimisations et surtout va pouvoir les appliquer de manière systématique pour garantir une prédictibilité des performances.

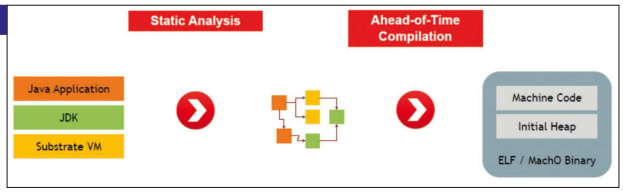
```
+ $GRAALVM_HOME/bin/native-image --pgo Streams
```

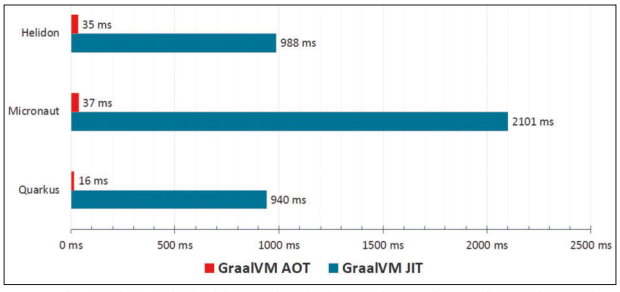
Ainsi le benchmark de l'API Stream tourne 2 fois plus vite avec cette technique :

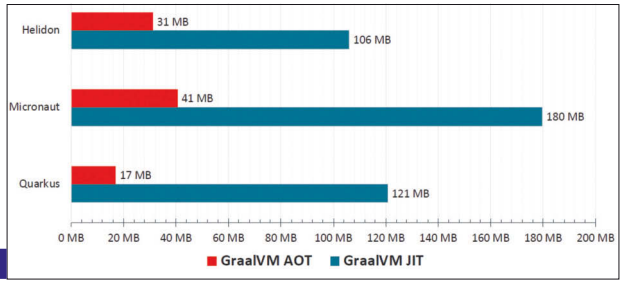
```
+ ./streams 1000000 200
...
Iteration 20 finished in 827 milliseconds with checksum 6e36c560485cdc01
```

Revenons sur le sujet de la performance prédictible. Il est possible de travailler ses profils pour cibler des comportements exceptionnels de son application : pensez à un comportement nominal pour une application de trading où il ne se passerait pas grand-chose et tout à coup, un pic d'activité survient. Dans ce cas précis, avec le compilateur JIT, il y a de fortes chances pour que la JVM reparte sur

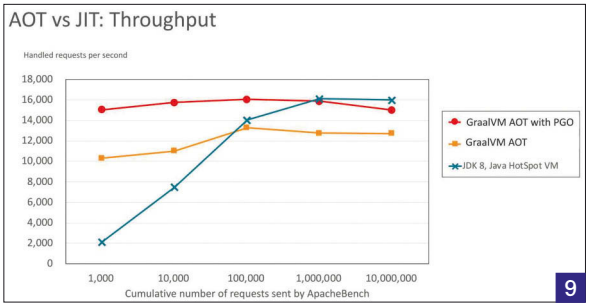
7







8



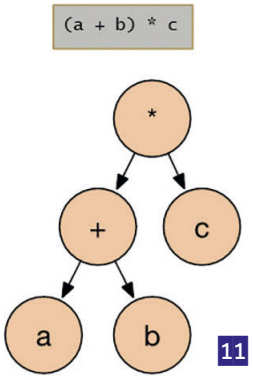
9

du code peu utilisé et donc interprété et lent. Avec un profil préparé spécifiquement pour un tel événement, la compilation native va pouvoir générer le code machine optimal pour ce genre de cas exceptionnel et obtenir des performances adaptées à des cas d'usage exceptionnels dans la vie de mon application !

Et le meilleur reste à venir, car toutes ces nouvelles optimisations : JIT, AOT, PGO sont disponibles pour les langages non Java !

Programmation polyglotte

Nous venons de voir que la GraalVM fournit des performances jamais atteintes auparavant pour les langages compatibles avec la JVM : Java, Scala, Groovy, Clojure, Kotlin… mais et pour les autres langages ? Comment la GraalVM pourrait apporter les mêmes niveaux de performance pour ceux-ci : Ruby, JavaScript, Python, R… et avec une seule et même approche ? 10



11

Comme indiqué au début de cet article, la programmation polyglotte, utilisant plusieurs langages, répond à de nombreux besoins notamment de productivité de la communauté Java ; pour preuve, l'existence de frameworks tels que Rhino, Nashorn, JRuby, Jython ou encore Renjin respectivement pour les langages dynamiques JavaScript, Ruby, Python et R. Rappelons que le point essentiel qui a jusqu'alors limité l'adoption de la programmation polyglotte à une échelle plus vaste est lié au coût de passer d'un langage à un autre en terme de performances. Ces frameworks écrits en Java génèrent du bytecode et tirent ainsi parti des capacités de la JVM (JIT, garbage collector…) ce qui améliore déjà l'interopérabilité entre ces langages. Cependant, chacun de ces frameworks possède sa propre architecture et ne peuvent interagir avec la JVM autrement qu'en émettant du bytecode : aucun contrôle sur le compilateur C2 de Hotspot n'est possible, c'est une boîte noire.

Avec GraalVM, nous avons vu que le compilateur a été réécrit en Java ce qui a permis d'y ajouter un ensemble d'API pour pouvoir contrôler son exécution très finement. Ensuite, le principe de programmation polyglotte franchit une nouvelle étape, car une seule architecture est utilisée pour prendre en charge tous ces langages : une seule manière de communiquer entre langages, une seule manière de gérer la mémoire, etc. et une seule interface avec la génération du bytecode : le framework Truffle. Truffle est framework pour écrire des interpréteurs d'Abstract Syntax Trees (AST : résultat du parsing de la grammaire d'un programme). Un AST est une structure arborescente qui possède des nœuds. Chaque nœud possède une méthode execute() qui permet d'évaluer le résultat de ce nœud par rapport à son/ses nœud(s) enfants. 11

Lors de l'exécution, un interpréteur écrit et annoté avec Truffle va permettre de produire du code annoté qui va ensuite être compilé par le compilateur Graal JIT. L'annotation de l'interprétation est importante, car c'est elle qui va faciliter notamment des prises de décisions sur le code machine à produire par Graal : spéculation liée à des types dynamiques (polymorphisme) ou non, branche finale dans une boucle, debugging, etc. 12

L'approche traditionnelle, comprendre la plus facile, lorsque l'on développe un langage dynamique est de s'arrêter à l'interpréteur. L'ajout d'un typage statique ou d'un compilateur dynamique est très complexe et ralentit en effet le développement d'un nouveau langage même si sur le long terme, cela devient nécessaire. Java et JavaScript en sont les parfaits exemples et seule l'adoption massive de la communauté a rendu nécessaire et possible la création de compilateurs dynamiques/JIT (vu les investissements nécessaires pour leur développement). Ainsi, Ruby et R sont d'autres exemples où les performances souffrent de l'absence de compilateurs dynamiques… enfin souffraient jusqu'à ce que la GraalVM arrive. 13

En résumé, Truffle fait le pont entre un interpréteur de langage dynamique et le compilateur Graal JIT pour fournir des performances dignes des implémentations les plus poussées. Par exemple, TruffleJS qui implémente le JavaScript rivalise sur des benchmarks avec la machine virtuelle V8 ; sauf que TruffleJS fait 80 000 lignes de code et V8, plus de 1 700 000.

12

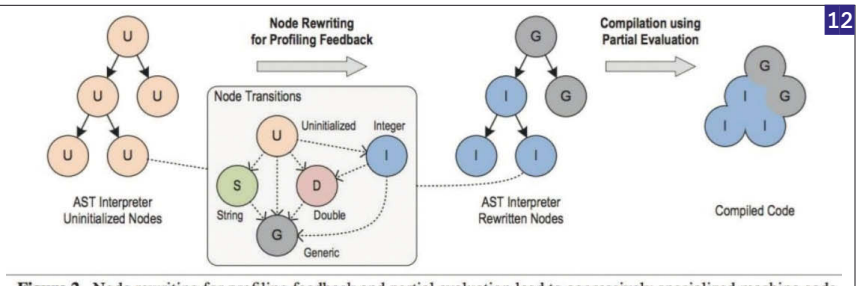


Figure 2. Node rewriting for profiling feedback and partial evaluation lead to aggressively specialized machine code.

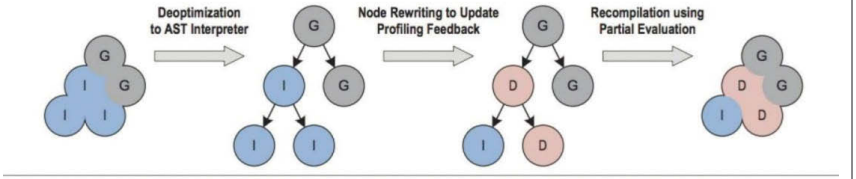
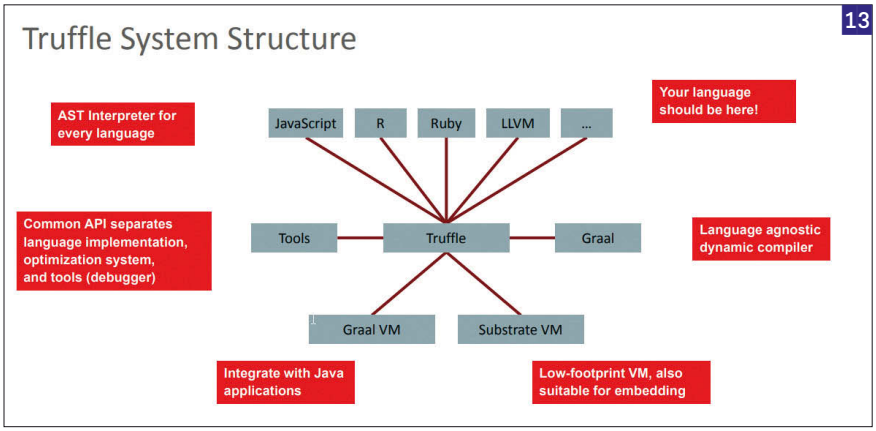
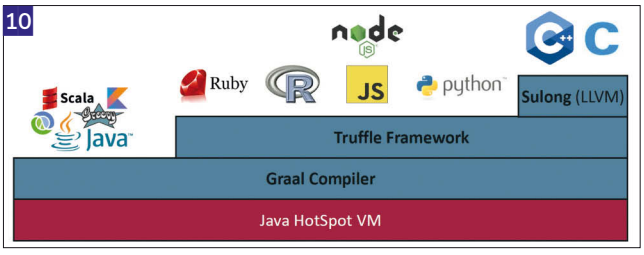


Figure 3. Deoptimization back to the AST interpreter handles speculation failures.

13



10



Autre exemple, en prenant le code Ruby suivant :

```ruby
def clamp(num, min, max)
  [min, num, max].sort[1]
end

def cmyk_to_rgb(c, m, y, k)
  Hash[{
    r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
  }.map { |k, v| [k, clamp(v, 0, 255)] }]
end

benchmark do
  cmyk_to_rgb(rand(255), rand(255), rand(255), rand(255))
end
```

L'on peut simplifier, l'écriture de la fonction clamp en JavaScript comme suit :

```ruby
require 'v8'

context = V8::Context.new

$clamp = context.eval("
  function clamp(num, min, max) {
    if (num < min) {
      return min;
    } else if (num > max) {
      return max;
    } else {
      return num;
    }
  }
  clamp;
")

def cmyk_to_rgb(c, m, y, k)
  Hash[{
    r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
  }.map { |k, v| [k, $clamp.call(v, 0, 255)] }]
end
```

Oracle Labs a testé ce code sur différentes implémentations natives Ruby, Java, JavaScript ; avec les frameworks Rhino, Nashorn et bien sûr avec la GraalVM (incluant Truffle donc). Les gains apportés par GraalVM sur cet exemple simple mixant du JavaScript et du Ruby sont illustrés sur le graphique suivant : 14

Ce qu'il faut en retenir : Truffle génère du bytecode qui est optimisé par le compilateur Graal JIT. Ce code est identique que l'on soit en pure Ruby ou un mixe de Ruby et JavaScript. Les autres implémentations telles que JRuby, Rhino, Nashorn, V8 ou Ruby natif démontrent soit la lenteur des machines virtuelles soit le coût en performance du passage de structure de données d'un langage à un autre, ici de Ruby à JavaScript.

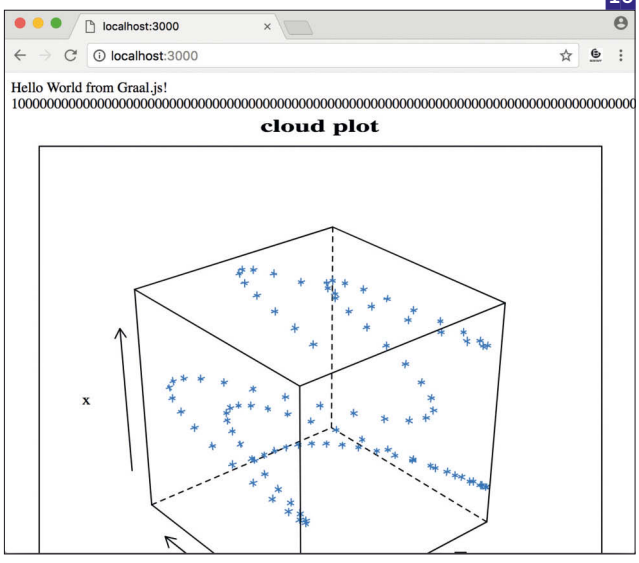
Pour tester rapidement la programmation polyglotte avec GraalVM, je vous recommande le repository github suivant : https://github.com/graalvm/graalvm-demos/ et notamment la démonstration polyglot-javascript-java-r. Cette application utilise l'environnement Node.js avec la GraalVM embarquée !

```javascript
const express = require('express')
const app = express()

const BigInteger = Java.type('java.math.BigInteger')


app.get('/', function (req, res) {
  var text = 'Hello World from Graal.js!<br> '

  // Using Java standard library classes
  text += BigInteger.valueOf(10).pow(100)
            .add(BigInteger.valueOf(43)).toString() + '<br>'

  // Using R methods to return arrays
  text += Polyglot.eval('R',
      'ifelse(1 > 2, "no", paste(1:42, c="|"))') + '<br>'

  // Using R interoperability to create graphs
  text += Polyglot.eval('R',
    `svg();
     require(lattice);
     x <- 1:100
     y <- sin(x/10)
     z <- cos(x^1.3/(runif(1)*5+10))
     print(cloud(x~y*z, main="cloud plot"))
     grDevices:::svg.off()
    `);

  res.send(text)
})

app.listen(3000, function () {
  console.log('Example app listening on port 3000!')
})
```

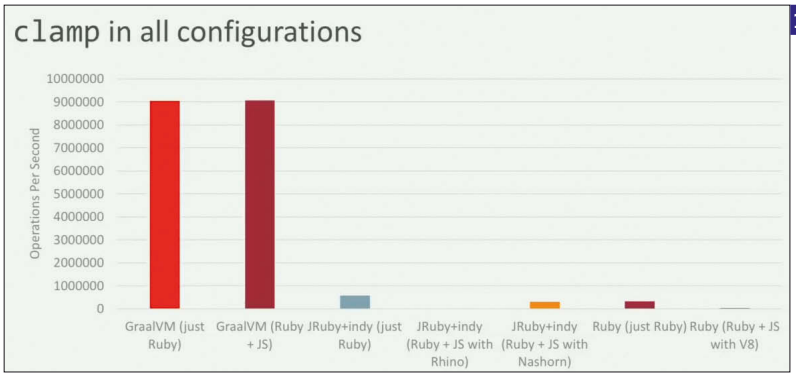
Un serveur HTTP est créé puis l'on s'interface avec le langage Java pour utiliser la classe java.math.BigInteger avant de demander la génération d'une image en SVG avec la librairie R lattice : 15

D'ailleurs au sujet du langage R, la version embarquant la GraalVM s'appelle FastR. Pour l'installer il suffit d'utiliser l'outil gu :

15



14



```
$ wget -nv https://github.com/oracle/graal/releases/download/vm-19.1.1/graalvm-ce-linux-amd64-19.1.1.tar.gz
...
[322434905/322434905] -> "graalvm-ce-linux-amd64-19.1.1.tar.gz.1" [1]

$ tar -xvf graalvm-ce-linux-amd64-19.1.1.tar.gz
graalvm-ce-19.1.1/lib/installer/installer.jar
graalvm-ce-19.1.1/lib/installer/bin/gu
graalvm-ce-19.1.1/lib/installer/components/polyglot/.registry
graalvm-ce-19.1.1/jre/lib/graalvm/launcher-common.jar
...

$ ./graalvm-ce-19.1.1/bin/gu install R
Downloading: Component catalog from www.graalvm.org
Processing component archive: FastR
Downloading: Component R: FastR  from github.com
Installing new component: FastR (org.graalvm.R, version 19.1.1)
...

$ ./graalvm-ce-19.1.1/jre/languages/R/bin/configure_fastr
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking for pwd... /usr/bin/pwd
checking whether builddir is srcdir... yes
...
```

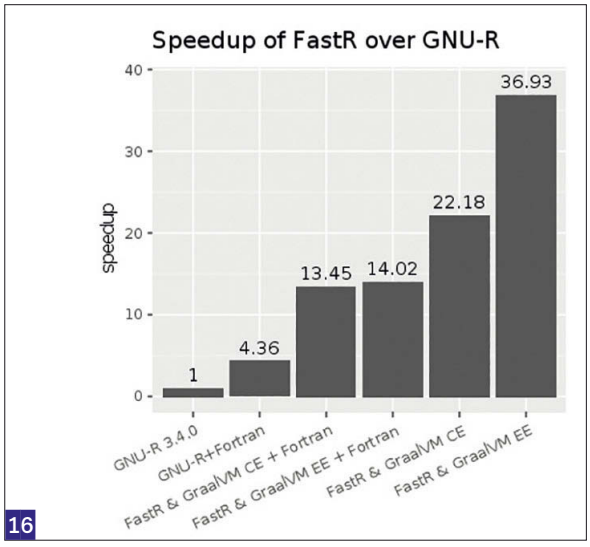
Et la console est accessible via le programme R :

```
$ ./graalvm-ce-19.1.1/bin/R
R version 3.5.1 (FastR)
Copyright (c) 2013-19, Oracle and/or its affiliates
Copyright (c) 1995-2018, The R Core Team
Copyright (c) 2018 The R Foundation for Statistical Computing
Copyright (c) 2012-4 Purdue University
Copyright (c) 1997-2002, Makoto Matsumoto and Takuji Nishimura
All rights reserved.

FastR is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information.

Type 'q()' to quit R.
>
```

FastR est une alternative à GNU-R, l'implémentation de référence de l'environnement d'exécution du langage R. Bénéficiant d'une très large bibliothèque de packages (CRAN), le langage R est connu pour sa lenteur à traiter de larges jeux de données. Ainsi, le code R est généralement optimisé en le réécrivant en Fortran ou en C.

Oracle Labs a encore ici réalisé des tests en comparant différentes implémentations pour exécuter un algorithme de raytracing en R : GNU-R, GNU-R et l'optimisation en Fortran, FastR (embarquant GraalVM CE puis EE) avec l'optimisation Fortran, FastR (CE puis EE) en réimplémentant l'optimisation Fortran en R natif. 16

Le niveau d'accélération est édifiant : facteur de 22 à 37. Imaginez maintenant les analyses que vous pourriez réaliser en bénéficier de telles performances : des jeux de données plus grands, des modèles plus précis… Imaginez maintenant l'approche polyglotte en utilisant les packages R dans un programme Python ou bien l'inverse, combinez ces packages R avec Spark…



16

Enfin la GraalVM EE peut aussi exécuter du code natif C, C++, Rust ou Fortran dans un mode « bac à sable » grâce à son intégration avec le framework Sulong. Un cas d'usage très utile : faire fonctionner du code natif dans un espace sécurisé qui contrôle les accès mémoires et qui remonte les exceptions en cas d'accès à des zones mémoires interdites (pointeurs douteux, tableaux non initialisés, dépassement de limites, etc.).

GraalVM embarquée

Dernière utilisation de la GraalVM, et non des moindres : la possibilité d'être embarquée. Nous l'avons déjà vu un peu plus tôt avec l'environnement Node.js et FastR mais depuis 2018, Oracle a fourni une image Docker avec une bêta version de la base de données multi-model Oracle incluant quelques fonctionnalités appelées Multilingual Engine (MLE) : https://www.oracle.com/technetwork/database/multilingual-engine/overview/index.html

Cette bêta version offre la possibilité de créer des procédures stockées écrites en JavaScript ou Typescript. Pour les entreprises et développeurs qui ont investi sur le langage JavaScript, c'est une véritable opportunité de compléter leur approche « Full-Stack » en développant dans un même langage le front-end, le back-end et le data store mais aussi en réutilisant leur code et les packages (npm) existant sur toutes ces couches.

Exemple de code JavaScript :

```
helloword.js:
module.exports.helloworld = function () { return "Hello World"; }

helloword.d.ts:
export function helloworld() : string;

shell:
shell> dbjs deploy helloworld.js -u myuser -p mypasswd -c mydb:1521/globaldbname

sqlplus:
sql> select helloworld() from dual;
```

A noter que Python est également disponible en bêta test. Imaginez alors cette future plateforme avec des capacités de gestion de données uniques sur le marché : un seul runtime, ultra performant, une intégration avec la meilleure des bases de données multimodèles du marché et un nombre énorme de packages (pip, npm… et CRAN) à portée de code !

Conclusion

GraalVM offre une multitude de possibilités tant aux nouveaux développeurs qu'à ceux plus aguerris. C'est une petite révolution de l'écosystème Java mais aussi Scala, Kotlin, JavaScript/Node.js, Python, R, Ruby, etc. qui est en train d'arriver. C'est aussi le résultat d'années de R&D qui portent enfin leurs fruits.

Le meilleur conseil que je puisse vous donner : téléchargez la GraalVM, testez votre code, essayez les images natives, imaginez des approches polyglottes, jouez avec les procédures stockées Oracle en JavaScript et Python, donnez du feedback et faites vivre cette communauté !

Remerciements

Merci à Thomas Wuerthinger et Oleg Šelajev pour leur aide dans l'écriture de cet article.

Julian Fischer,
anynines

Déploiements multicloud avec Cloud Foundry

Cloud Foundry est une plateforme se prêtant aux déploiements multicloud. Cet article démontre comment déployer des applications vers plusieurs plateformes cloud.

niveau 200

Application boursière type

Supposez qu'un microservice simple permettant de stocker et de retrouver les derniers cours boursiers soit mis au point et déployé sur deux plateformes différentes dans deux zones géographiques distinctes.

L'application est écrite en langage Go [go-lang] et la programmation réduite au strict minimum. Les nouveaux cours boursiers et actualisations sont acceptés dans ce format :

```
curl -d "name=BASF&value=6523" -X POST http://localhost:8080/stock
```

Pour retrouver des cours boursiers, il suffit d'exécuter cette commande :

```
curl http://localhost:8080/stocks
```

1

Code complet sur : https://github.com/fischerjulian/gostock

Référencement sur la plateforme

Pour opérer un déploiement sur Cloud Foundry, un accès développeur est requis. La plupart des fournisseurs proposent un essai ou un niveau gratuit pour tester cette plateforme et/ou exécuter une application simple, comme l'application boursière.

Il existe de multiples fournisseurs Cloud Foundry et le processus de référencement varie de l'un à l'autre. Parmi les offres proposées figure celle de paas.anynines.com. Le processus de référencement est explicite. 2

Au terme du processus d'inscription, l'environnement Cloud Foundry cible peut être défini :

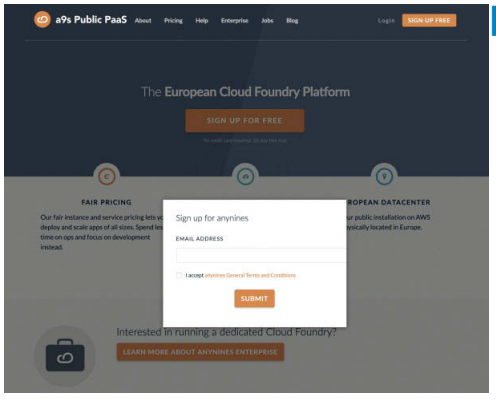
```
cf api https://api.de.a9s.eu
```

Voilà qui indique à la commande cf en local l'environnement Cloud Foundry auquel adresser les appels API. Un système d'identification reconnaît les développeurs d'applications autorisés.

```
cf:gostock demo$ cf login
Noeud final de l'API : https://api.de.a9s.eu

Email> demo@anynines.com

Password>
Identification...
OK

Sélectionnez une organisation (ou appuyez sur Entrée pour ignorer) :
1. demo_anynines_com

Org> 4
```

```
{
  - stocks: {
      - 3: {
            ID: 3,
            name: "Apple",
            value: 17780,
            CreatedAt: "2019-05-11T18:37:53+02:00",
            UpdatedAt: "2019-05-11T18:37:53+02:00"
        },
      - 4: {
            ID: 4,
            name: "Alphabet Inc Class A",
            value: 102140,
            CreatedAt: "2019-05-11T18:37:53+02:00",
            UpdatedAt: "2019-05-11T18:37:53+02:00"
        },
      - 5: {
            ID: 5,
            name: "BASF",
            value: 6523,
            CreatedAt: "2019-05-12T10:34:43+02:00",
            UpdatedAt: "2019-05-12T10:34:43+02:00"
        },
      - 6: {
            ID: 6,
            name: "BASF",
            value: 6523,
            CreatedAt: "2019-05-12T10:34:47+02:00",
            UpdatedAt: "2019-05-12T10:34:47+02:00"
        }
    }
}
```

1 *Capture d'écran illustrant la valorisation boursière des titres extraits en retour au format JSON.*



2 *Capture d'écran illustrant le processus d'inscription.*

```
Organisation ciblée demo_anynines_com

Sélectionnez un espace (ou appuyez sur Entrée pour ignorer) :
1. production
2. staging
3. test

Space> 1
```

```
Espace ciblé production



Noeud final d'API :   https://api.de.a9s.eu (Version de l'API : 2.131.0)
Utilisateur :       demo@anynines.com
Organisation :      demo_anynines_com
Espace :            production
```

Le processus d'identification aboutit à la sélection de l'*organisation* et de l'*espace* Cloud Foundry. Les organisations et espaces sont autant de locataires sur une plateforme Cloud Foundry qui permettent à chacun d'eux de représenter d'autres unités organisationnelles ou fonctionnelles. Le scénario le plus courant consiste à diviser une organisation en différents environnements de développement (*production*, *staging* et *test)* au moyen d'espaces. Pour autant, ce modèle n'a rien d'obligatoire, et son utilisation n'est pas de rigueur. À partir du moment où une organisation et un espace sont sélectionnés, le déploiement applicatif peut commencer.

Déploiement de l'application via cf push

En définitive, le déploiement de l'application avec Cloud Foundry est simple :

```
cf push
```

Concrètement, la commande cf charge l'application sur Cloud Foundry, déclenchant ainsi le processus de *staging*. Le *staging* (ou simulation) consiste à analyser une liste de *buildpacks* configurés par le fournisseur CF jusqu'à trouver celui correspondant à l'application. Un buildpack est une bibliothèque qui vise à doter une application donnée de l'environnement d'exécution approprié. Si, pour une application Ruby, cet environnement englobe un interpréteur Ruby et le *bundler* Gem, pour une application Go, il suffit d'un …. comme …. L'analyse de l'application a permis de définir le *buildpack* Go comme une correspondance de … .
À l'issue de cette phase de simulation applicative, un *droplet* est créé, contenant une version exécutable de l'application boursière. Cloud Foundry ne fait aucun distinguo entre les droplets qui, tous, sont associés à des applications exécutables, indépendamment de leur contenu.
Au terme de la simulation, plutôt que d'exécuter l'application directement, l'environnement *staging* est détruit et un processus d'orchestration est lancé. Cloud Foundry signifie son intention d'exécuter l'application boursière sur 2 instances applicatives. Grâce à l'orchestration, l'application est lancée en deux endroits via le runtime applicatif Cloud Foundry baptisé Diego []. **L'application est, en principe, en cours d'exécution. Il convient de s'en assurer :**

```
cf push

renvoie :

Start unsuccessful
```

Une analyse plus approfondie de :

```
cf logs gostock --recent
```

met en évidence une liste relativement longue d'entrées de journal :

```
Cannot create db schema: %!(EXTRA *net.OpError=dial tcp 127.0.0.1:5432: connect:
 connection refused)
```

L'application n'a pas été lancée car le *staging* a échoué en raison d'une **dépendance manquante : la base de données**.

Persistance via des services de données

La plateforme Cloud Foundry exige du service web qu'il soit *stateless* et délègue la persistance à une **instance de service de données** distincte [application 12 facteurs]. L'application boursière utilise PostgreSQL [] comme magasin de données.
L'application s'appuie sur une variable d'environnement dans son environnement d'exécution présentant la structure suivante :

```
{
 "VCAP_SERVICES": {
  "a9s-postgresql10": [
   {
    "binding_name": null,
    "credentials": {
     "host": "pgd673318-psql-master-alias.node.dc1.a9ssvc",
     "hosts": [
      "pgd673318-pg-0.node.dc1.a9ssvc"
     ],
     "name": "pgd673318",
     "password": "a-random-password",
     "port": 5432,
     "uri": "postgres://a9sb9438b9545053269280dcfa501d9a713b6624d39:a-random-password
@pgd673318-psql-master-alias.node.dc1.a9ssvc:5432/pgd673318",
     "username": "a9sb9438b9545053269280dcfa501d9a713b6624d39"
    },
    "instance_name": "gostockdb",
    "label": "a9s-postgresql10",
    "name": "gostockdb",
    "plan": "postgresql-single-nano",
    "provider": null,
    "syslog_drain_url": null,
    "tags": [
     "sql",
     "database",
     "object-relational",
     "consistent"
    ],
    "volume_mounts": []
   }
  ]
}
}
```

L'application interprète les identifiants d'accès au SGBDR à partir de la variable d'environnement VCAP_SERVICES et les analyse conformément à la structure de données JSON. Si l'organisation générale de cette structure JSON est définie dans l'API Open Service Broker, les détails peuvent varier d'un fournisseur à l'autre.
Mais quelle est l'origine du service PostgreSQL ? Techniquement, le *runtime* applicatif Cloud Foundry ne fournit pas de services de don-

nées. En revanche, la plupart des prestataires Cloud Foundry proposent un jeu standard de services de données. Ceux-ci sont intégrés au moyen de Service Brokers représentant des API standard [OSB API] pour décrire, allouer et relier les instances de services de données appelées à être utilisées par les développeurs d'applications. Par conséquent, pour stocker continuellement des cours boursiers, il est nécessaire de créer au préalable une instance de service PostgreSQL :

```
cf create-service a9s-postgresql10 postgresql-single-nano gostockdb
```

Dans les minutes qui suivent, une machine virtuelle de base de données PostgreSQL dédiée est allouée. L'instance de service nouvellement créée se nomme *gostockdb*, conformément à ce qui est indiqué dans la commande de création. Pour autant, l'application ne peut encore l'utiliser, faute des identifiants d'accès indispensables mentionnés plus haut. Pour allouer un accès applicatif, un utilisateur de base de données doit être créé. Ce processus se nomme création d'une *liaison de service* dans Cloud Foundry.

```
cf bind-service gostock gostockdb
```

La liaison de service correspond à un ensemble unique d'identifiants spécifiquement réservés à la connexion de cette application donnée à cette instance de service précise.
À présent que les dépendances de l'application boursière sont définies, nous pouvons reprendre le déploiement applicatif.

```
cf restage gostock
```

La commande restage déclenche à nouveau l'exécution du *buildpack*. Le *buildpack* Java, par exemple, tient compte des liaisons de service et injecte des fonctionnalités de base de données dans le conteneur.

Accès à l'application

Le déploiement de l'application est à présent effectif et celle-ci est prête à l'emploi.

Déploiement sur la seconde plate-forme

Pour l'instant, l'application est déployée sur une seule plate-forme. Le moment est venu de s'intéresser aux modalités de son déploiement sur une seconde plateforme — run.pivotal.io ou *PWS*.
Les identifiants obtenus à l'issue de la création d'un compte peuvent servir à définir le point terminal (*endpoint*) et le login de l'API :

```
cf login -a api.run.pivotal.io
```

Il convient à nouveau de choisir une *organisation* et un *espace*.
Comme sur paas.anynines.com, une base de données PostgreSQL est indispensable à l'exécution de l'application. Cependant, chaque prestataire Cloud Foundry peut fournir différents services de données assortis d'implémentations diverses. Sur PWS, une base de données PostgreSQL peut être créée en faisant appel à *ElephantSQL*, laquelle fera l'affaire pour notre application de démonstration. Pour les charges de travail de production, néanmoins, il convient d'examiner de près l'implémentation des services de données. Deux bases de données PostgreSQL peuvent être aux antipodes l'une de l'autre en termes de performances et de sécurité. Si une base de données sur un serveur PostgreSQL commun peut suffire pour des applications gadgets, une application très sollicitée risque de nécessiter un environnement PostgreSQL spécialisé et robuste, avec trois nœuds dédiés et une fonction de réplication assurant une bascule rapide. La création de la base de données PostgreSQL est tout aussi simple :

```
cf create-service elephantsql turtle gostockdb
```

Idem pour la mise en production de l'application :

```
cf push gostock
```

et sa liaison à l'instance du service de base de données.

```
cf bind-service gostock gostockdb
```

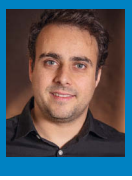
Une recréation rapide de l'exécutable

```
cf restage gostock
```

et l'application peut être exploitée sur une seconde plateforme : https://gostock.cfapps.io/stocks

Synthèse

Cet exemple montre à quel point les déploiements multicloud peuvent être facilités avec Cloud Foundry. L'expérience utilisateur *cf push* facilite la mise en œuvre des applications. Les applications déployées sur une plateforme Cloud Foundry donnée s'exécuteront également sur d'autres platesformes éponymes. Ce principe est vrai dès lors que l'opérateur de la plateforme a résolu la problématique des données en proposant un jeu standard de services de données, en libre-service à la demande entièrement automatisé, parfaitement intégrés à la plateforme via des service brokers. Les développeurs d'applications peuvent ainsi recourir à *cf marketplace* pour créer des instances de services de données venant compléter Cloud Foundry pour en faire une plateforme adaptée à des applications multicloud véritablement portables. •



Steve Houël
Architecte Cloud & DevOps chez Ippon Technologies, évangéliste Serverless et contributeur sur des projets Open Source comme JHipster et daSWAG. Il est fier d'être un #GeekEnthusiast.

IPPON
Discovery to Delivery

Architecture Serverless : de la Théorie à la pratique 2/3

Aujourd'hui, de nouveaux services font parler d'eux dans le monde de l'IT. Lorsque l'on parle de fournisseur de services Cloud, c'est principalement le Serverless. Dans cet article, nous nous intéresserons aux différentes étapes qui constituent la réalisation d'un projet basé sur les services Serverless et aux impacts que ce type de technologie peuvent avoir sur notre conception.

niveau 200

Le choix du langage

L'un des points importants dans le cycle de vie de votre fonction est sa phase de "chauffe" (que l'on nomme 'cold start'). Celle-ci dépend de plusieurs facteurs :

- La taille de votre package contenant votre fonction ;
- Le runtime d'exécution ;
- La configuration (mémoire allouée, réseau, …).

Nous verrons dans une autre partie les différentes astuces pour optimiser ces éléments, ici nous ferons un focus sur le runtime. Comme vous devez sûrement le savoir tous les langages ne sont pas égaux face à ce démarrage, nous ferons ainsi la différence entre un langage comme le Java qui s'exécutera au travers d'une machine virtuelle à la différence d'un langage comme le Python, le NodeJs ou le Go. Aujourd'hui, dans un contexte Serverless où votre fonction va possiblement s'arrêter et se charger plusieurs fois par jour ou subir des appels concurrents, il y aura un impact direct sur l'expérience utilisateur de par cette latence de démarrage. Pour vous donner un ordre d'idée, nous parlons ici de millisecondes voire de secondes selon les configurations. Il sera de votre responsabilité d'optimiser ce temps, ainsi que votre code bien sûr, afin de garantir la meilleure expérience pour vos utilisateurs.

Étudions maintenant un exemple concret pour évaluer ces différences.

Méthodologie

Pour la réalisation de cette évaluation nous utiliserons une fonction de type HelloWorld et nous testerons les langages suivants sur le service AWS Lambda :

- Java
- NodeJS
- Python3
- Go
- C#

Pour le calcul de ce temps de chargement nous prendrons le temps d'exécution total de la fonction, étant donné que l'ajout d'une librairie pour monitorer précisément ce temps l'impactera aussi. Nous utiliserons aussi différentes configurations mémoires pour vous montrer l'impact de ce type de paramètre.

Résultats

Après exécution de ce test, nous obtenons les résultats suivants : 1

	Runtime	Memory	Duration (ms)
	nodejs8.10	1536	0,46
	nodejs8.10	1024	0,58
	nodejs8.10	512	1,17
	go1.x	1536	1,86
	go1.x	1024	2,78
	java8	1536	3,00
	java8	1024	3,28
	go1.x	512	3,99
	nodejs8.10	256	4,19
	java8	512	5,29
	nodejs8.10	128	5,38
	go1.x	256	5,72
	python3.7	1536	5,91
	go1.x	128	6,43
	python3.7	1024	9,23
	python3.7	512	10,91
	python3.7	256	12,23
	java8	256	17,20
	python3.7	128	26,08
	java8	128	45,20
	dotnetcore2.0	1536	241,87
	dotnetcore2.0	1024	367,48
	dotnetcore2.0	512	778,70
	dotnetcore2.0	256	1 577,58
1	dotnetcore2.0	128	3 211,52

Nous pouvons faire 2 observations sur ces résultats. La première est que NodeJS, à ce jour, bénéficie du plus petit cold start avec 0.46ms et 1536Mb de mémoire comparée à 241ms pour une fonction équivalente en .Net. Le deuxième point est que le temps de cold start évolue linéairement selon la configuration mémoire de votre fonction, attention cependant à bien prendre en compte l'impact coût dans ces configurations. Lors de l'utilisation d'une mémoire plus importante, le coût en termes d'exécution augmentera. Il vous faudra trouver le bon ratio coût / performance. L'autre point à garder en tête concerne le contenu de la fonction. Lors du développement d'un projet nous aurons forcément besoin de librairies pour simplifier nos interactions avec les autres services, ainsi le Java par exemple, aura un temps plus long de par l'utilisation du framework Spring et l'initialisation de son contexte.

Une autre observation concerne les efforts faits par AWS pour réduire en permanence ces temps. Si nous prenons l'étude faite par Yan Cui en 2017 dans son article (https://theburningmonk.com/2017/06/aws-lambda-compare-cold start-time-with-different-languages-memory-and-code-sizes/), nous avions pour le runtime Java (128Mb) un temps de plus de 4500ms, aujourd'hui nous avons pour une configuration équiva-

Architecture	Fréquence des requêtes par fonction	Nombre de fonctions	Requêtes en parallèle par fonction	Cold starts par fonction	Cold starts global
Foncton unitaire	1 / seconde	1	1	1	1
Fonction indépendante	1 / seconde	10	1	1	10
Foncton unitaire	100 / seconde	1	10	100	100
Fonction indépendante	100 / seconde	10	10	10	100

2

lente un temps d'environ 45ms. L'intégralité du code source de ce benchmark est téléchargeable sur mon GitHub (https://github.com/stevehouel/benchmark-serverless-function).

Votre projet Serverless

Etant donné que nous avons la capacité de déployer des fonctions de façon unitaire sur nos environnements, nombreux sont ceux qui veulent les rendre complètement indépendantes et parlent même de micro-services à tort. La réalité en est toute autre quand on a plus de 200 fonctions en production avec des liens de dépendances entre elles. Vous auriez entre autres des soucis de complexité et de duplication de code sans parler de la complexité de gestion du cycle de vie de vos fonctions. Pour ma part voici mon approche sur mes projets :

- les fonctions qui constituent les points finaux d'une API sont regroupées dans un seul et unique projet,
- les fonctions de traitement en arrière-plan sont regroupées dans un projet,
- chaque projet a son propre répertoire (un seul répertoire Git),
- les fonctions d'un projet sont testées et déployées ensemble (cycle de vie unique).

Les raisons d'être de cette stratégie de regroupement sont :

- d'atteindre une grande cohésion pour les fonctions connexes,
- d'améliorer le partage de code là où cela a du sens (les points finaux d'une API sont susceptibles de partager une certaine logique puisqu'ils opèrent dans le même domaine, cf Bounded context du Domain Driven Design).

Bien que les fonctions soient regroupées en projets, elles peuvent toujours être déployées individuellement. En revanche je préfère un déploiement unique de l'ensemble des fonctions du fait de :

- la simplicité : toutes les fonctions associées dans un projet ont le même numéro de version,
- la rapidité de déploiement : dans une certaine limite, nous pouvons paralléliser le déploiement de nos fonctions.

Un autre anti-pattern lors de l'utilisation de ce type de service est l'utilisation d'une fonction globale avec un point d'entrée unique pour l'intégralité de l'application. L'exécution du code par la suite se fera uniquement suite à un routage interne propre à la fonction. On peut légitimement penser qu'avoir une fonction unique puis une redirection vers un code dédié permettrait d'exécuter plus souvent notre fonction et ainsi d'éviter ce fameux "cold start" si bloquant, cependant cela ne représente pas la réalité des faits. Prenons le scénario d'une application :

- avec une montée en charge graduelle (sans pic inopiné qui provoquerait encore plus de chargements),
- avec une durée d'exécution par fonction courte, environ 100ms,
- composée de 10 services, ce qui conduira à 2 architectures pour nos tests,
  - une composée de 10 fonctions indépensdantes,
  - une autre avec une unique fonction globale.

Cela nous donne l'expérience suivante : 2

Nous observons que nous atteignons le même nombre de chargements dans les deux cas à partir du moment où le nombre de requêtes concurrentes augmente. Un résultat indirect sur la version monolithe sera l'augmentation de la durée de chargement vu le nombre plus important de modules ou de code (exemple de celui dédié au routage).

Votre stratégie de tests

De nombreuses personnes se posent la question de la stratégie de tests à adopter sur ce type de projet et sont parfois effrayées par l'ampleur que peut prendre la mise en place des tests d'intégrations, principalement sur la partie Mock. Dans le développement, il existe plusieurs typologies de tests, mais dans notre cas nous ne nous intéresserons qu'à ces 3 types de tests :

- **Tests unitaires** : tester unitairement vos fonctions pour garantir que les cas aux limites de votre domaine Métier sont respectés ;
- **Test d'intégration :** tester l'interaction entre les différents blocs de votre application, cela peut se faire entre différents modules au sein de votre code ou entre votre application et vos services tiers. L'objectif n'étant pas de valider le comportement de tous les cas mais de valider la gestion des cas nominaux et des erreurs (exemple : des erreurs de communication) ;
- **Test d'acceptance** : tester l'intégralité de votre système et de votre métier.

L'objectif ici n'est pas de relancer le débat sur la nécessité des tests dans votre code pour garantir le bon fonctionnement de votre application. Ici, nous étudierons plus particulièrement l'impact des services Serverless sur ces tests. La principale problématique à ce jour est la simulation du comportement de ces nouveaux services managés. Ils sont souvent basés sur des technologies propriétaires et sont difficilement simulables. Certains projets de la communauté Open Source ont cependant réussi et ont pu simuler le comportement de certains services comme DynamoDB (exemple du projet **LocalStack :** https://github.com/localstack/localstack). Pour ma part, ma conviction est que l'on doit au maximum utiliser des services distants, et cela pour plusieurs raisons :

- Les fonctionnalités de ces services évoluent très rapidement, tellement rapidement qu'une communauté Open Source ne peut pas être tout le temps identique à la version présente chez le fournisseur de services Cloud,
- De plus, au sein même d'un hébergeur, vous pouvez potentiellement avoir plusieurs versions de ce même service avec une disponibilité variable de certaines fonctions selon les régions,
- La problématique des droits d'accès et permissions IAM pour l'accès aux ressources est aussi un sujet important qui doit être pris en considération dès la phase d'intégration.

Je vous préconise d'utiliser directement les services de votre hébergeur pour vous éviter tous ces problèmes d'intégration qui pourraient rapidement devenir votre pire cauchemar. Si toutefois vous

avez des réticences à utiliser ce genre de tests, par les coûts qu'ils peuvent générer, il faut se rappeler que les fournisseurs de services Cloud possèdent des offres de free-tier très avantageuses (exemple pour AWS Lambda : 400 000 GO-secondes par mois sur une période illimitée).
Le point le plus important à retenir est de bien penser à remettre à l'état initial le(s) service(s) utilisé(s) lors de vos tests et ainsi de garantir une exécution isolée. Pour ma part, je supprime l'intégralité des données entre les tests et l'intégralité de mon infrastructure à la fin de mon pipeline.

Le déploiement de l'infrastructure

L'intégration d'une fonction est relativement simple. Il s'agit grossièrement d'un bout de code, dans un langage compatible avec le service (généralement NodeJS, Java, C#, Go et Python), qui est compilé (ou non), compressé et déployé. La partie complexe se trouve dans l'interaction et la configuration des fonctions avec les autres services (exemple de la gestion des permissions).
Nous avons aujourd'hui deux approches dans l'écosystème Serverless pour le déploiement de notre code :

- Une approche orientée infrastructure avec des technologies de type InfraAsCode (IaC),
- Une approche orientée framework à intégrer directement dans le code source.

Pour avoir une bonne vision de cet écosystème, je vous conseille d'aller voir le lien suivant : https://github.com/anaibol/awesome-serverless.

Approche Infrastructure

Cette première approche plus orientée infrastructure vous permettra de gérer facilement l'intégration de vos fonctions et de toutes les ressources qui leur sont liées. Ces technologies peuvent se comparer à de l'Infrastructure as Code (IaC) vu que l'intégralité du paramétrage se fait au travers d'un ou plusieurs fichiers de configuration via un format comme le Yaml ou le Json. Nous retrouverons bien sûr les services IaC proposés par les fournisseurs de services eux-mêmes comme SAM (Serverless Application model) basé sur Cloudformation ou ceux proposés par la communauté OpenSource comme le framework Serverless (https://www.serverless.com) qui permet d'avoir une configuration agnostique du fournisseur de services.
Vous trouverez ci-joint un exemple de configuration en utilisant SAM (https://github.com/awslabs/serverless-application-model) :

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Sample SAM Template

# More info about Globals: https://github.com/awslabs/serverless-application-model/blob/master/docs/globals.rst
Globals:
  Function:
    Timeout: 3
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function # More info about Function Resource: https://github.com/awslabs/serverless-application-model/blob/master/versions/2016-10-31.md#aws
serverlessfunction
    Properties:
      CodeUri: hello_world/
      Handler: app.lambda_handler
      Runtime: python3.6
      Environment: # More info about Env Vars: https://github.com/awslabs/serverless-application-model/blob/master/versions/2016-10-31.md#environment-object
        Variables:
          PARAM1: VALUE
      Events:
        HelloWorld:
          Type: Api # More info about API Event Source: https://github.com/awslabs/serverless-application-model/blob/master/versions/2016-10-31.md#api
          Properties:
            Path: /hello
            Method: get

Outputs:
  HelloWorldApi:
    Description: "API Gateway endpoint URL for Prod stage for Hello World function"
    Value: !Sub "https://${ServerlessRestApi}.execute-api.${AWS::Region}.amazonaws.com/Prod/hello/"

  HelloWorldFunction:
    Description: "Hello World Lambda Function ARN"
    Value: !GetAtt HelloWorldFunction.Arn

  HelloWorldFunctionIamRole:
    Description: "Implicit IAM Role created for Hello World function"
    Value: !GetAtt HelloWorldFunctionRole.Arn
```

Approche Développeur

Cette approche plus orientée framework de développement s'implémente directement dans le code de votre application et fait abstraction des aspects infrastructures. Vous n'aurez donc plus à développer des scripts InfraAsCode au sein de vos projets mais simplement d'implémenter vos différentes fonctions et utiliser les annotations fournies par le framework.
Un des exemples de ce type de framework développé par AWS se nomme Chalice (https://github.com/aws/chalice) :

```
from chalice import Chalice

app = Chalice(app_name="helloworld")

@app.route("/")
def index():
  return {"hello": "world"}
```

Et l'automatisation dans tout ça ?

Nous l'avons vu plus haut, lors de l'utilisation de ce type de service, vous serez amené à faire un choix de technologie pour vous aider au déploiement de vos fonctions. Que ce soit via l'approche infrastructure ou développeur, ces outils vous fourniront certaines fonctionnalités basées généralement sur une CLI pour vous simplifier l'exécution des différentes tâches d'intégration et de déploiement.

Le packaging de vos fonctions

Point important à savoir, la phase de packaging ne se définit pas simplement par le fait de compresser l'ensemble de vos librairies avec votre code source. Il faut garder en tête certaines subtilités qui pourraient devenir gênantes par la suite. Prenons l'exemple de l'installation des dépendances du langage Python. Il faut savoir que ces dépendances s'installent et dépendent de librairies de l'OS d'exécution, en l'occurrence Amazon Linux. Si vous ne prenez pas cela en compte, vous serez amené à rencontrer régulièrement une erreur de fichier avec l'extension **.so** invalide.

Heureusement pour nous, il y a un moyen rapide et efficace de contourner ce problème grâce à Docker et au projet OpenSource de *Lambci* (https://github.com/lambci/docker-lambda). Vous devrez lancer une image Docker qui contiendra les mêmes dépendances que l'environnement Lambda cible et vous serez alors en mesure de lancer la phase de construction et d'installation des dépendances directement à l'intérieur et ainsi de garantir le respect de l'environnement cible de votre fonction.

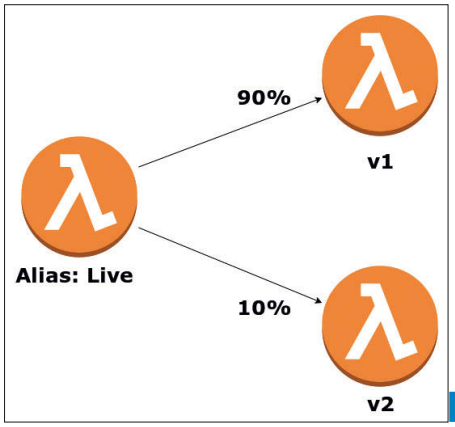
Voici à quoi ressembleraient des tâches de packaging Python dans un Makefile : 3

Stratégie de déploiement

On parle souvent d'intégration et de déploiement continu pour garantir la qualité de vos livrables et leurs déploiements sur vos différents environnements. L'objectif ici, sera de parler de cette stratégie de déploiement pour garantir le maintien fonctionnel de votre application sans impacter les utilisateurs.

Généralement, nous appliquons des techniques telles que **Canary Releases** et **Blue/Green Deployments** pour réduire les risques liés à vos mises en production. Dans le cadre du Serverless il est désormais possible et vivement recommandé d'effectuer ce genre d'opérations.

Heureusement pour nous, certains fournisseurs de services proposent nativement ces fonctionnalités de déploiement. Habituellement, le déploiement d'une fonction Lambda implique que toutes les invocations de fonctions exécuteront le nouveau code, soit parce



4

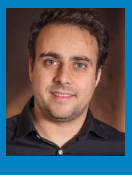
que nous mettons à jour $Latest (nous pouvons comparer cette notion au tag *latest* lors de la construction d'une image Docker), soit parce que nous pointons un alias défini vers une nouvelle version. Cela signifie que si quelque chose tourne mal, 100 % des invocations seront erronées et nous devrons rapidement revenir à la version précédente. Encore pire, nous pourrions ne pas remarquer le bug et laisser notre système dans un état incohérent. Cependant, avec l'introduction de l'**alias traffic shifting**, nous pouvons maintenant spécifier les poids de versions sur un alias, de sorte que toutes les invocations n'atteignent pas la nouvelle version et que seulement une certaine quantité de trafic soit destinée à la dernière version. 4

Nous pourrons par la suite augmenter progressivement la quantité de trafic vers la nouvelle version puis basculer totalement sur cette version, nous permettant de garantir son bon fonctionnement. Cette stratégie de déploiement n'est pas uniquement réservée aux fonctions et peut s'appliquer à d'autres services comme le déploiement de votre API. Malheureusement pour nous, ce mode de déploiement est encore très compliqué à mettre en oeuvre sans l'utilisation de services natif type **CodeDeploy** (https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/automating-updates-to-serverless-apps.html).

Suite et fin dans le numéro 233

3

```
build:
  $(VIRTUALENV) "$(VENV_DIR)"
  mkdir -p "$(VENV_DIR)/lib64/python$(PY_VERSION)/site-packages"
  touch "$(VENV_DIR)/lib64/python$(PY_VERSION)/site-packages/file"
  "$(VENV_DIR)/bin/pip$(PY_VERSION)" --disable-pip-version-check install -Ur requirements.txt
  find "$(VENV_DIR)" -name "*.pyc" -exec rm -f {} \;

bundle.local:
  zip -r -9 "$(ZIP_FILE)" src
  cd "$(VENV_DIR)/lib/python$(PY_VERSION)/site-packages" \
    && zip -r9 "$(ZIP_FILE)" *
  cd "$(VENV_DIR)/lib64/python$(PY_VERSION)/site-packages" \
    && zip -r9 "$(ZIP_FILE)" *

bundle:
  docker run -v $$PWD:/var/task -it lambci/lambda:build-python3.6 /bin/bash -c 'make clean build bundle.local'
```



Steve Houël
Architecte Cloud & DevOps chez Ippon Technologies, évangéliste Serverless et contributeur sur des projets Open Source comme JHipster et daSWAG. Il est fier d'être un #GeekEnthusiast.

L'observabilité appliquée au Serverless

Partie 2

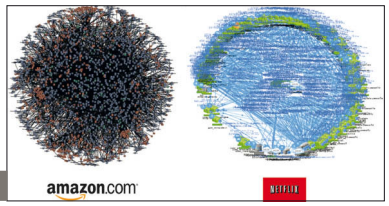
Aujourd'hui, de nouveaux services font parler d'eux dans le monde de l'IT et principalement lorsque l'on parle de fournisseur de services Cloud; c'est le Serverless. Nous nous attaquerons à un sujet bien précis : l'observabilité. Nous verrons ensemble les impacts sur votre conception lorsque l'on applique ces principes à ce type d'architecture Serverless.

niveau 100

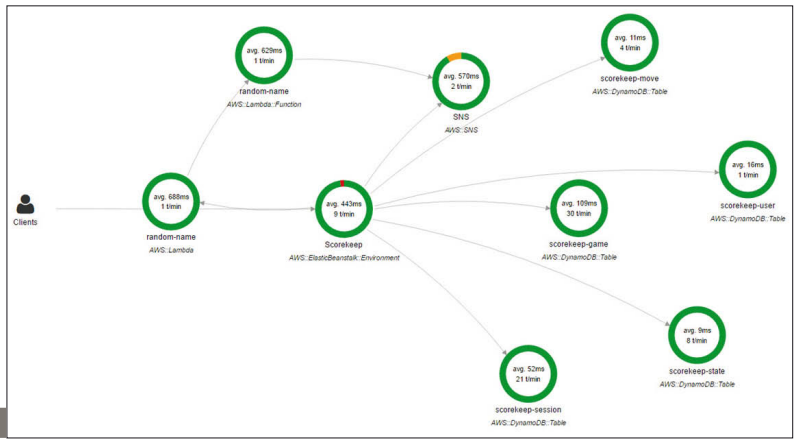
La transmission d'informations de bout en bout

De nos jours les architectures deviennent de plus en plus complexes. L'une des conséquences de cette complexité est le nombre grandissant d'interactions entre les services. Voici 2 parfaits exemples d'architectures contenant plus de 500 micro-services avec Netflix et Twitter : 3

Lorsque l'on réalise ce type d'architecture, nous nous heurtons forcément à la problématique de la gestion des journaux d'évènements et de notre capacité à garantir un suivi de bout en bout lors de l'exécution d'une requête sur plusieurs services. Ce cas est d'autant plus vrai lorsque l'on parle d'architecture Serverless où chacune de nos fonctions correspond à une entité indépendante. Une autre problématique se trouve aussi sur notre capacité à garantir ce lien au travers des briques d'infrastructures managées parfois méconnues. Certains services proposent nativement ce genre de fonctionnalité à un certain niveau. Nous prendrons l'exemple ici d'AWS X-Ray qui permet simplement d'avoir une représentation des interactions entre nos services : 4



3



4

Toutefois, ce service n'apporte pas encore toutes les fonctionnalités que l'on souhaite surtout lorsque l'on souhaite investiguer et identifier un problème spécifique dans notre chaîne de traitement. Prenons en exemple un cas réel d'utilisation. Pour ce faire nous utiliserons l'architecture suivante comme modèle : 5

Cette architecture illustre un cas typique de communication entre plusieurs services métiers composés de différentes briques Serverless. Lorsqu'un développeur va vouloir identifier la cause d'une erreur survenue en production il devra :

- Récupérer les derniers journaux d'événements avec le contexte d'exécution pour analyser les différents entrants et sortants,
- Connaître la chaîne d'exécution de sa requête et possiblement identifier les points de blocages.

Dans un contexte Serverless, si vous avez prévu de positionner le flag DEBUG au travers des variables d'environnements, vous arriverez rapidement au problème de la prise en compte de ce changement. Rappelons-le, une fonction bénéficiera de cette mise à jour uniquement lors de la création d'une nouvelle instance, ce qui induira un Cold start et donc une latence pour vos utilisateurs finaux. L'autre problème est de devoir pousser cette modification sur l'intégralité de votre chaîne, ce qui selon sa complexité, peut être un travail fastidieux. Si vous souhaitez mettre en place une solution plus pérenne, je vous conseille d'adopter la solution qui consiste à passer directement un paramètre au sein de votre requête et de le transmettre sur les différentes briques technologiques. Vous serez alors capable de positionner plusieurs flags comme celui du DEBUG lors de l'envoi de votre requête ou de générer un ID de corrélation lors du premier traitement afin d'avoir un suivi sur l'ensemble de votre chaîne d'exécution. Dans le cas de notre architecture, nous aurions le traitement suivant : 6

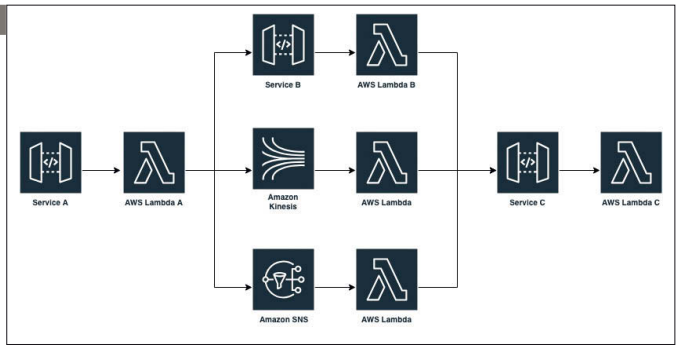
Nous observons que la valeur "*debug-enabled*" initialisée lors du premier appel API du Service A est transmise au travers de l'ensemble du système à l'identique de notre ID de corrélation "x-correlation-id" généré au sein de la Lambda A.

Maintenant, si nous détaillons un peu plus cette transmission par service, nous avons :

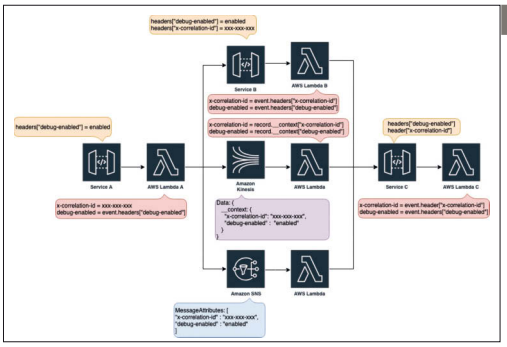
- **ApiGateway :** comme toute requête HTTP, nous pouvons transmettre ce style de données via les headers de la requête,
- **Lambda :** les informations sont récupérées directement depuis l'évènement source,
- **Kinesis :** malheureusement, ce service ne permet pas d'ajouter des informations de contexte à nos enregistrements. Pour ce faire, il nous faudra surcharger le contenu des données transmises dans l'élément "*Data*",
- **SNS :** sur ce service, nous pouvons utiliser le champ "*MessageAttributes*" pour envoyer nos informations.

Il vous suffira désormais de récupérer et fournir le traitement adéquat pour ce type d'information. Vous aurez ainsi la capacité de déclencher le mode DEBUG sur l'ensemble de votre chaîne uniquement via le positionnement du header "*debug-enabled*" par le développeur. Attention toutefois à ajouter une couche d'autorisation pour éviter tous les abus possibles et générer des coûts non voulus sur votre système d'observabilité.

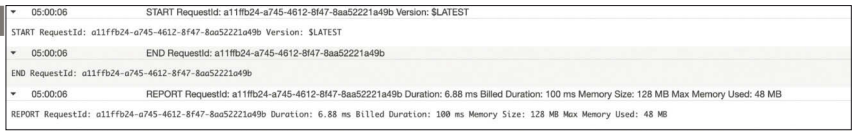
5



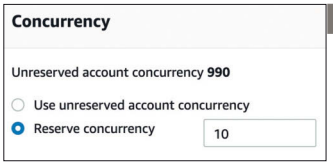
6



7



8



L'envoi de métriques personnalisées

Il est important de pouvoir récupérer des informations non standard sur l'exécution de nos fonctions, nous prendrons l'exemple de la consommation mémoire et du temps de facturation de notre fonction.

Ces deux valeurs mises en corrélation vont vous permettre d'optimiser votre fonction en trouvant le meilleur ratio mémoire / performance. Prenons l'exemple d'une fonction de 512Mo qui s'exécute en 42ms. Sachant qu'AWS Lambda fonctionne par bloc de 100ms, vous avez une perte de 58ms. Vous pourrez alors réévaluer votre configuration et descendre la mémoire utilisée à 256Mo ce qui aura un impact direct sur votre facture.

Un autre avantage d'avoir ce temps de facturation sous forme d'une métrique, est votre capacité à l'intégrer directement dans votre processus de développement, voire même dans votre pipeline d'intégration afin d'identifier les augmentations soudaines de complexité de votre code ou un mauvais algorithme. Rappelons-le, l'optimisation de votre code a un impact direct sur votre facture (exemple d'une fonction qui s'exécute en 300ms au lieu de 500ms vous fera réduire votre coût en fin de mois de 40%). Pour l'extraction, nous nous baserons sur l'approche de Datadog concernant l'envoi de métriques personnalisées vu sa simplicité de mise en place.

Le format

Si nous reprenons l'explication fournie par Datadog au sein de sa documentation, pour envoyer des métriques personnalisées, il faut envoyer un message avec le format DogStatsD (https://www.datadoghq.com/blog/statsd/).

```
MONITORING|unix_epoch_timestamp|metric_value|
metric_type|my.metric.name|#tag1:value,tag2
```

où *metric_type* est soit count, gauge, histogram, ou check.

Vous pouvez également utiliser l'API HTTP de Datadog pour les soumettre à partir de votre fonction Lambda directement.

Extraction des informations

Revenons à nos données. Pour extraire les 2 métriques souhaitées qui sont la mémoire consommée et le temps de facturation, nous allons devoir traiter les journaux d'évènements de notre fonction.

Pour rappel, lors de l'exécution d'une fonction AWS Lambda, nous pouvons récupérer la trace suivante : 7

À la fin de chaque invocation, AWS Lambda publie un message REPORT détaillant la mémoire consommée par votre fonction pendant cette invocation, et votre temps facturé. Concernant l'extraction, AWS nous permet de créer directement un flux de nos évènements et de les traiter par le biais d'une fonction AWS Lambda. Cette fonctionnalité est faite pour agréger nos messages et les envoyer à un système d'observabilité, mais ici nous utiliserons l'extraction de nos informations.

Et une lambda, une !

Intéressons-nous au code de notre fonction d'extraction. Vous trouverez ci-joint un exemple du code Python pour le traitement de ce rapport, l'extraction des informations et l'envoi sur votre serveur Datadog.

Code complet sur : https://github.com/steve-houel/datadog-serverless-functions-metrics.

Attention toutefois à vos limites

Je dois vous mettre en garde sur un point : la limite d'exécutions concurrentes. Comme vous devez sûrement le savoir, AWS Lambda est soumis à une soft limit (https://docs.aws.amazon.com/fr_fr/lambda/latest/dg/concurrent-executions.html) qui est de 1000 exécutions simultanées pour l'ensemble de vos fonctions. Selon votre volume de messages et de fonctions, vous pourrez être amené à rapidement l'atteindre. Cependant, l'une des fonctionnalités de ce service vous permet de fixer cette limite par fonction : 8

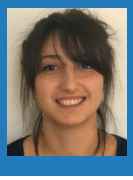
Vous pourrez alors maîtriser vos exécutions et ne pas vous retrouver dans le cas d'une panne en cascade de votre système.

Conclusion

Nous l'avons vu, l'observabilité ne peut pas être comparée au simple fait d'ajouter une couche de monitoring à son système. Cela va au-delà. Elle doit donner la capacité, au travers de différents types d'informations, d'avoir la connaissance nécessaire à la compréhension de son système ainsi que d'identifier et résoudre les problèmes qui peuvent survenir. Les systèmes basés sur les services Serverless ne dérogent pas à ces règles; la complexité en revanche se trouve dans la jeunesse de son écosystème et de l'intégration des différents outils spécialisés comme *Datadog, Espagon, Thundra ou Dashbird* avec les fournisseurs de services Cloud.

Toutefois, il y a une réelle mobilisation de la communauté et des entreprises pour rapidement remédier à ces problématiques et apporter toutes les fonctionnalités avancées existantes (Machine Learning, Prédiction de pannes, ...) et les appliquer aux cas d'utilisations spécifiques du Serverless. •

Designer des APIs REST avec Flask-RESTPlus



Dorra Dhouib
Consultante Sénior Python
INVIVOO BEYOND TECH

niveau 200

Partie 2

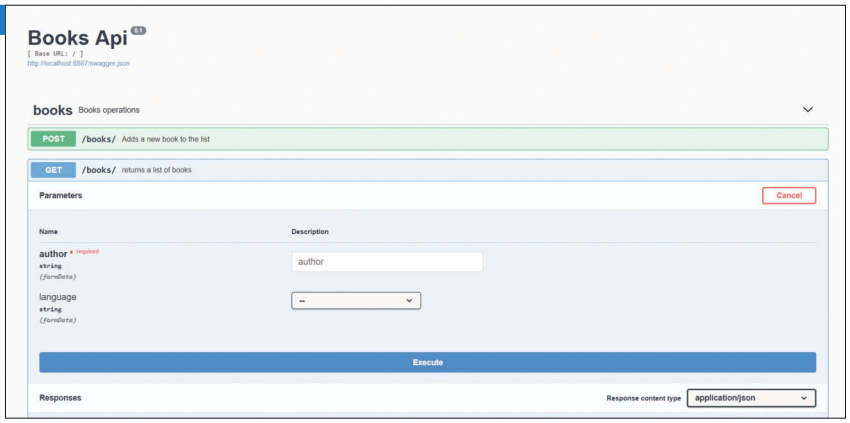
Validation

Certaines des méthodes http implémentées ont des paramètres d'entrées et/ou des valeurs de sorties qui seront peut-être utilisées à leur tour comme paramètres d'entrée.
Comment pouvons-nous valider le format de ces inputs / outputs ?
On peut se lancer dans des contrôles à base de *if* dans le corps de nos méthodes, mais il existe beaucoup plus simple.
Flask-RESTPlus fournit tous les outils nécessaires pour facilement valider les paramètres d'entrée et les données de sortie.

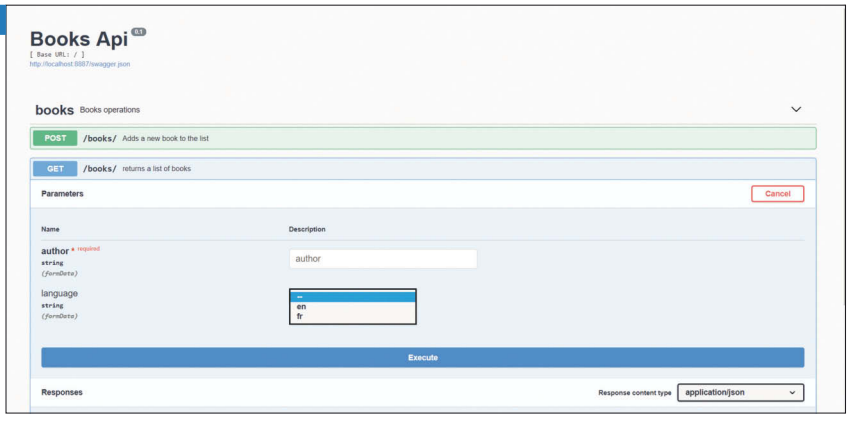
Validation des paramètres d'entrées pour les méthodes « GET »

Afin de définir ces paramètres, nous utilisons un objet appelé RequestParser. Cet objet a une fonction nommée add_argument(), qui nous permet de spécifier le nom du paramètre et ses valeurs autorisées. Une fois défini, nous pouvons utiliser le décorateur @api.expect pour attacher cet objet à une méthode.
Une fois que la méthode est décorée, l'interface utilisateur Swagger de la méthode affiche un formulaire pour spécifier les valeurs des arguments.
Le RequestParser permet de valider les valeurs d'arguments. Si une validation de valeur échoue, l'api renvoie une erreur HTTP 400 avec un message approprié.

5



6



À savoir que vous pouvez, activer ou désactiver la validation d'argument pour chaque méthode à l'aide de l'argument validate de la méthode @api.expect.
Vous pouvez également activer la validation à l'aide de la variable de configuration RESTPLUS_VALIDATE lors du démarrage de votre application Flask.

```
app.config('RESTPLUS_VALIDATE')=True
```

Pour créer un argument qui accepte plusieurs valeurs, utilisez le mot clé action et indiquez 'append' comme valeur :

```
parser.add_argument('multiple', type=int, action='append', required=True)
```

Pour spécifier une liste de valeurs d'arguments valides, utilisez le mot-clé choices et spécifiez un itérateur :

```
parser.add_argument("language", choices = ['en, 'fr'])
```

Reprenons notre api :

```
books_arguments = reqparse.RequestParser()
books_arguments.add_argument('author', type=str, required=True)
books_arguments.add_argument('language', type=str, choices=['en', 'fr'])


@ns_books.route("/")
class BooksList(Resource):
  @api.expect(books_arguments)
  def get(self):
    """
    returns a list of books
    """
    data = books_arguments.parse_args(request)

    if data.get('language'):
      cursor = mydb.books.find({'author': data.get('author'), 'language': data.get('language')},
{"_id": 0})
    else:
      cursor = mydb.books.find({'author': data.get('author')}, {"_id": 0})
    data = []
    for book in cursor:
      data.append(book)

    return {"response": data}
```

5 6

Validation des paramètres d'entrées pour les méthodes « POST » et « PUT »

Si vous souhaitez mettre à jour ou créer un nouveau livre, vous devez envoyer les données sérialisées de l'élément au format JSON dans le corps d'une requête.
Flask-RESTPlus vous permet de documenter et de valider automatiquement le format des objets JSON entrants à l'aide de modèles d'API. Un modèle d'API RESTPlus définit le format d'un objet en

répertoriant tous les champs attendus. Chaque champ a un type associé (par exemple, String, Integer, DateTime).
Définissons le modèle correspondant aux informations d'un livre :

```
from flask_restplus import fields

book_definition = api.model('Book Informations', {
  'author': fields.String(required=True),
  'title': fields.String(required=True),
  'language': fields.String(required=True)
})
```

Une fois le modèle défini, vous pouvez l'associer à une méthode à l'aide du décorateur @api.expect().

```
@ns_books.route("/")
class BooksList(Resource):
  @api.expect(books_arguments)
  def get(self):
    """
    returns a list of books
    """

  @api.expect(book_definition)
  def post(self):
    """
    Add a new book to the list
    """
    data = request.get_json()
    title = data.get('title')
    if title:
      if mydb.books.find_one({"title": title}):
        return {"response": "book already exists."}
      else:
        mydb.books.insert_one(data)
```

Nous le remarquons tout de suite, utiliser un modèle simplifie amplement le corps de la méthode POST en nous épargnant tous les tests sur les données reçues. 7

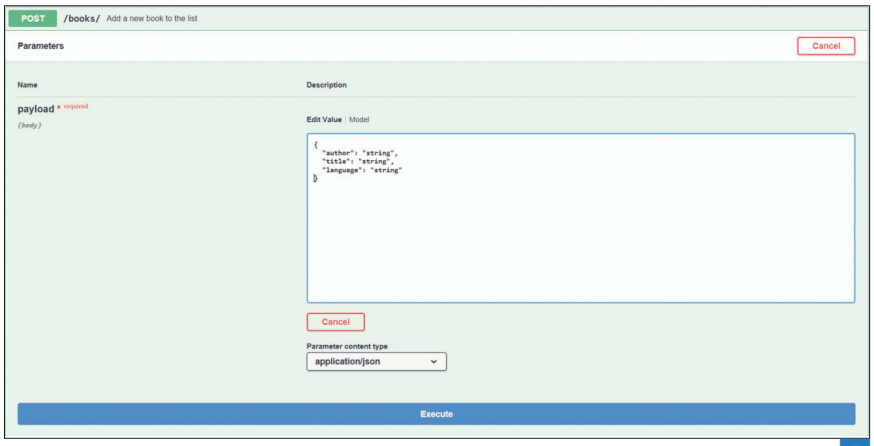
Options supplémentaires pour les fields :

- **required** : True si le champ est obligatoire
- **default** : valeur par défaut pour le champ
- **description** : description du champ (apparaîtra dans l'interface utilisateur Swagger)
- **exemple** : valeur d'exemple optionnelle (apparaîtra dans l'interface utilisateur Swagger)

Des options de validation supplémentaires peuvent être ajoutées aux fields :

- **String:**
  - **min_length** et **max_length** : longueur minimale et maximale d'une chaîne ;
  - **pattern** : une expression régulière à laquelle le string doit correspondre.
- **Nombres (entier, flottant, fixe, arbitraire) :**
  - **min** et **max** : valeurs minimales et maximales ;
  - **multiple** : le nombre doit être un multiple de cette valeur.

Un field d'un modèle d'api peut utiliser un autre modèle comme valeur attendue. Vous devez ensuite fournir un objet JSON en tant



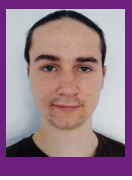
7

que valeur valide pour ce field.
items: fields.Nested (item)
Un champ peut également nécessiter une liste de valeurs voire une liste d'objets imbriqués.
'item_ids': fields.List (fields.Integer),
'items': fields.List (fields.Nested (item)

Validation des output JSON

Les modèles d'API peuvent également être utilisés pour les outputs JSON des méthodes.
Si vous décorez une méthode avec @api.marshal_with(modèle), **Flask-RESTPlus** générera un objet JSON avec les mêmes champs que ceux spécifiés dans le modèle.
La méthode doit simplement renvoyer à un objet ayant des attributs portant les mêmes noms que les champs. Sinon, la méthode peut renvoyer à un dictionnaire avec des valeurs attribuées aux mêmes clés que les noms des champs de modèle. •

```
list_books = api.model('Books', {
  'books': fields.List(fields.Nested(book_definition))
})


@ns_books.route("/")
class BooksList(Resource):
  @api.marshal_with(list_books)
  def get(self):
    """
    returns a list of books
    """
    data = books_arguments.parse_args(request)

    if data.get('language'):
      cursor=mydb.books.find({'author':data.get('author'),'language':data.get('language')}, {"_id":0})
    else:
      cursor = mydb.books.find({'author': data.get('author')}, {"_id":0})
    data = []
    for book in cursor:
      data.append(book)

    return {"books": data}
```



Maxime Ellerbach
lycéen
maxime@ellerbach.net

Mini voiture autonome avec un Raspberry Pi en Python

L'été dernier, j'ai fait un stage chez Magic Maker https://www.magicmakers.fr/. Le thème du stage portait sur l'intelligence artificielle. A l'issue du stage les animateurs nous ont proposé de participer à une course de mini-voitures autonomes impliquant de l'intelligence artificielle : http://ironcar.org. Etant intéressé, je me suis aussitôt inscrit.

niveau 200

Présentation du projet

Pour concourir, il me fallait donc une voiture et un programme basé sur de l'intelligence artificielle. Je vais tout d'abord vous parler de la voiture, sa construction ainsi que son évolution, par la suite, j'aborderai la partie code utilisant Python, récupération d'image et intelligence artificielle basée sur Keras et TensorFlow. Enfin, je conclurai sur le projet et sur les difficultés rencontrées.

Construction et évolution de la voiture

Ne voulant pas faire une voiture de A à Z, j'ai utilisé comme base une voiture radio télécommandée que j'ai eue à noël quand j'étais plus jeune.

En suivant les règles imposées, ma voiture devait embarquer :

- Une à deux cartes type Raspberry ou Arduino ;
- Une webcam ou un autre capteur de mon choix ;
- Le matériel nécessaire pour les faire fonctionner (câbles, alimentation…).

J'ai enlevé toute la partie radiocommandée ne laissant que le châssis, les moteurs et j'ai replacé le système permettant de gérer la direction par un servomoteur. Pour le premier prototype, j'ai utilisé du carton auquel j'ai fixé un Arduino, un RockPro64 et une caméra. L'Arduino pour gérer la partie moteurs et servomoteur et le RockPro64 pour s'occuper de la partie traitement d'images.

Note

le RockPro64 est une carte embarquant 4Go de Ram et un processeur hexacœur. Pour plus d'info : https://www.pine64.org/?page_id=61454

Voici le schéma électrique : 1

Lors d'une course, à cause des vibrations de la voiture, la vis qui bloquait le régulateur à 12V s'est dévissée causant une hausse de la tension qui a cramé la carte. À la suite de cet accident horrible, j'utilise maintenant un Raspberry Pi 3B+, une carte plus standard mais moins puissante.

Avec ce changement, il a fallu revoir la partie alimentation. Le Raspberry Pi pouvant être alimenté par un connecteur micro USB, j'ai utilisé une batterie externe de téléphone en guise d'alimentation à la place du régulateur de tension. Ayant remarqué que le H-bridge avait un peu de mal lorsque la voiture accélérait, je l'ai changé et opté pour un H-bridge pouvant supporter un plus grand ampérage.

J'ai profité de tous ces changements pour passer du carton à de l'acrylique pour assurer la solidité de la voiture.

Voici à quoi ressemble la voiture actuellement : 2

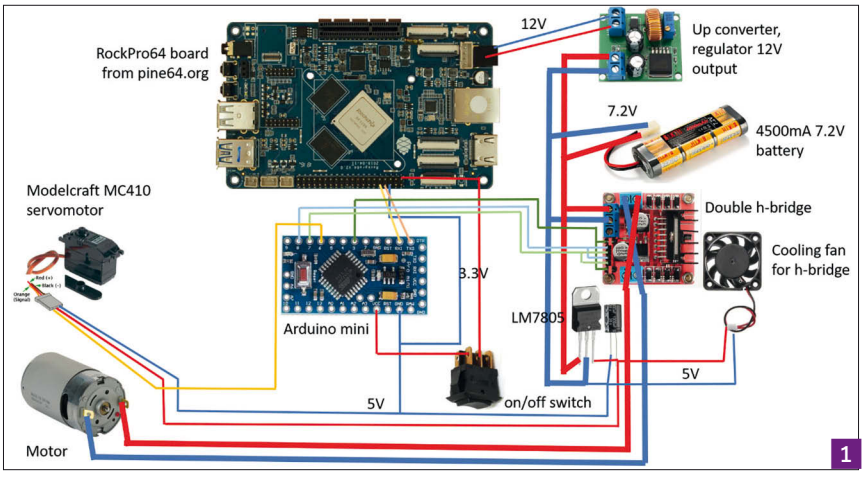
En bas, on retrouve la batterie pour les moteurs et le servomoteur. Au premier étage, on retrouve le H-bridge et l'Arduino. Au deuxième étage, il y a le Raspberry Pi et la caméra qui est inclinée pour offrir une vision d'ensemble de la piste. Enfin, au dernier étage, on retrouve la batterie externe pour alimenter le Raspberry Pi.

Contraintes de vitesse

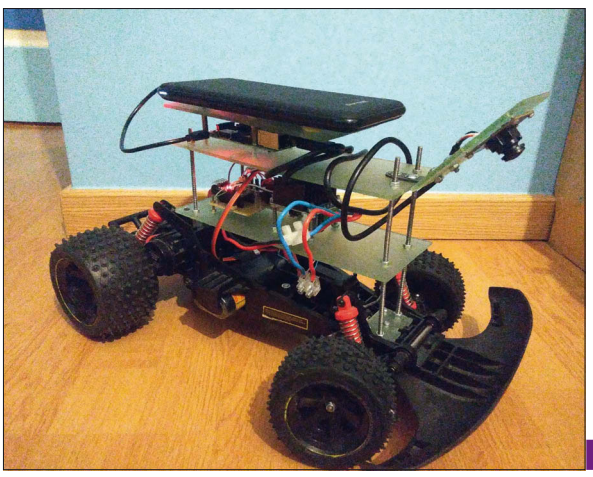
N'ayant qu'un Raspberry Pi pour effectuer les calculs, il fallait repenser mon code pour faire quelque chose de léger, un programme qui puisse traiter assez images par secondes pour avoir une réactivité suffisante. Dans un premier temps, je me suis fixé un objectif de 10 images par secondes.

Faire un graphe s'impose donc ! 3

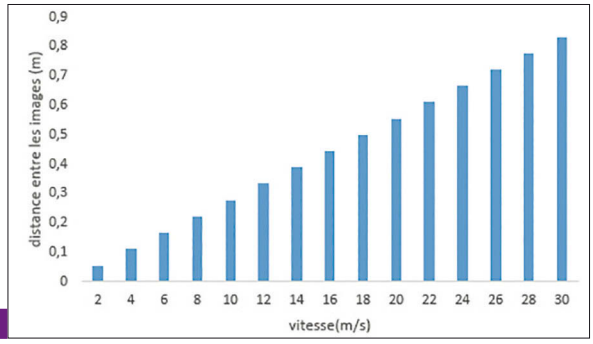
Ce graphe représente la distance entre deux images en fonction de la vitesse de la voiture avec 10 images par secondes. On constate



1



2



3

que si l'on veut rester précis, il faudrait rester en dessous de la barre des 0.3 m entre chaque image, ce qui nous impose d'avoir une vitesse maximum de 10 km/h environ.

Avec ces contraintes en tête, on peut passer au code mais d'abord à un peu de théorie !

Théorie autour du réseau de neurones à convolution

Pour ce projet, j'ai utilisé un réseau de neurones à convolution (CNN) pour diriger la voiture. C'est-à-dire, à partir d'une image qui provient de la caméra sur la voiture, prédire la direction que doit prendre la voiture.

La partie convolutive du CNN va servir à détecter des concepts, des formes d'une image (lignes, bords de la route, anomalies…), puis le réseau neuronal va servir à ajouter du sens à ces données et décider en fonction des observations, la direction par laquelle la voiture doit aller.

Afin d'entraîner notre CNN, il nous faut récupérer des images et les associer à un label (étiquette). Chaque label correspond à une direction que vont prendre les roues : maximum gauche, gauche, tout droit, droite et maximum droite.

Récupération d'images et labellisation

Pour la récupération d'image, je préfère utiliser OpenCV. C'est une librairie de manipulation d'images très complète. Le problème, c'est qu'elle n'est pas disponible sur Raspberry Pi avec l'outil d'installation de module « pip » de Python. Il a donc fallu le compiler manuellement, le guide pas à pas d'installation est disponible sur mon GitHub : https://github.com/Maximellerbach/AutonomousCar/blob/master/AutonomousCar/software.md

Une fois OpenCV installé, on peut procéder à la récupération d'images.

```
import cv2
import time
cap= cv2.VideoCapture(0)
while(True):
  _, img = cap.read()
  cv2.imwrite('img_sans_label\\'+str(time.time())+'.png',img)
```

On importe OpenCV avec « cv2 » et time, on initialise notre camera « cv2.VideoCapture(0) » puis dans une simple boucle « while », on récupère l'image « img » qui provient de la caméra. On la redimensionne puis l'enregistre dans le dossier « img_sans_label » avec comme nom la date en seconde.

Ce petit programme est fonctionnel, mais il faut pousser la voiture à la main et à force, ça fait mal au dos… Vous retrouverez sur mon Github un programme plus complexe qui dirige la voiture à partir d'une manette de Xbox 360, récupère les images et les labellise avec les valeurs du joystick : https://github.com/Maximellerbach/AutonomousCar/blob/master/AutonomousCar/JoyControl/RaspberryControl.py

Une fois nos images récupérées, il faut les labelliser. Pour ma part, je rajoute dans le nom de l'image son label associé (exemples : 0_1558543281.16499.png, 3_1558543281.305123.png). C'est plus simple pour trier les images selon leurs labels dans l'explorateur de fichier, mais il est tout à fait envisageable d'utiliser une autre méthode. Pour démarrer et faire quelques tests, je recommanderais de labelliser au moins 1000 images.

C'est un processus long mais nécessaire au bon fonctionnement de notre réseau de neurones à convolution. L'auto-labellisation est envisageable, mais requiert tout de même un réseau de neurones entraîné. Dans tous les cas, une labellisation manuelle au départ est toujours nécessaire. Maintenant que nos données sont labellisées, on peut passer au code !

Chargement des données et préparation des données

Pour tout ce qui concerne notre intelligence artificielle, nous utiliserons « keras ». Keras est une API de réseau de neurones, Keras peut s'utiliser avec Tensorflow, Theano ou encore CNTK. Nous utiliserons Keras avec Tensorflow. En utilisant Keras, la syntaxe change mais le code, au fond reste le même que si nous utilisions Tensorflow !

Si vous installez Tensorflow, sachez qu'il existe une version GPU pour les cartes graphiques Nvidia. Si vous en possédez une renseignez-vous sur l'installation de la version GPU ici : https://www.tensorflow.org/install/pip

Sinon, installez simplement la version processeur avec : « pip install tensorflow »

Vous pouvez aussi opter pour Google collab. C'est un environnement Jupyter Notebook qui ne demande aucune installation/réglages et qui exécute vos programmes dans le cloud. Tensorflow GPU y est nativement installé, lors de l'utilisation, un GPU vous sera attribué. Plus d'informations : https://www.tensorflow.org/install/ ; https://colab.research.google.com/notebooks/welcome.ipynb

Avant de créer notre CNN, il nous faut charger nos données labellisées pour constituer deux listes :

X qui contiendra nos images ;

Y qui contiendra nos labels correspondant aux images.

Pour cela, nous allons utiliser le module « Glob ». Il va nous permettre de récupérer toutes les images présentes dans un dossier.

On considère le programme suivant :

```
from glob import glob

import cv2
import numpy as np
from keras.layers import Conv2D, Dense, Dropout, Flatten
from keras.models import Sequential
from keras.utils import plot_model, to_categorical
from sklearn.model_selection import train_test_split
```

```
import autolib  # custom lib

X = []
Y = []

dos = 'images\\*'

for img_path in glob(dos):
img = cv2.imread(img_path)
img_flip = cv2.flip(img, 1)
    labels = autolib.get_label(img_path, flip=True, dico=[0,1,2,3,4])
X.append(img)
X.append(img_flip)

Y.append(labels[0])
    Y.append(labels[1])
```

On initialise nos listes X et Y, on définit le dossier dans lequel se trouvent nos images labellisées, puis, pour chaque image dans notre dossier, on va répéter les instructions suivantes :
Lecture de l'image ;

- Création d'une image « miroir » ;
- Récupération des labels de l'image « normale » et « miroir » à partir du nom de l'image grâce à une fonction du programme « autolib » qui comporte quelques fonctions que j'utilise régulièrement dans mes programmes ;
- Ajout de l'image « normale » puis « miroir » à X ;
- Ajout du label « normal » puis du label « miroir » à Y.

La création d'une image miroir est très importante. Elle permet d'équilibrer nos labels, c'est-à-dire d'avoir autant de labels vers la gauche que de labels vers la droite. Ce qui résulte en une meilleure convergence du modèle et une conduite plus fluide de la voiture.
Préparons maintenant nos données :

```
X = np.array(X) / 255
Y = to_categorical(np.array(Y), 5)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1)
```

On transforme nos listes X et Y en « array » en utilisant le module « numpy » puis on va normaliser nos images en les divisant par 255.
Du coté des Y, on applique la fonction « to_categorical ». C'est une fonction qui, à partir de nos labels 0,1,2,3 ou 4 (5 classes), va créer des labels sous forme de tableau (exemple avec 3 classes : label 1 = [1,0,0] ; label 2 = [0,1,0] ; label 3 = [0,0,1]).
Ensuite, on va séparer nos données en plusieurs sets qui nous serviront à entraîner notre CNN et à le tester. J'ai choisi de mettre dans les données de test 1/10 de X. Le reste se retrouvera dans les données d'entraînement.

Création du CNN

Pour ce modèle qui reste simple, nous allons utiliser un modèle dit séquentiel, c'est-à-dire que chaque couche va avoir pour entrée la sortie de la couche précédente.

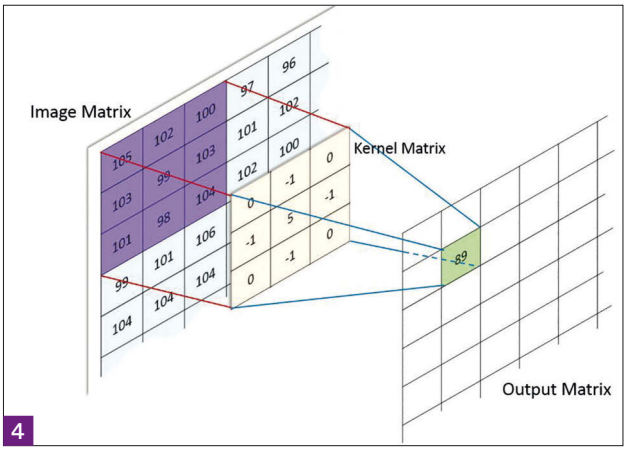
```
model = Sequential()
```

Avec « Sequential() » pour rajouter une couche, il suffit d'écrire « model.add » suivi de la fonction que vous voulez ajouter.
Le modèle devant tourner sur un Raspberry Pi, doit être léger. Pour qu'il soit le plus léger possible, il faut rapidement baisser la résolution de notre image d'entrée et limiter le nombre de filtres.

```
  input_shape=(120,160,3)))
model.add(Dropout(0.2))

model.add(Conv2D(4, kernel_size=(5,5), strides=2, activation="relu"))
model.add(Dropout(0.2))

model.add(Conv2D(8, kernel_size=(5,5), strides=2, activation="relu"))
model.add(Dropout(0.2))

model.add(Conv2D(16, kernel_size=(5,5), strides=2, activation="relu"))
model.add(Dropout(0.2))

model.add(Flatten())

model.add(Dense(32, use_bias=False, activation="relu"))
model.add(Dense(16, use_bias=False, activation="relu"))

model.add(Dense(5, use_bias=False, activation="softmax"))
```

Pour rappel, la convolution agit de cette façon sur une image : 4
Source : https://i.stack.imgur.com/vxEa3.jpg

Elle effectue une multiplication entre le kernel (le filtre) et l'image. Dans notre modèle, nous utilisons la convolution avec des filtres de 5 par 5 et un pas de déplacement de 2, c'est ce déplacement qui réduit la résolution.
Pour éviter de perdre trop d'informations lors de la baisse de résolution, le nombre de filtres de convolution augmente, il passe de 2 à 4 puis 8 et enfin 16.
Entre chaque convolution, on va venir ajouter du « Dropout ». Cette couche va simuler la disparition de quelques poids entre chaque couche, elle a pour effet d'éviter « l'overfit » (le sur-apprentissage). Elle va donc homogénéiser les performances du modèle sur des données de test et rendre ainsi la conduite plus fluide.



4

Le sur-apprentissage, c'est comme pour un humain l'apprentissage par cœur sans vraiment comprendre ce que l'on apprend. Et cela ne permet pas en général de s'adapter à des changements de situations.
Une fois la résolution de l'image assez basse, pour passer à la partie neuronale, on « aplatit » l'image avec la couche « flatten() ». Dans notre cas, l'image de forme (1, 4, 7, 16) va être transformée en image « plate » de forme (1, 448).
Cette nouvelle forme est compatible avec l'utilisation de neurones dit « Denses ». Chaque neurone reçoit en entrée tous les poids des neurones précédents, d'où la notion de densité.
On vient donc ajouter deux couches de neurones Dense, une de 32 neurones et une autre de 16.
Vient s'ajouter enfin une dernière couche de Dense avec 5 neurones (5 directions) qui, elle, utilise l'activation « softmax ». L'activation softmax fait en sorte que la somme de chaque neurone dans cette couche soit égale à 1.
Pour ces couches de Dense, je désactive l'utilisation du « bias ». Le bias est une valeur qui va venir s'ajouter systématiquement à un neurone, les bias ne sont pas affectés par les neurones des couches précédentes. Avec peu de neurones, le bias peut influencer en grande partie la prédiction, ce qu'il faut éviter à tout prix ! C'est un peu comme conduire les yeux fermés ;-).
Note : pour le reste du CNN, j'ai choisi l'activation « relu » c'est une activation semi linéaire très répandue qui prend en compte que les valeurs strictement positives, si la valeur est négative alors l'activation renvoie 0.
Pour visualiser l'architecture de notre modèle on peut utiliser :

```
model.summary()
```

Qui nous renvoie :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 58, 78, 2)         150
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 27, 37, 4)         200
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 17, 8)         800
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 7, 16)          3200
_________________________________________________________________
flatten_1 (Flatten)          (None, 448)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                14336
_________________________________________________________________
dense_2 (Dense)              (None, 16)                512
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 80
=================================================================
Total params: 19,278
Trainable params: 19,278
Non-trainable params: 0
```

C'est un bon moyen de visualiser la résolution des images et le nombre de filtres de notre modèle.

Compilation et entraînement du modèle

Avant d'entraîner le modèle, il nous faut le compiler, c'est-à-dire préciser la fonction de « loss » (la fonction qui va calculer l'erreur par rapport au résultat attendu) et l'optimiseur (la fonction qui va choisir comment rectifier le modèle par rapport à l'erreur).

```
model.compile(loss="categorical_crossentropy",optimizer= "adam"
        ,metrics=['accuracy'])
```

Puisque nous faisons un « classifier » (classificateur) et que l'on a utilisé l'activation « softmax » pour notre dernière couche. Il nous faut donc utiliser la fonction de loss « categorical_crossentropy ».
Pour l'optimiseur, j'ai choisi « adam » c'est un optimiseur très répandu. C'est le meilleur optimiseur dans la plupart des cas.
Plus d'informations sont disponibles sur les optimiseurs et algorithmes pour calculer la loss sur : https://keras.io/optimizers/ et https://keras.io/losses/ .
On peut enfin entraîner notre modèle !

```
model.fit(X_train, Y_train, batch_size=16, epochs=20
     ,validation_data=(X_test,Y_test))
```

Pour entraîner, on utilise « model.fit », on précise nos données d'entraînement (X_train, Y_train) et nos données de test (X_test, Y_test). On précise le batch size (c'est le nombre d'images qui sont traitées en une fois) et l'on choisit le nombre d'époques (le nombre de fois que le modèle va s'entraîner entièrement sur les sets de données d'entraînement).
Je préfère avoir un batch size assez petit pour éviter le sur-apprentissage. Plus il sera proche de 1, moins le modèle va être précis, plus il sera grand, plus il sera précis mais risque de sur-apprendre. Il faut donc trouver un compromis entre les deux. De plus, le batch size influe sur le temps que va prendre votre modèle à converger. Par exemple, en une époque, avec un batch size de 4, le modèle va être corrigé 4 fois plus qu'avec un batch size de 16.
Le nombre d'époques dépend du nombre de données que vous avez et le temps de convergence de votre modèle, il faut souvent faire 2 à 3 entraînements avant de trouver le bon nombre d'époques à utiliser. Attention à ne pas trop en mettre, encore une fois pour éviter le sur-apprentissage.
A présent, nous pouvons lancer le programme et admirer !
Si tout se passe comme prévu, vous devriez avoir quelque chose comme ça : 5

Le temps d'entraînement varie en fonction du CPU/GPU utilisé, pour ma part, j'utilise une gtx 970. C'est une carte graphique sortie fin 2014, elle fait tourner encore aujourd'hui pratiquement tous les jeux en haute qualité en 1080p voir 1440p. C'était un très bon compromis entre performance et prix à l'époque.

5

```
Train on 14461 samples, validate on 1607 samples
Epoch 1/20
2019-05-31 11:44:33.298733: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
2019-05-31 11:44:33.483481: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1356] Found device 0 with properties:
name: GeForce GTX 970 major: 5 minor: 2 memoryClockRate(GHz): 1.329
pciBusID: 0000:01:00.0
totalMemory: 4.00GiB freeMemory: 3.30GiB
2019-05-31 11:44:33.489167: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1435] Adding visible gpu devices: 0
2019-05-31 11:44:34.203557: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:923] Device interconnect StreamExecutor with strength 1 edge matrix:
2019-05-31 11:44:34.206936: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:929]      0
2019-05-31 11:44:34.209449: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:942] 0:   N
2019-05-31 11:44:34.212757: I T:\src\github\tensorflow\tensorflow\core\common_runtime\gpu\gpu_device.cc:1053] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 3031 MB memory) -> physical GPU (device: 0, name:
GeForce GTX 970, pci bus id: 0000:01:00.0, compute capability: 5.2)
 5040/14461 [=========>....................] - ETA: 23s - loss: 1.4951 - acc: 0.3853
```

Une fois l'entraînement terminé qui m'a pris environ 4 minutes, on peut admirer le résultat : 6
D'un côté nous avons les performances sur le set d'entraînement (loss et acc) et de l'autre nous avons les performances sur le set de test (val_loss et val_acc).
Notre modèle est moins précis sur ses données d'entraînement que sur celles de test, mais comment est-ce possible ? Lorsque notre modèle s'entraîne, il prend en compte le dropout alors que pendant l'évaluation sur les données de test, le dropout est enlevé.
Dans l'idéal, il faudrait que nos deux « acc » soient les plus proches possible et que l'acc soit légèrement plus élevé que la « val_acc », si vous êtes dans le même cas que moi, essayez de baisser le «Dropout » 7
Voilà qui est mieux !
Vous pouvez maintenant enregistrer votre modèle en mettant à la fin de votre programme :

```
model.save('model.h5')
```

Le code pour l'entrainement est disponible ici : https://github.com/Maximellerbach/AutonomousCar/blob/master/AutonomousCar/create_model/convolution/article.py
Une autre version plus complexe sous forme de classe mis à jour régulièrement : https://github.com/Maximellerbach/AutonomousCar/blob/master/AutonomousCar/create_model/convolution/train.py

Test du modèle

Une fois notre modèle créé et entraîné, il nous faut pouvoir le tester ! Pour diriger notre voiture, l'idée serait de réaliser les actions suivantes en boucle :
- Acquisition d'une image par la webcam ;
- Préparation de l'image ;
- Prédiction du label à partir de l'image ;
- Appliquer la direction en fonction du label ;
- Enregistrer l'image et le label (optionnel).

Voici les modules qui nous serviront dans ce programme :

```
import os
import time

import cv2
import numpy as np
from keras.models import load_model

import SerialCommand
```

Le module « SerialCommand » sert à transmettre les informations de direction et de vitesse à l'Arduino par le port série du Raspberry Pi. Code complet du module ici : https://github.com/Maximellerbach/AutonomousCar/blob/master/AutonomousCar/test_model/convolution/SerialCommand.py

```
dico = [11,9,7,5,3]
speed = 70

model = load_model("model.h5")

cap = cv2.VideoCapture(0)

ser = SerialCommand.control("/dev/ttyS0")
ser.ChangeMotorA(2)
```

On vient initialiser notre dictionnaire de direction pour transformer le label prédit (0,1,2,3,4) en direction allant de 0 (droite) à 14 (gauche) et on définit la vitesse allant de 0 à 255. On charge le modèle précédemment entraîné et on initialise l'utilisation de notre webcam.
On vient aussi préciser le port série utilisé sur lequel est connecté l'Arduino puis on met les moteurs sur le mode « avancer ».
Jetons un œil à notre boucle !

```
while(True):
  try:
    _, cam= cap.read()

    #PREPARE IMAGE
    img = cv2.resize(cam,(160, 120))
    img_pred = np.expand_dims(img, axis= 0)

    #PREDICT
    predicted = np.argmax(model.predict(img_pred))
    direction = dico[predicted]

    ser.ChangePWM(speed)
    ser.ChangeDirection(direction)


  except:
    print("error in program's loop")
ser.ChangePWM(0)
cap.release()
```

Pour chaque itération, les étapes suivantes sont exécutées :
- Récupération de l'image « cam » ;
- Redimension de « cam » ;
- Changement de forme de notre image, on passe de (120,160,3) à (1,120,160,3). Cette étape est nécessaire car 4 dimensions sont requises pour la prédiction ;
- Prédiction des labels, ici, on va venir prendre le label qui représente la plus grande probabilité avec np.argmax().
- Récupération de la direction à partir du label.
- Application de la vitesse, la vitesse étant constante dans ce pro-

6
```
Epoch 20/20
14461/14461 [==============================] - 11s 756us/step - loss: 0.5360 - acc: 0.7809 - val_loss: 0.4802 - val_acc: 0.7990
```

7
```
Epoch 20/20
14461/14461 [==============================] - 10s 703us/step - loss: 0.4463 - acc: 0.8183 - val_loss: 0.4719 - val_acc: 0.8164
```

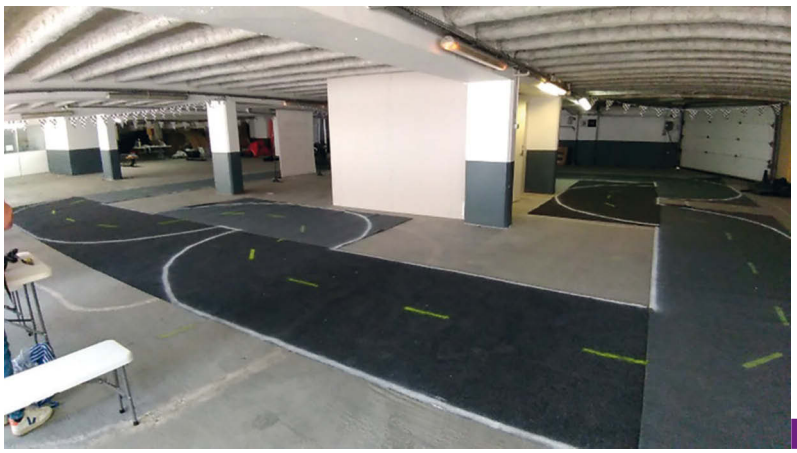
gramme aurait pu être appliquée au début du programme en dehors de la boucle, mais en la mettant dans la boucle, je m'assure que la voiture démarre uniquement si la caméra et la prédiction marchent.

- Changement de la direction des roues à partir de la direction prédite.

Le code est mis dans un « try » pour qu'au cas où il y aurait une erreur, le programme ne s'arrête pas et m'alerte juste que quelque chose ne va pas.

Enfin, lorsque je mets un terme au programme j'arrête la voiture en mettant la vitesse à 0 et j'éteins la webcam.

Le programme est disponible ici : https://github.com/Maximellerbach/AutonomousCar/blob/master/AutonomousCar/test_model/convolution/drive_article.py



8

Déroulement des courses

Au moment où je vous écris, je suis actuellement 2ème au classement général. Lors de la dernière course seulement 5 équipes étaient présentes et les 3 équipes qui sont normalement sur le podium (Axionable, l'école 42 et Epita) étaient à la Vivatech, laissant le podium libre !

A chaque course, c'est un réel défi de faire un modèle en 3 heures car le circuit change d'une course à l'autre et il est donc indispensable de récupérer des images et de les labelliser.

C'est une étape cruciale, par exemple : lors de la dernière course, j'ai oublié de vérifier si ma manette marchait, bien évidemment elle ne marchait pas ! Il m'a donc fallu labelliser le plus d'images possible en un temps imparti… Résultant en une labellisation partielle du circuit.

La voiture c'est très bien débrouillée sur la partie labellisée en ne sortant aucune fois, par contre, sur l'autre partie c'était un désastre ! 8

Conclusion

Ce projet est très complet, il faut aussi bien travailler la programmation que l'électricité, l'électronique, la conception et le bricolage en général.

Côté programmation, il n'y a pas seulement 3-4 programmes qui font la labellisation, l'entraînement et le test (je vous ai montré ici les plus importants) ; il y a une panoplie de petits programmes qui me servent à visualiser des sets de données, visualiser les endroits les plus importants selon le modèle dans l'image. Récemment j'ai réalisé un programme qui analyse des images et les trie en fonction de leurs caractéristiques, ainsi, la labellisation « manuelle » est beaucoup plus simple et rapide. Au lieu de labelliser 7000 images, je labellise 200 dossiers comportant des images similaires.

Ce projet m'a appris à programmer avec des ressources limitées : au départ, je devais tourner au maximum à 10 images traitées par seconde, maintenant j'atteinds le plafond des 30 images par secondes. Le hardware ne fait pas tout !

C'est le projet qui me pousse le plus à faire des recherches de mon côté, me faire découvrir des nouveaux domaines de l'intelligence artificielle.

Côté hardware, ce projet m'a fait découvrir plus en profondeur les technologies des cartes embarquées, leurs avantages, leurs défauts.

J'ai aimé notamment passer du temps à confectionner la voiture : au début découper du carton pour le prototypage, ensuite rendre le tout solide avec de l'acrylique, des vis et un peu de soudure pour la partie Arduino !

Je trouve qu'actuellement avec toutes les librairies de machine learning telles que Tensorflow, Keras, Scikit-learn, le machine learning est accessible à tous !

Si ce projet vous a intéressé, je vous invite à aller faire un tour sur mon Github : https://github.com/Maximellerbach

Jérémie Patonnier
Développeur Advocate et Architect front à **Zenika**

Accessibilité web : WCAG démystifié

Quand on veut mettre en œuvre une démarche d'accessibilité web, il y a pas mal de choses à faire et à savoir. On peut être tenté de partir bille en tête sur des questions d'outillage et de processus, mais on va souvent retrouver des mentions à un document qui revient encore et encore, une norme qui semble représenter l'alpha et l'oméga de la question : WCAG (Web Content Accessibility Guidelines – Principes d'Accessibilité des Contenus Web).

niveau 100

Cette spécification est tellement centrale qu'elle vaut la peine de se pencher un peu dessus, histoire de s'armer correctement avant de passer à la mise en œuvre. Aujourd'hui on va donc voir ce qu'est concrètement WCAG : qu'est-ce qu'il y a dedans ? Quels en sont les avantages ? Et finalement, quelles en sont les limites ?

Pour qui, pour quoi ?

Créée dans le cadre de la *Web Accessibility Initiative* (initiative pour l'Accessibilité Web) du W3C, WCAG est clairement une des spécifications d'accessibilité web les plus fondamentales. D'une part, elle s'applique à tout contenu créé avec les technologies web (HTML, CSS, JavaScript, SVG, etc.) que ce soit pour un site web, une application en une page (SPA: *Single Page Application*) ou une application native web (créée avec Electron par exemple). D'autre part, elle est le fondement d'à peu près toutes les réglementations officielles, de la section 508 américaine au RGAA (Référentiel Général d'Accessibilité des Administrations) français.

Contrairement à beaucoup de spécifications du W3C, WCAG n'est pas une spécification technique. Elle est une spécification fonctionnelle qui fournit un ensemble de principes (*guidelines*) pour rendre les contenus web plus accessibles sans pour autant être une solution ultime. Son préambule est totalement transparent à cet égard :

Web Content Accessibility Guidelines (WCAG) 2.1 covers a wide range of recommendations for making Web content more accessible. Following these guidelines will make content more accessible to a wider range of people with disabilities, including accommodations for blindness and low vision, deafness and hearing loss, limited movement, speech disabilities, photosensitivity, and combinations of these, and some accommodation for learning disabilities and cognitive limitations; but will not address every user need for people with these disabilities. These guidelines address accessibility of web content on desktops, laptops, tablets, and mobile devices. Following these guidelines will also often make Web content more usable to users in general.

Web Content Accessibility Guidelines (WCAG) 2.1 couvre un large spectre de recommandations pour rendre les contenus web plus accessibles. Suivre ces principes rendra les contenus plus accessibles à un ensemble plus important de personnes en situation de handicap, ce qui inclut des aménagements pour la cécité ou la déficience visuelle, la surdité et la perte d'audition, les contraintes motrices, les difficultés d'élocution, la photosensibilité et la combinaison de tous ces états ; elle inclut également des dispositions pour les problèmes d'apprentissage ou de cognition. Cependant, elle ne répond pas à l'exhaustivité des besoins des personnes en situation de handicap. Ces principes couvrent l'accessibilité des contenus web sur les ordinateurs de bureau, les ordinateurs portables, les tablettes et les appareils mobiles. Suivre ces principes rendra généralement les contenus web plus faciles à utiliser pour tous les utilisateurs.

Un des grands enjeux de l'accessibilité c'est la capacité à pouvoir anticiper les situations de handicap pour en réduire l'impact. Le truc c'est que quand on n'est pas soi-même confronté à ces questions, il est difficile d'anticiper ces situations. C'est en partie pour résoudre ce problème que WCAG existe. Les principes qui y sont répertoriés sont là pour répondre à nombre de situations qu'on aurait du mal à imaginer. Cependant, comme toute norme fonctionnelle, sa mise en œuvre est affaire de contexte et d'appréciation. Et justement WCAG est bien conçue pour nous aider à évaluer la pertinence des principes à suivre pour nos projets.

Un temple pour l'accessibilité

WCAG repose sur trois grands piliers :

- Les principes à suivre (souvent la partie émergée de l'iceberg dont on a plus ou moins entendu parler) ;
- Les critères de succès des principes et le niveau de conformité associé ;
- Les conseils de mise en œuvre.

Voyons de quoi il s'agit exactement.

Les principes : la nourriture de l'âme

Les fondements de WCAG sont ses principes de mise en œuvre qui vont être regroupés au sein de quatre grandes lignes directrices. Comme on le disait, il est difficile d'imaginer toutes les situations de handicap possibles. Pour cette raison, WCAG adopte un modèle générique pour ses principes plutôt que d'essayer de répondre à des questions spécifiques. Voici une version vulgarisée de ces lignes directrices et de leurs principes :

La perception

Tout contenu web doit être perceptible (*Perceivable*). En clair, quelles que soient les conditions, un utilisateur doit pouvoir percevoir le contenu – le voir, l'entendre ou le toucher. Prenez le temps d'y réfléchir, c'est beaucoup moins évident que ça en a l'air… Indice : tout le monde n'a pas les mêmes capacités de perception. Les principes associés étant :

- Principe 1.1 : tout contenu non textuel doit avoir une alternative textuelle.

- Principe 1.2 : tout média temporel (audio, vidéo, animation, etc.) doit avoir une alternative (en général textuelle, mais il y a d'autres cas).
- Principe 1.3 : tout contenu doit pouvoir être représenté de différentes manières (mise en page alternative par exemple) sans perte d'informations.
- Principe 1.4 : tout type de contenu doit être facile à distinguer d'un autre (couleurs, formes, etc.).

L'usage

Tout contenu web doit être utilisable (*Operable*). Ça veut dire que, encore une fois, quelles que soient les conditions, l'interface utilisateur doit proposer un moyen d'interaction à l'utilisateur – là aussi c'est délicat, car tout le monde n'a pas les mêmes capacités d'interaction. Par exemple, quiconque a déjà utilisé une interface tactile pour manipuler un contenu optimisé pour l'usage d'une souris devrait vite comprendre de quoi on parle. Les principes associés étant :

- Principe 2.1 : toutes les fonctionnalités doivent être utilisables avec un clavier.
- Principe 2.2 : l'utilisateur doit avoir assez de temps pour lire et utiliser le contenu.
- Principe 2.3 : ne concevez pas un contenu dont on sait qu'il peut induire des convulsions ou toute autre réaction physique.
- Principe 2.4 : aidez les utilisateurs à naviguer, à trouver les contenus et à déterminer où ils sont.
- Principe 2.5 : facilitez l'usage des fonctionnalités via diverses méthodes en plus du clavier.

La compréhension

Tout contenu web doit être compréhensible (*Understandable*). Que ce soient les contenus eux-mêmes ou la capacité à y accéder, l'utilisateur doit pouvoir tout comprendre. Cette fois, le piège se situe dans une remarque anodine : « Mais enfin ! C'est évident. »… Mmh, vraiment ? Les principes associés étant :

- Principe 3.1 : assurez-vous que les contenus textes sont lisibles et compréhensibles.
- Principe 3.2 : assurez-vous que les contenus apparaissent et agissent de manière prédictible.
- Principe 3.3 : aidez les utilisateurs à éviter et corriger les erreurs de saisie.

La robustesse

Tout contenu web doit être robuste (*Robust*). Là, il s'agit de s'assurer que le contenu puisse être affiché de manière fiable sur tout appareil, ce qui inclut les appareils d'assistance, actuels aussi bien que futurs. Le principe associé étant :

- Principe 4.1 : maximisez la compatibilité avec les agents-utilisateurs, y compris les technologies d'assistance.

Les critères de succès : l'épreuve du feu

La difficulté est moins dans la compréhension de ces principes somme toute assez génériques et de bon sens (même s'il est vrai que ça demande quand même pas mal de jus de cerveau pour être sûr qu'on en mesure bien toute la portée) que dans l'évaluation de leurs différents critères de succès. En d'autres termes : comment sait-on que ces principes sont correctement suivis dans notre contexte ?

Ces critères de succès commencent à être assez précis et il serait fastidieux de les passer en revue ici. D'autant plus qu'il existe moult ressources qui en font une véritable transcription opérationnelle, à commencer par le référentiel technique du RGAA pour le lectorat français. On ne va donc pas les détailler ici, mais on va quand même voir comment ces critères sont utilisés pour tester l'accessibilité d'une page web.

Il y a au total 78 critères de succès dans WCAG 2.1 pour évaluer l'accessibilité d'un contenu web. Alors évidemment, tous ne s'appliquent pas en fonction du contexte. Par exemple, s'il n'y a pas de champ de formulaire, aucun des critères liés au principe 3.3 (prévention et correction des erreurs de saisie) ne s'appliquera. C'est d'ailleurs souvent la première chose à faire dans une évaluation d'accessibilité : évaluer quels sont les critères de succès pertinents à tester.En plus de la pertinence individuelle de chaque critère d'évaluation, ceux-ci sont classifiés en trois niveaux de conformité : A, AA (double A), et AAA (triple A). Il n'y a pas de définition formelle de ce que représente chacun des niveaux de conformité. Cependant on peut considérer les choses comme suit :

- **Niveau de conformité A**
  Ce niveau de conformité regroupe tous les critères de succès qui sont généralement assez simples à mettre en œuvre ou avec un impact significatif pour tous les utilisateurs. Par exemple, le critère de succès 3.1.1 : la langue d'une page web peut être déterminée programmatiquement.
- **Niveau de conformité AA**
  Il s'agit des critères qui sont plus exigeants en termes de mise en oeuvre ou qui ont un impact plus spécifique pour les utilisateurs. Par exemple, le critère de succès 1.4.4 : le texte peut être redimensionné jusqu'à 200% de sa taille sans perte de contenu ni de fonctionnalité.
- **Niveau de conformité AAA**
  Ici on regroupe les critères de conformité qui sont soit très spécifiques pour une population donnée, soit qui ont un impact de mise en œuvre majeur (une expertise spécifique sera requise), soit avec des conséquences potentiellement contre-productives pour certains utilisateurs. Par exemple, le critère de succès 1.2.6 : tous les contenus audio/vidéo sont accompagnés d'une transcription en langue des signes.

Ce qu'il faut bien noter c'est que ces niveaux de conformités sont cumulatifs. C'est-à-dire que même si tous les critères AAA sont conformes vous ne pouvez pas déclarer un contenu conforme AAA si un seul critère A applicable n'est pas conforme. On notera également que la plupart des législations existantes demandent un niveau d'accessibilité des contenus qui soit conforme AA.

À ce stade nous allons atteindre le point où les choses se compliquent vraiment. Une fois que l'on a identifié quels sont les critères de succès applicables à notre cas et le niveau de conformité que l'on veut atteindre, il reste à faire l'évaluation et il faut donc se poser la question de comment tester la conformité à proprement parler.

Les conseils de mise en œuvre : la voie de l'initié

Sur le chemin qui mène à la sagesse, WCAG nous donne des armes qui prennent la forme de conseils de mise en œuvre. En effet, pour un même critère de succès, il existe plusieurs solutions techniques ou fonctionnelles pour y répondre. Là encore, tout est affaire de contexte.

Ces conseils de mise en œuvre prennent donc deux formes. D'une part un guide de compréhension avec des fiches explicatives qui détaillent les objectifs de chaque critère de succès et donnent des exemples concrets. D'autre part un ensemble de techniques (et d'éventuelles conditions d'échec) applicables à chaque critère de succès.

Tout ça donne un jeu de poupées russes de documents assez touffus, mais finalement plus compréhensibles que ce qu'on pourrait croire. Reprenons l'exemple du critère de succès 3.1.1 : la langue d'une page web peut être déterminée programmatiquement.
Déjà, la notion de "détermination programmatique" n'est pas ce qu'il y a de plus évident à comprendre. Pour clarifier ça, un petit tour sur la fiche de compréhension du critère va nous éclairer. En gros, l'objectif c'est que les outils qui vont exploiter le contenu soient capables d'en déterminer la langue le plus facilement possible. Par exemple, pour qu'un lecteur d'écran puisse choisir le bon algorithme de prononciation du texte.
OK, et comment fait-on ça ? C'est là que les techniques de mise en œuvre vont nous aider. Pour ce critère c'est assez facile, il y a 4 techniques incontournables (*sufficient*) pour valider le critère et 2 techniques additionnelles (*advisory*) qui apportent du plus sans pour autant permettre de valider le critère. Parmi les quatre techniques incontournables, toutes ne sont pas pertinentes selon le contexte (eh oui, encore le contexte). En l'occurrence il y a une technique spécifique pour les documents HTML, une pour les documents Flash et deux pour les documents PDF. Ainsi, si on travaille sur un site web, pour valider le critère de succès 3.1.1 il suffit de renseigner l'attribut lang de la balise <html> sur toutes les pages. Et c'est tout.

Alors c'est sûr que dit comme ça, ça fait un peu pétard mouillé ! Tout ça juste pour un attribut HTML ! Certes, la solution technique est finalement assez triviale, mais l'enjeu, c'est moins la solution que la compréhension des besoins et des impacts et de ce point de vue, WCAG est irréprochable. En outre, garantir que cet attribut est présent sur toutes les pages d'un site et avec une valeur pertinente peut être finalement assez complexe à mettre en œuvre s'il s'agit d'un site multilingue. Pour peu que le contexte s'y prête, ce qu'on imagine facile peut devenir difficile à réaliser.

Kamehameha

WCAG n'est pas compliqué à appréhender / à appliquer, c'est plutôt la ramification infinie des possibilités de mise en œuvre qui peut vite faire tourner la tête du développeur (pour ne pas dire : rendre complètement dingue).
L'enjeu de la mise en œuvre de cette spécification va finalement tourner autour des trois problématiques suivantes :

- La compréhension des principes et des critères de succès de la norme.
- L'appréciation des contraintes à suivre en fonction du contexte de chaque projet.
- La compétence des créateurs de contenu pour mettre en œuvre les solutions techniques adaptées.

Finalement, pour devenir un super saiyan de l'accessibilité il n'y a pas vraiment de secret : il faut se former en continu, interroger régulièrement ses pratiques de travail et prendre le temps de relire les passages de WCAG qui correspondent à votre tâche du moment, en particulier les fiches de compréhension et les critères de succès.

Conclusion

Cet article a vocation à vous familiariser avec WCAG qui sera toujours la référence ultime dans toute démarche d'accessibilité web. Ceci dit, sa complexité et sa flexibilité n'en font pas une spécification véritablement opérationnelle. Pour cette raison il existe des outils plus pratiques à utiliser au quotidien. Pour n'en citer que quelques-uns :

- Le référentiel technique du RGAA (sans doute la meilleure référence pour les Français en particulier et les francophones en général).
- Les checklists accessibilité premier pas et second pas de Opquast.
- Les ressources du site WebAIM (en particulier leur checklist WCAG 2).
- Et bien d'autres, du plus simple au plus complet, listés sur le site du W3C.

Mais ne nous y trompons pas, pour pratiques qu'ils soient, **ces outils ne doivent pas être utilisés aveuglément**. Si WCAG est si flexible et sujette à interprétation – et de son propre aveu, incomplète – ce n'est pas par hasard ou par erreur. L'accessibilité est un sujet complexe en ce qu'il requiert d'être confronté en permanence à la réalité d'un contexte donné.
Les meilleurs contenus accessibles sont ceux créés par des personnes qui maîtrisent à la fois les enjeux contextuels et les techniques de mise en œuvre. WCAG répond d'abord fondamentalement à la question des enjeux en forçant les créateurs de contenu web à se poser des questions qu'ils n'imagineraient pas par ailleurs. •

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

Docker et Containers : introduction et concepts

Les conteneurs sont une (autre) façon de faire tourner un système d'exploitation et une application de manière isolée et virtualisée. À chaque démarrage, on part sur un nouvel environnement tout neuf. Il est possible de faire tourner plusieurs containers sur une même machine (host). En termes de montée en charge, une solution à base de containers a des perspectives intéressantes. Les containers offrent les bénéfices de l'isolation, de la portabilité, de l'agilité et une isolation entre les Dev et les Ops.

niveau 100

Les containers et Docker

Docker est un projet Open-Source pour faire tourner des containers Linux ou Windows sur le Cloud ou sur une machine standard (on-promise). Sous Windows, les développeurs peuvent faire tourner des images Linux ou Windows.

Docker et VM

Docker c'est un peu comme une VM mais en plus souple. Dans une VM, il y a un OS, et N applications. Sous Docker, on a une application. Et il est possible de faire tourner plusieurs containers Docker sur la même machine. 1

L'avantage d'utiliser Docker c'est que le syndrome du « ça marche sur ma machine ! » ne tient plus. Si ça tourne sur Docker, ça tourne tout le temps.

Petit lexique

Une image de Container : c'est un package avec toutes les dépendances pour créer le container. Une image contient les Frameworks, les DLL, les fichiers de configurations et tout ce qui est nécessaire au runtime du container. Une fois créée, une image ne peut plus être modifiée.

Un fichier Dockerfile : un fichier texte qui contient les instructions pour construire une image Docker. Ce fichier est comme un fichier CMD avec des commandes de copy, d'exécution pour définir l'environnement.

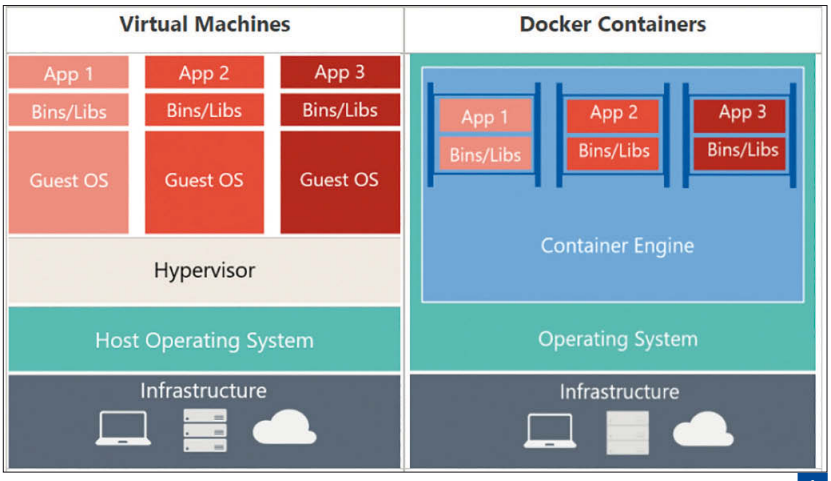
Build : la commande docker build permet de construire une image Docker à partir d'un Dockerfile.

Container : c'est une instance d'une image Docker. Il exécute un service, une application et les modules de l'OS.

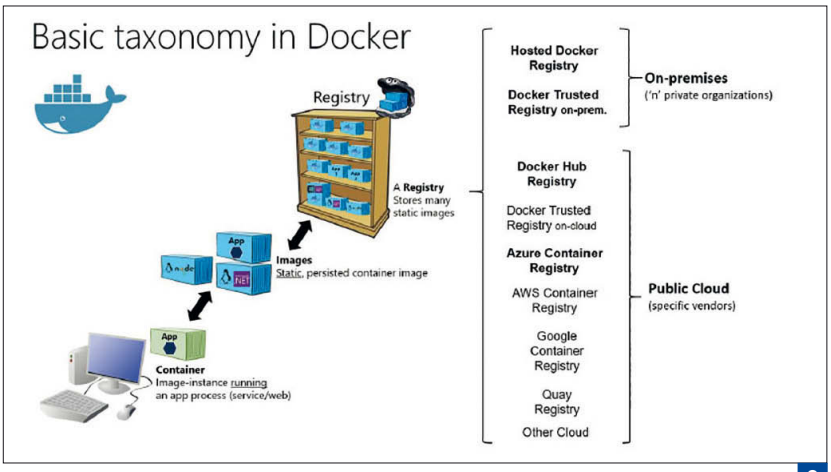
Volumes : C'est un espace de fichiers que le container peut utiliser.

Tag : l'identifiant d'une image docker

Repository : un endroit ou l'on peut stocker les images Docker.



1



2

Registry : Un service qui permet d'accéder aux repositories. Il existe des registries sur le Cloud.

Docker containers, images, registries

Une image Docker est une représentation statique d'une application ou d'un service avec sa configuration et ses dépendances. Pour faire tourner un service, l'image de l'application est instanciée pour créer un container. 2

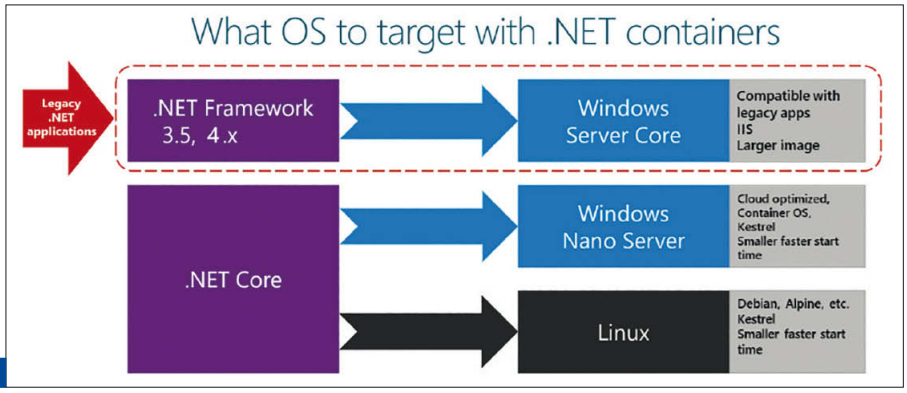
.NET Core ou NET Framework

Pour utiliser les containers Windows, on peut utiliser soit l'un soit l'autre. Si on a des grosses dépendances à Windows, NET Framework est le choix par défaut. Par contre, pour faire tourner un container Linux, il faut utiliser NET Core. 3 4

Au maximum, essayez de dépendre de Windows Nano Server qui est léger. Léger veut dire que l'image est moins grosse en taille.

Architecture / App Type	Linux containers	Windows Containers
Microservices on containers	.NET Core	.NET Core
Monolithic app	.NET Core	.NET Framework .NET Core
Best-in-class performance and scalability	.NET Core	.NET Core
Windows Server legacy app ("brown-field") migration to containers	–	.NET Framework
New container-based development ("green-field")	.NET Core	.NET Core
ASP.NET Core	.NET Core	NET Core (recommended) .NET Framework
ASP.NET 4 (MVC 5, Web API 2, and Web Forms)	–	.NET Framework
SignalR services	.NET Core 2.1 or higher version	.NET Framework .NET Core 2.1 or higher version
WCF, WF, and other legacy frameworks	WCF in .NET Core (only the WCF client library)	.NET Framework WCF in .NET Core (only the WCF client library)
Consumption of Azure services	.NET Core (eventually all Azure services will provide client SDKs for .NET Core)	.NET Framework NET Core (eventually all Azure services will provide client SDKs for .NET Core)

3



4

Mise en œuvre

Voici à quoi ressemble un fichier Dockerfile :

```
FROM microsoft/iis:windowsservercore-ltsc2016
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
```

La première ligne indique l'image de l'OS qui va être utilisée. Ensuite, il s'agit de lancer des commandes comme COPY, etc. La commande EXPOSE permet d'ouvrir un port explicitement. Dans ce cas, 7001, car mon service LMDBService expose ce port comme end-point.

Les prochaines étapes

Docker requiert que l'image soit buildée puis instanciée. On récupère son adresse IP et on peut interagir avec.

Run en local sous Docker

Ensuite, il faut builder l'image Docker :

```
docker imagebuild –tag mydocker/myserverd:\dev\docker
```

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

```
docker run -d-p7001:7001-it mydocker/myserver
```

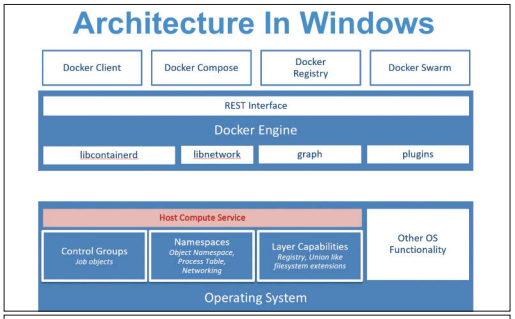
Pour pouvoir tester le container, il nous faut son adresse IP avec son ID en utilisant :

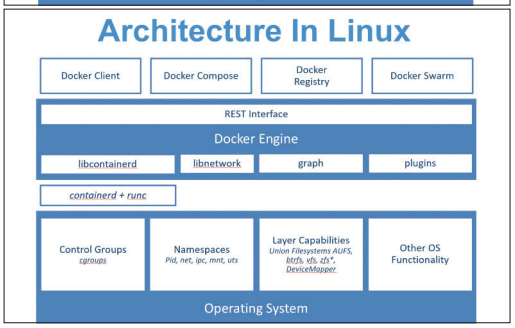
```
dockerps –a
```

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 renvoyée sera => 172.26.255.203





Il est maintenant possible de tester le container, car on connaît son adresse IP. Pour arrêter le container, la commande est très simple :

```
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.

Conclusion

Docker est un système souple et fiable. Faites le test, cela permet de faire tourner des environnements légers et des images diverses. •

Loïc Mathieu
Consultant à **Zenika Lille**

Zoom sur Quarkus

Quarkus est un nouveau framework Java, orienté développement de microservices, sorti début mars 2019. Il est développé par RedHat et est pensé pour le développement d'applications à déployer dans le cloud en supportant nativement Kubernetes (et OpenShift) et en permettant la création facilitée d'un package natif de votre application. Un package léger en termes de taille de livrable et en utilisation mémoire. L'utilisation mémoire a été particulièrement optimisée pour la mémoire non-heap (metaspace et autre) en réduisant le nombre de classes chargées au runtime et donc en optimisant le Resident Set Size (RSS) de votre application (la taille totale de la mémoire utilisée par le process de votre application sur votre système). À sa sortie, il a suscité pas mal d'étonnements, explications.

Quarkus : quoi, comment ?

Au développement, Quarkus est basé sur MicroProfile (via SmallRye : https://smallrye.io/), on développe alors des services standards Jax-RS qui seront servis par la couche HTTP de Undertow (le serveur HTTP de JBoss) et dont certaines fonctionnalités réactives et asynchrones seront apportées par Vert.X. Ce dernier permet aussi l'utilisation de client HTTP non bloquant et de messaging asynchrone.

Dans ces services, la donnée peut être retournée directement, ou via un CompletionStage<T> ou un Publisher<T> pour permettre un appel synchrone, asynchrone ou réactif au sein de la même API : **trois paradigmes, une seule implémentation, et ça c'est top** ! Plus d'info ici : https://quarkus.io/vision/continuum

Quarkus vient avec un plugin Maven (ou Gradle) qui va se charger de toute la tuyauterie nécessaire pour le moteur d'injection de dépendances et le démarrage de l'application (le bootstraping), il va faire ça à la compilation (et pas au runtime comme Spring), via génération de bytecode. Ce qui permet un lancement plus rapide de votre application, et une empreinte mémoire plus faible. Par contre, cela rend obligatoire la compilation via Maven de votre projet (et donc pas via votre IDE) ; pour rendre ça plus attractif pour les développeurs, le plugin de compilation de Quarkus peut se lancer en mode dev avec du livereload !

Pour le déploiement, on peut déployer Quarkus via un jar autoporté à packager avec ses librairies, ou via une application native. C'est le plugin Maven de Quarkus qui va se charger de générer pour nous une image native de notre application via GraalVM si on le désire. Et il fait ça très bien, sans aucune configuration de notre part. On peut après cela construire une image Docker de quelques dizaines de Mo qui démarrera en quelques dizaines de ms avec une empreinte mémoire très faible !

Voici une comparaison, en termes d'empreinte mémoire et de taille d'image, d'applications classiques cloud native avec et sans Quarkus (source : https://quarkus.io/) 1

On peut voir que Quarkus démarre plus rapidement et avec une empreinte mémoire plus faible que les solutions standard du marché même sans compilation native, avec compilation native l'empreinte mémoire et le temps de démarrage sont époustouflants !

Tout ça semble prometteur non ? Pour visualiser comment ça marche, voici un tutoriel pour vos premiers pas en Quarkus !

QUARKUS : TUTORIEL

niveau 100

Étape 1 : générer un projet depuis l'artifact maven

```
mvn io.quarkus:quarkus-maven-plugin:0.13.1:create \
    -DprojectGroupId=fr.loicmathieu.demo.quarkus \
    -DprojectArtifactId=quarkus-demo \
    -DclassName="fr.loicmathieu.demo.quarkus.rest.PersonRest" \
    -Dpath="/persons" \
    -Dextensions="resteasy-jsonb"
```

Pour cela, on peut suivre le tutoriel suivant https://quarkus.io/guides/getting-started-guide, j'ai juste ajouté l'extension qui permet la sérialisation en Json.

On peut remarquer les points suivants en regardant ce qui a été généré :

- Pas de classe Application ;
- Dépendances gérées via le BOM de Quarkus ;
- Seules dépendances (hors test) : quarkus-resteasy et quarkus-resteasy-jsonb ;
- Dépendances de tests (junit5 pour les TU et rest-assured pour les tests d'intégration/tests fonctionnels) ;
- Le plugin Maven de Quarkus ;
- Un profil maven spécifique pour le natif.

Étape 2 : démarrer le projet généré

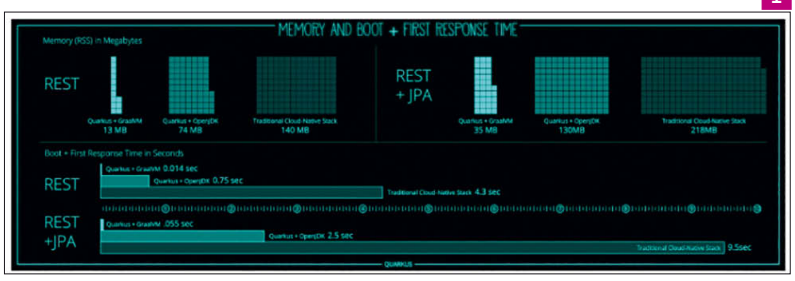
```
mvn compile quarkus:dev
```

Il n'y a pas de support par les IDE, et comme Quarkus doit faire des étapes spécifiques à la compilation pour fonctionner (via un plugin Maven), il doit être lancé en ligne de commande via Maven. Mais heureusement pour nous, il y a du livereload très rapide (moins d'une seconde) !

Voici les logs de démarrage :

```
Listening for transport dt_socket at address: 5005
[io.qua.dep.QuarkusAugmentor] (main) Beginning quarkus augmentation
[io.qua.dep.QuarkusAugmentor] (main) Quarkus augmentation completed in 359ms
[io.quarkus] (main) Quarkus 0.13.1 started in 0.648s. Listening on: http://[::]:8080
[io.quarkus] (main) Installed features: [cdi, resteasy, resteasy-jsonb]
```

1



On peut remarquer ici :

- Par défaut, le port de debug est ouvert (5005) ;
- Quarkus Augmentation : la magie !!! On en reparlera… ;
- Démarrage en 650 ms ;
- Écoute sur le port 8080 ;
- Les features chargées : cdi (pour l'injection de dépendance) et resteasy (pour les WS JAX-RS) ;
- Un appel sur http://localhost:8080/persons nous retournera "hello".

Étape 3 : coder…

Un petit CRUD sur un objet Person avec un service et un repository… Pour cela, il nous faut ajouter le jar de quarkus-orm-hibernate, et en fonction de la BDD utilisée, le jar correspondant (ici quarkus-jdbc-postgres). Pendant le développement, le hot reload se fait à chaque appel (et pas à chaque sauvegarde d'un fichier, ce qui est bien) en moins de 500 ms la première fois, chiffre qui tombe à 200ms au deuxième rechargement !

Pour développer ça, on peut suivre le tutoriel qui est ici : https://quarkus.io/guides/hibernate-orm-guide

Voici les principales étapes :

1. Créer un objet Person : source complèt via lien en fin d'article

```
@Entity
public class Person implements Serializable {
    private Long id;
    private String firstName;
    private String lastName;

     @Id
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

     // autres getters et setters
}
```

2. Créer un repository JPA

```
@ApplicationScoped
public class PersonRepository {
   @Inject
    private EntityManager em;

     public void set(Person person){
        em.persist(person);
    }

     public Person get(Long id){
        return em.find(Person.class, id);
    }

     public List&lt;Person&gt; list(){
        return em.createQuery("from Person").getResultList();
    }

     public void delete(Long id){
        em.remove(em.find(Person.class, id));
    }
}
```

3. Créer un service (interface et implémentation)

```
public interface PersonService {
    public void set(Person person);
    public Person get(Long id);
    public List&lt;Person&gt; list();
    public void delete(Long id);
}

@ApplicationScoped
public class PersonServiceImpl implements PersonService {
    @Inject
    private PersonRepository personRepository;

    @Transactional
    public void set(Person person) {
        personRepository.set(person);
    }

    public List&lt;Person&gt; list() {
        return personRepository.list();
    }

    public Person get(Long id) {
        return personRepository.get(id);
    }

    @Transactional
    public void delete(Long id) {
        personRepository.delete(id);
    }
}
```

Ici on peut remarquer un support des transactions très facile à utiliser via une simple annotation.

J'ai choisi ici de faire un repository JPA "à la main", mais Quarkus vient avec Panache qui permet de faire de l'hibernate *avec panache*. Panache va permettre de faire des repository très facilement (via une interface unique proche de ce que fait Spring Data) ou des Entity améliorés à la *Active Record* : https://quarkus.io/guides/hibernate-orm-panache-guide

4. Le service REST ensuite est assez simple

```
@Path("/persons")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class PersonRest {
    @Inject
    private PersonService personService;

    @GET
    public List&lt;Person&gt; list() {
        return personService.list();
    }

    @GET
    public Person get(@PathParam("id") Long id) {
        return personService.get(id);
    }

    @POST
    public void create(Person person) {
        personService.set(person);
    }

    @PUT
    @Path("/{id}")
    public void update(@PathParam("id") Long id, Person person) {
        if(! id.equals(person.getId())){
            throw new BadRequestException("id in path and in body must be the same");
        }
        personService.set(person);
    }

    @DELETE
    @Path("/{id}")
    public void delete(@PathParam("id") Long id) {
        personService.delete(id);
    }
}
```

Et voici notre CRUD codé ! Il reste juste un petit peu de configuration à réaliser, il faut ajouter dans l'application.properties les informations de configuration de la BDD. Comme tout est dans le repository github, je vous laisser aller voir si vous êtes curieux !

Maintenant que notre webservice est fini d'être développé, on veut le lancer ! Même si pendant le développement un simple mvn compile quarkus:dev était lancé, nous voulons maintenant packager notre application.

Étape 4 : packaging

Via maven :

```
mvn package
```

Lancement de l'application via son runner :

```
java -jar target/quarkus-demo-1.0-SNAPSHOT-runner.jar
```

Résultat : un jar de 10 Mo qui se lance en près d'une seconde, mais attention sa taille est trompeuse, il utilise un répertoire local de bibliothèques qui fait lui-même 24 Mo, on a donc une application d'une taille totale de 34 Mo.

Via docker

Il y a plusieurs manières de packager via Docker, allons directement à la plus pratique, la plus simple, la plus puissante : packager un natif via un multi-stage build !

Quoi ? natif ? multi-stage build ???

Natif : Quarkus vient avec tout ce qu'il faut pour générer un package natif (c'est à dire compilé en natif dans une version optimisée pour un OS / une architecture) en se basant sur l'outils *native-image* du projet Graal, et la SubstrateVM qui est une JVM permettant de faire tourner uniquement des applications natives et qui comprend le JIT Graal. Plus d'informations sur ce projet hautement intéressant : https://www.graalvm.org/.

Le problème c'est que pour compiler en natif, il faut que GraalVM soit installé en local, et il faut donner à l'outil *native-image* de GraalVM un fichier de configuration assez complexe. Le plugin

maven de Quarkus s'occupe pour nous d'appeler l'outil *native-image* avec les informations de configuration nécessaires, mais il nous faut quand même GraalVM installé en local sauf si… on réalise le build Maven dans Docker ce qui est possible avec un build multi stage. Un build docker multi-stage permet d'avoir plusieurs FROM dans son Dockerfile, le premier permettant de builder notre application et le second permettant de générer l'image finale. On imagine aisément l'intérêt ici : le premier stage permettra de builder via Maven et GraalVM notre package natif et le deuxième stage de packager notre application avec un FROM d'une image **distroless** (image minimaliste non basée sur une distribution Linux et qui ne contient qu'un ensemble très restreint d'outils).

C'est pour ces raisons que, même si l'archétype Maven nous a généré deux Dockerfiles (un pour un build standard et un autre pour un build natif), nécessitant à chaque fois de packager via Maven notre application en amont, je préfère utiliser le Dockerfile suivant :

```
## Stage 1 : build with maven builder image with native capabilities
FROM quay.io/quarkus/centos-quarkus-maven AS
COPY src /usr/src/app/src
COPY pom.xml /usr/src/app
# we will build a native image using the native maven profile
RUN mvn -f /usr/src/app/pom.xml -DskipTests -Pnative clean package

## Stage 2 : create the docker final image form a distroless image !
FROM cescoffier/native-base
# we copy from the previous build the artifacts
COPY --from=build /usr/src/app/target/*-runner /application
EXPOSE 8080
ENTRYPOINT ["./application", "-Dquarkus.http.host=0.0.0.0"]
```

Il suffit ensuite de builder l'image pour avoir une image minimaliste, contenant uniquement le binaire de notre application compilée et intégrant la SubstratVM.

Avant ça, il faut modifier le fichier. dockerignore, car celui généré par l'artefact Maven pose des soucis avec cette technique de build. Il faut remplacer son contenu par : src/main/docker/*

Ensuite le build se fait tout simplement via cette commande Docker :

```
docker build -t loicmathieu/quarkus-demo \
    -f src/main/docker/Dockerfile .
```

Ici, l'inconvénient est qu'à chaque build on va retélécharger via Maven tous les artefacts nécessaires, j'ai déjà quelques idées pour contourner en partie ça…

Après lancement de notre conteneur, **il démarre en… 47 ms** ! Comme notre BDD a été lancée de manière externe il faut surcharger la valeur de configuration de son URL, ceci est possible directement depuis la ligne de commande via -Dmy.config=value

Étape 5 : lancement

Voici un exemple de lancement :

```
#Lancement d'un conteneur Postgres
docker run --ulimit memlock=-1:-1 -it --rm=true \
  --memory-swappiness=0 --name postgres \
  -e POSTGRES_USER=sarah -e POSTGRES_PASSWORD=connor \
  -e POSTGRES_DB=skynet -p 5432:5432 postgres:10.5

#Lancement de notre conteneur avec surcharge de l'URL vers la BDD
docker run -ti -p 8080:8080 --link postgres \
  loicmathieu/quarkus-demo \
  -Dquarkus.datasource.url=jdbc:postgresql://postgres:5432/mydatabase
```

Et le résultat :

```
WARN  [org.hib.eng.jdb.spi.SqlExceptionHelper] (main) SQL Warning Code: 0, SQLState: 00000
WARN  [org.hib.eng.jdb.spi.SqlExceptionHelper] (main) table "person" does not exist, skipping
INFO  [io.quarkus] (main) <b>Quarkus 0.13.1 started in 0.047s</b>. Listening on: http://0.0.0.
INFO  [io.quarkus] (main) Installed features: [agroal, cdi, hibernate-orm, jdbc-postgresql, na
```

Quarkus : et la magie dans tout ça ?

Vert.X est magique, mais ça on en a déjà un peu parlé !

La principale magie de Quarkus est son système d'injection de dépendances et de bootstraping d'application (rappelez-vous, pas de classe Application) à la compilation. Il n'y a pas encore beaucoup de documentation sur le sujet, mais la page du guide de développement d'extension donne un petit aperçu de sa philosophie (en trois phases) :

- **Augmentation** : c'est fait via Maven via des *BuildStep* qui vont, en fonction de chaque extension (CDI, resteasy, etc.) générer le bytecode (le code compilé Java) nécessaire pour le boostraping de notre application. La phase d'augmentation va enregistrer des Record (suites de bytecode) qui seront ensuite exécutés dans les phases successives. Ces Record sont le résultat de l'exécution du code de boostraping de l'extension et pas un rejeu de celui-ci.
- **Static Init** : initialisation statique du framework et de ses extensions. Ce qui a été enregistré comme Record(STATIC_INIT) dans la phase d'augmentation. Par exemple, si un framework se configure via XML on va charger dans cette phase le résultat du parsing du XML et de la configuration du framework via celui-ci (le parsing ne sera pas réalisé au lancement de l'application, car réalisé en phase de compilation). Cette phase s'exécute via le plugin de compilation Maven.
- **Runtime Init** : l'initialisation de l'application au démarrage de celle-ci. Ce qui a été enregistré en phase d'augmentation en tant que Record(RUNTIME_INIT) s'exécutera ici.

C'est la responsabilité des extensions de permettre un maximum d'initialisations dans la phase de **Static Init** pour permettre aux applications de s'exécuter le plus rapidement possible.

En mode dev, toutes ces phases se font au démarrage pour permettre le livereload.

Attention : ceci est ma compréhension du mécanisme, en attendant plus d'informations cette description peut être en partie inexacte !!!

Pour aller plus loin :

https://quarkus.io/guides/extension-authors-guide#three-phases-of-bootstrap-and-quarkus-philosophy

https://lescastcodeurs.com/2019/03/26/lcc-207-interview-sur-quarkus-avec-emmanuel-bernard/

Pour finir :

Quarkus vient avec déjà un bel écosystème pour un framework sorti il y a même pas deux mois, en plus de ce qu'on a déjà vu, on peut citer :

OpenAPI (Swagger) : https://quarkus.io/guides/openapi-swaggerui-guide ;

OpenTracing (Jaeger) : https://quarkus.io/guides/opentracing-guide ;

HealthCheck : https://quarkus.io/guides/health-guide ;

Métrologie (Prometheus) : https://quarkus.io/guides/metrics-guide ;

Kafka : https://quarkus.io/guides/kafka-guide ;

Infinispan : https://quarkus.io/guides/infinispan-client-guide ;

Kotlin ;

Kubernetes et OpenShift : https://quarkus.io/guides/kubernetes-guide ;

Un Rest Client basé sur CXF (et ça c'est cool) : https://quarkus.io/guides/rest-client-guide.

Chaque guide est réalisé avec un artefact Maven qui génère tout ce qu'il faut pour implémenter la fonctionnalité en question. L'apprentissage est donc très aisé. Parmi les grands manquants on trouve le support des BDD NoSQL telle que MongoDB et Elasticsearch, même si leur intégration manuelle ne pose pas spécialement de problème on aurait voulu quelque chose de packagé (entre autres pour la gestion des connexions et le health check).

Mais j'ai su d'une source proche du projet que c'est déjà dans les cartons pour MongoDB et *avec panache* ! •

Le code source complet de cet exemple est sur :

https://github.com/loicmathieu/quarkus-demo

Laetitia Avrot
Experte PostgreSQL, Lætitia est co-fondatrice du mouvement Postgres Women et membre du comité du code de conduite de la communauté PostgreSQL. Elle contribue au projet PostgreSQL de différentes manières (patch, mentor pour Google Code-in Contest, speaker dans des conférences, bénévolat dans les évènements…)

PostgreSQL et la réplication

niveau 200

Comme beaucoup de SGBDR (Système de Gestion de Bases de Données Relationnelles), PostgreSQL s'est doté de plusieurs systèmes de réplication permettant d'adresser différents cas d'usages. Après quelques définitions de base, nous verrons comment c'est implémenté dans PostgreSQL et quel système de réplication utiliser dans quel cas.

PostgreSQL et la durabilité

Les SGBDR garantissent la durabilité des données. Cela signifie que lorsque les données ont été commitées et que l'utilisateur a eu le retour de commit sans erreur, les données ne doivent plus être perdues.

Une solution très simple permettant de garantir la durabilité des données serait d'écrire les données en synchrone sur le disque systématiquement au commit. Bien que solide, ce type de fonctionnement pose des gros problèmes de performance qui le rend inutilisable.

La solution qui a été définie est d'utiliser un fichier dans lequel on stocke les informations permettant d'appliquer ces modifications plus tard. Ces données sont écrites dans ce fichier particulier en mode synchrone sur disque. Cela pose moins de problèmes de performance, car ce fichier est séquentiel. Ce fichier s'appelle le journal de transactions et quasiment tous les SGBDR ont fait le choix de cette solution. Dans le monde Postgres, on utilise l'acronyme WAL (Write Ahead Log).

Le chemin d'écriture

Voici le chemin d'écriture d'une donnée dans PostgreSQL: 1

- Un utilisateur envoie une requête d'écriture à PostgreSQL ;
- Les données sont montées en mémoire ;
- Les données sont modifiées (c'est le petit carré rouge en lieu et place du carré vert en haut à droite) ;
- L'utilisateur envoie l'ordre de commit ;
- Postgres écrit la modification dans les journaux de transactions ;
- PostgreSQL renvoie l'acquittement du commit à l'utilisateur.

Plus tard, PostgreSQL recopiera ces données modifiées dans le fichier de données. En cas de crash de l'instance avant que cette opération (appelée

UN PEU DE VOCABULAIRE

Maître/Esclave

Dans la communauté PostgreSQL, les dénominations maître/esclave ne sont pas utilisées, même si elles sont extrêmement courantes avec les autres SGBD. Il y a deux raisons principales à la non-utilisation de ces termes dans le monde Postgres :

- C'est une mauvaise analogie. Si un maître meurt, on ne va pas chercher un esclave pour prendre sa place, on prend un nouveau maître ;
- Ces termes font référence à des faits peu glorieux pour l'humanité qui peuvent être offensants pour certaines personnes, donc autant ne pas les utiliser.

En lieu et place de maître/esclave, la communauté a proposé différents termes. Voici mes préférés (mais il y en a d'autres) :

- Primaire/secondaire, ce qui permet éventuellement de nommer tertiaire une réplication en cascade (réplication basée sur un nœud secondaire) ;
- Reine/princesse/ouvrière, terminologie basée sur le monde des abeilles qui permet de séparer des nœuds secondaires en fonction de leur utilisation : haute disponibilité (princesse) ou load balancing des lectures (ouvrière) ;
- Primaire/standby ou réplica, cette dénomination a l'avantage d'être comprise sans explication supplémentaire par beaucoup de monde.

Pour le reste de l'article, nous utiliserons la dénomination reine/princesse/ouvrière.

La Reine

La reine est l'instance de base de données sur laquelle des opérations d'écriture peuvent avoir lieu. Ces opérations sont ensuite envoyées aux ouvrières et princesses qui les appliquent.

L'ouvrière

L'ouvrière est une standby ouverte en lecture seule. Elle est souvent asynchrone (avec un lag de l'ordre de 200 ms si la reine et l'ouvrière sont sur le même réseau et pas trop éloignées géographiquement). Son but est de soulager la reine de certaines tâches (lecture), ce qui permet à la reine de se concentrer sur son travail (écriture).

La princesse

La princesse est une ouvrière qui, en temps normal, ne fait rien. On pourrait se poser la question de la pertinence d'avoir une instance qui ne fait rien et effectivement, c'est un coût. Ce coût permet cependant de modérer le risque de perte de données et/ou de perte de disponibilité. Une princesse est souvent synchrone (pour limiter le risque de perte de données). Dans une ruche, lorsque la reine meurt, le premier rôle de la princesse qui souhaite être reine est de tuer toutes les autres princesses. Dans les bases de données, c'est un peu pareil, on ne veut surtout pas que deux princesses se croient reines en même temps, sinon il y a risque de perte de données (syndrôme du "split brain").

Réplication physique

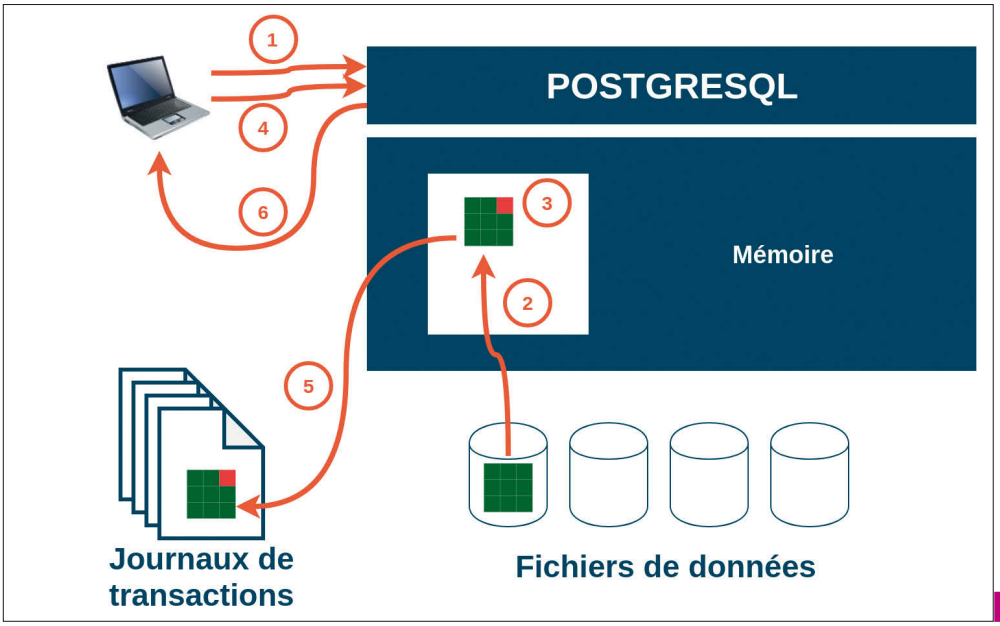
Une réplication physique est un système de réplication dans lequel les fichiers de la reine et les fichiers des ouvrières/princesses sont identiques au bit près. Une réplication physique ne peut concerner que toutes les bases de données d'une instance. On ne peut pas choisir d'exclure telle ou telle base ou telle ou telle table. La réplication physique dans Postgres peut être synchrone ou asynchrone. Si elle est synchrone, cela signifie que, avant même de renvoyer le message de commit signifiant que l'opération a pu avoir lieu, il faut que la princesse soit notifiée du changement. Cela diminue les performances des requêtes d'écriture, mais diminue également le risque de perte de données.

Réplication logique

A contrario, dans une réplication logique, les fichiers de la reine et ceux des ouvrières/princesse ne sont pas forcément identiques, même si leur contenu est le même. Une réplication logique peut permettre de choisir de répliquer uniquement certaines données (certaines bases, certaines tables, certaines colonnes, voire même certaines lignes seulement).

C'est la même différence qu'entre une sauvegarde physique (copie des fichiers binaires des bases de données) et une sauvegarde logique (export SQL des données d'une ou plusieurs bases de données).

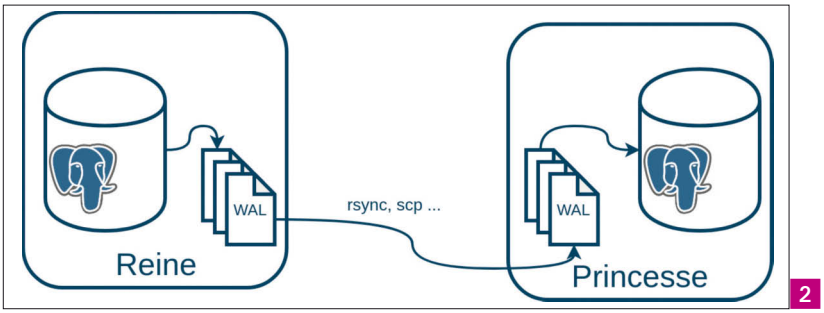
checkpoint) n'ait pu avoir lieu, les données ne sont toujours pas écrites dans les fichiers de données. Cependant, PostgreSQL garde toujours dans son fichier de contrôle le numéro de la dernière transaction prise en compte par le dernier checkpoint. Au redémarrage de l'instance, Postgres vérifie qu'il n'y a pas de données stockées dans les journaux de transactions qu'on ne retrouve pas dans les fichiers de données en comparant le numéro de la dernière transaction prise en compte par le dernier checkpoint et le numéro de la dernière transaction écrite dans les WAL. Si des données sont dans ce cas au démarrage, c'est qu'il y a eu un crash et Postgres procède à un checkpoint. Cette opération s'appelle un recovery. L'accès aux données n'est possible qu'à la fin du recovery.



1

Et la réplication dans tout ça ?

La réplication physique est basée sur le même principe. On part d'une copie binaire de la reine et on applique les journaux de transactions en permanence (recovery permanent sans fin). Historiquement, on a commencé par une technique qui s'appelait le WAL shipping : on attendait qu'un journal de transaction soit complet (16Mo sous Postgres) pour l'envoyer à la princesse (les ouvrières n'existaient pas à ce moment-là). L'envoi se faisait avec un outil externe comme rsync.



2

Cela posait le problème d'une perte de données qui pouvait être importante (16Mo). On a donc inventé la streaming réplication : une réplication qui envoie les blocs de données modifiées au fil de l'eau à la princesse (ou à l'ouvrière). En effet, les ouvrières sont arrivées à ce moment-là dans l'univers Postgres. Aujourd'hui, plus personne ne fait de WAL shipping, à moins d'être resté sur une version obsolète de Postgres qui ne permet pas la réplication streaming.

À partir de là, on a pu aller encore plus loin : la réplication synchrone. Il s'agit de vérifier que la princesse a bien reçu le journal de transactions avant d'envoyer le message de commit à l'utilisateur. De manière optionnelle, on peut également aller jusqu'à attendre que la princesse ait appliqué le journal de transactions pour renvoyer le message de commit. Bien sûr, cela ajoute du délai entre l'envoi de l'ordre du commit par l'utilisateur et la réception de la réponse, mais cela permet de s'assurer que la princesse ne sera pas en retard sur la reine.

Pour la réplication logique (native à PostgreSQL depuis Postgres 10), il a fallu ajouter plus d'informations dans les journaux de transactions pour pouvoir écrire le SQL à rejouer sur l'instance secondaire. Mais le principe est exactement le même : les journaux de transactions sont envoyés en streaming, lus sur le secondaire et on reconstruit les ordres SQL pour pouvoir les rejouer.

Les cas d'usage

Maintenant que nous sommes d'accord sur le vocabulaire utilisé dans cet article, voyons quand utiliser tel ou tel type de réplication.

Diminuer le risque de perte de données

Pour diminuer le risque de perte de données, il est préférable d'opter pour une princesse synchrone sur le même datacentre doublée d'une princesse synchrone sur un deuxième datacentre. Normalement, on évite de mettre en place des princesses synchrones sur un datacentre distant, le réseau présentant un goulet d'étranglement qui va ralentir les flux et rendre le lag assez insupportable. Mais dans ce cas de figure, on va déclarer à Postgres qu'il n'a pas besoin que les deux serveurs répondent pour que ce soit synchrone, une seule réponse suffit (Cela s'appelle le "quorum" dans le monde Postgres). 3

Dans le cas nominal, on aura donc certainement la princesse synchrone qui est sur le même datacentre qui répondra, ce qui permettra de ne pas trop impacter les performances des requêtes d'écriture sur la reine. Dans le cas où la princesse sur le même réseau ne répond plus, la deuxième princesse pourra répondre, même si cela dégradera certainement les performances des écritures sur la reine. C'est un

compromis entre la diminution du risque de perte de données et les performances.

Diminuer le risque d'indisponibilité

Pour diminuer le risque d'indisponibilité, il faut mettre en place une à plusieurs instance·s princesse·s pour permettre de basculer sur un nouveau serveur en cas d'indisponibilité de la reine. Dans la configuration la plus simple, une seule princesse est installée. Étant donné qu'on souhaite diminuer le risque d'indisponibilité, il est préférable que cette princesse se situe dans un autre datacentre, à bonne distance du premier.
Comme cette princesse est sur un datacentre distant, il est préférable de la garder en asynchrone, afin de ne pas réduire les performances de la reine. 4

Cependant, comme cette princesse est en asynchrone, il faudra attendre qu'elle ait fini d'appliquer les WAL de la reine pour pouvoir promouvoir la princesse en cas de défaillance de la reine. À défaut, il y a un risque de perte de données (perte des dernières transactions).

Il n'y a pas d'outil qui permette de gérer le failover automatique dans le cœur de PostgreSQL, donc il existe de nombreuses solutions. Voici la liste de ces outils, chacun ayant ses faiblesses et ses forces :

- PAF ;
- Patroni ;
- Pg_autofailover ;
- RepMgr.

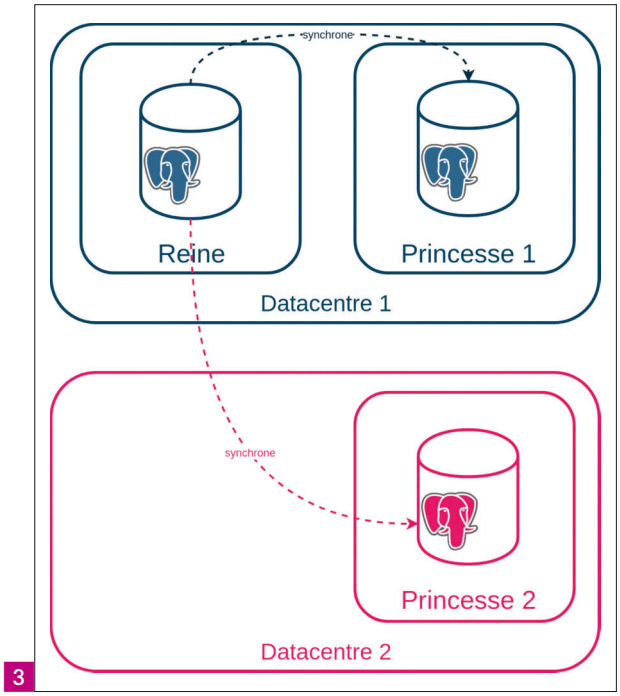
Load balancing en lecture

L'architecture de PostgreSQL est faite de telle manière que chaque connexion crée a minima un processus sur la machine. De cette manière, on comprend que PostgreSQL en natif ne peut pas maintenir des centaines de milliers de connexions simultanées à la base de données. Cependant, de très nombreux sites web très largement visités fonctionnent sous PostgreSQL.
Le secret consiste tout d'abord à utiliser un pool de connexions. C'est un outil qui maintient un certain nombre de connexions à la base de données, ce qui permet de les réutiliser tout en évitant de réitérer la phase de connexion. Le pool de connexions n'est pas inclus dans le cœur de Postgres. Cependant, de nombreux drivers de connexions en comportent un. De plus, l'excellent pg_bouncer est là pour ça.

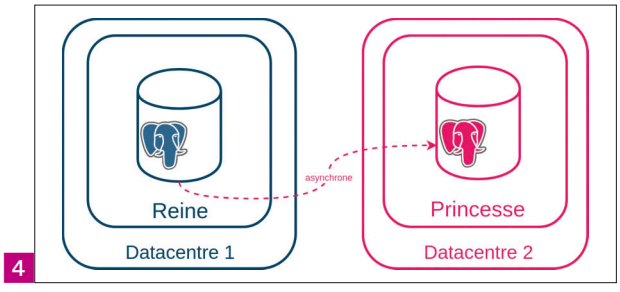
Parfois, le pool de connexions n'est pas suffisant. Souvent, ce cas se présente sur des applications web. D'après le type de workload moyen d'une application web, il y a peu d'écritures et beaucoup de lectures. Il semble donc opportun de créer plusieurs ouvrières pour pouvoir lire les données sur différentes machines. Il est préférable que ces ouvrières soient synchrones pour la consistance des données (au sens du théorème CAP, c'est-à-dire pour qu'une lecture rapporte systématiquement la dernière version committée de la ligne).
Si elles sont synchrones, il est préférable qu'elles se trouvent sur le même réseau et dans le même datacentre, sous peine de réduire les performances de la reine. 5

Load balancing en écriture

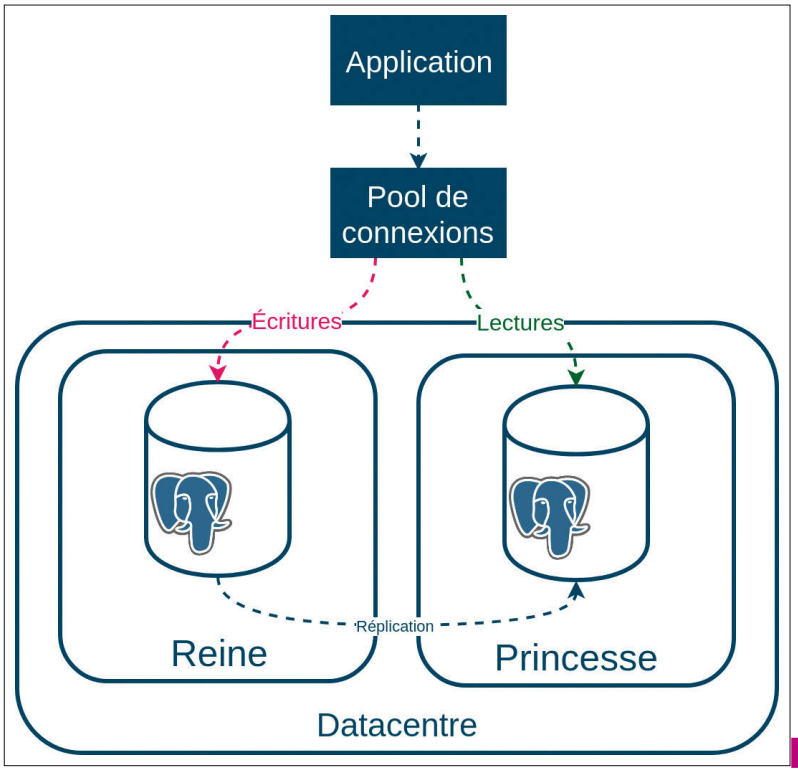
Sur les SGBDR, il n'est pas possible d'écrire sur une ouvrière. Si une telle opération avait lieu, l'ouvrière serait considérée comme corrompue, étant donné que ses fichiers doivent être identiques au bit près avec ceux de la reine.



3



4



5

Si vous avez beaucoup d'écritures (application très transactionnelle), vous pouvez jouer sur un autre levier : tenter de réduire le temps d'exécution d'une écriture pour que l'applicatif puisse en gérer plus en un temps donné. Si vraiment vous arrivez au bout de ce que vous pouvez faire dans ce domaine et que vous ne pouvez pas passer sur une machine plus puissante (et/ou des disques plus rapides), il va falloir trouver une solution pour pouvoir écrire sur plusieurs nœuds en même temps.

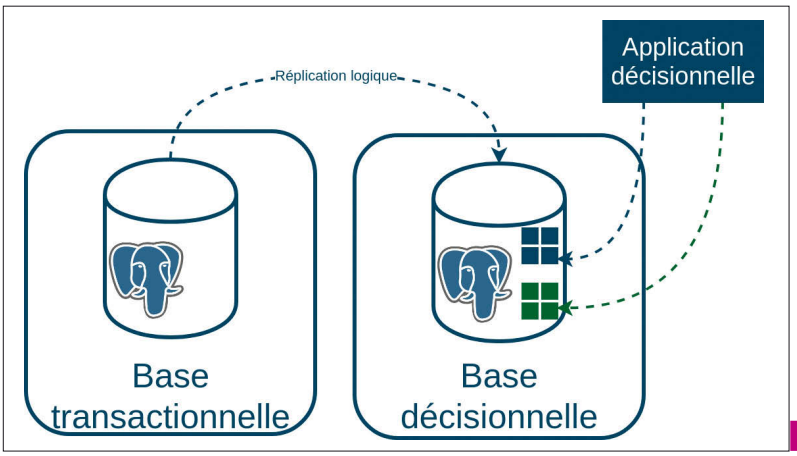
Il existe des solutions à base de réplication logique qui permette de faire de la réplication reine-reine, mais dans 99,9% des cas, ce n'est pas la solution que vous voulez mettre en place. En effet, si vous mettez en place une solution compliquée à base de réplication bidirectionnelle, il va falloir être capable de gérer les conflits et croyez-moi, vous ne voulez pas faire ça.

Dans la majorité des cas, vous aurez certainement déjà partitionné vos grosses tables. La solution est donc devant vos yeux ! Il "suffit" de répartir vos partitions sur différentes machines. Une extension de PostgreSQL permet ça. C'est Citus.

Créer une base décisionnelle

Les entrepôts de données nécessitent une structure spécifique, mais doivent être mis à jour régulièrement. Dans les années 2000, il était parfaitement acceptable de fournir un entrepôt de données mis à jour toutes les nuits (donc basé sur les données de la veille). Étrangement, cela ne paraît plus acceptable aujourd'hui.

Une bonne manière de mettre à jour un entrepôt de données est de mettre en place une réplication logique et de construire dessus des vues matérialisées en mode blue-green. L'idée est de créer un schéma blue et un schéma green. Sur chaque schéma, on crée les mêmes vues matérialisées basées sur les tables mises à jour par la réplication logique. On configure le search_path sur blue (par exemple) et on lance un recalcul des vues matérialisées sur le schéma green. À la fin du recalcul, on configure le search_path sur green et on lance le recalcul sur le schéma blue. 6



6

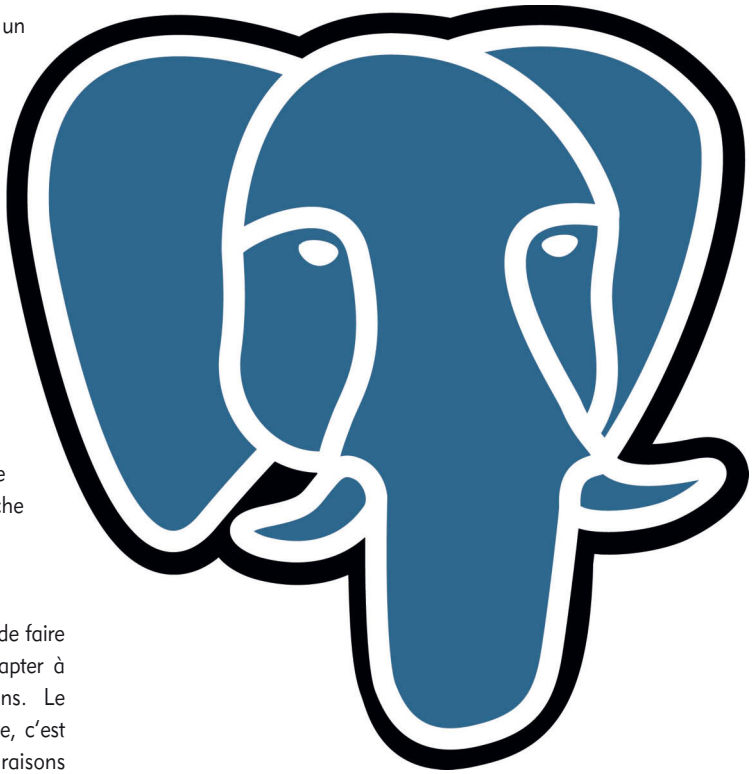
Il est à noter que si vous utilisez un pool de connexions, vous risquez d'avoir des problèmes à cause du changement de search_path qui n'est pris en compte que pour les nouvelles connexions. Dans ce cas, vous pouvez mettre en place un système qui permet de définir quel est le schéma actif (détermination du schéma à utiliser par l'applicatif).

Cependant, si vous faites du décisionnel, vous devriez avoir peu de connexions et un pool de connexions semble une surcouche inutile.

Conclusion

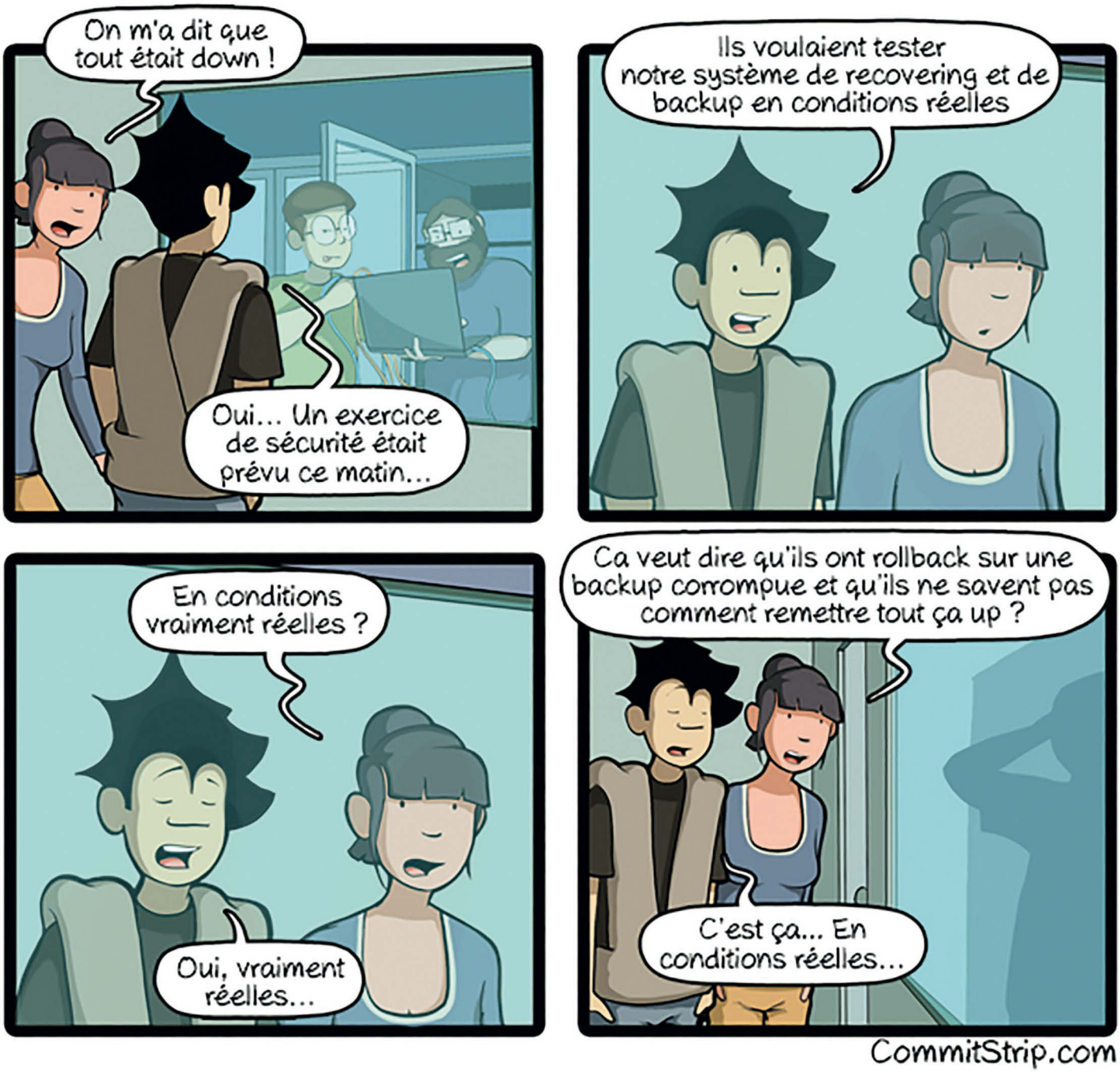
Pour conclure, PostgreSQL permet de faire beaucoup de choses et peut s'adapter à énormément de cas d'utilisations. Le principal problème que je rencontre, c'est lorsqu'on souhaite, pour des raisons d'économie parfaitement compréhensibles, fusionner les usages.

Par exemple, il est fréquent de voir des ouvrières qui sont prévues pour prendre la place de la reine si celle-ci a une défaillance. C'est parfaitement possible, mais il faudra dans ce cas accepter la perte potentielle de données. Si on la refuse, il faudra mettre en place une réplication synchrone et on s'expose à une perte de disponibilité. Il faut donc à chaque fois, bien peser le pour et le contre lorsqu'on dessine une architecture.



Une architecture n'est pas gravée dans le marbre et, le besoin évoluant, il est possible qu'un risque qui semblait acceptable l'année dernière ne le soit plus aujourd'hui. Il est donc important de prévoir des révisions de l'architecture également. Comme Postgres sort une nouvelle version chaque année, il semble opportun de se poser des questions au moment de mettre à jour Postgres. Cela permet également de se demander si telle fonctionnalité gérée par un composant externe n'a pas été intégrée au cœur de Postgres. •

En condition réelle



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 : ZDNet
Nos experts techniques : M. Hage Chahine, C. Vergne, B. Gosset, A. Outlioua, O. Bouzereau, F. Pradines, N. Bouteillier, P. Boulanger, P-Y Pamart, A. Thiefaine, L. Lefevre, J. Fischer, S. Houel, D. Dhouib, M. Ellerbach, J. Patonnier, C. Pichaud, L. Mathieu, L. Avrot, CommitStrip.
Couverture : ©Estradaanton - **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, août 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

L'Expo Inria

Revivez l'histoire de l'informatique et la révolution numérique dont Inria est un acteur majeur (premiers ordinateurs, réseaux, arrivée d'Internet, du web, de l'IA) en découvrant l'histoire d'Inria depuis son installation sur le site historique de Voluceau (ancien Quartier général de l'OTAN).

Journées européennes du patrimoine
Portes ouvertes pour tous publics

Samedi 21 septembre 2019
9 h 30 - 18 h 00

Accès gratuit sur présentation d'une pièce d'identité
Stationnement gratuit

Inria
Domaine de Voluceau - RD 307
78150 Rocquencourt
Expo@inria.fr
www.inria.fr

#JEP 2019

Inria