PROGRAMMEZ!

PROGRAMMEZ!

Le magazine des développeurs

N°245
03/04
2021

#1# Core features

PHP 8
QUASAR
Java : GraalVM
DRUPAL 9

#2# Performances

Domotique : Gladys Assistant
Python sur les calculatrices !

#3# Other fixes

Algorithme : A-star
Mission failed : comment crasher son projet ?
Je code sur Fitbit !

© Mordoliff

Le seul magazine écrit par et pour les développeurs

M 04319 - 245 - F: 6,50 € - RD

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

Contenus

Divers

Programmez! est une publication bimestrielle de Nefer-IT.
Adresse : 57, rue de Gisors 95300 Pontoise – France. Pour nous contacter : redaction@programmez.com

EDITO#245*

Cherche DevOps Full Stack.
C'est moi ou le truc ne veut rien dire ?

Il y a des jours où je me demande si je suis sur la même planète technologique que certaines entreprises et même certains recruteurs ?

« cherche DevOps fullstack » : comment comprendre cette recherche ? J'avoue avoir cherché longtemps. Est-ce un agiliste ? Un développeur ? Un être hybride à 42 bras ? Je mise plutôt pour le dernier. Rappelons tout de même que le DevOps est une philosophie, une approche, une méthodologie. Malheureusement, on accole ce terme à tout et n'importe quoi. Au point que plus personne ne sait de quoi on parle. Et cela confirme aussi que certains recruteurs et d'entreprises ne maîtrisent pas du tout les termes.

« fullstack » est du même genre. On le trouve à toutes les sauces possibles : développeur JS fullstack. On pourrait dire de même pour le développeur web, mobile, cloud computing, desktop, Java, .net, etc. Bref, on veut un développeur connaissant tout et n'importe quoi. Le développeur-pieuvre omniscient ! Là encore, il faudrait se rendre compte du non-sens. Être un bon développeur maîtrisant les piles techniques que l'on utilise au quotidien, c'est déjà assez difficile. Exiger qu'il maîtrise l'ensemble des couches et tout ce qui déborde, c'est illusoire. Développeur fullstack JavaScript ? Comment le définir ? De quoi parle-t-on ? Un dév qui maîtrise aussi bien le front et le back et tous les principaux frameworks ?

© Excelsior, 17 février 2021. Tournage du Project X

Bref, il est expert en tout. Au risque, dans la réalité d'être expert en rien.

Je peux considérer la notion fullstack sur des domaines plus définis mais pas quand on cherche un développeur JavaScript qui peut être vu comme généraliste. Je n'utile pas péjorativement (en voilà un joli mot) le terme.

Bien entendu, je cherche un dév fullstack qui sait tout faire, plutôt profil junior car c'est moins cher qu'un senior, et le tout avec beaucoup de promesses (c'est un peu comme en politique : une promesse reste une promesse, on croit ce que l'on veut bien croire) et avec un salaire, dans la moyenne. Mais si je peux proposer moins, pas de souci !

HB SILICON VALLEY

50 ans ! Voilà 50 ans que la Silicon Valley s'appelle la Silicon Valley. En janvier 1971, le journaliste Don Hoeffler publie une série d'articles sur la baie de San Francisco en utilisant le nom de « Silicon Valley ». On oublie trop souvent qu'à l'origine, la vallée était tournée vers les semi-conducteurs dès la fin des années 1950 avec Shockley puis Fairchild Semiconductor. Dans les années 60, des dizaines de fabricants s'y étaient installés.

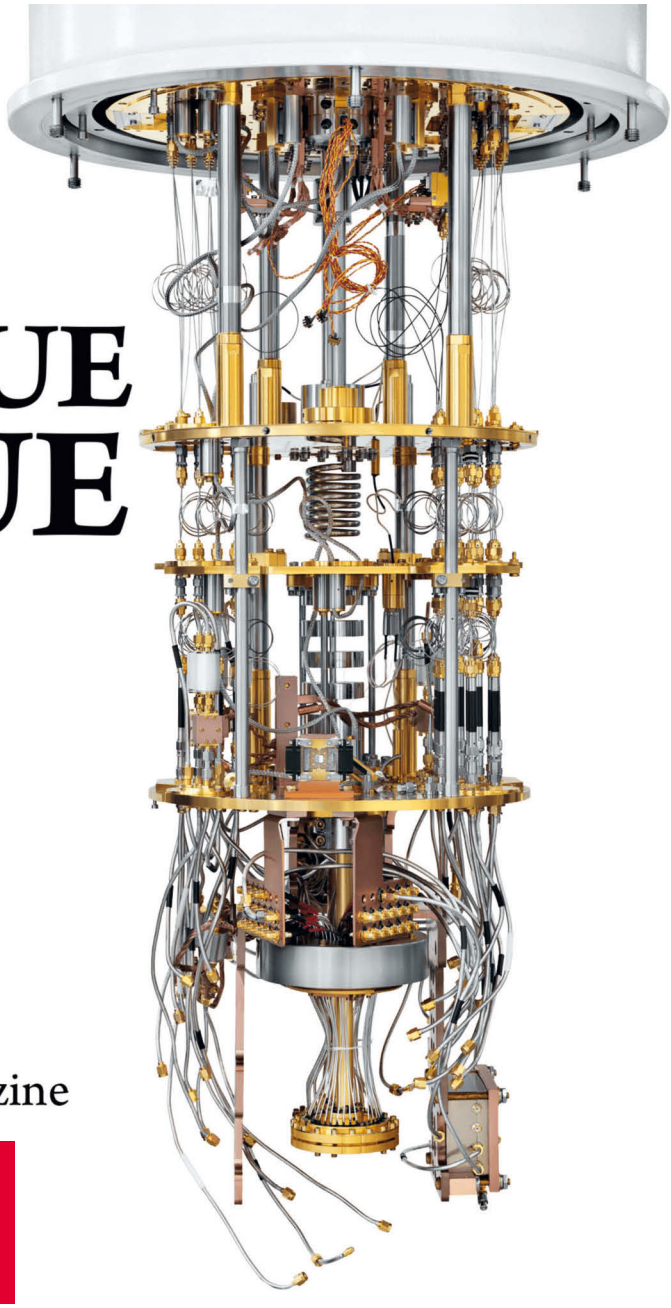
La technologie est une longue histoire dans la vallée. Grâce aux efforts de l'université de Stanford, les premières startups technologiques s'installent et s'y créent à la fin des années 30. HP est sans doute la première startup jamais créée dans la baie de San Francisco.

*245+3=248

François Tonic
Big boss de fin de niveau depuis 20 ans
Rédacteur en chef
ftonic@programmez.com

DevCon#10

INFORMATIQUE QUANTIQUE



25 MARS 2021

À PARTIR DE 13H30

7 sessions - 5 experts

Conférence pour les développeurs du magazine

PROGRAMMEZ!
Le magazine des développeurs

En partenariat avec

EPITECH.
LE FUTUR DE L'INFORMATIQUE
LE MEILLEUR DE L'INNOVATION

ZDNet.fr

Inscription :

https://tinyurl.com/1w1ylmn4

Agenda complet :

https://www.programmez.com/page-devcon/devcon10-speciale-informatique-quantique

Attention : agenda pouvant changer à tout moment selon les interdictions.

Les événements Programmez!

Nos prochains meetups :
23 mars : l'art du refactoring et le code legacy
27 avril : Flutter
29 juin : sujet à venir

Nos prochaines DevCon :
25 mars : conférence spéciale informatique quantique. À partir de 13h30.
17 juin : DevCon .Net 2e édition

Informations & inscription : programmez.com

mars

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	**16**	**17**	**18**	**19**	20	21
IoT Week 2021						
22	**23**	24	**25**	26	27	28
	Meetup		DevCon quantique (Programmez)			
29	30	31				

avril

			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	**17**	**18**
					Makerfight (Mulhouse)	
19	20	21	22	23	24	25
26	**27**	28	29	30		
	Meetup					

mai

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	**18**	19	20	21	22	23
	Devopsdays Paris					
24	25	26	27	28	29	30
31						

juin

	1	2	3	4	5	6
7	8	9	10	**11**	12	13
				Devfest Lille 2021		
14	**15**	16	**17**	18	19	20
BlendWebMix (Lyon)			DevCon .Net			
21	22	23	24	25	26	27
28	**29**	**30**	**1 juil.**	**2 juil.**		
Hack in Paris						
		Devoxx France				

juillet

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
			1	**2**	**3**	**4**
			Hack in Paris			
			Devoxx France			
				Nantes Maker Campus		
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

août

septembre

octobre

				1	2	3
4	5	6	7	8	9	10
11	12	13	**14**	**15**	16	17
			VolCamp			
18	19	20	21	22	23	24
25	26	27	28	29	30	31

novembre

décembre

SANS DATE

- MixIT
- NCrafts Paris
- BreizhCamp
- DevFest du bout du monde
- RivieraDev
- Best of Web
- Flutter Con Paris
- Sunny Tech
- DevFest Toulouse
- DevFest Nantes

Revoir les vidéos de nos dernières conférences sur YouTube
https://tinyurl.com/ygmzlh9e

Page meetup.com
https://www.meetup.com/fr-FR/Meetup-Programmez/

N°5
Disponible !

Technosaures n°5

Atari ST - Commodre SX-64 - Amiga 2000
Apple IIGS - Alan Kay - GOTEK

Commandez directement sur **programmez.com**

6,66 € *(+frais de port*)* **36 pages**

*Revue trimestrielle. Editée par Nefer-IT. *Avec frais de port : 7,66 €*

Abonnement 1 an : 30 €

Sale temps pour les cybercriminels

En matière de cybersécurité, les bonnes nouvelles sont rares alors autant les souligner : Europol a annoncé le démantèlement du botnet Emotet, généralement considéré comme le réseau d'ordinateurs infectés le plus actif et le plus dangereux. Les opérateurs du botnet revendaient l'accès aux appareils infectés à d'autres groupes cybercriminels, qui les utilisaient pour diffuser des ransomwares et autres logiciels malveillants. Les forces de l'ordre ont pris le contrôle des serveurs de commande du réseau, et prévoiraient une désinfection automatique des appareils compromis au mois d'avril. En revanche, les forces de l'ordre n'ont pas annoncé l'arrestation des personnes à la tête de l'organisation.

OvhCloud : la sécurité avant tout

OVHcloud

OvhCloud a fièrement annoncé avoir obtenu la qualification SecNumCloud, le visa de l'Anssi attestant que la société d'Octave Klaba respecte un haut niveau de sécurité pour un hébergeur cloud. Et comme si cela ne suffisait pas, l'hébergeur a également annoncé un nouveau partenariat avec Atos visant à proposer une offre de cloud sécurisée s'appuyant sur son infrastructure et sur les produits d'Atos, à destination des clients d'entreprise. OvhCloud souhaite poursuivre sur sa lancée et vise toute une série de certifications de sécurité similaire en Europe et sur d'autres marchés, SecNumCloud n'étant reconnu qu'en France.

Brexit : les transferts de données sur la sellette

Le Brexit a finalement bien eu lieu au 1er janvier 2020, et la question des transferts de données personnelles entre la Grande-Bretagne et le continent n'est pas tout à fait tranchée. Le pays bénéficie d'une période de grâce de six mois pendant laquelle les règles en vigueur avant le Brexit s'appliquent, mais celle-ci sera levée à compter du mois de juin. Et en l'absence d'une décision de la Commission européenne en sa faveur (une « décision d'adéquation », pour être précis), cela signifiera que les entreprises qui souhaitent traiter des données personnelles européennes devront proposer des garanties en matière de protection des données au moins équivalentes à celles offertes dans le droit européen. L'ICO, équivalent britannique de la CNIL, a d'ailleurs invité les entreprises britanniques à se préparer à toutes les éventualités en la matière, ce qui laisse entendre que la Commission pourrait avoir des réticences à offrir le blanc-seing tant désiré permettant la libre circulation des données. C'est en tout cas un levier de négociation que l'UE ne se privera pas d'exploiter.

Whatsapp : les CGU de la discorde

Difficile de passer à côté : Whatsapp a mis à jour ses conditions générales d'utilisation, indiquant aux utilisateurs que les données de connexion allaient dorénavant pouvoir être partagées avec les entreprises du groupe Facebook, auquel l'application appartient. Une évolution déjà mise en œuvre au cours des dernières années, mais qui restait jusque-là optionnelle. Mais cette fois, Whatsapp souhaite forcer la main à ses utilisateurs non européens pour les pousser à accepter. L'annonce a fait bien plus de bruit que ce que prévoyait Whatsapp, et ceux qui s'en réjouissent le plus sont leurs concurrents : Signal et Telegram ont ainsi trusté les tops des applications téléchargées sur les magasins d'applications depuis l'annonce.

Loon : Google se déballonne

Google a annoncé la fin de son projet Loon. Si vous n'aviez pas suivi les multiples projets de la société, un petit rappel : Loon était un projet visant à proposer de la connectivité internet en exploitant des ballons dirigeables en haute altitude. Une idée qui venait faire concurrence à Facebook et son projet Aquila, qui visait lui à utiliser un drone à l'énergie solaire pour connecter des internautes. Google a donc annoncé la fin de partie pour ce projet, estimant que « la route vers une viabilité commerciale s'est révélée beaucoup plus longue et risquée qu'espéré. »

Arcep : Laure de la Raudière prend la tête du gendarme des télécoms

L'Arcep a une nouvelle dirigeante, l'ex-députée Laure de la Raudière. Les habitués de l'Assemblée Nationale et du numérique la connaissent bien : ancienne ingénieure télécom, elle s'est illustrée dans de nombreux projets de loi ayant trait au numérique et a également été membre du Conseil National du Numérique. L'Élysée a proposé sa candidature, approuvée sans trop de difficultés par l'assemblée nationale. Seul Xavier Niel y trouvait à y redire, soulignant que Laure de la Raudière avait commencé sa carrière chez France Telecom-Orange. Si elle était passée chez Free, pas sûr qu'il se soit autant ému.

Au panthéon des génies de l'informatique

ALICE RECOQUE

Elle avait été au cœur du développement d'un des plus importants mini-ordinateurs de la CII : le Mitra 15. Elle en fut le chef de projet et mena à bien son développement. Elle s'intéressa aussi à l'intelligence artificielle. Elle fut nommée à la direction de la mission IA de Bull en 1985.

BRAD COX

Au début des années 1980, Brad Cox va créer les fondations d'un langage orienté objet qui sera utilisé par NeXT puis par Apple : Objective-C. Ce langage s'inspire du précurseur de l'OO : Smalltalk, tant en reposant sur des bases C.

La fondation Raspberry Pi dans la guerre des MCU avec la PICO

François Tonic

Si la Pi 400 était un pavé dans la marre des ordinateurs pas chers avec un modèle intégré sur lequel on rajoute l'écran, le modèle Pico est un véritable MCU killer. Oui, mais par rapport à qui ? Arduino ? ESP ? RISC-V ? La réponse n'est pas aussi claire qu'il n'y paraît. Et les lacunes du MCU pèsent sur la Pico.

À -5 $, que peut-on attendre de ce MCU à la framboise ? Le design rappelle d'autres cartes du marché, notamment les Arduino Pro / Nano. Elle embarque un MCU ARM M0+ 133 MHz, 264 Ko de RAM, 2 Mo de QSPI Flash, 26 GPIO dont 3 analogiques (ça, c'est une bonne nouvelle), 16 broches sont alimentées. On dispose de I2C, SPI et UART. Là-dessus, pas de surprise. Pour la partie alimentation / connexion, on passe par le connecteur micro-USB. Le voltage de fonctionnement est le 3,3V.

Vous l'aurez compris, Pico n'est pas très communicante : pas de WiFi ou de Bluetooth. Par rapport à d'autres MCU, c'est une grosse lacune et limite son usage en contexte IoT sauf à rajouter un module réseau. Nous verrons plus loin pourquoi la carte est muette.

Un design light

Ce qui étonne visuellement c'est le peu de composants sur le PCB. Un tiers de la Pico est vide, avec uniquement le logo. Il y a donc de la place pour d'autres capteurs, une antenne wifi, etc. La Pico embarque peu de choses : le MCU, un bouton, le connecteur USB, une LED et un capteur de température. C'est minimaliste.

La carte arrive nue, sans documentation, sans headers. Les headers ne sont même pas livrés en standard ! La fondation aurait pu faire un effort. A ce prix-là, de nombreuses ESP arrivent avec les headers (non soudés). Rajoutez environ 1 € et un peu de soudure.

Autre petit manque : aucune LED ne vous dit si la Pico fonctionne ou non. Dommage.

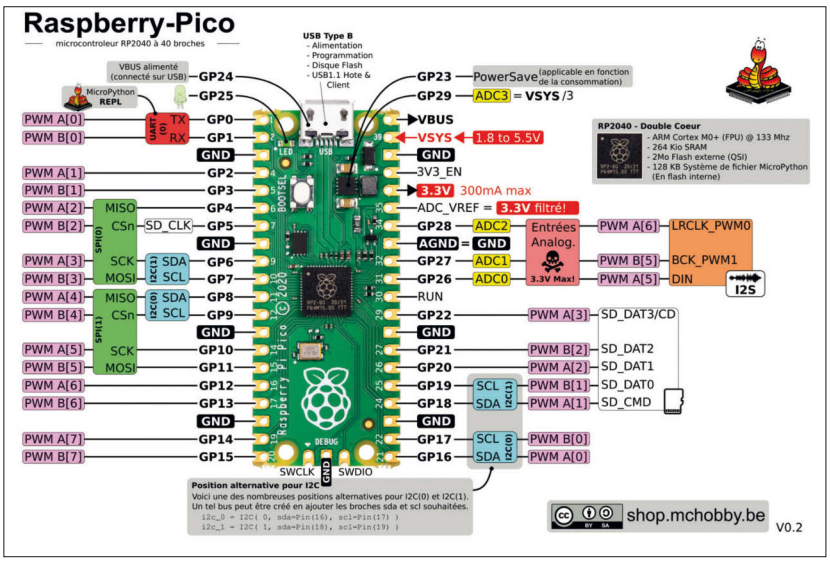
Développement classique

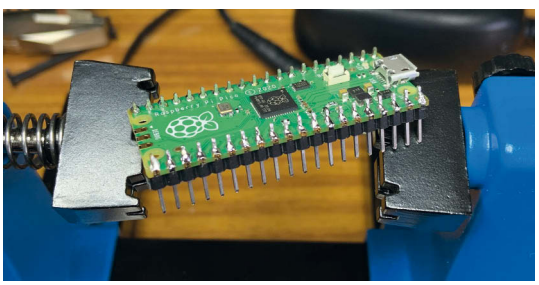
Côté développement, peu de surprises : C, C++, MicroPython (et non Python comme on peut parfois le lire). On ne retrouve pas ici la souplesse et l'intégration d'Arduino ou des ESP. Il faudra jouer du Terminal et de la ligne de commandes pour installer les SDK, configurer et exécuter les codes. Dommage que la fondation n'ait pas amélioré ce point surtout pour les novices.

Connectez tout d'abord la Pico à un autre Pi ou un PC / Mac. Pour l'IDE Python, nous pouvons utiliser Thonny IDE. Il faut installer le pico-setup puis le pico-sdk. En soi, rien de compliqué, mais il faut cloner les référentiels GitHub.

Consommation : le problème de tout MCU

La carte en elle-même a une consommation élevée à cause des différents composants. Pour la partie purement RP2040,





voir l'encadré dédié. La documentation technique annonce une moyenne de 85-95 mA à pleine charge (mode popcorn avec VGA, SD, audio). Le boot exige jusqu'à 10 mA. Le mode repos est léger avec une moyenne entre 0,8 à 1,4 mA. Attention, le mode sommeil est consommateur : + 1 mA, jusqu'à 1,9.

Si on prend les chiffres d'Espressive sur l'ESP32, la RP2040 propose une consommation intéressante. Même si l'ESP apparaît plus complet entre 160 et 260 mA en pleine charge (WiFi / Bluetooth).

Arduino admet l'arrivée de Pi, mais Raspberry est sur un plateau

La réaction d'Arduino, dans les heures suivant les annonces autour de la Pico, était assez amusante à suivre. Car Arduino reprend le RP2040 de la Pico. Ce MCU était déjà annoncé et disponible chez d'autres constructeurs de cartes. Arduino a déjà utilisé des MCU beaucoup plus puissants que les Atmel. L'annonce de la future Arduino RP2040 a quelque chose de bizarre comme improvisé. Arduino a beau dire bonjour à

PIMORONI COMPLÈTE LA PICO

Le constructeur Pimoroni est bien connu des makers. Il propose de nombreuses shields et des capteurs. Pimoroni a rapidement dévoilé toute une gamme de cartes supplémentaires pour faciliter la vie du maker – développeur :

- Écran LCD 1,14''
- Matrice LED
- Carte d'extension
- Carte VGA
- Carte audio
- Etc.

Malheureusement, cette gamme a été largement en rupture de stock en quelques heures… Mais au final, pour compléter sa Pico, il faut dépenser minimum 50 € ! Pimoroni est malheureusement cher, notamment leurs capteurs breakout.

Raspberry Pi dans le monde MCU, c'est tout de même une claque pour la carte italienne. Surtout que la future carte utilisera le MCU dessiné par la fondation ! L'annonce a beau mettre en avant les capteurs supplémentaires, c'est tout de même une gifle. Surtout que Arduino n'a pas été claire sur le prix et la disponibilité.

Après ne soyons pas trop négatifs. Car Arduino reste plus simple d'approche pour un débutant, un enfant. Et surtout elle est facilement trouvable. La Uno, malgré son âge et le peu de puissance, offre une souplesse incomparable. Et c'est open source et open hardware. Ce que la Pi n'est pas totalement.

Cependant, une des faiblesses de la fondation est son incapacité chronique à être disponible. Le marché du MCU est très actif et hyper concurrentiel. La disponibilité, et le prix sont vitaux. Pour nous, la Pico se positionne plus face aux ESP. Mais contrairement aux ESP, la Pico n'est pas adaptée aux IoT / objets produits en grande série. Et la form factor de la Pico peut poser un problème d'intégration. La fondation Raspberry n'a pas vocation à rentrer dans le monde IoT / grande série. D'autres le font très bien. Et honnêtement c'est un autre métier.

Les +

- **Prix**
- **Compacité de la carte**
- **Les langages supportés**
- **Documentation**

Les –

- **Pas de headers fournis**
- **Pas de WiFi / Bluetooth**
- **RAM et stockage**
- **Pas de LED de fonctionnement**

Documentation de référence :

https://datasheets.raspberrypi.org/rp2040/rp2040_datasheet.pdf
https://datasheets.raspberrypi.org/rp2040/hardware_design_with_rp2040.pdf

RP2040 : UN MCU JEUNE

Quelle est donc cette puce avec le logo à la framboise ? C'est la première fois que logo apparaît ainsi. La base du design est connue et mature : ARM Cortex-M0 2 cœurs à 133 MHz. Il est possible de pousser la fréquence, à vos risques et périls. Ce MCU doit concilier performance / consommation / prix. Il doit pouvoir faire aussi bien que les autres MCU du marché. Une des références est ST Micro. La fondation met en avant l'optimisation en virgule flottante pour améliorer les performances.

En lisant la documentation de référence, le RP2040 seul (nous disons bien le MCU uniquement), en sommeil, la puce consomme 0,39 mA, mais peut monter à 4,5. En utilisation intensive, le MCU consomme jusqu'à 35,5 mA. Le boot est comme toujours consommateur. Il est intéressant de constater que le document de référence montre bien que plus la fréquence est élevée, plus la puce consomme. Donc méfiance. Un des critères d'un bon MCU est la consommation.

Le RP2040 supporte par défaut 2 bus I2C, 2 SPI, 2 UART. Ce qui est généralement suffisant. Pour des besoins spécifiques ou pour hacker le hardware, vous pouvez passer par le Programmable Input & Output (PIO).

Le PIO est une interface avant tout dédiée aux fabricants et aux développeurs pour s'interfacer avec des capteurs ou du matériel non supporté par les autres bus. Vous pouvez aussi créer votre propre carte autour du RP2040. Pour ce faire, la fondation propose un document très intéressant : hardware design with RP2040. Un exemple de carte est même accessible (modèle KiCAD).

La RP2040 est donc un MCU récent par rapport aux autres offres du marché. Il doit donc prouver sa pertinence et sa stabilité. Au niveau des spécifications, ce MCU souffre de plusieurs lacunes : pas de WiFi et de Bluetooth, 264 Ko de RAM, une fréquence relativement basse, pas d'Ethernet géré par défaut.

DÉMARRAGE EN DOUCEUR

Pour coder et utiliser la Pico vous pouvez utiliser une Pi ou un PC. Pour ce faire, nous utilisons l'IDE recommandé : Thonny. Cela évite de jouer avec le Terminal.

1. On installe l'IDE puis on le lance ;
2. En bas à droite, on clique sur Python… On choisit MicroPython Pico. Il faut tout d'abord mettre à jour la PICO et installer le langage ;
3. On connecte un câble USB à la PICO, on appuie sur le bouton Bootsel de la PICO ;
4. On lance l'installation. On relâche le bouton. Après quelques minutes, l'installation indique un discret « done ».

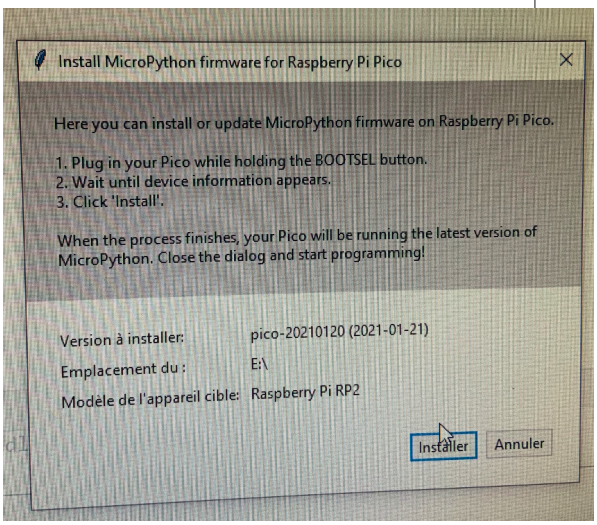
C'est tout. La PICO est prête à utiliser notre serpent favori. Vérifiez bien que nous sommes « MicroPython (Raspberry Pi Pico) » pour pouvoir interagir avec la carte.

Dans l'éditeur de code, on tape :

```
from machine import Pin
led = Pin(25, Pin.OUT)

led.toggle()
```

On clique sur Run. Et la LED de la carte s'allume. Pour sauvegarder les fichiers, il suffit de sauvegarder le fichier : (nom fichier).py. N'oubliez pas le .py. Ce n'est pas automatique… Comme sur toute bonne carte MicroPython, pour exécuter le code automatiquement quand la carte est alimentée : on nomme le fichier : main.py.



Roadmap des langages

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

Attention : à l'heure où nous bouclons ce numéro, certaines roadmaps n'étaient pas totalement fixées.

Déjà disponible

TypeScript 4.2

La principale nouveauté du langage consiste en de nouvelles possibilités pour définir des tuples avec éléments supplémentaires. Initialement, le type tuple est destiné à modéliser des tableaux avec des longueurs et des types d'éléments spécifiques. La date de la version finale n'était pas connue au moment d'imprimer ce numéro.

Tokio

Tokio est un runtime asynchrone pour le langage de programmation Rust. Ce projet est développé depuis 4 ans, même s'il vient seulement de sortir en version 1.0. Aucune version 2.0 n'est planifiée avant trois ans, et le projet garantit un support d'au moins cinq ans pour cette version 1.0.

PHP 8.0.1

Quelques mois après la sortie de PHP 8, le premier patch a été déployé. Il corrige les premières grosses remontées de problèmes et de bugs sur le core, la gestion IMAP, l'Opcache, SSL, XML...

Rust 1.50

Rust 1.50.0 est disponible. Cette version ajoute des implémentations de ops::Index et IndexMut pour les tableaux [T; N] pour n'importe quelle longueur de const N. L'équipe Rust souligne : L'opérateur d'indexation [] travaillait déjà sur les tableaux grâce à la magie du compilateur intégré, mais au niveau du type, les tableaux n'ont pas implémenté les caractéristiques de la bibliothèque jusqu'à présent.

Rust enrichit sa bibliothèque de 9 fonctions stables :

- bool::then
- btree_map::Entry::or_insert_with_key
- f32::clamp
- f64::clamp
- hash_map::Entry::or_insert_with_key
- Ord::clamp
- RefCell::take
- slice::fill
- UnsafeCell::get_mut

Bootstrap 5 bêta

La bêta 2 a été distribuée mi-février avec beaucoup de corrections et de modifications. Les plugins JS fut le gros focus des équipes notamment à cause du support de Vanilla JS à la place de jQuery.

Parmi les corrections apportées :

- Dropdown émet désormais des événements sur le .dropdown-toggle au lieu du .dropdown .
- Restauration de l'option de décalage pour les listes déroulantes.
- Correction du basculement modal en cliquant sur data-bs-toggle="modal" .
- Le composant de base est maintenant construit dans un fichier .js séparé .
- getSelector ne renvoie plus d'URL en tant que sélecteur, ce qui auparavant provoquait des erreurs dans les plugins dropdown et scrollspy.
- Composants refactorisés.

mars

Java 16

Le prochain OpenJDK est prévu pour mars. Les équipes travaillent sur plusieurs évolutions et nouveautés :

- Migration vers Git ;
- Disponibilité des fonctions de C++14 dans OpenJDK ;
- Disponibilité sur Alpine Linux & Windows/AArch64, sans oublier Apple Silicon.

Cette version, et la v17, doit aussi apporter plusieurs projets, dont Panama et Amber. Panama doit pouvoir assurer l'interconnexion entre la JVM et le code natif. Il doit faire des appels natifs depuis la JVM, accès natif aux données, optimisation du JIF (côté natif), librairies dédiées.

septembre

Java 17

Parmi les nouveautés attendues, on trouve la proposition JEP356 pour un générateur de nombres pseudo-aléatoires pour remplacer le mécanisme actuel. L'intention de cette proposition est de fournir de nouveaux types d'interface et des implémentations pour les générateurs de nombres pseudo-aléatoires (PRNG), y compris les PRNG sautables et une classe supplémentaire d'algorithmes PRNG séparables (LXM).

octobre

Python 3.10

Python change sa cadence de mise à jour. La 3.10 devrait sortir en octobre prochain. Parmi les nouveautés prévues : opérateur Union, nouveauté sur le TypeAlias, amélioration sur plusieurs modules (base64, codecs, py compile). Il n'est pas prévu d'ajouter de nouveaux modules. Les listes des retraits et dépréciations sont assez longues : checker la liste. Plusieurs changements sont indiqués sur les API. Là encore, soyez vigilant(e) sur la migration. Mais, il y a largement le temps pour le portage / la migration des codes actuels.

Date inconnue

PHP 8.1

Les développeurs travaillent à la 8.1 depuis plusieurs semaines. Cette version doit apporter des évolutions par rapport à la 8.0 et des corrections. On s'attend au support du MurmurHash (en v3). Pour le moment, la liste des modifications / ajouts n'est pas fixée.

Swift 5.4

Le langage Swift continue d'évoluer avec l'annonce de la version 5.4. Les nouveautés ne sont pas claires. Plusieurs dizaines de propositions ont été acceptées par la gouvernance du langage. La 5.4 sera dans la lignée de la précédente version et de la 5.x : stabilité des API, performances. L'arrivée officielle de la version Windows est un réel avantage pour le langage, surtout depuis le retrait d'IBM.

Angular vNext

Les prochaines versions apporteront beaucoup d'évolutions :

- Support de TypeScript 4 ;
- Généralisation d'Ivy dans Angular ;
- Changement et évolution du support RxJS ;
- Support natif de Trusted Types ;
- Intégration de MDC Web dans Angular Material (GUI).

React 18

Pour la v18, les équipes React annoncent des casses de compatibilité même si les développeurs espèrent en limiter les effets. Pour en savoir plus :
https://blog.logrocket.com/whats-new-in-react-v17-and-the-road-to-v18/

Maël Morel
Consultant agilité à Zenika Nantes

Mission Failed : ça n'arrive pas qu'aux autres

Et voilà, c'est arrivé : le produit sur lequel je travaillais a échoué, mort avec pertes et fracas. Que s'est-il passé ? Ce n'est évidemment pas la première fois qu'une intégration de progiciel dans un cycle en V échoue lamentablement. Tous les éléments classiques de l'échec étaient réunis : intégration de progiciel, guéguerres managériales, cycle en V non assumé, contractualisation à outrance, manque de délégation, etc. Il n'y aurait rien de plus à dire que ce qui a déjà été dit et qui a fait que l'agilité a émergé.

Pourquoi ces échecs à plusieurs millions d'euros existent-ils encore ? Reprenons une discussion qui court sur plus de cinquante ans sur le sujet de la gestion de projet et essayons de voir pourquoi il existe encore des décideurs qui choisiraient rationnellement une méthode prédictive dans des contextes complexes, et pourquoi c'est probablement une erreur.

Un échec cuisant, état des lieux

Le produit avait deux objectifs majeurs : rationaliser le développement d'offres d'assurance de plusieurs départements et améliorer sensiblement le Time To Market. Comment ? En intégrant un progiciel qui promettait assez de souplesse pour pouvoir créer une offre d'assurance en moins de trois mois. Après avoir développé un MVP (Minimum Viable Product), la MOE et la MOA s'engagent sur un cahier des charges à trois ans. En attendant, les utilisateurs refusent de se servir du livrable en l'état. La MOA insiste sur le fait que les délais devront être tenus et la MOE promet que les délais seront tenus. Quelques mois plus tard, le projet est arrêté. Un an et demi de travail de plusieurs dizaines de personnes mis à la poubelle, plusieurs millions d'euros perdus, des prestataires sortis en catastrophe, etc. Le bilan est lourd. Comment en est-on arrivés là ? Un atelier de rétrospective nous apprend que ce fiasco serait dû à de mauvais choix techniques, des erreurs de management et d'organisation en général. Mais un point a particulièrement retenu mon attention : comment se fait-il qu'en 2019 on puisse encore développer des produits informatiques complexes avec une approche prédictive et un cahier des charges sur trois ans ?

Un échec cuisant, le plan à trois ans

En reprenant l'état du backlog au temps T et le débit d'items livrés en moyenne par semaine par les équipes, une première projection me donnait un travail restant à faire à plus de 6 mois, alors que tout devait déjà être terminé. Cela n'empêcha pas les décideurs de demander un cahier des charges qui contenait plus de 150 fonctionnalités planifiées sur 3 ans. Puisqu'on considérait que l'utilisateur ne pouvait pas utiliser une application qui ne serait pas « complète », et puisque les mises en production se font tous les 6 mois, chaque partie prenante chargeait au maximum ses fonctionnalités de peur de rater le train et de devoir attendre de longs mois. Ainsi s'est formé un plan détaillé avec ses livrables, ses jalons, le tout jusqu'en 2021. Une date qui évidemment allait être tenue, puisqu'elle devait être tenue, selon les principes de M. Coué. S'il y eut des tentatives d'adoucir ce monolithe, comme une priorisation à 3 mois et une communication rapprochée avec le prestataire du progiciel, elles étaient réduites à néant par les engagements contractuels du cahier des charges. Heureusement la mascarade fut arrêtée suffisamment tôt pour éviter des pertes encore plus importantes. Au-delà de l'état de fait, qu'est-ce qui pourrait soutenir l'efficacité de l'utilisation d'un cahier des charges sur plusieurs années ?

La Hiding Hand : principe

Dans une publication de 1967 appelée « The principle of the Hiding Hand » (1), l'économiste Albert O. Hirschman expose comment, selon lui, il est bon de planifier à long terme sur de gros projets et de se tromper sur ses prédictions. Le principe qu'il appelle la Hiding Hand –qu'on pourrait traduire littéralement par « la main qui cache » mais aussi « la main qui donne une raclée »– forcerait les décideurs à rencontrer des difficultés non prévues et à s'y adapter, les transformant en entrepreneurs capables de prises de risque. Il explique que l'erreur de planification s'explique selon 2 modes : la « pseudo-imitation » et le « pseudo-programme-complet ». La « pseudo-imitation » fait croire au décideur qu'un modèle ayant déjà été appliqué avec succès ailleurs pourra donc se dérouler sans problème majeur ici avec un ratio de 90/10 entre le pareil et le différent. Hirschman nous explique qu'en fait ce ratio se retrouve souvent inversé avec 90 % de spécifique, et donc de différent, et 10 % de pareil. On retrouve typiquement ce mode dans l'intégration de progiciel, avec la promesse du ratio entre ce qui est disponible sur étagère et ce qui reste de spécifique à développer. Le second mode qui permet la Hiding Hand est le « pseudo-programme-complet ». Ici le décideur prend pour argent comptant le discours de l'expert qui planifie les étapes, un budget, les personnes impliquées, le matériel à déployer, le tout sur plusieurs années. En bref un programme exhaustif qu'il suffira de dérouler pour aboutir au succès du projet. L'application du plan dévoilera bien souvent que cette expertise était en fait l'équivalent d'une boule de cristal et que ses prédictions ne se dérouleront pas comme annoncées. Toute personne ayant une expérience d'un cycle en V d'au moins 6 mois sur un projet informatique reconnaîtra le mode « pseudo-programme-complet ». En résumé, le mode « pseudo-imitation » fait

apparaître un projet comme moins difficile, et le mode « pseudo-programme-complet » donne l'illusion aux décideurs qu'ils peuvent déjà savoir ce qu'il va se passer. Et ils sont tout à fait compatibles sur un même projet. Hirschman considère que ces deux modes entraînent bien des difficultés par leurs erreurs d'appréciations, mais que c'est dans cette adversité que les décideurs seront forcés d'être créatifs et de prendre des décisions pertinentes et rapides, en résumé de s'adapter comme dans la fameuse maxime de Nietzsche : « ce qui ne te tue pas te rend plus fort ».

L'exemple de l'usine de bambou pakistanaise

Hirschman donne l'exemple d'une grosse exploitation de bambou située dans un Pakistan au sortir de l'indépendance. Après quelques déboires techniques, l'usine était prête à fonctionner et exploiter les centaines d'hectares de bambou au milieu desquels elle était installée. Mais suite à une rarissime floraison du bambou le bois fut rendu inexploitable et l'usine s'est retrouvée sans matière première. Rapidement, les responsables lancèrent des recherches pour identifier une autre espèce de bambou qui soit fiable et dont la pousse est rapide, et ils arrivèrent ainsi à se sortir de ce guêpier et même à dépasser les prévisions de départ. Les décideurs avaient donc surestimé la disponibilité de matière première, mais ils avaient aussi sous-estimé leur capacité d'adaptation. C'est dans ce mécanisme que se déploie la Hiding Hand. On peut retrouver des exemples similaires dans l'informatique. Citons l'entreprise Ludicorp qui, confrontée au manque d'intérêt suscité par le jeu vidéo qu'ils développaient, ont dû s'adapter et ont créé une application web d'échange de photos très populaire : Flickr. Alors si la Hiding Hand fonctionne, pourquoi mon dernier projet a-t-il échoué ?

Le problème statistique

En 2014, Bent Flyvbjerg et Cass R. Sunstein répondent à Hirschman dans une publication nommée « The principle of the Malevolent Hiding Hand; or, the Planning Fallacy Writ Large » (2). Ils éprouvent la Hiding hand en confrontant statistiquement ses résultats. Ils observent d'abord qu'Hirschman ne compile que onze projets dans sa démonstration, ce qui est évidemment trop faible pour prouver un phénomène général. Ils entreprennent donc d'étudier 2062 grands projets et leur réussite au regard de leur coût prévu et de leur bénéfice final. Si Hirshman a raison, on devrait observer que les projets dont les coûts sont sous-évalués devraient voir leurs hausses de bénéfices largement compenser ces pertes grâce à la créativité de leurs décideurs. Qu'en est-il ? On constate que non seulement les bénéfices ne compensent que très rarement les coûts, mais qu'il n'y a en moyenne pas de hausse des bénéfices du tout. Les auteurs décrivent une Hiding Hand malveillante : les erreurs de planification ne sont pas couvertes par une créativité émergente qui, soit est absente, soit arrive trop tard, soit qui est simplement insuffisante. Cette Hiding Hand malveillante est observée 3,5 fois plus que la Hiding Hand bienveillante, ce qui les conduit à la conclusion suivante : la Hiding Hand est un cas particulier, et un plan optimiste n'est pas une condition suffisante à la réussite d'un projet.

Se faire mal, mais pas trop mal

En reprenant ce dialogue d'économistes sur un demi-siècle, j'ai cherché à trouver ce qui pourrait encore justifier, dans l'industrie logicielle, l'utilisation des méthodes prédictives dans un environnement complexe. En dehors des problématiques de contractualisation ou de psychologie (comme l'habitude ou le biais d'optimisme comparatif), qu'est-ce qui pourrait expliquer qu'on utilise encore des Cycles en V dans des développements logiciels complexes ? Tout le monde convient qu'il est difficile de planifier la création d'un logiciel sur 3 ans, et les béquilles que sont la «pseudo-imitation » et le « pseudo-programme-complet » sont des illusions dangereuses. Et il apparaît d'après Flyvbjerg et Sunstein que se sortir d'une planification fallacieuse est statistiquement difficile. Je repense aux mots d'un des décideurs de mon projet raté : « on contractualise un planning sur trois ans, avec une variation maximum de 20 % sur les jalons ». Mais pourquoi rigidifier le poids de la planification dans une industrie, le logiciel, où nous avons la chance de travailler un produit suffisamment souple et évolutif pour pouvoir faire des choix et s'adapter –dans certaines limites– tout au long du développement ? Le but de la Hiding Hand est de rendre les décideurs plus créatifs et plus capables de prise de risque. Et c'est précisément ce que propose l'empirisme, en essayant puis en analysant le résultat. On peut allier de cette manière des essais audacieux à une prise de risque mesurée. Dans un an le Manifeste Agile aura 20 ans, et pourtant l'oubli de sa quatrième valeur « l'adaptation au changement plus que le suivi d'un plan » aura encore conduit à l'échec d'un projet. Nous n'avons pas besoin d'une mauvaise planification : nous avons l'empirisme. Les méthodes prédictives avec leur planification rigide sont dangereuses ? Heureusement nous pouvons accompagner le changement ! Nous voulons fabriquer des acteurs entreprenants ? Nous avons le terrain de jeu pour essayer. J'ai une vision, un objectif, et j'avance pas à pas en m'inspectant et en m'adaptant régulièrement.

Never say never again

L'optimisme naïf de la Hiding Hand d'Hirschman n'apporte manifestement pas de solution intéressante, mais elle décrit étonnamment bien deux modes d'échec courants des méthodes prédictives. L'inspection et l'adaptation sont évidemment des conditions insuffisantes à la réussite d'un logiciel complexe, mais elles en sont les conditions nécessaires. Ce sont elles qui permettent de prendre les risques qui permettront d'avancer, au pire en se faisant mal mais jamais « trop » mal. J'aimerais conclure en partageant un contre-exemple à cet échec. Il y a quelques années un projet de compteurs électriques communicants était lancé en fanfare. Les attentes étaient fortes. Un département majeur de ce SI démarra le projet comme il savait le faire : MOA, MOE, cahier des charges sur plusieurs années, etc. Mais au bout d'un an, les indicateurs étaient au rouge. Le directeur du département était expérimenté et il pressentait l'échec à venir du projet. Il prit donc une décision courageuse : se faire accompagner, changer de méthode, faire des itérations et livrer un premier service de bout en bout. Aujourd'hui plus de 13 millions de compteurs Linky fournissent de l'électricité à 50 millions de Français.

(1) Albert O. Hirschman, « The principle of the Hiding Hand »

(2) Bent Flyvbjergand et Cass R. Sunstein, « The principle of the Malevolent Hiding Hand; or, the Planning Fallacy Writ Large »

Cobol : un langage plus que jamais actif

Échange avec Bob Yee, expert COBOL chez Compuware

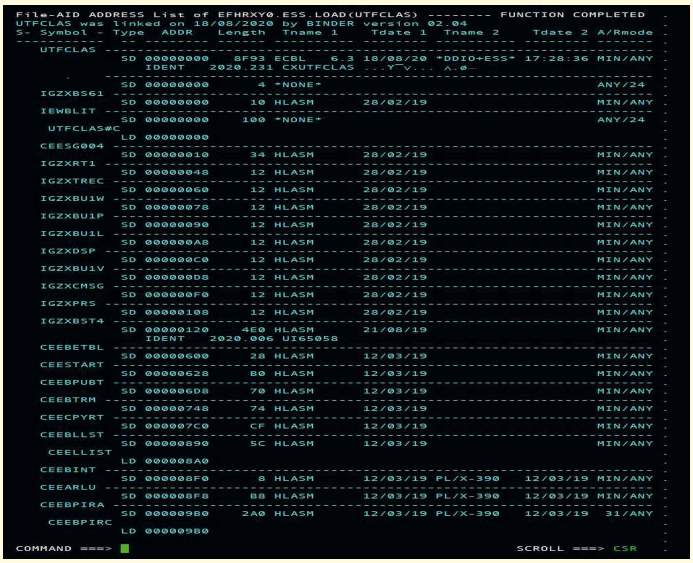
Qui aurait parié que COBOL, un langage de programmation mainframe développé il y a plus de six décennies, serait toujours un sujet tendance en 2021 ? COBOL, ou Common Business-Oriented Language, est toujours le garant du traitement de gros volumes de données dans les secteurs public et privé du monde entier - des guichets automatiques aux programmes de vols aériens en passant par le commerce électronique. Durant les premiers jours de la pandémie COVID-19, COBOL a été injustement pointé du doigt lors de l'indisponibilité de certains systèmes d'assurance-chômage américains à la suite d'un grand nombre de demandes de chômage déposées en ligne. Mais COBOL n'était pas du tout en faute.

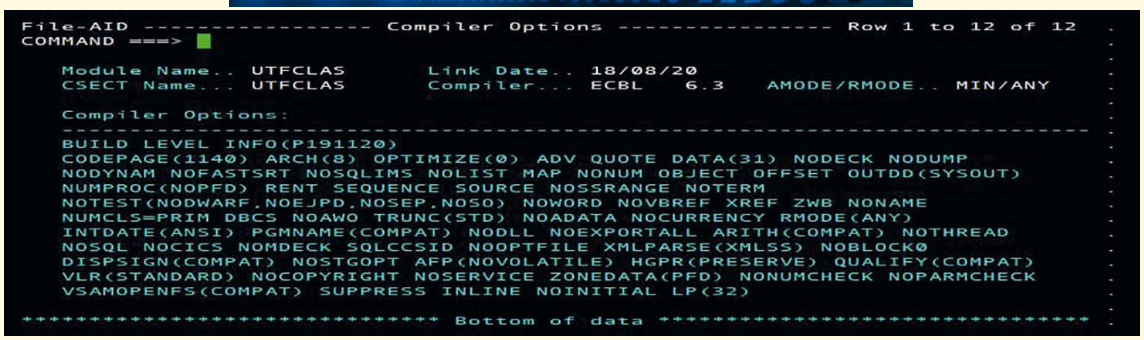
COBOL, un bouc émissaire qui ne le mérite pas !

COBOL est moderne, car il a été conçu pour rester d'actualité. Le terme « Common » reflète la demande des premiers clients mainframe pour un langage commun, hors des contraintes propriétaires. COBOL fonctionne sur différentes plateformes et continue à fonctionner au fil des mises à jour régulières, génération après génération.

Si les programmes COBOL ne sont pas perçus à la hauteur des défis des organisations modernes, c'est souvent lié aux enjeux de maintenance des applications existantes, de la modernisation de l'environnement de développement ou encore de la mise à jour matérielle. La cause première est donc davantage due à une insuffisance de budgets et aux manques de volonté des entreprises qu'à proprement parler à une limitation technologique. IBM continue à investir dans ses mainframes IBM Z, apportant toujours plus de puissance et les dotant de politiques de sécurité de données avancées. IBM fait également évoluer son compilateur COBOL pour permettre à ses clients d'améliorer considérablement la performance des anciens programmes, simplement en les recompilant, bien souvent sans changer une seule ligne de code !

En cas de renouvellement matériel, les entreprises peuvent faire face à des difficultés de maintenance des programmes COBOL. Encore une fois, l'enjeu n'est pas lié à au langage lui-même, mais bel et bien aux outils ou méthodes utilisés par les développeurs. Dans certains cas, la méthodologie de développement logiciel en cascade de type cycle en V peut en partie être responsable. Cette approche est lente et ne répond pas bien aux besoins en agilité des entreprises – d'autant plus en période de crise. Dans d'autres cas, ce sont les outils de développement et de test qui sont obsolètes. Lorsque les développeurs sont expérimentés et utilisent des interfaces de développement sur « écran vert » depuis des années, il n'y a pas de difficultés majeures. Mais dans le cas de développeurs de nouvelle génération, des outils de développement et de test modernes sont impératifs, à l'instar d'un développement Cloud.





Compuware.
a BMC company

FOCUS SUR UNE MIGRATION COBOL V6 RÉUSSIE GRÂCE À LA PLANIFICATION AVEC BOB YEE, EXPERT COMPUWARE

Où en sont la plupart des entreprises dans le processus de migration ? Que recherchent-elles ?
BY : La plupart des entreprises sont en cours de planification. Elles ont besoin d'une ligne directrice et de documents de référence. Il est important de bien comprendre que la plupart des programmes COBOL n'auront pas besoin d'être recodés ou recompilés. En fait, la majorité des entreprises ne recompileront pas tout, mais choisiront plutôt les programmes qui, une fois recompilés, apporteront un gain de performance significatif ou encore dans le cas de modification métier obligatoire. Il est important de bien noter que toute recompilation impliquera de nouveaux tests et de bons scénarios de test.

Quels conseils donneriez-vous aux entreprises pour les aider dans leur migration ?
BY : Le but est de réaliser la migration la plus fluide possible. Pas de Big Bang ! Continuer à exécuter sa propre méthodologie de développement. Il est particulièrement important de soigner les tests de non-régression puis de planifier correctement - c'est la clé. Pour l'ensemble des clients accompagnés qui ont migré vers Cobol V6, le processus a été relativement fluide quand la planification était adéquate.

Y a-t-il beaucoup d'indisponibilité applicative lors de la migration ?
BY : Non, pas du tout. Cela dépend de l'approche que vous adoptez. L'idéal est de commencer par une seule application et d'identifier les programmes à convertir vers la nouvelle version. Compilation, tests, adaptation, recompilation et utilisez des outils pour surveiller l'impact sur les performances. Une fois que l'application est en production et que vous avez finalisé votre pilote, la prochaine application va tirer parti du nouveau savoir-faire. La méthodologie d'adaptation de la première application métier doit donc servir de modèle à toutes les autres. Plus tard, lorsqu'une application sera en maintenance ou projet, la compilation de ses composants modifiés n'utilisera plus l'ancienne version du compilateur, mais Cobol V6. En gardant cela en tête, vous développez et effectuez des migrations en même temps. Donc, il n'y a pas d'indisponibilité.

Quels sont les meilleurs outils à utiliser lors de la migration ?
BY : Avant toute chose, il faut comprendre l'état du patrimoine applicatif, identifier les composants qu'il sera nécessaire de recompiler, que ce soit pour des raisons de compatibilité ou d'options de compilation. Pour cela, il est nécessaire d'utiliser l'utilitaire File-AID Library Utility (Option 3.1) pour faire un état des lieux des bibliothèques d'exécutables. L'utilisateur spécifie le nom d'une bibliothèque d'exécutables pour générer un fichier rapport au format CSV qu'il peut ouvrir dans MS Excel. Ce rapport contient, pour chaque module exécutable, toutes les CSECTS, le langage, la version de compilateur et, dans le cas des modules COBOL, les options de compilation. Ceci est indispensable, car si vous avez par exemple des programmes OS / VS COBOL et VS COBOL II, vous devez impérativement les vérifier pour vous assurer qu'ils fonctionneront dans le nouvel environnement. Vous pouvez également utiliser Strobe, la solution d'analyse et de diagnostic de performance pour les applications mainframe de Compuware. Strobe permettra de réduire les coûts matériels et logiciels en identifiant les inefficacités des applications qui entraînent une consommation excessive de CPU. Cela peut être intéressant dans le cadre de la surveillance des performances lors des phases de tests des composants avant/après migration Cobol pour comparer les économies générées.

productivité. Alors pourquoi les programmes COBOL ne répondraient-ils pas aux attentes des organisations ? Le principal défi des entreprises est de faire évoluer le patrimoine de programmes existants - certains tournant depuis de nombreuses années. À l'issue des mises à jour, de nombreux problèmes surviennent : la nouvelle fonctionnalité n'apporte pas le résultat attendu, le programme fonctionne, mais va provoquer le bug dans un programme plus ancien, la nouvelle fonctionnalité est performante avec de faibles volumes, mais entraîne une paralysie totale du système en cas de plus fort volume transactionnel… Si l'équipe de développement n'a pas les moyens d'anticiper ces problèmes, elle sera naturellement réticente à apporter des changements. Il est donc impératif que les développeurs gagnent en visibilité sur tout changement introduit dans le parc applicatif COBOL, ainsi que sur les effets que ces changements auront sur le reste des programmes, le tout, au travers d'une expérience de développement moderne et familière. À la lumière de la crise mondiale que nous vivons aujourd'hui, les compétences COBOL sont plus plébiscitées que jamais, donnant de formidables perspectives d'emploi aux experts comme aux développeurs débutants.

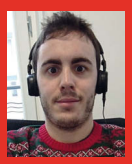
Actuellement, IBM encourage les entreprises à migrer au plus vite vers les versions 5 et 6 de COBOL pour améliorer la performance des applications sans modifier le code source, mais aussi réduire l'utilisation CPU et réaliser des économies non négligeables.

Kubernetes pour les développeurs

Kubernetes s'impose désormais comme une sérieuse option dès lors que l'on commence à réfléchir à des problématiques de déploiement et de passage à l'échelle. Kubernetes a clairement gagné la guerre de l'orchestration de conteneurs, que ce soit chez soi ou sur le cloud. Mais ne nous le cachons pas : Kubernetes reste complexe à aborder par soi-même. Cet article vous donnera les clés pour débuter avec Kubernetes sereinement en tant que développeur, et mieux comprendre les concepts qui lui sont uniques !

Alexis "Horgix" Chotard

Alexis "Horgix" est coach DevOps, SRE et formateur. Tant développeur qu'ops et passionné par l'automatisation au service des équipes, les notions d'expérience développeur et de craftsmanship lui sont chères. Kubernetes lui facilite la vie depuis 3 ans, que ce soit on-premise ou sur du cloud, à tel point qu'il a même passé 2 certifications ! Disponible sur Twitter (@Horgix) pour discuter du contenu de cet article.



Franck Cussac

Franck a suivi une formation de développeur, mais s'est pris très vite de passion pour l'intégration et le déploiement automatisé. Les conteneurs ont complètement changé sa façon de travailler. 3 ans plus tard, le voilà doublement certifié sur Kubernetes et prêt à partager tout ce qu'il a appris !

Les Conteneurs

Nous sommes en 2021 et il n'est plus nécessaire de présenter les conteneurs Linux. Bien sûr, les prémices de ceux-ci ont été amorcées il y a bien longtemps et par d'autres systèmes d'exploitation, mais c'est **Docker** qui a réellement permis de démocratiser l'utilisation des conteneurs grâces à trois choses : la **facilité d'utilisation** (le fameux *docker pull/run*), l'approche « **une seule application dans un conteneur** », et enfin la **notion d'image facilement partageable**.

Le terme « conteneur » représente aujourd'hui deux choses dans le langage de tous les jours de notre écosystème :

- la notion d'**image** : des métadonnées associées à un root filesystem, comprenant tous les fichiers nécessaires à démarrer un processus bien précis ;
- la notion de **processus isolé** : sur les distributions GNU/Linux, il s'agit d'une isolation réalisée par le kernel Linux lui-même, au travers de la notion de *namespaces* et de *control groups*.

L'outillage autour des conteneurs est désormais une commodité, notamment grâce aux standards définis par l'Open Container Initiative (https://opencontainers.org/). Bref, les conteneurs sont aujourd'hui **le** moyen standard de livrer et de faire tourner des services.

Les prémices de l'orchestration

Les conteneurs, c'est super. Mais si l'on ne s'en sert que localement, leur intérêt est vite limité (mais pas pour autant négligeable !). Puisque les conteneurs servent de nouveau **standard de packaging d'application**, transcendant les langages et les frameworks, il serait dommage de ne pas pouvoir en profiter côté serveur.

Faire tourner des applications de manière **distribuée** sur plusieurs serveurs ne date pas d'hier. Mais justement : comment était-ce fait « hier », à l'aube des orchestrateurs de conteneurs ? Revenons dans les années 2013-2014. Le mouvement DevOps commence à vraiment se populariser. Et pourtant… bien souvent, ce sont encore des administrateurs système ou « ops » qui mettent en production les applications de manière très manuelle :

- gestion de configuration (Ansible, Puppet, Saltstack, etc.) ;
- placement « manuel » sur les « bons » serveurs ;
- scripts shell de déploiement ;
- configuration de load balancing relativement manuelle ;
- des débuts de conteneurs lancés localement via systemd.

Bref, conteneurs ou pas, le déploiement de services ne disposait alors pas du même niveau de maturité et d'automatisation qu'aujourd'hui.

Au final, la problématique à traiter est la suivante : disposant de ressources sous la forme de serveurs d'une part et d'applications à faire tourner d'autre part, comment est réalisée l'association des deux ? **Sur quels serveurs vont tourner quelles applications ?** C'est cette problématique, simple en apparence, que l'on appelle orchestration.

Bien sûr, à petite échelle, pas besoin de sortir l'artillerie lourde – sans pour autant lancer vos applications manuellement, nous vous exhortons à ne pas créer de cluster Kubernetes si vous n'avez que deux serveurs et une petite dizaine d'applications dans votre garage.

Mais qu'en est-il lorsque l'on change d'échelle ? **Est-il réaliste de s'imaginer placer manuellement des centaines d'applications sur des centaines ou milliers de serveurs ?** De déplacer les applications d'un serveur qui viendrait de prendre feu vers plusieurs autres ? Bien sûr, la réponse est non. Outre le côté particulièrement fastidieux d'une telle tâche, les erreurs humaines viennent rapidement s'ajouter dans l'équation.

Finalement, cette opération d'**association ressources applications** peut très bien être réalisée **automatiquement**, probablement plus efficacement de surcroît. C'est en se basant sur ce constat que les orchestrateurs ont vu le jour, avec un objectif simple : **vous permettre d'exprimer un simple souhait tel que « je veux que cette application qui nécessite telles ressources tourne en tant d'exemplaires » et laisser une machine faire le reste** afin de faire de ce souhait une réalité.

Si ce concept vous rappelle les gestionnaires de parcs de machines virtuelles, la différence ici est bien que l'on raisonne en termes de conteneurs **applicatifs** et non de machines virtuelles bien souvent à longue durée de vie.

Guerre de l'Orchestration

Le besoin de lancer directement des applications conteneurisées « à l'échelle », ou en tout cas de manière **résiliente** sur une infrastructure **distribuée**, s'impose rapidement.

En 2015, c'est le grand boom des orchestrateurs de conteneurs, avec notamment :

- **Nomad**, de Hashicorp (vous savez, ceux qui font Terraform, Vault, etc.) ;
- **Swarm**, de chez Docker Inc. ;

- **Mesos** (+ Marathon), géré par la fondation Apache et issu de Twitter ;
- **Kubernetes**, tirant ses origines de chez Google (Borg, Omega et Project 7).

Nous n'entrerons pas ici dans un comparatif détaillé. Retenez simplement que **Kubernetes a « gagné » cette guerre des orchestrateurs**, notamment pour les raisons suivantes :

- « **hype** » de la provenance de Google ;
- rapidement **géré de manière ouverte et indépendante**, notamment depuis que Google a **cédé Kubernetes à la Linux Foundation** afin de créer la **CNCF** (Cloud Native Computing Foundation - https://cncf.io/) ;
- efforts considérables de **standardisation** et d'**interopérabilité** avec l'écosystème ;
- **extensibilité** aisée, facilitant l'adoption dans tout contexte ;
- **évolution** très rapide, avec énormément de contributions communautaires ;
- perpétue une des choses qui a fait le succès de Docker : une **CLI** (*Command Line Interface*) complète, intuitive et bien documentée ;
- très vite arrivé en **service managé** sur les plus gros fournisseurs de cloud.

Nous voici donc, en 2021, avec **Kubernetes : un orchestrateur de conteneurs permettant de gérer ceux-ci de manière distribuée sur un ensemble de serveurs**. Et gérer, ce n'est pas simplement les faire tourner : il s'agit également de faciliter leur **exposition** sur le réseau, leur **monitoring**, la mise à disposition de **stockage persistant** et bien d'autres points encore. Tout ceci, bien sûr, **sans pour autant s'en attribuer la logique**, mais plutôt en facilitant l'interconnexion avec des composants dédiés. Entrons dans le cœur du sujet afin de voir tout ceci en détail.

Préambule

Périmètre de cet article

Cet article est écrit pour des **développeurs**, utilisateurs de clusters Kubernetes. Nous n'aborderons donc pas la partie « création et gestion de clusters » et partirons du principe, tout au long de cet article, que **vous disposez d'un cluster Kubernetes fonctionnel** et d'un accès à celui-ci.

Si ce n'est pas le cas, n'hésitez pas à regarder du côté des offres de Kubernetes *as a Service* : **GKE** chez GCP ; EKS chez AWS ; **Kapsule** chez Scaleway ; AKS chez Azure ; Playground Katacoda ; etc.

Vous pouvez également vous tourner vers des solutions permettant de faire tourner un cluster minimal sur votre propre machine de manière légère, telles que : **Minikube** ; MicroK8S ; K0s ; K3s ; KinD ; Docker Desktop ; *Docker for Mac et Docker for Windows embarquent désormais la capacité de lancer un mini-cluster Kubernetes !* etc.

Exemples et sorties de commandes

Tout au long de cet article, vous trouverez des exemples de commandes, de leurs sorties et de fichiers YAML (https://yaml.org/). Un certain nombre sera **tronqué** afin de ne garder que ce qui est contextuellement pertinent. N'hésitez pas à **essayer les exemples chez vous** et à **découvrir par vous-même les détails**.

Nous vous invitons également à vous appuyer sur la **documentation officielle de Kubernetes** (https://kubernetes.io/fr/docs) ; la traduction française est réalisée par de merveilleux contributeurs dont vous pourriez d'ailleurs faire partie !

Kubernetes et ses petits noms

Vous verrez fréquemment Kubernetes écrit de manière abrégée, de par la longueur du nom qui signifie à peu de choses près « navigateur » en grec (κυβερνήτης). Les abréviations les plus courantes sont *Kube* et ***K8s*** (il y a 8 lettres entre le *K* et le *s*). Nous ne les utiliserons pas dans cet article, mais ne soyez pas surpris si vous les croisez ailleurs, elles sont très répandues. Et pour ce qui est de la prononciation : « koubeurre-net-iz » !

Déployer une application sur Kubernetes

Vous voici devant votre cluster Kubernetes. **Comment lui parler ?** Tout d'abord, oubliez les dashboards et autres clicodromes. La force de Kubernetes réside dans les APIs qu'il expose, dans leur stabilité et le nombre d'outils les utilisant.

90% de vos interactions directes avec Kubernetes se passeront via la ligne de commande *kubectl*, sur laquelle vous ferez très probablement un "*alias k=kubectl*" assez rapidement au vu du nombre de fois que vous l'utiliserez.

kubectl fonctionne, comme bien des outils en ligne de commande de nos jours, avec un système de sous-commandes, les paramètres obligatoires directement passés sur la ligne de commande et les arguments optionnels avec les habituelles formes longues et courtes à savoir "-s / --long-option". Ainsi, jetons un œil à un exemple de ligne de commande fréquent :

```
kubectl create deployment my-programmez-app --image=sapientfr/hello --dry-run=client -o yaml
```

Ré-écrire la documentation de cet outil dans ces lignes n'aurait que peu de valeur, vous trouverez votre bonheur dans le ***kubectl --help*** et la documentation en ligne.

Retenez surtout qu'il existe deux grands types d'interactions avec Kubernetes d'un point de vue ligne de commande : les opérations de **lecture** (*get, describe*) et celles de **modification** (*create*, *delete*, *apply*, *run*, etc.).

Votre premier conteneur sur Kubernetes

Il serait temps que nous lancions réellement des choses sur Kubernetes, qu'en pensez-vous ? Rien de plus simple :

```
kubectl run demo-programmez --image=sapientfr/hello-programmez
```

Et c'est tout ! Vous vous retrouvez ainsi avec un conteneur nommé *demo-programmez* basé sur l'image *sapientfr/hello-programmez* du Docker Hub, qui, si tout va bien, est désormais démarré sur votre cluster Kubernetes. Pour le vérifier c'est facile :

```
kubectl describe pod demo-programmez
```

Vous noterez que, contre toute attente, la commande n'est pas "*kubectl describe **container***" mais "*kubectl describe **pod***". Kubernetes n'est-il pourtant pas censé gérer des conteneurs ?

Un concept à part : les Pods

La notion de **Pod** est un concept bien à part et relativement

unique à Kubernetes. Pour les comprendre, il est nécessaire de faire un bref rappel sur ce que sont les conteneurs sur Linux.

Un **conteneur Linux** n'est autre qu'un **processus** comme un autre, comme le serait votre navigateur, votre shell, une base de données, etc. mais qui est isolé des autres par votre **noyau** Linux sur un certain nombre de points. Ces isolations s'appellent des **namespaces**, et sont aujourd'hui au nombre de 8, permettant notamment à chaque conteneur d'avoir sa propre *stack* réseau, son propre *hostname*, sa propre liste de processus isolée du reste du système, ou encore son propre *filesystem*.

Sans trop détailler, sachez qu'un Pod n'est autre qu'un **ensemble de plusieurs processus conteneurisés, partageant certains namespaces** et n'étant donc pas totalement isolés les uns des autres. Ils représentent l'unité de travail de Kubernetes et sont un moyen de lier plusieurs processus fortement dépendants les uns des autres et nécessitant de toujours tourner ensemble. **Figure 1**

ATTENTION AUX PODS À CONTENEURS MULTIPLES !

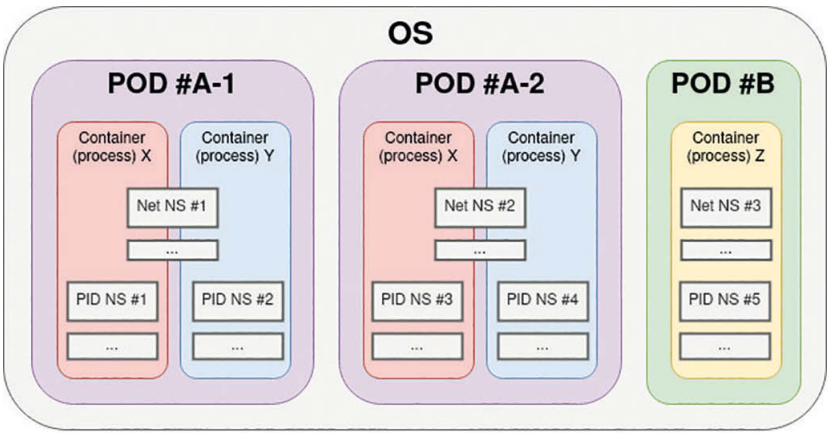
Bien qu'il soit possible de mettre différents conteneurs dans un Pod, **ce n'est pas la solution pour abstraire votre stack en entier**. Et pour cause : l'unité de déploiement et donc de scalabilité sur Kubernetes est le Pod. Avoir un Pod contenant plusieurs conteneurs, par exemple un pour votre application et un pour une base de données, signifierait que **les deux éléments vont scaler exactement de la même manière**. Vous voulez avoir votre application en 10 exemplaires ? Vous aurez aussi 10 instances de votre base de données ! Ce n'est probablement pas le comportement que vous souhaitez.

À titre indicatif, voici quelques cas d'usage valides pour les *multi-containers pods* :

- Réaliser la **terminaison TLS** au plus proche de votre application sans pour autant embarquer la logique dans celle-ci, via un reverse-proxy en *sidecar*.
- Injection d'un second processus pour **déboguer** un Pod existant en live.
- Éventuellement avoir un **cache** au plus près de votre application, bien que ce ne soit que rarement préférable à un cache partagé par toutes les instances de votre application.
- Envoyer les **logs** vers la sortie standard dans le cas d'une application legacy ne sachant loguer que dans un fichier.

Dans 95% des cas, vous n'aurez pas à avoir plusieurs conteneurs dans un même Pod ! Réfléchissez donc à deux fois à votre besoin avant de déclarer un Pod complexe et embarquant de multiples composants.

Figure 1 *Pods & Containers*



Les bases dans la vraie vie

Manifestes

Nous avons vu comment interagir avec Kubernetes de manière **impérative** via la ligne de commande *kubectl*. Bien que ceci soit courant et bien pratique pour explorer le cluster en lecture (*get* / *describe*), c'est en revanche **peu courant pour les opérations de modification.**

En effet, nous sommes à l'ère de l'automatisation et une chose est désormais indiscutable : les opérations manuelles et non reproductibles automatiquement sont à bannir ! Lorsqu'un Pod est déployé manuellement, que se passe-t-il si le cluster Kubernetes vient à disparaître subitement et doit être recréé ? Il serait souhaitable d'être en mesure de tout redéployer à l'identique et la solution n'est définitivement pas de maintenir un script bash avec des lignes de *kubectl run* à la suite les unes des autres !

Les **manifestes** Kubernetes permettent de **décrire textuellement des objets/ressources Kubernetes** (tels que des Pods) et de passer ces descriptions à Kubernetes qui s'occupera alors de faire converger l'état du cluster vers ce qui est demandé : **exactement le principe de l'infrastructure-as-code** !

Les manifestes en question sont écrits en YAML et possèdent systématiquement la structure suivante :

```
apiVersion: <API/Composant de Kubernetes gérant la ressource>
kind: <Type de ressource (Pod, Deployment, …)>
metadata: # Métadonnées de l'objet défini
  name: <Nom>
  labels:
    [...]
spec: # Spécification des paramètres de l'objet défini
  [...]
```

Bien évidemment, les éléments dans le champ *spec* seront drastiquement différents en fonction du type d'objet Kubernetes défini ! Ces manifestes sont, une fois écrits, facilement applicables via :

```
kubectl apply -f my-manifest.yaml
```

Voici par exemple un manifeste du Pod créé précédemment dans cet article :

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-programmez
spec:
  containers:
    - image: sapientfr/hello-programmez
      name: hello-world
```

Au-delà des Pods : les Deployments

Bien que les Pods soient un concept essentiel à comprendre pour utiliser Kubernetes, il n'est pas celui que vous manipulerez le plus souvent. En effet, faire tourner un seul exemplaire de votre application en isolation a rarement de l'intérêt. Si vous en êtes à dégainer Kubernetes, il y a de fortes chances que ce soit pour **faciliter la scalabilité et le déploiement** de votre application.

Kubernetes dispose donc de mécanismes (appelés *Controllers*) permettant d'ajouter de l'intelligence autour de sa fonction d'orchestrateur, à savoir placer des conteneurs sur des serveurs. Ces mécanismes incluent notamment le **passage à l'échelle** et la gestion de ***patterns* de déploiement progressif**.

Vous vous souvenez de la définition donnée en introduction ? « *Je veux que cette application qui nécessite telles ressources tourne en tant d'exemplaires* ». Les Pods permettent de définir l'application et les ressources, mais pas le nombre d'exemplaires. Pour ceci, il faut passer par la notion de ***ReplicaSet***. Mais vu que ceux-ci seront abstraits derrière des ***Deployments*** la plupart du temps, parlons directement des *Deployments* !

Voici un *Deployment* simple, créé par « *kubectl create deployment demo-programmez --image=sapientfr/hello-programmez --replicas=10* » :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-demo-for-programmez # Modifié pour lever toute confusion
spec:
  replicas: 10
  selector:
    matchLabels:
      app: demo-programmez
  template:
    metadata:
      labels:
        app: demo-programmez
    spec:
      containers:
        - image: sapientfr/hello-programmez
          name: hello-world
```

Ce *Deployment* indique à Kubernetes, comme on peut facilement le lire, de lancer 10 exemplaires du Pod défini dans la partie ***template***, qui reprend par conséquent strictement les champs que nous avons déjà vus auparavant.

Attention à l'indentation et aux *spec* imbriquées ! La *spec* de premier niveau est la spécification du *Deployment*, quant à la *spec* imbriquée c'est celle du template de Pod à lancer.

Les Deployments en action : stratégie de mise à jour et haute disponibilité

Si les *Deployments* viennent rajouter de l'abstraction par-dessus l'idée de *ReplicaSets*, ce n'est pas pour rien ! Ce sont également eux qui vont être responsables de la **mise à jour d'une application existante**.

Le challenge est le suivant : **comment mettre à jour une application sans perte de disponibilité ?** Aujourd'hui, il n'est plus acceptable d'avoir des plages de maintenance où un service n'est plus disponible pour ses utilisateurs.

Plusieurs stratégies de déploiement sont possibles pour arriver à cette finalité :

- Blue/Green deployment,
- Rolling update,
- Canary release,
- etc.

Nous ne verrons ici que le *rolling update* car c'est la stratégie par défaut des *Deployments* et la plus répandue.

Voici un exemple de rolling update d'un *Deployment* avec 10 replicas du Pod spécifié :

- Je mets à jour l'image donnée dans la spécification du Pod dans le *Deployment*.
- Kubernetes crée un nouveau *ReplicaSet* avec cette nouvelle spécification.
- Progressivement et en parallèle, le nombre de *replicas* de ce nouveau *ReplicaSet* est augmenté, tandis que celui de l'ancien *ReplicaSet* diminue.
- Une fois le nombre de nouveaux replicas atteint, l'ancien *ReplicaSet* est supprimé.

Bien sûr, il est possible de modifier cette stratégie de mise à jour ou d'influer sur cette politique de rolling update en modifiant certains paramètres (nombre de *replicas* à mettre à jour simultanément, etc.). **Figure 2**

Déployer sur Kubernetes : résumé

Nous voici donc en présence des éléments essentiels pour déployer une application sur Kubernetes : les ***Deployments***. Ils nous abstraient des notions de mise à jour de nos applications et se basent sur des ***ReplicaSets***. Ces derniers permettent de gérer le scaling horizontal de ***Pods***, agrégats potentiels de plusieurs conteneurs.

Tout ceci est défini au moyen de **manifestes** en **YAML** possédant une structure bien définie et ayant l'avantage d'être versionnables et applicables facilement au moyen de « ***kubectl apply -f my-manifest.yaml*** ».

Mon application est-elle en vie ?

Pour effectuer le déploiement progressif d'une nouvelle version d'un Pod, Kubernetes doit nécessairement savoir quand un des nouveaux *replicas* est **bel et bien « vivant »** et donc quand il lui est possible de passer au remplacement du suivant. Mais il y a également un autre moment auquel Kubernetes doit connaître l'état d'un Pod : tout au long de la vie de celui-ci. Après tout, un des buts primaires d'un orchestrateur est de redéployer les services en échec et nous ne pouvons pas nous contenter de l'arrêt d'un processus pour ceci ; une application peut très bien ne plus être en état de fonctionner mais avoir toujours son processus bel et bien vivant. Ces deux **périodes de la vie d'un Pod** sont considérées comme deux notions distinctes par Kubernetes :

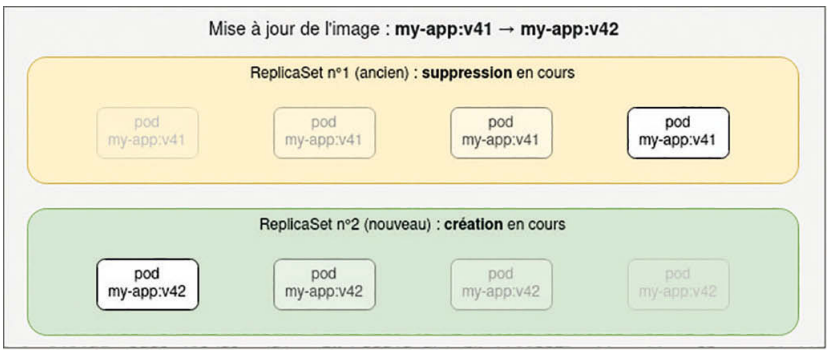
- ***Liveness***
- ***Readiness***

Readiness

Derrière le concept de *readiness* se cache l'idée d'attendre qu'une application soit **prête** (à répondre à des requêtes par exemple) avant d'être déclarée comme telle et exposée dans un *pool* de load balancing par exemple.

Ceci est typiquement pertinent pour certaines applications Java prenant du temps à démarrer. Sans cette notion, Kubernetes ne saurait pas faire la différence entre une application **n'arrivant pas à démarrer** et nécessitant d'être tuée puis redémarrée d'une application **normalement lente à démarrer**. C'est cette notion de *readiness* qui est attendue lors des *rolling updates* avant de passer au Pod suivant. Sans sa définition, Kubernetes considèrera un Pod comme *ready* dès lors que les ressources auront été allouées et le processus démarré.

Figure 2
Déploiement par rolling update



Liveness

Au-delà du moment du déploiement lui-même, il est possible à tout moment que votre application soit toujours en cours d'exécution alors qu'elle ne fonctionne plus. Quand cela arrive, Kubernetes est incapable de le détecter nativement excepté si le processus venait à se terminer/crasher. Il est donc nécessaire de lui indiquer comment savoir si un Pod est toujours en vie ou non.

Probes

Mais du coup, **comment savoir si un Pod est en vie ?** Et ça veut dire quoi « en vie » au-delà de l'existence du processus lui-même ?

La définition de « en vie / prête » est **fortement dépendante de l'application** elle-même : dans le cas d'une API, ce sera sa capacité à répondre en HTTP sur un port donné ; dans le cas d'une base de données, ce sera probablement la possibilité d'ouvrir une session TCP, voire de s'y connecter avec un réel client ; dans d'autres cas comme par exemple un batch de compression vidéo ce sera plus complexe à déterminer.

C'est pour cette raison que Kubernetes nous propose de définir nous-même la vérification de l'état d'un Pod, via la définition de ***Probes*** (sondes) sur les deux axes évoqués précédemment : « le Pod est-il *ready* ? » et « le Pod est-il *alive* ? ».

Ces vérifications peuvent se faire via 3 moyens :

- Requête HTTP ;
- Connexion TCP ;
- Commande locale (par exemple un script shell).

L'objectif d'un *LivenessProbe* est de surveiller périodiquement si un Pod est toujours fonctionnel, afin de le redémarrer si ce n'est pas le cas. Ceci nous donne par exemple la définition suivante :

```yaml
apiVersion: v1
kind: Pod
metadata: [...]
spec:
  containers:
    - name: demo-liveness
      image: k8s.gcr.io/liveness
      args:
        - /server
      livenessProbe:
        httpGet:
          path: /healthz
          port: 8080
        initialDelaySeconds: 30
        periodSeconds: 5
```

Kubernetes s'occupera ici de vérifier, toutes les 5 secondes en commençant 30 secondes après le démarrage du Pod, que celui-ci répond à une requête **HTTP** sur le port 8080 et le chemin */healthz* avec **succès** (code de statut compris entre 200 et 400).

Pour définir un *ReadinessProbe*, les paramètres sont exactement les mêmes : c'est seulement la manière dont il est utilisé par Kubernetes qui change.

Côté application

Vous l'aurez compris, il est indispensable que Kubernetes soit à même de déterminer l'état de votre application. Pour y parvenir, votre application doit bien évidemment **exposer son état !** Dans la plupart des cas, ceci se fera au moyen d'un endpoint HTTP exposé par votre application. Il est d'usage d'utiliser les chemins */health* ou */healthz*. Notez que si vous vous reposez sur un framework web moderne et relativement populaire, il est très probable que celui-ci embarque nativement ce type de fonctionnalité.

Bien sûr, **définir un bon *healthcheck* peut s'avérer compliqué** : un endpoint */healthz* doit-il aussi vérifier les dépendances de l'application, comme sa base de données, et prendre leur état en compte dans son propre état ? Doit-on inclure les *healthchecks* dans les logs d'accès au risque de générer beaucoup de bruit inutile ? Faut-il renvoyer des informations détaillées ou le code HTTP est-il suffisant ? À quel intervalle doit-on vérifier si l'application est toujours en vie ? Tout ceci étant très conceptuel et n'étant pas particulier à Kubernetes, nous ne le traiterons pas ici. Ce sont néanmoins des questions à se poser lorsque l'on souhaite déployer une application via un quelconque orchestrateur, Kubernetes compris.

Configurer son application

Nous savons désormais déployer nos applications sur Kubernetes et nous assurer que leur état est bien pris en compte. Super ! Mais **que serait une application sans sa configuration ?** Un certain nombre d'éléments vont varier d'un déploiement à l'autre de la même application : vous n'aurez sûrement pas le même nombre de *replicas* entre votre production et vos environnements de tests éphémères ; vous n'aurez pas non plus les mêmes bases de données pour vos tests et votre production ni les mêmes identifiants pour que vos applications s'y connectent. Les exemples sont nombreux. Mais **comment faire pour passer ces éléments proprement à notre application** en tirant profit d'une plateforme telle que Kubernetes ?

Configuration "cloud native"

Les **12 Factors** (https://12factor.net/fr/) ont depuis de nombreuses années posé un fait : **une application se configure désormais via ses variables d'environnement**. Cette approche a l'avantage d'être totalement **indépendante du langage** et de la manière dont est lancée votre application : que vous utilisiez Kubernetes, supervisord, systemd ou même un *docker run* à la main, vous aurez toujours la possibilité de passer facilement des variables d'environnement à tout processus. C'est une propriété de base des processus sur Linux.

Pods & variables d'environnement

Mais comment définir ces variables d'environnement pour une de nos applications sur Kubernetes ? Sans surprise, c'est vers la spécification d'un Pod et de ses conteneurs que l'on va se tourner :

```yaml
apiVersion: v1
kind: Pod
metadata: [...]
spec:
  containers:
    - name: envar-demo-container
      image: gcr.io/google-samples/node-hello:1.0
      env:
        - name: AWESOME_MAGAZINE
          value: "Programmez"
        - name: AWESOME_COMPANY
          value: "Publicis Sapient France"
```

Bien évidemment, rappelons que tout ceci reste utilisable dans le template de Pod défini dans un *Deployment*.

ConfigMaps

Définir directement les variables d'environnement dans le manifeste est bien pratique. Cependant, il peut arriver que nous souhaitions **externaliser la définition de ces variables d'environnements** hors de la responsabilité du déploiement de notre application. Cela peut être le cas lorsque certains paramètres sont communs à plusieurs applications (attention à ne pas en abuser), ou encore dans le cas de paramètres mis à disposition par une autre équipe.

C'est via un nouveau type de ressources Kubernetes que nous allons pouvoir répondre à ce besoin : les ***ConfigMaps***, qui ne sont rien d'autre que des objets contenant des paires de clé/valeur.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-demo
data:
  AWESOME_MAGAZINE: "Programmez"
  AWESOME_COMPANY: "Publicis Sapient France"
```

Pour l'utiliser, rien de plus simple :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: env-var-demo
spec:
  containers:
    - name: env-var-demo-container
      image: gcr.io/google-samples/node-hello:1.0
      envFrom:
        - configMapRef:
            name: env-demo
```

À noter qu'il est également possible de sélectionner certaines clés d'une *ConfigMap*, si l'on ne souhaite pas importer la totalité en tant que variables d'environnement. Ceci se fera via des paramètres *valueFrom* et *configMapKeyRef* que vous trouverez sans souci dans la documentation officielle. C'est également un moyen de donner d'autres noms aux variables d'environnement que celles des clés de la ConfigMap.

Si votre application se configure via un fichier, sachez qu'il est également possible de monter le contenu d'une *ConfigMap* en tant que fichier dans le *filesystem* du conteneur via un *volume* (nous ne les détaillerons pas dans cet article). Comme pour le reste, la documentation est là pour vous en apprendre les détails.

Les Secrets

Il existe un autre concept Kubernetes similaire aux *ConfigMaps* : les *Secrets*. La différence ? Simplement la manière dont ils sont stockés : contrairement aux *ConfigMaps*, le contenu des *Secrets* est encodé en base64 et ceux-ci peuvent être stockés au repos (*at rest*) de manière chiffrée.

Par exemple, voici un secret créé par *kubectl create secret generic my-secret --from-literal=password=thisisabadpassword* :

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
data:
  password: dGhpc2lzYWJhZHBhc3N3b3Jk
```

Les *Secrets* s'utilisent au niveau des Pods de la même manière que les *ConfigMaps*, avec des noms de paramètres légèrement différents.

Gérer proprement les secrets

Maintenant que nous savons configurer notre application, un dernier problème nous reste sur les bras. Cette idée de manifestes permettant de versionner la description du déploiement de nos applications, c'est vraiment super. **Mais qu'en est-il des informations sensibles à passer en configuration et définies dans des *Secrets***, tels que des identifiants de connexion à un système tiers (base de données, API d'un fournisseur, etc.) ?

Nous n'allons évidemment pas stocker dans Git une définition de *ConfigMap* avec ce genre d'information sensible en clair, ni une définition de *Secret* avec la valeur en base64 dans le fichier (pour rappel, **le base64 est parfaitement réversible** et n'est en rien du chiffrement).

Plusieurs options s'offrent à vous et le choix dépendra de votre contexte et des outils à votre disposition. Vous pourriez par exemple choisir de continuer à versionner ces éléments sensibles, mais de manière chiffrée au moyen de ***sops*** (https://github.com/mozilla/sops) ou équivalent.

Vous pourriez également envisager de **ne plus versionner ces éléments sensibles**, mais plutôt de les stocker et les mettre à disposition avec des moyens appropriés tels que Hashicorp Vault (https://www.vaultproject.io/) ou encore AWS Secret Manager (https://aws.amazon.com/secrets-manager/). Il existe toute une panoplie d'outils gravitant autour de Kubernetes permettant de faire facilement le lien entre vos manifestes / ressources Kubernetes et des secrets stockés dans des outils modernes, comme par exemple *vault-injector* (https://www.vaultproject.io/docs/platform/k8s/injector).

Enfin, avec une infrastructure hébergée chez un fournisseur de cloud le proposant, vous pourriez également considérer de ne plus avoir de secret à gérer du tout et déléguer un maximum de responsabilités à la partie *IAM* (gestion des utilisateurs et permissions) de votre fournisseur.

La gestion d'éléments de configuration sensibles reste un sujet à ne pas prendre à la légère, relativement ardu à automatiser proprement, mais qui en vaut la peine, promis !

Exposer ses applications

Jusqu'ici, nous avons pu voir comment déployer et configurer nos applications. Mais à l'heure des microservices, il y a de fortes chances pour que les applications que vous souhaitez déployer nécessitent d'être **exposées sur le réseau**, peu importe leur forme (API REST, gRPC, simple serveur web mettant des ressources front à disposition, ou même une base de données).

Dès lors que l'on parle d'exposer des applications distribuées, il devient nécessaire de parler de **load balancing**. En effet, c'est là tout l'intérêt de *scaler* une application horizontalement : répartir la charge sur différentes instances.

Services

Pour ce faire, Kubernetes propose la notion de ***Service***, se déclinant en 3 types différents : **ClusterIP, NodePort, LoadBalancer**.

Selon vos besoins, vous allez devoir faire le bon choix. Certains services auront besoin d'être exposés publiquement (ou en tout cas en dehors du cluster) tandis que d'autres pourront se contenter d'être disponibles à l'intérieur du cluster uniquement. Peut-être que votre cluster est déployé chez un fournisseur de cloud et que vous souhaitez bénéficier de son offre de load balancing managée (ELB, GCLB, etc.).

Dans le tableau ci-dessus, vous trouverez les cas d'usage de chaque type de service.

	ClusterIP	NodePort	LoadBalancer
Exposer une application à l'intérieur du cluster	✔	✖	✖
Exposer une application à l'extérieur du cluster	✖	✔	✔
Avoir un point d'entrée invariable pour mon application	✔	✖	✔

Nous n'allons pas nous appesantir dans les détails du fonctionnement de chacun. Voyons plutôt quelques exemples concrets.

Commençons par exposer une application via un ***ClusterIP*** :

```yaml
apiVersion: v1
kind: Service
metadata:
  name: catalog-svc
spec:
  ports:
    - port: 80
      targetPort: 1337
  selector:
    app: programmez-catalog
```

Dans ce manifeste, nous exprimons plusieurs choses :

- l'application à exposer/cibler est choisie via le *selector* sur le *label* « app=programmez-catalog » (ces concepts sont détaillés plus loin) ;
- nous souhaitons exposer le port 1337 des Pods de cette application ;
- nous souhaitons implicitement l'exposer de manière interne au cluster, le type de *Service* **par défaut** étant ***ClusterIP*** ;
- en tant que client contactant ce service, nous souhaitons nous adresser au port 80.

Kubernetes et le « network provider » configuré sur le cluster vont alors fournir une adresse IP *virtuelle* que nous pourrons contacter sur le port 80 et qui répartira *automatiquement* le trafic vers le port 1337 d'un des Pods sélectionnés. Cette adresse IP ne sera joignable **que depuis l'intérieur du cluster**. Nous ne détaillerons pas davantage le fonctionnement réseau ici, n'hésitez pas à contacter votre Ops préféré pour plus des détails.

Dorénavant, peu importe combien de Pods se cachent derrière notre service, ils seront toujours accessibles de la même manière.

De plus, Kubernetes associe une **entrée DNS** à cette IP et répondant au nom du service, résolvable directement depuis le cluster. C'est une implémentation de la notion de Service Discovery. Ainsi depuis un Pod partageant le même namespace, un *curl http://catalog-svc* sera parfaitement fonctionnel. Pour ce qui est de la résolution depuis d'autres namespaces, d'autres entrées DNS sont également créées pour l'interroger plus facilement :

Enregistrement DNS	Accessible depuis...
<service-name>	Le même namespace seulement
<service-name>.<service-namespace>	Tout le cluster
<service-name>.<service-namespace>.svc	Tout le cluster
<service-name>.<service-namespace>.svc.<cluster-domain>	Tout le cluster

Les *Services* de type ***NodePort*** sont les plus primaires et sont très rarement utilisés directement. Retenez simplement qu'ils nous permettent d'exposer une application sur un port de n'importe quel nœud du cluster et impliquent par conséquent de parler directement à un nœud. Il faut donc savoir lesquels existent, sont disponibles, etc.

Ces *NodePorts* sont les plus souvent utilisés derrière le rideau par les *Services* de type ***LoadBalancer***, qui iront créer un ou des load balancers externes sur votre cluster pour en exposer les applications. Le comportement sera donc totalement différent selon ce qui fournit ces load balancers externes, s'agissant le plus souvent d'un fournisseur de cloud.

La définition d'un *Service* de type *LoadBalancer* sera strictement la même que pour notre service de type *ClusterIP*, à l'exception bien sûr de l'ajout d'un champ « type: LoadBalancer » à la *spec*. Plutôt que d'aller créer une adresse IP interne au cluster, Kubernetes se chargera de provisionner un load balancer externe et de le pointer vers l'application ciblée grâce aux *selectors* (via des *NodePort*).

En résumé, voici les 3 types de *Services* : **figure 3**

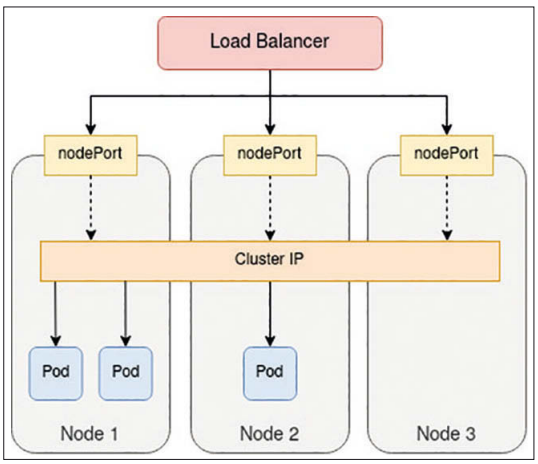


Figure 3 *Les 3 types de Services*

Ingress

Nous savons désormais exposer nos applications via des *Services*, ce qui s'avère très utile. Mais un inconvénient demeure : les services ne permettent de travailler qu'au niveau **TCP**/UDP (couche 4 du modèle OSI). Ceci est bienvenu pour des bases de données ou des files de messages possédant leur propre protocole, mais n'est pas suffisant. Avec l'apogée des microservices, il est fréquent de souhaiter répar-

tir les requêtes en fonction d'informations contenues au niveau **HTTP** : chemin dans l'URL, header, etc. C'est possible grâce à l'objet Ingress de Kubernetes !
Vous voulez avoir un seul et unique nom de domaine, avec différents chemins dans l'URL qui sont servis par différentes applications derrière ? Par exemple le classique */api* qui renverrait sur un backend tandis que les autres chemins seraient servis par votre front-end ? Ou encore un seul point d'entrée pour différents noms de domaine ? Avec de simples Services au sens Kubernetes, c'est impossible. Il sera nécessaire de passer par un composant **comprenant le protocole HTTP et sachant router/dispatcher les requêtes en fonction d'informations à ce niveau** : sur Kubernetes, il s'agit de la notion d'**Ingress**.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-http-exposition
spec:
  rules:
    - http:
        paths:
          - path: "/api"
            backend:
              service:
                name: my-backend
                port:
                  number: 80
    - host: "blog.engineering.publicissapient.fr"
      http:
        paths:
          - path: "/content"
            backend:
              service:
                name: blog
                port:
                  number: 80
```

L'*Ingress* précédent répartira le trafic entrant comme suit :

- Tout le trafic ayant comme préfixe d'URL */api* vers *my-backend*
- Tout le trafic ayant comme header *Host* « blog.engineering.publicissapient.fr » et ayant comme préfixe d'URL */content* vers le *Service* appelé *blog*

Vous n'aurez sans doute pas manqué de remarquer que l'on parle de « service » en tant que cible : eh oui, il s'agit bien de *Services* tels que nous les avons vus précédemment et ce sont bien souvent des *ClusterIP* !
L'exposition d'applications sur Kubernetes, nous l'admettons volontiers, n'est **pas vraiment évidente de prime abord**. Mais rassurez-vous, on s'y habitue rapidement et exposer des applications ne sera bientôt pour vous plus qu'une formalité ! Kubernetes a même largement simplifié la façon de gérer les problématiques réseaux en permettant à n'importe qui d'exposer une application distribuée.

Quelques concepts essentiels

Maintenant que les bases de Kubernetes en tant que développeur ont été couvertes, voyons quelques concepts essentiels pour la suite.

Les namespaces

Les namespaces sont aussi simples à comprendre qu'ils sont utiles. Le nom parle de lui-même une fois traduit : espace de nommage. Il s'agit simplement d'une manière de **ranger les ressources Kubernetes** : voyez-les comme des **boîtes** ou des dossiers.
Imaginez que vous ayez un Deployment appelé *my-awesome-db*. Il n'est évidemment pas possible d'en créer un deuxième portant le même nom car le nom est la manière principale d'identifier des ressources Kubernetes.
Pour autant, avoir plusieurs équipes qui souhaitent déployer des éléments avec le même nom n'est pas rare, sans forcément qu'il s'agisse du même composant mutualisé. Grâce aux namespaces, il est possible de ranger ces *Deployments* dans des boîtes virtuelles.
Nous aurions donc d'un côté un *Deployment* appelé *my-awesome-db* déployé dans le namespace / la boîte *team-1* et de l'autre côté un *Deployment* appelé lui aussi *my-awesome-db* déployé dans le namespace *team-2*. Ainsi, pas de conflit entre les noms !
Une bonne utilisation des namespaces est donc primordiale afin de ne pas se marcher sur les pieds en cas de **cluster Kubernetes partagé** par différentes équipes. Les namespaces servent d'ailleurs de base à la gestion des permissions en permettant de fortement isoler des équipes/projets si besoin. Ils ont également quelques cas d'utilisation un peu plus avancés que nous vous laissons découvrir par vous-même.
Quant à son utilisation, elle est simple :

- en CLI : ajout de l'option *-n* / *--namespace* aux différentes commandes ;
- dans un manifeste via la *metadata* éponyme ; cette pratique est cependant rare.

Les namespaces sont également un excellent moyen de séparer différents **environnements** d'une même application et de créer des environnements de test éphémères indépendants.

Labels & Selectors

Vous les avez peut-être remarqués dans certains manifestes d'exemples dans cet article : les **labels** et les **selectors**. Si vous êtes un habitué de l'infrastructure-as-code, la notion de « pet vs cattle » (animal de compagnie versus bétail) vous parle peut-être déjà. Si ce n'est pas le cas, disons en résumé qu'à l'ère de l'orchestration, nous ne considérons plus les applications et serveurs comme des éléments uniques, mais comme **des groupes** possédant des propriétés similaires.
En plus de désigner les différentes ressources (applications, serveurs, etc.) individuellement par leurs noms ou identifiants, Kubernetes permet de les étiqueter avec des métadonnées de type **label**, un système de taxonomie libre au format **clé/valeur**.
Exemple : nous ne verrons pas un couple d'instances d'une application comme *myapp-prod-1* et *myapp-prod-2*, mais plutôt comme deux Pods possédants les **labels (propriétés) « app=myapp » et « environment=prod »**.
On appelle **selectors** les requêtes permettant de retrouver toutes les ressources (d'un même type ou non) possédant un label donné. Les cas d'usage les plus classiques sont par exemple :

- les *Deployments et ReplicaSets* afin de savoir quels *Pods* gérer ;
- un *Service* devant cibler des *Pods* à exposer ;
- un filtre en CLI, par exemple *kubectl get all --selector=company=publicis-sapient*.

Cette manière de voir les choses est indispensable pour aborder sereinement Kubernetes et ses concepts.

Les à-côtés indispensables

Nous avons maintenant fait le tour d'une utilisation de Kubernetes « basique » en tant que développeur. Mais la vraie vie s'avère rarement simple et basique ! Dans cette dernière partie, vous découvrirez quelques **pistes d'éléments hors Kubernetes lui-même** qui vous seront, si ce n'est **indispensables**, au moins **pratiques**.

Certificats TLS

Chiffrer le trafic public avec du TLS (HTTPS) n'est désormais plus optionnel. Pour ce qui est des communications à l'intérieur de votre cluster, il s'agit davantage d'un petit plus, bien que de moins en moins optionnel. Toujours est-il que pour chiffrer les communications de vos applications, vous allez avoir besoin de **certificats TLS**.

Kubernetes n'intervient pas directement dans la génération de ceux-ci. En revanche, il vous facilite grandement le fait d'exprimer un besoin de certificat, peu importe la solution qui va le générer – fournisseur de cloud, **Let's Encrypt** (https://letsencrypt.org/) avec **cert-manager** (https://cert-manager.io/), etc. Une fois ces certificats en votre possession, Kubernetes vous facilite encore une fois la vie en vous permettant de les assigner aisément à un Ingress grâce à de simples ***annotations*** (métadonnées similaires aux *labels*). Ces annotations vous faciliteront également la configuration d'autres éléments nécessaires à un trafic plus sécurisé desservi par vos Ingress, comme la configuration des redirections HTTP→HTTPS par exemple.

Enregistrements DNS

Un autre élément est également inévitable dès lors que l'on parle d'exposition de services publiquement : les **noms de domaine**. Bien sûr, il est possible de diriger le trafic au niveau *Ingress* en se basant sur le *Host* ; mais qu'en est-il de la définition de l'enregistrement DNS approprié qui pointera sur cet *Ingress* et sera résolu par les clients afin de contacter nos applications ? Comment faire pointer automatiquement un nom de domaine sur un Load Balancer managé créé par un *Service* de type *LoadBalancer* et qui aurait comme nom par défaut quelque chose comme *a3ed633b0a[...]646536.eu-west-1.elb.amazonaws.com* ?

C'est là encore un outil tiers qui vient à notre secours, se reposant sur les primitives de Kubernetes que sont les annotations et les *Services*/*Ingress*. Son nom : **external-dns** (https://github.com/kubernetes-sigs/external-dns). Le principe : une annotation pour expliciter le nom de domaine, qui sera utilisée pour aller déclarer automatiquement l'enregistrement DNS correspondant auprès de votre DNS préféré, sous réserve bien sûr d'en avoir la permission.

« Packager » ses manifestes avec Helm

Nous avons vu de nombreuses ressources Kubernetes permettant, une fois mises bout à bout, de définir une stack applicative complète et parfois même complexe. Mais qu'en est-il de la **livraison de ces « stacks »** en tant que telles ? De leur paramétrisation pour s'adapter à différents environnements, voire être lancées par des tiers ?

Plusieurs réponses sont possibles à ces questions, mais la plus courante d'entre elles est sans conteste **Helm** (https://helm.sh/), le « gestionnaire de paquets » pour Kubernetes. De même que vous installez des paquets sur votre distribution Linux préférée via une seule ligne de commande, grâce à Helm vous pourrez déployer une stack applicative complète en une seule commande. Ces paquets sont appelés *Charts*.

On retrouve ici la force qui a fait de Docker son succès : la facilité à packager et distribuer un livrable ; dans ce cas il ne s'agit pas d'une image mais de la description d'une infrastructure logicielle et de ses dépendances. N'hésitez pas à également essayer **Kustomize** (https://kustomize.io/), moins répandu que Helm mais néanmoins plus simple pour des applications « maison », du moins de notre point de vue.

Le déploiement automatisé avec ArgoCD

Vous connaissez désormais les *Deployments* Kubernetes. Mais sont-ils suffisants ? Une notion de « stack » englobant toutes les ressources d'une application indépendamment du packaging (manifestes simples, Helm, Kustomize, autres équivalents) ne serait-elle pas la bienvenue ? Une petite interface web permettant de consulter l'historique des déploiements ne serait-il pas lui aussi un joli bonus ?

Pour tous ces éléments, **ArgoCD** (https://argoproj.github.io/argo-cd/) est aujourd'hui très répandu et viendra proposer aussi bien de l'outillage en CLI qu'une interface visuelle appréciables et pratiques. Il saura faciliter le déploiement de vos manifestes, de vos *Charts Helms* ou de nombreuses autres solutions de packaging d'applications pour Kubernetes. Sa popularité le rend incontournable et nous ne pouvions pas faire l'impasse sur cet outil ; vous pourrez cependant très bien vous en passer dans un premier temps au profit d'une bonne utilisation de votre outil de CI/CD favori.

Tekton (https://tekton.dev/) mérite également une mention honorable dans cette section.

Astuces autour de la CLI

Les sous-commandes et paramètres de *kubectl* n'auront bientôt plus de secret pour vous. Mais avec la pratique, vous découvrirez très rapidement à quel point il est pénible de sans arrêt préciser le *namespace* à cibler via « -n mon-namespace ». Les *context* Kubernetes vous permettront de définir le « namespace courant » afin de ne pas avoir à répéter ce paramètre encore et encore. Mais changer de *namespace* courant et/ou de contexte au travers des différentes sous-commandes de *kubectl config* peut rapidement s'avérer fastidieux ; **kubectx** et **kubens** (https://github.com/ahmetb/kubectx) vous faciliteront grandement la vie dans ces opérations de la vie courante. Installez-les au plus vite dès que vous vous serez familiarisé avec l'utilisation manuelle des *namespaces* et contextes !

Le projet **kubectl-aliases** (https://github.com/ahmetb/kubectl-aliases) vous fournira quant à lui quantité d'alias plus que bienvenus pour vous faciliter les commandes du quotidien. **Stern** (https://github.com/wercker/stern) vous facilitera la consultation en live de logs venant de Pods variés. Enfin, **kubeval** (https://www.kubeval.com/) vous permettra de valider rapidement et facilement la syntaxe de vos manifestes localement, si jamais votre IDE préféré ne le fait pas déjà.

En restant cette fois-ci dans les possibilités de base de *kubectl*, n'hésitez pas à vous servir de « *kubectl port-forward* » pour accéder aisément à des *Services* Kubernetes depuis votre poste de travail, vous gagnerez ainsi un temps précieux.

Pour aller plus loin avec d'autres outils

Vous l'aurez compris, un des avantages notables de Kubernetes est son écosystème. Ainsi, voici quelques outils « en vrac » concernant directement Kubernetes auxquels nous vous invitons à jeter un œil et à juger la pertinence en fonction de votre contexte, de vos expériences et de vos besoins :

- Telepresence (https://www.telepresence.io/) ;
- Open Policy Agent (https://www.openpolicyagent.org/) ;
- Istio (https://istio.io/) et Linkerd (https://linkerd.io/) ;
- Draft (https://draft.sh/).

Pour aller plus loin avec Kubernetes lui-même

Enfin, voici quelques concepts Kubernetes que nous n'avons pas détaillés voire mentionnés dans cet article, mais qui seront sans doute pour vous les prochaines étapes dans votre appréhension de Kubernetes :

- Persistence d'état via *Volumes* ;
- *Ressources* sur les Pods (*limits/requests* de CPU, RAM, etc.) ;
- *NetworkPolicies* ;
- *Taints* & *Tolerations* ;
- *Node Selector* & *(Anti)Affinity*.

Conclusion

Vous voici désormais avec toutes les cartes en main pour aborder Kubernetes sereinement en tant que développeur. Vous comprenez maintenant comment déployer vos applications sous la forme de ***Pods*** grâce à des ***Deployments*** décrits dans des **manifestes** en YAML. Vous connaissez les grandes lignes des interactions avec un cluster Kubernetes au travers de ***kubectl*** et ses sous-commandes. Vous avez une idée précise de comment vous assurer que vos applications restent en vie grâce aux ***ReadinessProbes*** et ***LivenessProbes*** et comprenez même la différence entre les deux !

Poussant plus loin, vous entrevoyez la manière de configurer une application via des variables d'environnements ou des fichiers grâce aux ***ConfigMaps*** et aux ***Secrets***. Vous appréhendez même l'exposition de vos applications de tout type grâce aux ***Services*** (ClusterIP, NodePort, LoadBalancer) et aux ***Ingress***, tâche aussi compliquée en apparence que simple en réalité. Pour boucler le tout, vous êtes armés des concepts essentiels que sont les ***namespaces*** et le duo ***labels/selectors*** et de pistes solides quant à l'impact de Kubernetes sur le design de vos applications : des endpoints de *healthchecks*, une scalabilité horizontale à prévoir, une vision d'ensemble au-delà de l'unicité d'un processus, etc.

Et comme si tout cela ne suffisait pas, vous avez le plaisir de repartir avec quelques points à creuser par vous-même pour vous rapprocher des pratiques les plus répandues avec notamment **Helm**, **ArgoCD**, de l'outillage en ligne de commande avancé, ainsi qu'une multitude de petites idées.

Cela dit, nous voudrions vous laisser sur quelques messages qui nous semblent essentiels :

Kubernetes n'est pas une solution magique. Vos applications ne seront pas magiquement scalables horizontalement. Les services à état (base de données par exemple) nécessiteront toujours une compréhension profonde de leur fonctionnement. La collaboration entre équipes et personnes sera toujours nécessaire.

Kubernetes n'est pas la seule solution. D'autres abstractions existent et permettront de vous faciliter la vie sur certains cas d'usage simples, notamment les *Platform-as-a-Service*, que ce soit chez votre fournisseur de cloud préféré (Beanstalk chez AWS, AppEngine chez GCP, etc.) ou chez des services dédiés (**Scalingo**, **CleverCloud**, Heroku, etc.). Dans une certaine mesure, des services tels que *Cloud Run* chez GCP ou *ECS+Fargate* chez AWS sont également des options pour lancer vos conteneurs sans vous soucier de ce qui se passe derrière le rideau.

Kubernetes est d'une complexité considérable. Bien qu'il soit séduisant de l'adopter pour du *on-premise* hébergé soi-même, ayez bien conscience qu'il s'agit de la clé de voûte sur laquelle tout le reste reposera. Vous ne pouvez pas vous permettre de ne pas savoir le déboguer si nécessaire. Considérez sérieusement les offres de cluster Kubernetes managés sur le cloud dont nous parlions déjà en introduction. Il y a peu de chances que le métier de votre entreprise soit de fournir des clusters Kubernetes, **concentrez-vous sur la valeur métier !**

Vous n'avez peut-être pas besoin de Kubernetes. Kubernetes reste « l'artillerie lourde » et tout le monde n'est pas Google. Pesez bien le pour et le contre avant de vous lancer tête la première et n'hésitez pas à expérimenter !

Cet article est loin d'être exhaustif, que ce soit quant aux fonctionnalités de Kubernetes lui-même ou quant aux cas d'usages envisagés dans les exemples.

Contribuez à Kubernetes si vous en avez l'occasion ; c'est ce merveilleux monde de l'open-source qui fait que nous avons aujourd'hui tant d'outils pour nous faciliter des problèmes complexes. Nous vous souhaitons une **bonne découverte de Kubernetes** pleine d'expérimentations : aucun article ne remplacera jamais la pratique !

"Ayant un pied dans le monde du développement et l'autre dans celui des opérations, il est difficile de manquer Kubernetes. Il s'agit pour moi d'un réel pas en avant pour notre industrie, mais nous ne devons pas perdre de vue qu'il ne s'agit pas d'une fin en soi. J'ai bon espoir que Kubernetes continue de se populariser, apportant une standardisation sur les déploiements équivalente à celle que Docker a pu apporter pour le packaging. Mais un jour, tout comme Docker, nous laisserons ces technologies derrière nous et irons de l'avant vers des abstractions toujours plus avancées. En attendant, Kubernetes facilite la vie de nombreuses équipes produit, et la/les vôtres en feront très certainement bientôt partie !"

Alexis "Horgix" Chotard

"En tant que développeur de formation, j'ai toujours eu beaucoup de mal avec les notions de réseaux. J'ai beaucoup appréhendé ces parties quand j'ai commencé mon apprentissage. Cependant une fois les notions de bases passées pour savoir ce que c'est qu'un certificat TLS, un DNS, les différences entre du HTTP et du TCP, j'ai pu immédiatement mettre en pratique tout ça grâce à l'outillage très simple mis à disposition par Kubernetes ! Alors mon conseil, si vous êtes comme moi, accrochez-vous ! Ça en vaut réellement la peine."

Franck Cussac

Gildas Morel des Vallons

Développeur depuis plus de 15 ans en technologies web et mobile, j'ai évolué en web agency et en startup. Spécialisé en JavaScript depuis 2015 avec de solides connaissances backend avec Node.js et front avec Vue.js. Actuellement Team Leader et Formateur Vue.js chez SFEIR.

Débuter avec Quasar Framework

Quasar est un framework qui a vu le jour il y a un plus d'1 an. Il permet de développer des frontend, sans être concurrent direct de Vue.js, React ou Angular. Il s'agit d'un environnement de développement complet intégrant parfaitement différents outils éprouvés dans le but de produire avec la même base de code : un site web, une PWA (Progressive web App), une application mobile iOS / Android et une application desktop Windows / Mac / Linux. Et ce, avec les 3 mots d'ordre : simplicité, performance, sécurité.

C'est une promesse très ambitieuse et sans pouvoir aborder tous les aspects de ce *framework*, car ils sont nombreux, cet article vous permettra d'en connaître les principaux atouts et, je l'espère, vous donnera envie de l'essayer.

Historique et popularité

Créé par Razvan Stoenescu, Quasar est un *framework* open source actuellement en v1.14. La version 1.0 a été publiée en bêta en février 2019 et la version 1.0 finale le 3 juillet 2019. Avec plus de 18 000 "stars" sur GitHub et 38 000 téléchargements hebdomadaires (source npmjs.org), il bénéficie de mises à jour très régulières sur ses différents composants :

- @quasar/app : le *framework* ;
- @quasar/cli : la CLI (*Command Line Interface*) globale incluant toutes les commandes de dev et de build ;
- @quasar/extras : les fonts, icônes et animations ;
- Icon Genie cli : une CLI permettant de gérer les icônes et *splashscreens* pour les applications mobile et tablette.

L'équipe derrière Quasar est composée d'une quinzaine de personnes : développeurs, designer, media manager et community staff entre autres. Le projet étant open source, il ne tient qu'à vous d'y contribuer ou de le sponsoriser. Le projet est soutenu par des sociétés et organisations telles que DigitalOcean.

Une promesse ambitieuse

La promesse de Quasar est très ambitieuse : "*Build high-performance VueJS user interfaces (SPA, PWA, SSR, Mobile and Desktop) in record time*".

	SPA(1)	PWA(2)	App mobile	App desktop	BEX(3)
Quasar	Oui	Oui	Cordova ou Capacitor	Electron	Oui
Ionic	Non	Oui	Cordova ou Capacitor	Non	Non
Flutter	Oui	Non	Oui	Oui	Non
React Native	Non	Non	Oui	Non	Non
Nuxt.js	Oui	Oui	Non	Non	Non
Meteor	Oui	Non	Oui	Non	Non

(1)SPA : Création d'une Single Page Application
(2)PWA : Création d'une Progressive web App
(3)BEX : Création d'une extension navigateur (Browser EXtension)

Pour appuyer cela, une vidéo présente le développement d'une simple application de liste de tâches réalisée en 30 mins. Elle s'exécute dans une page web, dans une application mobile iOS / Android et dans une application *desktop* Mac et Windows.
Vous pouvez retrouver cette vidéo sur la page "Why Quasar?" : https://quasar.dev/introduction-to-quasar
Pour obtenir une telle rapidité dans le flux de développement, Quasar assemble des technologies éprouvées :

- Le *framework* Vue.js : c'est la base principale de Quasar, tous les composants sont des composants Vue ;
- Webpack : pour créer et optimiser le *bundle* des sources ;
- Electron : pour la génération d'applications natives Mac / Windows / Linux ;
- Capacitor ou Cordova : pour la génération d'applications natives iOS et Android.

À l'aide de la CLI, il est très simple d'initialiser un projet, spécifier si l'on veut qu'il soit déployé également en application native mobile ou *desktop* et de lancer le serveur de développement pour commencer à construire son application.
Pour en savoir plus sur les raisons de la création de Quasar, n'hésitez pas à consulter la page "Why Quasar?" sur le site du *framework* : https://quasar.dev/.

Positionnement

Quasar Framework entre en compétition avec les *frameworks* proposant le déploiement d'application web, mobiles et/ou *desktop*. Parmi les plus connus et les plus utilisés, on retrouve :

- Ionic framework : développement mobile *cross-platform* utilisant Cordova et plus récemment Capacitor ;
- Flutter : boîte à outils créée par Google pour développer des applications natives avec l'accent mis sur la rapidité de développement, l'interface utilisateur expressive et flexible et des performances optimales ;
- React Native : le framework créé par Facebook permettant de créer des applications natives en utilisant React ;
- Nuxt.js : le framework Vue.js intuitif ;
- Meteor : une plateforme pour développer des applications mobiles, *desktop* et web de manière simple, efficace et *scalable*.

Matrice fonctionnelle

Cette matrice met en perspective la capacité de ces *frameworks* à générer des applications de type : SPA(1), PWA(2), mobile, desktop et des extensions navigateur (BEX(3)). (voir tableau ci-contre)

Matrice technique

Cette matrice met en perspective les caractéristiques techniques des *frameworks*.

	Open source	Langage	Framework JS utilisé	Runtime	Composants UI inclus	Bundler
Quasar	Oui	html, css, js	Vue.js	Navigateur	Oui	Webpack
Ionic	Oui	html, css, js	React, Angular, Vue.js ou Vanilla JS	Navigateur	Oui	Spécifique
Flutter	Oui	Dart	Aucun	Spécifique	Oui (Widgets)	Dart compiler
React Native	Oui	html, css, js	React	Natif iOS / Android	Non	Facebook/ metro
Nuxt.js	Oui	html, css, js	Vue.js	Navigateur	Non	Webpack
Meteor	Oui	html, css, js	React, Angular, Vue.js ou Svelte	Navigateur	Non	Spécifique

Un onboarding redoutable

Le démarrage avec le *framework* Quasar est simple et rapide grâce à la CLI. Concrètement, votre environnement de dev est prêt en seulement trois commandes. Voici les différentes étapes permettant d'initialiser une application et de la déployer sur le web et en application native iOS et Android.

Avant de démarrer, vous devez avoir installé la dernière version stable de Node.js sur votre poste de travail. Puis, ouvrez un terminal et créez un nouveau projet Node.js à l'aide des commandes :

> mkdir my-quasar-project

> cd my-quasar-project

> npm init -y

Installons à présent la CLI (*Command Line Interface*) du *framework* Quasar :

> npm i @quasar/cli

Créons le projet Quasar :

> npx quasar create my-quasar-app

Comme nous avons installé @quasar/cli en tant que dépendance du projet et non en global, vous devez utiliser "npx" pour exécuter "quasar dev". Il s'agit d'un exécutable fourni par Node.js, qui va trouver l'emplacement du binaire quasar dans les dépendances du projet.

La CLI va vous poser plusieurs questions pour configurer le projet : nom du projet, nom de l'application mobile, auteur, etc.

```
$ npx quasar create my-quasar-app

 ___
/ _ \ _   _  __ _ ___  __ _ _ __
| | | | | | |/ _` / __|/ _` | '__|
| |_| | |_| | (_| \__ \ (_| | |
 \__\_\\__,_|\__,_|___/\__,_|_|


? Project name (internal usage for dev) my-quasar-app
? Project product name (must start with letter if building mobile apps) Quasar App
? Project description A Quasar Framework app
? Author Gildas Morel des Vallons <morel-des-vallons.g@sfeir.com>
```

Puis viennent les questions techniques, à commencer par le choix du préprocesseur CSS.

Vous aurez le choix entre Sass, Stylus ou aucun.

```
? Pick your favorite CSS preprocessor: (can be changed later) (Use arrow keys)
> Sass with indented syntax (recommended)
  Sass with SCSS syntax (recommended)
  Stylus
  None (the others will still be available)
```

La prochaine étape consiste à choisir la stratégie d'import des composants et directives.

Plus il y a de composants dans votre application Quasar, plus son poids sera conséquent et potentiellement les temps de compilation seront longs (tout est relatif, on parle ici de quelques secondes). La CLI propose une configuration d'import automatique des composants permettant d'inclure uniquement ceux que vous utilisez ainsi qu'une configuration incluant tous les composants.

Sachant que vous n'aurez certainement pas besoin de tous les composants pour faire votre application, il est préférable de choisir la stratégie d'import automatique pour bénéficier du *treeshacking*, maximiser les performances et avoir un *bundle* le plus léger possible.

```
? Pick a Quasar components & directives import strategy: (can be changed later) (Use arrow keys)
> * Auto-import in-use Quasar components & directives
    - also treeshakes Quasar; minimum bundle size
  * Import everything from Quasar
    - not treeshaking Quasar; biggest bundle size
```

À présent, vous allez devoir choisir les fonctionnalités à activer. Pas d'inquiétude si vous en oubliez, il est possible de les ajouter par la suite.

```
? Check the features needed for your project:
 (*) ESLint
 (*) Vuex
 ( ) TypeScript
>(*) Axios
 ( ) Vue-i18n
 ( ) IE11 support
```

TypeScript n'est pas activé par défaut dans Quasar, c'est pourquoi il doit être sélectionné ici si vous souhaitez l'utiliser. Pour information, à l'activation de TypeScript, il vous sera demandé quel style de composant utiliser parmi : Composition API, Class-based ou Options API.

Pour en savoir plus, reportez-vous à la section "Supporting TypeScript" de la documentation : https://quasar.dev/quasar-cli/supporting-ts.

Si vous avez choisi ESLint, vous devrez choisir le préréglage.

Si vous n'avez pas de préférence, choisissez Prettier.

```
? Pick an ESLint preset:
  Standard (https://github.com/standard/standard)
  Airbnb (https://github.com/airbnb/javascript)
> Prettier (https://github.com/prettier/prettier)
```

Enfin, vous devrez choisir entre Yarn et NPM.

Si vous n'avez pas de préférence, choisissez NPM.

```
$ npx quasar create my-quasar-app

 ___
/ _ \ _   _  __ _ ___  __ _ _ __
| | | | | | |/ _` / __|/ _` | '__|
| |_| | |_| | (_| \__ \ (_| | |
 \__\_\\__,_|\__,_|___/\__,_|_|


? Project name (internal usage for dev) my-quasar-app
? Project product name (must start with letter if building mobile apps) Quasar App
? Project description A Quasar Framework app
? Author Gildas Morel des Vallons <morel-des-vallons.g@sfeir.com>
? Pick your favorite CSS preprocessor: (can be changed later) Sass
? Pick a Quasar components & directives import strategy: (can be changed later) Manual
? Check the features needed for your project: ESLint, Vuex, Axios
? Pick an ESLint preset: Standard
? Cordova/Capacitor id (disregard if not building mobile apps) org.cordova.quasar.app
? Should we run `npm install` for you after the project has been created? (recommended) (Use arrow keys)
> Yes, use Yarn (recommended)
  Yes, use NPM
  No, I will handle that myself
```

Une fois la dernière étape validée, la CLI va initialiser le projet et le paramétrer avec la configuration que vous venez de spécifier.

```
Quasar CLI · Generated "my-quasar-app".


[*] Installing project dependencies ...

[..................] \ fetchMetadata: sill pacote range manifest for external-editor@^3.0.3 fetched in 160ms
```

```
 [*] Quasar Project initialization finished!

To get started:

  cd my-quasar-app
  quasar dev

Documentation can be found at: https://quasar.dev

Quasar is relying on donations to evolve. We'd be very grateful if you can
read our manifest on "Why donations are important": https://quasar.dev/why-donate
Donation campaign: https://donate.quasar.dev
Any amount is very welcomed.
If invoices are required, please first contact razvan@quasar.dev

Please give us a star on Github if you appreciate our work:
https://github.com/quasarframework/quasar

Enjoy! - Quasar Team
```

À la fin du processus, votre application est prête, elle a été créée dans le répertoire my-quasar-app

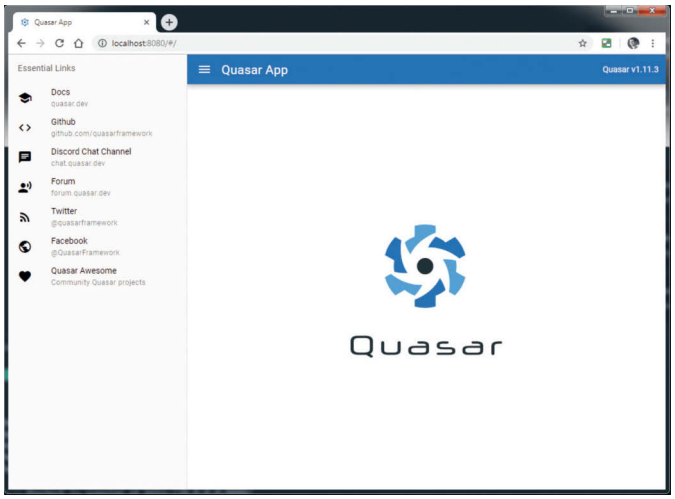
> cd my-quasar-app

Lancer l'application web

Pour lancer votre application Quasar en mode développement, exécutez la commande :

> npx quasar dev

La compilation va prendre quelques secondes, puis votre navigateur par défaut va s'ouvrir à l'adresse : http://localhost:8080.



L'application Quasar de démarrage contient un menu et une page avec le logo du projet.

Vous pourrez remarquer que si vous ouvrez l'inspecteur du navigateur et passez en vue mobile, le menu s'adapte automatiquement. Vous pouvez également redimensionner la fenêtre de votre navigateur pour le constater.

Tous les composants Quasar sont prévus pour un rendu sur le web et sur mobile. Cela paraît évident de nos jours, mais vous allez voir dans les prochaines sections de cet article qu'il y a quelques bonnes surprises qui vont encore plus vous faire aimer ce *framework*.

La commande "quasar dev" exécute l'application quasar en mode développement.

Pour créer l'application de production, il suffit d'exécuter la commande :

> npx quasar build

```
bash-3.1$ npx quasar dev

Dev mode..... spa
Pkg quasar..... v1.12.8
Pkg @quasar/app... v2.0.1
Debugging..... enabled

Configured browser support (at least 84.59% of global marketshare):
· Chrome for Android >= 81
· Firefox for Android >= 68
· Android >= 81
· Chrome >= 73
· Edge >= 79
· Firefox >= 68
· iOS >= 10.0-10.2
· Opera >= 64
· Safari >= 10.1

App · Reading quasar.conf.js
App · Checking listening address availability (0.0.0.0:8080)...
App · Transpiling JS (Babel active)
App · Extending SPA Webpack config
App · Generating Webpack entry point
App · Booting up...

• Compiling:
└── SPA                    100% done in 5801 ms
```

Cette commande va optimiser et construire l'application dans le répertoire dist/spa.

Lancer l'application mobile native iOS / Android

À présent, vous êtes à deux commandes d'avoir exactement la même application, mais sous forme d'application native pour iOS et Android :

- Pour Android : vous devez avoir Android Studio et l'Android SDK.
- Pour iOS : vous devez avoir Xcode.

Quasar laisse le choix entre Cordova et Capacitor pour générer les applications mobiles. Nous utiliserons Capacitor, car il est plus récent.

Pour ajouter capacitor au projet :

> npx quasar mode add capacitor

Vous n'avez plus qu'à compiler le projet avec la commande :

> npx quasar dev -m capacitor -T android

Dans la commande ci-dessus, remplacez "android" par "ios" pour compiler pour iOS. Cette commande va lancer Android Studio qui va configurer le projet en utilisant Gradle. Avant de cliquer sur le bouton "Run app" d'Android Studio, accédez au menu "Refactor" et choisissez "Migrate to Android X". Cliquez sur le bouton "Do refactor" pour valider les modifications, puis cliquez sur "Run app".

Une nouvelle application appelée "Quasar App" sera ajoutée sur votre appareil.

En mode développement, l'application mobile est liée à votre poste de travail qui sert de serveur à l'application. Lorsque vous faites des modifications dans le code, elles sont automatiquement visibles dans l'application mobile via le module HMR (Hot Module Reload) de Quasar.

Quasar est particulièrement efficace lorsqu'il s'agit de tirer parti de son module HMR. Cela rend le développement mobile aussi rapide et léger que le développement web.

Je vous invite à essayer en modifiant le contenu de la balise <q-toolbar-title> du fichier src/layouts/MainLayout.vue pour en être convaincu.

De même que pour le web, vous pouvez construire l'application mobile de production avec la commande :

> npx quasar build -m capacitor -T android

Cette commande va créer une application mobile native autonome que vous pourrez publier sur l'App Store.

Et maintenant ?

Nous avons pu voir qu'il est très rapide de démarrer un nouveau projet grâce à la CLI. Entrons à présent dans le détail, en abordant les différents aspects

du *framework* qui vont vous permettre de développer votre application sereinement et rapidement.

Le fichier de configuration unique : quasar.conf.js

Avant de s'intéresser aux styles et composants, il est important de regarder le fichier de configuration *quasar.conf.js* qui va déterminer toutes les dépendances de votre application.

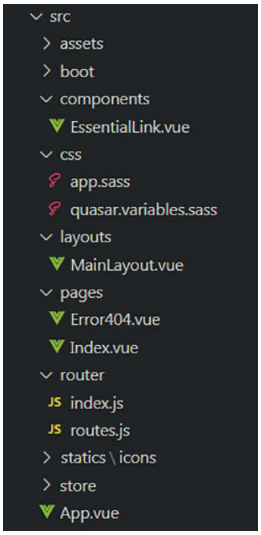
Il est structuré en plusieurs sections pour configurer :

- Les modules à charger au *boot* (noeud "boot") ;
- Le préprocesseur CSS (noeud "css") ;
- Les bibliothèques d'icônes (noeud "extras") ;
- La stratégie d'import des composants et la liste des plugins de Quasar (noeud "framework") ;
- Les variables d'environnements (noeud "build") ;
- Les *plugins* Webpack (nœud "build", fonction "extendWebpack") ;

La configuration par type d'application (noeud "pwa", etc.).

Ce fichier va vous permettre de configurer l'application Quasar et de l'étendre si vous avez besoin de mettre en place un module tiers, un *plugin* Webpack, etc.

L'arborescence des sources



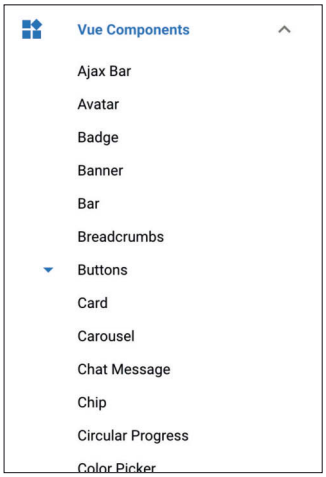
Pour démarrer, voici ce qu'il faut repérer au niveau des sources du projet :

- src/router/routes.js : il s'agit du routeur de votre application qui va faire correspondre une route (*url*) à un *layout* et une page ;
- src/pages : contient les pages de votre application ;
- src/components : contient les composants Vue de votre application. Ce sont les composants que vous allez utiliser dans vos pages ;
- src/layouts/MainLayout.vue : ce fichier correspond à la mise en page principale de votre application. Vous pouvez créer d'autres *layouts* dans ce répertoire et utiliser le routeur pour spécifier quel est le *layout* utilisé pour chaque page ;
- src/css/app.sass : c'est la feuille de style globale de l'application.

Par défaut, Quasar ne propose pas de répertoire "services" ou "helpers" pour ranger le code métier. C'est à vous de le créer. En ce qui me concerne, j'ai ajouté pour cela un répertoire "services" dans le répertoire "src".

La bibliothèque de composants

Une bibliothèque très complète



Quasar propose en standard plus de 70 composants : avatar, boutons, badges, fil de discussion, *color picker*, *infinite scroll*, tableaux, parallax, etc. Vous retrouverez tous les composants sur cette page : https://quasar.dev/vue-components/.

Chaque composant est un composant Vue : il présente des propriétés (*props*) pour le paramétrer et des événements pour le faire réagir. Ils sont reconnaissables facilement dans le code : ils commencent toujours par la lettre "q" de Quasar. Lorsque vous incluez le composant dans votre page, la balise du composant commence par "q-" et s'écrit en kebab case.

Exemple du composant Button

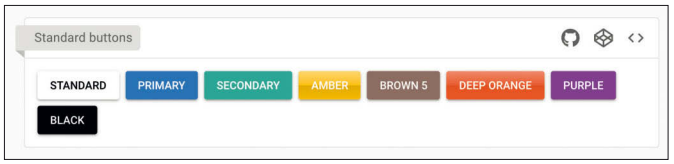
Prenons l'exemple d'un composant simple : le composant q-btn qui permet d'afficher un bouton.

Il s'inclut comme ceci dans un composant parent :

```
<q-btn color="primary" label="Primary" />
```

Source : https://github.com/GildasMorel/debuter-avec-quasar-framework/blob/q-btn/src/pages/Index.vue

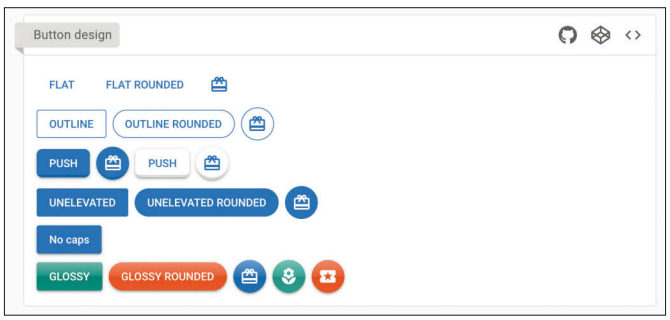
Vous obtiendrez le second bouton visible dans les boutons standards de la documentation :

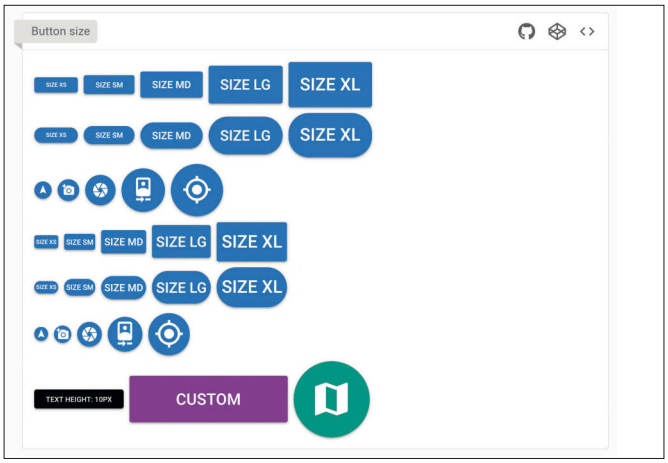


Pour voir le code permettant d'obtenir tous les boutons visibles sur cette image, cliquez sur l'icône "< >" en haut à droite du cadre.

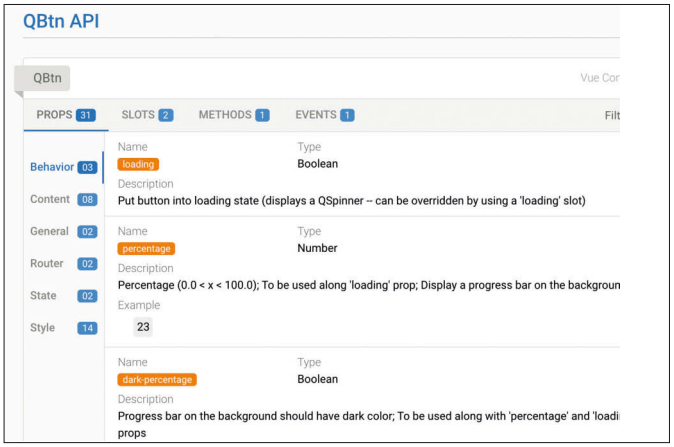
Cette icône est disponible pour chaque exemple de chaque composant et vous donne accès aux onglets *TEMPLATE* et *SCRIPT* afin de copier / coller le code html et JavaScript nécessaire pour reproduire l'exemple sur votre projet.

Sur la page du composant, vous retrouverez également les exemples de personnalisation : avec ou sans icône, différentes tailles, etc.





En bas de chaque page de composant, vous trouverez une section "API" précisant toutes les *props*, événements et méthodes permettant de le configurer.



Focus sur certains composants

Après vous avoir montré comment mettre en place un composant et connaître son API avec un composant basique qui est le bouton, j'ai choisi de vous présenter :

- Un composant classique : le tableau ;
- Un composant qui bénéficie de la réactivité de Vue : le carrousel ;
- Un composant qui met en valeur l'aspect multiplateforme intégré de Quasar : l'*uploader* ;
- La combinaison de deux composants : *infinite scroll* et *pull to refresh*.

Le composant table

Ce composant est un classique pour afficher un tableau de données.
Pour l'utiliser dans votre application Quasar, ajoutez le code suivant dans la partie <template> du composant parent :

```
<q-table
    title="Fichiers"
    :data="data"
    :columns="columns"
    row-key="filename"
/>
```

Source : https://github.com/GildasMorel/debuter-avec-quasar-framework/blob/q-table/src/pages/Index.vue

Puis, tout se joue dans la partie <script> où vous devez déclarer les colonnes (*columns*) et les données (*data*) sous forme de tableaux d'objets.
Exemple d'un tableau comportant deux colonnes :

```
import { format } from 'quasar';
const { humanStorageSize } = format;

export default {
    data () {
        return {
            columns: [
                {
                    name: 'filename',
                    label: 'Nom du fichier',
                    align: 'left', // 'right' par défaut
                    field: row => row.filename,
                    sortable: true
                },
                {
                    name: 'storageSize',
                    label: 'Poids du fichier',
                    field: row => row.storageSize,
                    format: val => val ? humanStorageSize(val) : val,
                    sortable: true
                }
            ],
            data: [
                {
                    filename: 'readme.md',
                    storageSize: 15000
                },
                {
                    filename: 'demo.avi',
                    storageSize: 237000000
                }
            ]
        };
    }
}
```

Source : https://github.com/GildasMorel/debuter-avec-quasar-framework/blob/q-table/src/pages/Index.vue

Vous obtiendrez ceci :

Quasar se charge d'ajouter par défaut la pagination et de mettre en place les curseurs permettant de trier sur les colonnes qui ont la propriété *sortable* à *true*.
Pour la colonne "Poids du fichier", la propriété *format* permet de mettre en forme la valeur définie dans la propriété *field*.
La fonction *humanStorageSize()*, fournie par Quasar, permet de représenter une valeur en KB, MB ou GB.
Notez que le tri fonctionne correctement sur cette colonne, car il s'effectue sur la valeur de la propriété *field*.
Vous retrouverez tous les paramètres de personnalisation du composant *table* sur cette page : https://quasar.dev/vue-components/table.
Tout est configurable : la pagination, les colonnes invisibles, l'en-tête, les états de chargement, les *body slots*, l'export CSV, la navigation au clavier et les *popups* d'édition de champs.

Le composant carrousel

Pour afficher des informations et notamment des images, le composant *carrousel* est un très bon choix. Voyons comment l'utiliser pour afficher rapidement une petite galerie d'images avec un bouton pour les voir en plein écran et des vignettes pour la navigation.

Dans la section <script> de votre composant, ajoutez :

```
export default {
    data () {
        return {
            slide: 1,
            fullscreen: false
        }
    }
}
```

Source : https://github.com/GildasMorel/debuter-avec-quasar-framework/blob/q-carousel/src/pages/Index.vue

Enfin, dans la section <template>, ajoutez :

```
<q-carousel
    swipeable
    animated
    arrows
    thumbnails
    infinite
    v-model="slide"
    :fullscreen.sync="fullscreen"
    :style="`width: ${fullscreen ? '100vw' : '60vw'}`"
>
    <q-carousel-slide :name="1" img-src="https://cdn.quasar.dev/img/mountains.jpg" />
    <q-carousel-slide :name="2" img-src="https://cdn.quasar.dev/img/parallax1.jpg" />
    <q-carousel-slide :name="3" img-src="https://cdn.quasar.dev/img/parallax2.jpg" />
    <q-carousel-slide :name="4" img-src="https://cdn.quasar.dev/img/quasar.jpg" />

    <template v-slot:control>
        <q-carousel-control
            position="bottom-right"
            :offset="[18, 18]"
        >
            <q-btn
                push round dense color="white" text-color="primary"
                :icon="fullscreen ? 'fullscreen_exit' : 'fullscreen'"
                @click="fullscreen = !fullscreen"
            />
        </q-carousel-control>
    </template>
</q-carousel>
```

Source : https://github.com/GildasMorel/debuter-avec-quasar-framework/blob/q-carousel/src/pages/Index.vue

Voici ce que vous obtiendrez :

Comme vous pouvez le constater, l'essentiel du code est dans la partie template. On notera :

- *swipeable* : permet de définir que le carrousel peut être manipulé par un geste de la souris ou du doigt sur mobile.
- *animated* : permet de déclencher une animation en cas de changement d'image. Par défaut l'animation est *fade*, elle peut être remplacée par un *slide-right* ou *slide-down*.
- *thumbnails* : permet d'afficher les vignettes. Au clic sur une vignette, le carrousel s'anime jusqu'au *slide* correspondant.
- *infinite* : permet de revenir à la première image lorsque l'on clique sur "suivant" sur la dernière image, et inversement.
- Les balises *q-carousel-slide* permettent de définir les différents *slides* à afficher. Si vous récupérez dynamiquement le contenu du carrousel, vous pouvez utiliser la directive *v-for* pour boucler sur les données à afficher.
- Le *template v-slot:control* permet de définir le bouton permettant de passer en plein écran. Au clic sur le bouton, la propriété *fullscreen* change de valeur. Dans les attributs du carrousel, *fullscreen.sync* est une propriété réactive qui est liée à la valeur de la propriété *fullscreen*. Le carrousel passe donc automatiquement en plein écran dès que *fullscreen* passe à *true*.
- Pour les besoins du test, la propriété *style* est présente pour définir la largeur du carrousel. Elle a été rendue réactive grâce aux ":" présents en préfixe. Elle réagit à la valeur de la propriété *fullscreen* de façon à régler différemment la largeur du carrousel s'il est en plein écran ou non. La même technique peut être utilisée, par exemple, pour afficher deux résolutions d'images différentes en fonction de l'affichage : vignette ou plein écran.

Vous retrouverez toutes les options de personnalisation du composant carrousel sur la page du composant : https://quasar.dev/vue-components/carousel

Le composant uploader

Dans le cadre d'une application de souvenirs que je suis en train de développer, j'avais besoin d'implémenter un formulaire avec un *drag'n drop* de médias (photos et vidéos). Mon application étant à destination du web et du mobile, mon premier réflexe a été d'identifier des outils permettant de répondre à ce besoin :

- Dropzone.js : pour la gestion du *drag'n drop* et de l'*upload* multiple pour la version web ;
- Un plugin Cordova pour récupérer un média de la galerie du téléphone et implémenter une barre de chargement pour le suivi de l'*upload*.

Après avoir implémenté Dropzone.js dans mon application Quasar via le composant vue-dropzone et la possibilité d'ajouter des librairies externes, j'ai eu besoin de trouver une meilleure façon de représenter l'*upload* des fichiers, car celle de Dropzone.js ne me convenait pas.

Et c'est là que j'ai découvert le composant *uploader* en parcourant la liste des composants Quasar.

Ce composant fournit exactement les fonctionnalités que je recherchais : *drag'n drop* de fichiers, suivi des *uploads* et surtout la sélection des fichiers sur le disque dur si on l'utilise via un navigateur et la sélection dans la galerie du téléphone si on l'utilise sur mobile.

Nul besoin d'implémenter une librairie externe ni de gérer la spécificité mobile iOS / Android. Le composant s'adapte automatiquement à son contexte d'exécution et propose une expérience utilisateur simple et performante.

Et bien évidemment, votre projet contient de fait moins de dépendances et de code grâce à cela.

Pour utiliser QUploader dans votre application, ajoutez dans la section <template> :

```
<q-uploader
    class="q-ma-lg"
    label="Photos / Vidéos / Audio"
    method="put"
    :factory="getSignedUrl"
    multiple send-raw
    hide-upload-btn
    flat bordered
    accept="image/*, video/*, audio/*"
/>
```

Source : https://github.com/GildasMorel/debuter-avec-quasar-framework/blob/q-uploader/src/pages/Index.vue

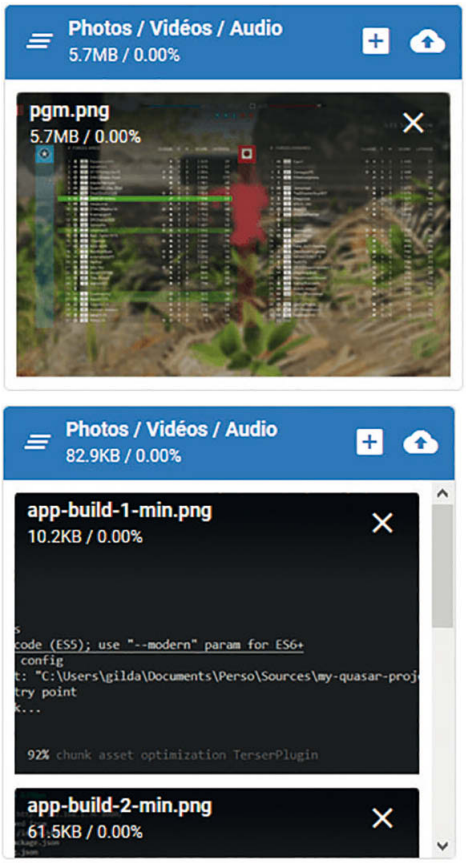
Dans sa version basique, vous n'avez pas besoin de spécifier quoi que ce soit dans la section <script> de votre composant. En ce qui me concerne, j'envoie les médias sélectionnés sur la Google Cloud Platform, j'avais donc besoin de générer une *url* signée dédiée à chaque transfert de fichier. C'est ce que j'ai fait en utilisant la propriété *factory* de q-uploader, qui appelle ici la méthode *getSignedUrl* de mon composant à chaque transfert de fichier, pour récupérer l'*url* cible.

Parmi les autres attributs utilisés ci-dessus :

- *method* : permet de spécifier le verbe de la requête qui envoie chaque fichier (par défaut c'est *post*) ;
- *multiple* : permet de spécifier que l'on souhaite transférer plusieurs fichiers ;
- *send-raw* : envoie le contenu brut des fichiers, sans en-têtes ;
- *accept* : permet de filtrer les types de fichiers au moment de la sélection sur l'appareil ;
- *flat* et *bordered* : permettent d'ajuster le style du composant.

L'action de glisser / déposer des fichiers sur le composant est activée par défaut.

Voici le rendu du composant :



Il inclut l'affichage du poids de chaque fichier, un bouton d'ajout de nouveaux médias, un bouton d'*upload* et un bouton pour supprimer un fichier.
En termes de design, il peut être retravaillé pour afficher les médias sous forme de grille en utilisant un composant q-card.
Vous retrouverez toutes les options de personnalisation de q-uploader sur la page du composant : https://quasar.dev/vue-components/uploader.

Combiner les composants infinite scroll et pull to refresh

Voici les deux derniers composants que je souhaitais vous présenter : *Infinite Scroll* et *Pull to refresh*. Le premier permet, lors de l'affichage d'une liste, de gérer l'apparition des éléments suivants lorsque l'on arrive en bas de la liste. Le second, de rafraîchir la liste lorsque l'on effectue une action de *scroll* depuis le haut de l'écran, comme si on souhaitait remonter en haut de la liste. De nos jours, ces deux comportements sont des standards que l'on retrouve dans quasiment toutes les applications.
Leur implémentation avec Quasar est très simple.
Dans la section <template>, ajoutez :

```
<q-infinite-scroll @load="infiniteScroll" :offset="nbItemsPerPage" class="q-mt-xl">
    <q-pull-to-refresh @refresh="pullToRefresh">
        <p v-for="block in blocks" :key="block.id">{{ block.message }}</p>
    </q-pull-to-refresh>
    <template v-slot:loading>
        <div class="row justify-center q-my-md">
            <q-spinner-dots color="primary" size="40px" />
        </div>
    </template>
</q-infinite-scroll>
```

Source : https://github.com/GildasMorel/debuter-avec-quasar-framework/blob/q-infinite-scroll-and-q-pull-to-refresh/src/pages/Index.vue

Ce code définit un *infinite scroll* avec une icône de chargement (composant q-spinner-dots) qui s'affiche pendant le chargement des éléments suivants. Et au sein de l'*infinite scroll*, il ajoute la fonctionnalité de "tirer pour rafraîchir" sur la liste des blocs grâce au composant q-pull-to-refresh. Pour remplir la liste et rendre tout cela dynamique, ajoutez dans la section <script> :

```
export default {
    data() {
        return {
            nbItemsPerPage: 50,
            blocks: []
        }
    },
    created() {
        this.blocks = [
            ...this.getNewBlocks(0, this.nbItemsPerPage)
        ];
    },
    methods: {
        getNewBlocks(cursor, nb) {
            const newBlocks = [];
            for (let i = cursor + 1; i <= cursor + nb; i++) {
                newBlocks.push({ id: i, message: `Line ${i}` });
            }

            return newBlocks;
        },
        pullToRefresh(done) {
            setTimeout(() => {
                this.blocks = this.blocks.splice(0, this.nbItemsPerPage);
                done();
            }, 1500);
        },
        infiniteScroll(index, done) {
            setTimeout(() => {
                this.blocks = [
                    ...this.blocks,
                    ...this.getNewBlocks(this.blocks.length, this.nbItemsPerPage
)               ];
                done();
            }, 1500);
        }
    }
}
```

Source : https://github.com/GildasMorel/debuter-avec-quasar-framework/blob/q-infinite-scroll-and-q-pull-to-refresh/src/pages/Index.vue

La liste des blocs comporte 50 blocs par page et à chaque déclenchement de l'*infinite scroll*, 50 nouveaux éléments sont ajoutés dans la liste. L'utilisation de *setTimeout* permet de simuler un temps d'attente de manière à voir apparaître l'icône de chargement.
L'appel à la fonction "done()" fournie par Quasar permet de déclencher la disparition de l'indicateur de chargement (composant q-spinner-dots) et l'affichage des 50 nouveaux blocs. L'utilisation de cette fonction est la bonne pratique en cas de traitement asynchrone afin de déclencher la fin du traitement après chargement des nouvelles données.
À partir de ce code, vous obtiendrez une liste affichant 50 éléments par page avec l'ajout de 50 nouveaux éléments chaque fois que vous approchez le bas de la liste.
Et la possibilité de réinitialiser les éléments en tirant le haut de la liste.
Vous retrouverez toutes les options de personnalisation de ces 2 composants sur leurs pages respectives :

Infinite scroll : https://quasar.dev/vue-components/infinite-scroll
Pull to refesh : https://quasar.dev/vue-components/pull-to-refresh

Le layout et les styles

Maintenant que nous avons vu comment initialiser un projet et y ajouter des composants, nous allons voir comment utiliser les *layouts* et les styles proposés par Quasar pour pouvoir maîtriser la mise en page de votre application.

Vue d'ensemble

Quasar utilise Flexbox et un ensemble de classes CSS pour sa gestion du *layout*. La documentation est très complète : https://quasar.dev/layout
Vous y retrouverez toutes les explications dont vous avez besoin ainsi qu'un *layout builder* pour vous aider à configurer la mise en page de votre application.
Vous y trouverez également une *layout gallery* proposant des mises en page préconfigurées, dont :

- YouTube ;

• Google Play ;
• Google Photos ;
• Whatsapp.

Cela peut être un excellent point de départ et un gain de temps significatif pour débuter. Enfin, Quasar propose un ensemble de classes CSS permettant de modifier les styles :

• Tailles de police : https://quasar.dev/style/typography ;
• Palette de couleurs : https://quasar.dev/style/color-palette ;
• Positionnement : https://quasar.dev/style/positioning ;
• Etc.

Pour en savoir plus, visitez la section *Style & Identity* de la documentation : https://quasar.dev/style

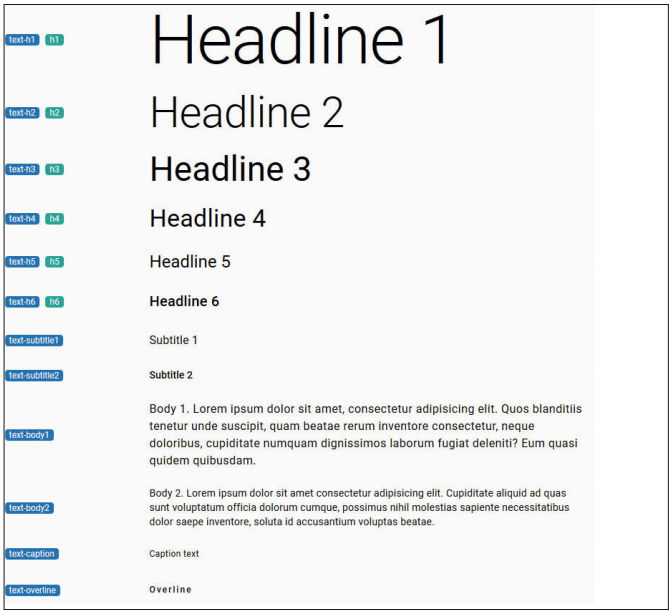
Typographie et couleurs

Quasar applique automatiquement un style sur les balises titre h1 à h6 et propose ces styles pour une utilisation sur les autres balises, sous la forme "text-hX".

Par exemple, les deux lignes suivantes affichent la même taille de texte :

```
<h1>Titre de la page</h1>
<p class="text-h1">Titre de la page</p>
```

Pour compléter les styles de titres, vous trouverez des styles de sous-titres et de *body*.



Vous retrouverez, sur la page *Typography* de la documentation, comment changer la graisse, ajouter une typo personnalisée et utiliser les *helpers* CSS disponibles.

Pour coloriser le texte, choisissez une couleur dans la palette de couleur et appliquez la classe sur votre élément. Préfixez avec "text-" pour coloriser le texte et/ou "bg-" pour coloriser le fond.

Par exemple :

```
<!-- Afficher un paragraphe avec la taille de texte d'un titre h1
et la couleur orange -->
<p class="text-h1 text-orange-8">Titre de la page</p>

<!-- Afficher un paragraphe avec la taille de texte d'un titre h5
et un fond bleu -->
<p class="text-h5 bg-light-blue-2">Sous-titre</p>
```

Margins et paddings

Sur le même principe que la plupart des framework CSS (Bootstrap, …), Quasar propose plusieurs mots-clés pour définir les tailles : xs, sm, md, lg et xl. Pour régler les *margins* et *paddings* d'un bloc il faut combiner ces tailles avec un type et une direction, comme ceci :

q-[type][direction]-[taille]

Le type est "p" pour *padding* ou"m" pour *margin*.

Les valeurs pour la direction sont :

• t (*top*), r (*right*), b (*bottom*), l(*left*)
• a (*all*), x (*left and right*), y (*top and bottom*)

Dans la pratique, cela donne :

```
<!-- Margin small dans toutes les directions -->
<div class="q-ma-sm"></div>

<!-- Margin extra small en haut -->
<div class="q-mt-xs"></div>

<!-- Padding medium à gauche et à droite -->
<div class="q-px-md"></div>
```

Pour aller plus loin dans la gestion des styles de Quasar avec notamment les variables Sass / SCSS / Stylus, les *breakpoints* et autres classes de *body*, reportez-vous à la documentation des styles.

Les icônes

Quasar utilise par défaut les *Material Icons*. Il est tout à fait possible d'utiliser d'autres bibliothèques d'icônes (Font Awesome, Ionicons, Themify, etc.) ou d'en utiliser plusieurs en même temps. Il suffit de préciser la bibliothèque d'icônes à utiliser dans le fichier de configuration quasar.conf.js, dans la section "extras".

Par défaut : extras: ['material-icons']

De base, les composants Quasar référencent des icônes de la bibliothèque *Material*. Si vous souhaitez utiliser uniquement Font Awesome et désactiver Material, pensez à modifier les icônes qui viennent avec le code des composants pour conserver un affichage correct.

Exemples d'affichage d'une icône :

```
<!-- Le plus simple -->
<q-icon name="accessibility" />
<!-- Avec un modificateur de taille -->
<q-icon name="accessibility" size="md"/>
<!-- Avec une marge medium de tous les côtés et une couleur -->
<q-icon name="accessibility" size="md" color="positive" class="q-mt-md"/>
```

Voici le résultat :

Conclusion

Il resterait encore tant à dire sur Quasar. Nous n'avons pas abordé les *helpers* ni parlé des *plugins*, des *utils*, du SSR, de l'intégration d'une librairie externe, des tests unitaires et *end to end*, des fonctionnalités spécifiques aux PWA (*Progressive web Apps*), de la génération d'applications *desktop* avec Electron ou d'extensions navigateurs (*Browser extensions* — BEX). Il faudrait continuer cet article en y ajoutant de nouvelles parties, mais vous savez quoi ? Le mieux est directement de se lancer et d'apprendre au fur et à mesure de vos besoins. J'espère que cette mise en jambe vous aura donné envie de donner une chance à Quasar.

Pour finir, sachez que le 5 juillet 2020 a eu lieu la première conférence dédiée à ce framework : Quasar.conf.

Vous pouvez retrouver le *relpay* de la conférence sur YouTube : https://www.youtube.com/watch?v=6ZKBZ3k4Ebk

Franck Dubois
Video Game Codeur
Développeur agile
Formateur JavaScript / Unity / GDevelop

Astar, un algorithme pour le jeu vidéo

A star (A*) est un algorithme de recherche de chemin dans un graphe datant de 1968. Un graphe est un ensemble de nœuds dont certains sont reliés entre eux. Il trouve son emploi dans de multiples domaines, notamment dans la recherche de chemins entre 2 nœuds.

A* a l'excellence de trouver une solution quand il y en a une, et rapidement. Son emploi est donc un bon choix dans le cadre du jeu vidéo. Cependant, la solution trouvée ne sera pas toujours la meilleure. A* est dérivé de l'algorithme de Dijkstra (1959) qui est un algorithme de recherche du plus court chemin. Il est plus consommateur en ressources et en temps. Eternel compromis à trouver !

L'environnement d'expérimentation de A star (A*)

Pour tester cet algorithme, nous utilisons un labyrinthe en 2D que nous modélisons et affichons dans un canvas html5. Pour rendre cette modélisation simple et facilement modifiable, nous créons une structure de données comme ceci :

```
const labyrinth = {
 0: " C                    ",
 1: "                  #   ",
 2: "                  #   ",
 3: "                  ###  ",
 4: "                       ",
 5: "                       ",
 6: "                  #   ",
 7: "            #     #   ",
 8: " #          #     #   ,
 9: "    #               #  ,
 10: "                  #   ",
 11: "           ######      ",
 12: "           #     #     ",
 13: "           #  #P #     ",
 14: "              ####     ",
 15: "   #####      #        ",
 16: "      ##     ######## ",
 17: "      ##        #      ",
 18: "      ##        ###    ",
 19: "                       "
}
```

Dans cette structure, le caractère « # » symbolise un mur, « C » le poursuivant contrôlé par le programme et « P » le joueur poursuivi. Pour la démonstration lors de l'affichage du labyrinthe, les murs seront symbolisés par des carrés gris, le joueur par un carré jaune et le poursuivant par un carré rouge. Le chemin sera symbolisé par des carrés verts.
Nous créons donc une série de fonctions dédiées à cet affichage :

- ***createCanvasContext*** crée un canvas pour gérer l'affichage du labyrinthe ;
- ***clearCanvas*** efface un canvas ;
- ***initGameGrid*** initialise le labyrinthe à partir de la structure de données, affiche le labyrinthe avec les 2 protagonistes poursuivant et poursuivi ;
- ***showSquare*** affiche un carré à une position et une couleur données dans le labyrinthe ;
- ***showPlayer*** affiche le poursuivi (la fin du chemin) ;
- ***showComputer*** affiche le poursuivant ;
- ***showWall*** affiche un mur ;
- ***showPath*** affiche un bloc du chemin solution ;
- ***displayLabyrinth*** affiche le labyrinthe.

Nous embarquons toutes ces fonctions dans un fichier JavaScript dédié que nous appelons ***display.js***.
Que nous utilisons comme ci-dessous au sein d'un fichier html avec un canvas html5 encapsulé dans une balise « div » :

```
<html>
 <body>
  <div id="gameDiv" style="margin-left:auto;margin-right:auto;width:
1168px;height:740px;">
  </div>
 </body>

<script src="display.js"></script>
<script>
"use strict";
{
 const gameDiv = document.getElementById("gameDiv");
 const gameCanvasContext = createCanvasContext("game",200,200,1,game
Div,"#000000");

 const labyrinth = {
  0: " C                    ",
  1: "                  #   ",
  2: "                  #   ",
  3: "                  ###  ",
  4: "                       ",
  5: "                       ",
  6: "                  #   ",
  7: "            #     #   ",
  8: " #          #     #   ,
  9: "    #               #  ,
  10: "                  #   ",
  11: "           ######      ",
  12: "           #     #     ",
  13: "           #  #P #     ",
  14: "              ####     ",
```

```
 15: "    #####      #          ",
 16: "       ##      ######## ",
 17: "       ##         #       ",
 18: "       ##         ###     ",
 19: "                          "
};

// read game grid
const gridData = initGameGrid(labyrinth,gameCanvasContext);

// clear
clearCanvas("game",gameCanvasContext);

// show walls
displayLabyrinth(labyrinth,gameCanvasContext);

// show player
showPlayer(gameCanvasContext,gridData.player.x,gridData.player.y);

// show enemy
showComputer(gameCanvasContext,gridData.computer.x,gridData.computer.y);
}
</script>
</html>
```

Le labyrinthe s'affiche comme prévu, l'environnement d'expérimentation est en place. Que la partie commence !

Que faire avec l'algorithme A* ?

A* permet de déterminer un chemin entre 2 points A et B, qu'il y ait des obstacles ou pas. Pour aller de A vers B, il va falloir passer par une série de points intermédiaires, et c'est **A*** qui se charge de les trouver. En faisant en sorte que le chemin soit le plus court possible. On appelle nœuds, les points A et B ainsi que les points intermédiaires.

Les nœuds

Le chemin solution :

- commence à un nœud A : le point A, lieu où commence le chemin, là où est par exemple le poursuivant ;
- se termine à un nœud B : le point B, lieu où se termine le chemin, là où se trouve par exemple le poursuivi ;
- est une liste des nœuds par lesquels il faut passer pour aller de A ver B, A étant le point (nœud) départ et B étant le point (nœud) d'arrivée.

En dehors des murs, des points de départ et d'arrivée, tous les lieux de passage possibles (tous les autres points du labyrinthe) sont des nœuds.

Trois piliers constituent l'**algorithme** :

- une liste de nœuds dite ouverte ;
- une liste de nœuds dite fermée ;
- une fonction d'évaluation de la qualité d'un nœud.

La liste ouverte contient tous les nœuds candidats pouvant appartenir au chemin solution.

Cette liste se remplit au fur et à mesure de l'exploration du labyrinthe. Lorsqu'un nœud de la liste ouverte est considéré comme faisant partie du chemin à l'instant où il est examiné, il est basculé dans la liste fermée.

La liste fermée comporte tous les nœuds qui à un moment ou un autre ont fait partie de la solution. Un nœud se trouve donc dans la liste ouverte ou la liste fermée. Jamais dans les 2 à la fois. Un fois qu'un nœud a basculé dans la liste fermée, il n'en ressort jamais.

Evaluation de la qualité d'un nœud

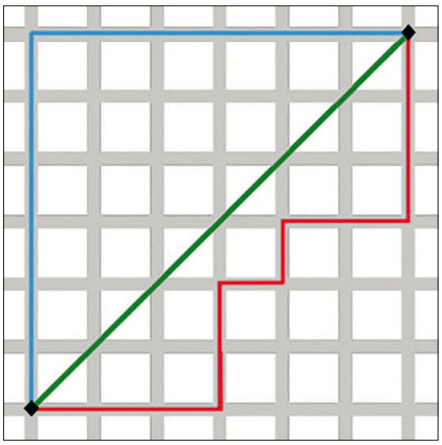
Pour évaluer la qualité d'un nœud, on utilise la somme de :

- la distance entre lui et le dernier nœud du chemin, le point d'arrivée ;
- et la distance entre lui et leur dernier nœud considéré comme valide pour le chemin.

Il y a plusieurs façons de faire pour calculer ces distances, entre autres :

- la distance à vol d'oiseau appelée aussi la **distance euclidienne** calculée sur la base du théorème de Pythagore ;
- la **distance de Manhattan** appelée aussi taxi-distance (la distance entre deux points parcourue par un taxi lorsqu'il se déplace dans une ville américaine où les rues sont agencées selon un réseau ou quadrillage).

Illustration avec en vert la **distance euclidienne**, en bleu, rouge la **distance de Manhattan**



Plus le calcul sera précis, meilleur sera le chemin trouvé. Alors pourquoi choisir la distance de Manhattan ? La **distance euclidienne** fait appel à des nombres à virgule flottante, ce qui rend le temps de calcul plus long qu'avec la **distance de Manhattan** qui n'utilise que des nombres entiers.

A l'échelle de cet exercice, la différence est imperceptible. Mais sur des parcours plus longs avec des calculs plus nombreux, cela peut devenir critique en temps d'exécution rendant le calcul par la **distance de Manhattan** plus pertinent.

L'algorithme A* en JavaScript

Pour nous faciliter un peu la tâche, nous ajoutons un objet utilitaire que nous appelons ***utils*** et que nous définissons aussi dans un fichier dédié ***util.js***.

Cet objet comporte 3 méthodes :

- ***equalJsonByValue*** compare 2 objets Json par leurs valeurs ;
- ***isNodeByValueInList*** indique en renvoyant un booléen si un nœud se trouve dans une liste ;
- ***getNodeFromList*** renvoie depuis une liste et un nœud passés en paramètres, un nœud de la liste dont le positionnement (propriétés x et y) dans le labyrinthe est identique à celui passé en paramètre.

Cette dernière méthode est appelée lorsqu'un nœud analysé est déjà dans la liste ouverte. Elle permet de renvoyer le

nœud de la liste ouverte avec sa qualité.
L'algorithme quant à lui prend forme par le biais de la classe ***Astar***.
Comme dit précédemment, notre environnement est composé :

- d'une position de départ (qui est un nœud [***nodeFrom***]) ;
- d'une position d'arrivée (qui est aussi un nœud [***nodeTo***]) ;
- du labyrinthe avec ses obstacles (composé aussi de nœuds).

Pour l'algorithme, nous utilisons :

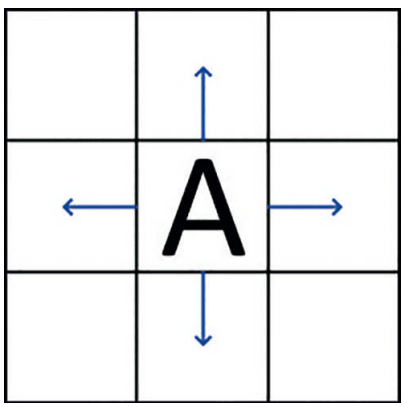
- une liste ouverte vide à l'initialisation (***this.openedList = new Array()***) ;
- une liste fermée vide à l'initialisation (***this.closedList = new Array()***) ;
- une fonction d'évaluation de la qualité d'un nœud (***setNodeQualityValue***).

Un nœud est déterminé par :

- sa position dans le labyrinthe exprimée avec l'abscisse et l'ordonnée, et prenant pour origine le point en haut à gauche du labyrinthe ;
- sa qualité ;
- son nœud parent.

```
{
 x : node.x,
 y : node.y,
 f : node.f,
 parent : node.parent
}
```

L'algorithme commence à partir de la position/nœud de départ, qu'il place d'emblée dans la liste fermée et de là sont déterminés les points/nœuds adjacents.



Ici, l'exploration se fait à la verticale et l'horizontale, mais rien n'interdit de le faire aussi sur les diagonales.
Quatre nœuds possibles, dont les coordonnées sont respectivement :

- 1er nœud à droite : même ordonnée que le point de départ, abscisse + 1 ;
- 2ème nœud au-dessus : même abscisse que le point de départ, ordonnée – 1 ;
- 3ème nœud à gauche : même ordonnée que le point de départ, abscisse – 1 ;
- 4ème nœud en dessous : même abscisse que le point de départ, ordonnée + 1.

Ensuite, vient l'analyse de chacun des 4 nœuds :

- s'il est un obstacle, il est oublié ;
- s'il appartient à la liste fermée, cela indique qu'il a déjà été analysé, il est oublié.

En dehors des 2 cas précédents :

- si le nœud examiné n'est pas dans la liste ouverte, on calcule sa qualité et on l'insère dans la liste ouverte ;
- si le nœud examiné est dans la liste ouverte, on calcule sa qualité, on la compare à la qualité du même nœud de la liste ouverte : si la qualité du nœud de la liste est moindre, il est remplacé par le nœud examiné.

C'est la méthode de ***listNextCellsAndPushToOpenedList*** qui se charge de cette partie.
Avec la méthode ***searchBestNodeInOpenedListAndPushItToClosedList***, on parcourt ensuite la liste ouverte pour en extraire le nœud ayant la meilleure qualité et le mettre dans la liste fermée. Si la liste ouverte est vide, il n'y a pas de chemin solution.
Si ce nœud correspond au nœud du point d'arrivée, le chemin solution est trouvé. Il suffit de remonter de ce nœud jusqu'au nœud de départ par le biais des nœuds parents. L'algorithme s'arrête là.
Sinon ce même nœud devient le nœud à partir duquel on reprend l'exploration à l'étape où sont déterminés les nœuds voisins à la verticale et à l'horizontale qui seront alors analysés à leur tour.
La méthode ***searchPath*** de la classe ***Astar*** renvoie le chemin sous la forme d'une liste de nœuds.
Il suffit de parcourir cette liste dans l'ordre pour afficher dans le labyrinthe le chemin trouvé ***aStar.path***.
L'utilisation de l'instruction ***setTimeout*** n'a pour objet que de montrer le parcours du chemin pas à pas.

```
const aStar = new Astar("#",labyrinth);
aStar.searchPath(gridData.computer,gridData.player);
let time = 100;
for (const element of aStar.path) {
  time += 100;
  setTimeout( () => { showPath(gameCanvasContext,element.x,element.y);} , time);
}
```

Le code de cet article est disponible sur www.programmez.com et notre Github.
Vous avez le loisir de tester différents labyrinthes et aussi les 2 méthodes de calcul de chemin que sont la distance euclidienne et la distance de Manhattan.
A vous de jouer maintenant !

PHP 8 : une version que tout le monde attendait

L'année 2020 est une année importante pour notre langage favori : les 25 ans de PHP et surtout la sortie de la version 8.0. Cette version était attendue avec impatience, car elle introduit de nombreuses évolutions.

Christophe Villeneuve

Consultant IT pour Atos, Mozilla Rep, auteur publié aux éditions Eyrolles et aux Éditions ENI, PHPère des elePHPants PHP, membre des Teams DrupalFR, AFUP, LeMug.fr (MySQL/MariaDB User Group FR), Lizard...

PHP 8 est sorti le 26 novembre 2020. Cette version introduit des changements de rupture, ainsi que de nombreuses nouvelles fonctionnalités et améliorations de performances.
En raison des profondes ruptures que la v8 implique, il y a plus de chances que vous deviez apporter des modifications plus ou moins importantes à vos codes / projets, avant de supporter et de déployer PHP 8. Ces évolutions rappellent la fin du support des versions précédentes que vous pouvez retrouver comme le montre l'image 1. **Figure 1**

La performance

Hourra ! PHP 8 se voit doter d'un compilateur juste à temps. C'est le fameux JIT que l'on trouve dans de nombreux langages interprétés.

JIT

Le compilateur JIT, ou just in time, est là pour compiler à la volée le code. Il permet d'améliorer les performances d'exécution et donc de vos codes en production. Comme vous le savez, PHP n'est pas un langage compilé, mais interprété. Le compilateur JIT compile des parties du code pendant l'exécution pour agir comme une version en cache du code. **Figure 2**
Pour bénéficier du JIT dans votre application ou projet web, les réglages s'effectuent du côté de l'administrateur serveur, car pour le développeur, c'est transparent, et il n'y a pas de lignes de codes supplémentaires à ajouter. Ouf ! Pour l'administrateur, il faut modifier le fichier de configuration de php.ini et au moins ces 2 lignes :

```
opcache.jit_buffer_size=100M
opcache.jit = CRTO
```

Pour mesurer le gain, il a été établi un tableau comparatif : **figure 3**
Code source :
https://gist.github.com/PedroEscudero/b64ea7409c0cc483c44f0773b6aebbdb
Le tableau montre un gain important quand JIT est actif ou pas. Bien entendu, si le compilateur est actif, vous pouvez déboguer votre code avec la fonction suivante :

```
$ gdb php
```

Source : https://wiki.php.net/rfc/jit

Les nouveautés

Trois grandes catégories rassemblent les nombreuses nouveautés de cette version

- Les méthodes ;
- Les fonctions ;
- La gestion des erreurs.

Les méthodes

Union types

Jusqu'à présent les « unions types » peuvent former une union avec 2 types. La nouvelle méthode « union type» accepte des valeurs de plusieurs types différents au lieu d'un seul.
Toutefois il existe des restrictions comme :

- Le type void ne pourrait pas faire partie d'une union, car void signifie qu'une fonction ne retourne aucune valeur.
- Les unions nulles peuvent être déclarées avec null ou ?

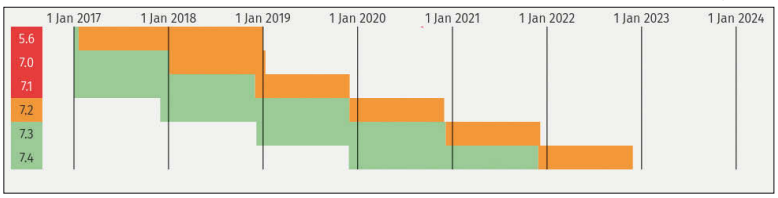
Au niveau du code, la propriété de classe peut se présenter pour manipuler les nombres, soit un entier, soit un nombre flottant :
* Avant PHP 8

```
class Number {

  /**
   * @var int|float $number
   */
  private $number;

 /**
  * @param int|float $number
  */
 public function setNumber($number) {
     $this->number = $number;
 }
```

Figure 1



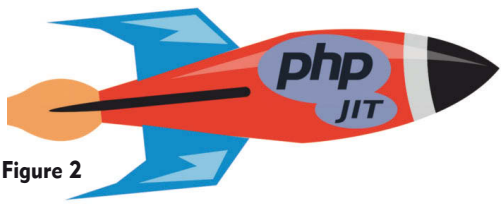


Figure 2

Figure 3

PHP-5.3	PHP-7.4	PHP-8	PHP-8 JIT
0.64574003219604	0.10253500938416	0.098223924636841	0,04458594322

Comparative of the average performance with 100 runs of the script

```
/**
 * @return int|float
 */
public function getNumber() {
    return $this->number;
}
}
```

* PHP 8

```
class Number {
  private int|float $number;

  public function setNumber(int|float $number): void {
    $this->number = $number;
  }

  public function getNumber(): int|float {
    return $this->number;
  }
}
```

Source : https://wiki.php.net/rfc/union_types_v2

Weak maps

Weak Maps est une extension de WeakRefs. Il permet de créer une map d'objets à des valeurs arbitraires (similaire à SplObjectStorage) sans empêcher les objets d'être utilisés comme clés de récupérations. L'intérêt est d'éviter les fuites de mémoire et que chaque instance de la classe reste ainsi itérable. Il améliore la fonction WeakRefs (disponible avec PHP 7.4) pour permettre de supprimer des objets lorsque seul le cache fait référence aux classes d'entités des objets.
Il permet de conserver une référence à un objet qui n'empêche pas la destruction de celui-ci. C'est utile lorsque seul le cache fait référence à cet objet. On économise des ressources lors de la manipulation des objets et d'augmenter la productivité en réduisant le temps d'attente.
Au niveau du code, nous créons une classe comme ceci :

```
class Foo
{
  private WeakMap $cache;

  public function getSomethingWithCaching(object $obj): object
  {
    return $this→cache[$obj] ??= $this->computeSomethingExpensive($obj);
  }
}
```

L'utilisation se fait en tableau :

```
$map = new WeakMap;    // objet instancé
$obj = new stdClass;
$map[$obj] = 42;
```

Nous vérifions le contenu de l'objet :

```
var_dump($map);

object(WeakMap)#1 (1) {
 [0]=>
 array(2) {
  ["key"]=>
   object(stdClass)#2 (0) {  }
  ["value"]=> int(42)
 }
}
```

Nous détruisons cette variable comme ceci :

```
unset($obj);// détruit variable
```

Nous vérifions le contenu qui montre un résultat vide :

```
var_dump($map);

Résultat :

object(WeakMap)#1 (0) {}
```

Source : https://wiki.php.net/rfc/weak_maps

Type «static» pour les résultats

Le nom de classe statique fait référence à la classe sur laquelle une méthode a été réellement appelée, même si la méthode est héritée. Il existe un certain nombre de cas d'utilisation typiques ou des types de retours statiques. Avant, les types retournaient 'self' ou 'static'. Avec PHP 8, le type static devient un type de retour valide. A partir d'une classe A, que nous étendrons sous le nom B, nous effectuerons différents appels

```
<?php
class A {
  public static function getSelf(): self
  {
    return new self();
  }

  public static function getStatic(): static
  {
    return new static();
  }
}

class B extends A {}
```

Résultat :

```
get_class(B::getSelf());  // A
get_class(B::getStatic()); // B
get_class(A::getSelf());  // A
get_class(A::getStatic()); // A
```

Source : https://wiki.php.net/rfc/static_return_type

Mixed type

A partir de la version 7 de PHP, le type mixte a bénéficié des types scalar, nullables, objet. Avec la version 8 de PHP, la valeur peut être n'importe quelle valeur dont le type union (voir plus haut). Ainsi, les développeurs peuvent déclarer des informations de type :

- Une fonction ne renvoie rien ou null ;
- Nous attendons un de plusieurs types ;
- ≠Nous attendons un type qui ne peut pas être indiqué en PHP.

Ainsi, le type mixte signifie l'un de ces types :

- array
- bool
- callable

- int
- float
- null
- object
- resource
- string

Il peut être utilisé comme paramètre ou type de propriété, pas seulement comme type de retour. Comme le type mixte inclut null, il n'est pas permis de le rendre nullable, et provoquera une erreur comme ceci :

```
function bar(): ?mixed {}
```

Source : https://wiki.php.net/rfc/mixed_type_v2

Opérateur nullsafe

Si vous connaissez l'opérateur de fusion nul, vous connaissez déjà ses inconvénients : il ne fonctionne pas sur les appels de méthode. Au lieu de cela, vous avez besoin de vérifications intermédiaires ou comptez sur des aides facultatives fournies par certains frameworks :

```
$startDate = $booking->getStartDate();

$dateAsString = $startDate ? $startDate->asDateTimeString() : null;
```

Avec l'ajout de l'opérateur nullsafe, nous pouvons maintenant avoir un comportement de type null sur les méthodes!

```
$dateAsString = $booking->getStartDate()?->asDateTimeString();
```

Source : https://wiki.php.net/rfc/nullsafe_operator

Attributes

Les attributs (ou appelés annotations), correspondent à une forme de métadonnées structurées. Ils sont utilisés pour spécifier les propriétés des objets, des éléments ou des fichiers. A partir de PHP 8, vous pouvez introduire des attributs pour PHP en les définissant comme une forme de métadonnées, sans avoir à analyser les docblocks. Vous pouvez les ajouter aux déclarations de classes, propriétés, fonctions, méthodes, paramètres et constantes. Au niveau du code, cela s'utiliserait comme ceci :

```
<<ExampleAttribute>>
class Foo
{
    <<ExampleAttribute>>
    public const FOO = 'foo';

    <<ExampleAttribute>>
    public $x;

    <<ExampleAttribute>>
    public function foo(<<ExampleAttribute>> $bar) { }
}

$object = new <<ExampleAttribute>> class () { };

<<ExampleAttribute>>
function f1() { }

$f2 = <<ExampleAttribute>> function () { };

$f3 = <<ExampleAttribute>> fn () => 1;
```

Les attributs sont ajoutés avant la déclaration à laquelle ils appartiennent, comme les commentaires d'un bloc doc. Ils peuvent être déclarés avant ou après un commentaire de bloc doc qui documente une déclaration.

```
<<ExampleAttribute>>
/** docblock */
<<AnotherExampleAttribute>>
function foo() {}
```

Source : https://wiki.php.net/rfc/attributes_v2

Constructor property promotion

L'ergonomie des objets en PHP a été simplifiée. La syntaxe est plus concise. Elle simplifiera la déclaration de propriété, la rendant plus courte et moins redondante. La construction concerne les paramètres promus (promoted parameters) au niveau de la méthode préfixée par des mots-clés de visibilité publique, protégée et privée. Cela se traduit qu'au lieu de spécifier des propriétés de classe et un constructeur, PHP peut combiner en une seule :
* Avant

```
class Money
{
    public Currency $currency;

    public int $amount;

    public function __construct(
        Currency $currency,
        int $amount,
    ) {
        $this->currency = $currency;
        $this->amount = $amount;
    }
}
```

* PHP 8:

```
class Money
{
    public function __construct(
        public Currency $currency,
        public int $amount,
    ) {}
}
```

Source : https://wiki.php.net/rfc/constructor_promotion

Héritage avec des méthodes privées

Auparavant, PHP appliquait les mêmes contrôles d'héritage sur les méthodes publiques, protégées et privées. En d'autres termes : les méthodes privées doivent suivre les mêmes règles de signature de méthode que les méthodes protégées et publiques. Toutefois, les méthodes privées ne seront pas accessibles par les classes enfants. PHP 8 a changé ce comportement, de sorte que ces vérifications d'héritage ne sont plus effectuées sur les méthodes privées.
Par ailleurs, l'utilisation de la fonction privée déclenchera maintenant un avertissement, comme ceci :

```
Warning: Private methods cannot be final as they are never overridden
```

```
by other classes
```

Source : https://wiki.php.net/rfc/inheritance_private_methods

Autoriser la virgule de fin

Lorsque vous appelez une fonction, vous pouvez être confronté au problème de la virgule de fin dans la liste des paramètres. PHP 8 autorise la possibilité d'avoir une virgule à la fin, comme ceci :

```
public function(
  string $parameterA,
  int $parameterB,
  Foo $objectfoo,
) {
  // …
}
```

Source : https://wiki.php.net/rfc/trailing_comma_in_parameter_list

Ajout interface Stringable

PHP 8 introduit une nouvelle interface Stringable qui est automatiquement ajoutée aux classes et implémente la méthode __toString (). Chaque fois qu'une classe implémente __toString(), elle implémente automatiquement l'interface en arrière-plan et il n'est pas nécessaire de l'implémenter manuellement. L'opération se programme de la manière suivante :

```
class Foo
{
  public function __toString(): string
  {
    return 'foo';
  }
}

function bar(string|Stringable $stringable) { /* … */ }

bar(new Foo());
bar('abc');
```

Source : https://wiki.php.net/rfc/stringable

Méthode abstraction des traits

Les traits peuvent spécifier des méthodes abstraites qui doivent être implémentées par les classes qui les utilisent. Avant PHP 8, la signature de ces implémentations de méthodes n'était pas validée, et se traduit en code comme ceci :

```
trait Test {
  abstract public function test(int $input): int;
}

class UsesTrait
{
  use Test;

  public function test($input)
  {
    return $input;
  }
}
```

PHP 8 effectuera une validation de signature de méthode appropriée lors de l'utilisation d'un trait et de l'implémentation de ses méthodes abstraites, pour obtenir ceci :

```
class UsesTrait
{
  use Test;

  public function test(int $input): int
  {
    return $input;
  }
}
```

Source : https://wiki.php.net/rfc/abstract_trait_method_validation

Les fonctions

PHP 8 voit arriver de nouvelles fonctions avec des améliorations pour certaines d'entre elles, que nous allons voir.

Match()

L'expression de correspondance ajoute une nouvelle expression (proche de la fonction switch) avec une sémantique plus sûre et la possibilité de renvoyer des valeurs. Cela se traduit dans le code :
* Avant :

```
switch ($statusCode) {
 case 200:
 case 300:
   $message = null;
   break;
 case 400:
   $message = 'not found';
   break;
 case 500:
   $message = 'server error';
   break;
 default:
   $message = 'unknown status';
   break;
}
```

A partir de PHP 8, vous pouvez effectuer de la façon suivante :

```
$message = match ($statusCode) {
  200, 300 => null,
  400 => 'not found',
  500 => 'server error',
  default => 'unknown status code',
};
```

Source : https://wiki.php.net/rfc/match_expression_v2

str_contains()

Cette fonction vérifie si une chaîne est contenue dans une autre. Jusqu'à présent vous utilisiez la fonction 'strpos' ou 'strstr. Maintenant, vous pouvez utiliser str_contains comme ceci :

```
<?php

str_contains("abc", "a"); // true
str_contains("abc", "d"); // false
```

```
// Les arguments avec une chaîne vide
str_contains("abc", ""); // true
str_contains("", "");    // true
str_contains('', 'abc'); // false

// Un argument variable
str_contains('abc', $variable);
str_contains($variable, 'a');
```

str_starts_with() et str_ends_with()

Ces 2 fonctionnalités sont proches, car elles déterminent le point de départ d'une chaîne. Le point de départ pour la fonction str_start_with() sera le début de la chaîne. Par contre, pour la fonction str_end_with(), son point de départ sera la fin de la chaîne. Au niveau du code, nous nous appuyons sur une chaîne appelée $str :

```
<?php
$str = "gaucheCentreDroit";

str_starts_with($str, "gau") //true
str_starts_with($str, "Gau") //false

str_ends_with($str, "Droit")   //true
str_ends_with($str, "droit")   //false
```

Comme le montre l'exemple, si la chaîne recherchée correspond, le résultat sera à TRUE, sinon à FALSE.
Source : https://wiki.php.net/rfc/add_str_starts_with_and_ends_with_functions

fdiv()

Cette fonctionnalité s'inspire des fonctions fmod() and intdiv() et de la norme IEEE-754 sur l'arithmétique à virgule flottante. Ainsi, Fdiv() autorise la division par 0. Cette fonction ne renvoie plus d'erreurs, car la valeur retournée sera INF, -INF , NAN selon le cas. Au niveau du code, nous pouvons effectuer :
* + INF

```
fdiv(1, 0); // float(INF)
$num = 1 / 0; // float(INF)
// Warning: Division by zero in ... on line ...
```

* - INF

```
fdiv(-1, 0); // float(-INF)
$num = -1 / 0; // float(-INF)
// Warning: Division by zero in ... on line ...
```

Concaténation

Cette fonctionnalité a été introduite à PHP 7.4 . Avec PHP 8, la fonction réagit désormais de manière plus intelligente, lorsqu'elle travaille avec plusieurs opérateurs.
Jusqu'à présent pour effectuer une somme, vous faisiez cela :

```
echo "sum: " . $a + $b;
```

L'interprétation par PHP s'effectue :

```
echo ("sum: " . $a) + $b;
```

Maintenant, en PHP 8, vous pouvez effectuer ceci :

```
echo "sum: " . ($a + $b);
```

Source : https://wiki.php.net/rfc/concatenation_precedence

get_debug_type()

Inspiré de la fonction gettype(), get_debug_type() renvoie le type d'une variable. Ainsi, vous l'utilisez pour générer des messages de débogage plus précis comme lors du traitement de types qui ne peuvent pas être gérés par la vérification d'exécution PHP existante basée sur des types de paramètres, tels que ceux provenant d'un tableau. Au niveau du code, si vous obtenez un résultat sous la forme d'un tableau, vous effectuez :
Avant :

```
$bar = [1,2,3];

if (!($bar instanceof Foo)) {
 throw new TypeError(
    'Expected ' . Foo::class .
    ', got ' . (is_object($bar) ? get_class($bar) : gettype($bar)));
}
```

PHP 8 :

```
if (!($bar instanceof Foo)) {
      throw new TypeError(
    'Expected ' . Foo::class .
    ' got ' . get_debug_type($bar));
}
```

Suivant le résultat obtenu, la valeur retournée sera différente et représentée dans le tableau suivant : **figure 4**
Source : https://wiki.php.net/rfc/get_debug_type

Objet ::class

Il s'agit d'un alias de get_class(), dans le but d'attribuer une classe aux objets. L'intérêt de cet alias, permet de réduire la taille du code source. Cela fonctionne de la même manière.
Exemple :

```
<?php 8

$foo = new Foo();

var_dump($foo::class);
```

Source : https://wiki.php.net/rfc/class_name_literal_on_object

Arguments nommés

Les arguments nommés vous permettent de passer des valeurs à une fonction, en spécifiant le nom de la valeur, afin que vous n'ayez pas à prendre leur ordre en considération. Avec la nouvelle version, vous pouvez également ignorer les paramètres optionnels. Pour cela, vous programmez comme ceci :

Valeur	gettype()	get_debug_type()
1	integer	int
0.1	double	float
true	boolean	bool
false	boolean	bool
null	NULL	null
« WordPress »	string	string
[1,2,3]	array	array
Une classe qui s'appelle « Foo\Bar »	object	Foo\Bar
Une classe anonyme	object	class@anonymous

Figure 4

Les anciens numéros de PROGRAMMEZ!

Le magazine des développeurs

226

236

238

239

240

241

HS 01 été 2020

242

243

HS 02

Tarif unitaire 6,5 € (frais postaux inclus)

TECHNOSAURES

Le magazine à remonter le temps !

N°1

N°2

N°3

N°4

N°5

Prix unitaire : **7,66 €** (frais postaux inclus)

N°4 Standard **10 €**

N°4 Deluxe **15 €**

☐ 226 : |__| ex ☐ 241 : |__| ex
☐ 236 : |__| ex ☐ HS 01 : |__| ex
☐ 238 : |__| ex ☐ 242 : |__| ex
☐ 239 : |__| ex ☐ 243 : |__| ex
☐ 240 : |__| ex ☐ HS 02 : |__| ex

soit |__|__| exemplaires x 6,50 € = |__|__|__|__| €

Technosaures ☐ N°1 ☐ N°2 ☐ N°3 ☐ N°5
soit |__|__| exemplaires x 7,66 € = |__|__|__|__| €
☐ N°4 Deluxe..... 15 €
☐ N°4 Standard .....10 €

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

..... |__|__|__|__| € soit au **TOTAL** = |__|__|__|__| €

☐ M. ☐ Mme ☐ Mlle Entreprise : |__|__|__|__|__|__|__| Fonction : |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Prénom : |__|__|__|__|__|__|__|__|__|__|__|__| Nom : |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Adresse : |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Code postal : |__|__|__|__|__| Ville : |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|

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

Offres pouvant s'arrêter à tout moment, sans préavis PROG 245

```
function foo(string $a, string $b, ?string $c = null, ?string $d = null)
{ /* … */ }

foo(
  b: 'value b',
  a: 'value a',
  d: 'value d',
);
```

Source : https://wiki.php.net/rfc/named_params

convertir DateTime()

Les classes DateTime et DatetimeImmutable obtiennent une nouvelle méthode ::createFromInterface. Le but va faciliter la conversion de ces 2 classes, dans un sens comme dans l'autre. Au niveau du code, la fonction s'effectue de la manière suivante :

```
DateTime::createFromInterface(DateTimeInterface $object) ;
DateTimeImmutable::createFromInterface(DateTimeInterface $object) ;
```

get_resource_id()

La récupération d'un identifiant d'une ressource imposait aux développeurs de convertir la ressource en entier. Avec PHP 8, vous pouvez obtenir celle-ci directement.
C'est pourquoi chacune de ces ressources se voit attribuer un ID, bien qu'auparavant le seul moyen de connaître cet id était de convertir la ressource en int, comme ceci :

```
$resourceId = (int) $resource;
```

Avec cette fonctionnalité, PHP 8 rend l'opération plus évidente et plus sûre :

```
$resourceId = get_resource_id($resource);
```

Source : https://github.com/php/php-src/pull/5427

Object dans token_get_all()

La fonction token_get_all () existe depuis la version 4.2 et renvoie un tableau de valeurs. Avec PHP 8 l'implémentation fonctionne avec des objets au lieu de valeurs simples. L'intérêt de cela va réduire la consommation de la mémoire. Au niveau du code, vous utiliserez la méthode PhpToken :: tokenize ()

```
<?php
$tokens = PhpToken::tokenize('<?php echo; ?>');

foreach ($tokens as $token) {
  echo "Line {$token->line}: {$token->getTokenName()} ('{$token->
text}')", PHP_EOL;
}
```

Pour obtenir le résultat :

```
Line 1: T_OPEN_TAG ('<?php ')
Line 1: T_ECHO ('echo')
Line 1: ; (';')
Line 1: T_WHITESPACE (' ')
Line 1: T_CLOSE_TAG ('?>')
```

Source : https://wiki.php.net/rfc/token_as_object

ext-json

Avant PHP 8, il était possible de compiler PHP sans avoir activé l'extension JSON. Maintenant l'extension est automatiquement présente. Source : https://wiki.php.net/rfc/always_enable_json

La gestion des erreurs

La gestion des erreurs reste un sujet sensible. Car cela concerne les erreurs du langage, mais aussi les erreurs de codes. Pour cela, PHP 8 apporte des améliorations pour vous aider dans vos développements au niveau des erreurs et des exceptions.

Les erreurs

Auparavant, seules les fonctions définies par l'utilisateur déclenchaient les « TypeErrors », les fonctions internes émettaient un avertissement et « null ». La représentation des messages d'erreurs était sous la forme d'un warning ou d'une notice. Avec PHP 8, la majorité des fonctions internes renvoient également des « TypeErrors ». Pour cela, les avertissements disponibles sont :

- **Variable indéfinie** : déclenche une erreur d'exception au lieu d'un avis ;
- **Indice de tableau indéfini** : déclenche un avertissement au lieu d'un avis ;
- **Tentative d'incrémentation/diminution de la propriété "%s" d'un non-objet** : déclenche une erreur d'exception au lieu d'un avertissement ;
- **Tentative de modification de la propriété "%s" d'un non-objet** : déclenche une erreur d'exception au lieu d'avertissement ;
- **Tentative d'attribution de propriété "%s" d'un non-objet** : déclenche une erreur d'exception au lieu d'un avertissement ;
- **Création d'un objet par défaut à partir d'une valeur vide** : déclenche une erreur d'exception au lieu d'un avertissement ;
- **Impossible d'ajouter un élément au tableau, car l'élément suivant est déjà occupé** : déclenche une erreur d'exception au lieu d'un avertissement ;
- **Ne peut pas annuler le décalage d'une variable qui n'est pas un tableau** : déclenche une erreur d'exception au lieu d'un avertissement ;
- **Ne peut pas utiliser une valeur scalaire comme tableau** : déclenche une erreur d'exception au lieu d'un avertissement.

Une autre amélioration dans la gestion des erreurs avec @ qui ne supprime plus les fatal errors. Avant, vous supprimez les erreurs avec l'opérateur @. Maintenant , vous pouvez le paramétrer comme ceci :

```
display_errors=Off
```

Source : https://wiki.php.net/rfc/engine_warnings

catch

Lors de l'utilisation de try/catch, vous devez intercepter l'erreur dans une variable que vous l'utilisiez ou non. Avec PHP 8, il n'est plus nécessaire de déclarer une variable. Par exemple avant PHP 8 :

```
try {
  // Something goes wrong
} catch (Exception $e) {
  Log::error("Wrong : ".$e->getMessage() );
}
```

Avec PHP 8 :

```
try {
  // Something goes wrong
} catch (Exception) {
```

```
    Log::error("Something went wrong");
}
```

Notez qu'il est nécessaire de toujours spécifier le type, vous n'êtes pas autorisé à avoir une capture vide. Si vous souhaitez intercepter toutes les exceptions et erreurs, vous pouvez utiliser Throwable comme type de capture.
Source : https://wiki.php.net/rfc/non-capturing_catches

Throw

Le comportement de « Throw » est passé d'instruction en expression. Cette modification permet de lever une exception dans le code. Voici quelques exemples :

```
// Les fonctions fléchées n'acceptant que des expressions simples :
$triggerError = fn () => throw new MyError();

// $value est seulement définie que si le tableau n'est pas vide.
$value = !empty($array)
  ? reset($array)
  : throw new InvalidArgumentException();

// Une déclaration IF pourrait rendre l'intention plus claire
$condition && throw new Exception();
$condition || throw new Exception();
$condition and throw new Exception();
$condition or throw new Exception();
```

Source : https://wiki.php.net/rfc/throw_expression

Ajustements de syntaxe variable

La syntaxe des variables dans PHP 8 corrige un certain nombre d'incohérences. Il prend en charge 4 principaux types d'opérations de « déréférencement », qui sont :

```
Array: $foo[$bar], $foo{$bar}
Object: $foo->bar, $foo->bar()
Static: Foo::$bar, Foo::bar(), Foo::BAR
Call: foo()
```

Source : https://wiki.php.net/rfc/variable_syntax_tweaks

Les retraits

Avec la nouvelle version, PHP a supprimé quelques fonctionnalités qui sont :

- __autoload ;
- convert_cyr_string (sans dépréciation) ;
- create_function ;
- each ;
- ezmlm_hash (sans dépréciation) ;
- fgetss ;
- get_magic_quotes_gpc ;
- hebrevc (sans dépréciation) ;
- money_format ;
- restore_include_path (sans dépréciation).

Toutes ces fonctionnalités étaient annoncées depuis plusieurs versions comme dépréciées (Deprecated) et il était important de ne plus les utiliser.

Dépréciations

Avec cette nouvelle version, il a été annoncé que certaines fonctionnalités passaient en version dépréciée, c'est-à-dire que les fonctions existent toujours, mais qu'il ne faut plus les utiliser. Une liste est disponible, car certaines vont être renommées ou remplacées par de nouvelles (ou alias) qui ont fait leur apparition.
Source: https://www.php.net/manual/fr/migration80.deprecated.php

Incompatibilités

Par ailleurs, avec la nouvelle version, les évolutions ont apporté un certain nombre d'incompatibilités :

- La comparaison des chaînes à nombres ;
- La manipulation des nombres ;
- La liste des fonctions à utiliser par rapport aux fonctionnalités supprimées ;
- Etc.

L'ensemble est disponible sur la page dédiée.
Source : https://www.php.net/manual/fr/migration80.incompatible.php

Conclusion

Comme vous le voyez, cette nouvelle version apporte beaucoup d'améliorations. C'est pourquoi les frameworks, CMS… proposeront de nouvelles versions pour bénéficier de ces nombreux avantages. N'oubliez pas de suivre les mises à jour et les patchs. Comme toujours, testez toute migration sur un serveur de tests et de préproduction. Vérifiez les casses de codes pour modifier et adapter le code.
Toutes les RFC : https://wiki.php.net/rfc#php_80

MISE À JOUR DE VOTRE PHP

La mise à jour de votre version PHP en 8 comporte toujours des risques, même mineurs. Si vous mettez régulièrement les versions du langage, l'évolution sera transparente. Par contre si vous possédez une version plus ancienne, l'évolution peut avoir plus d'impacts.
Tout d'abord, il faut définir si vous utilisez des librairies, frameworks, CMSpour lesquels il faudra aussi prévoir une mise à jour. En effet, si vous utilisez une fonctionnalité qui n'existe plus dans une mise à jour, votre projet fonctionnera moins bien. Pour installer PHP 8 sur votre ordinateur à la place de la version actuelle, vous pouvez effectuer ceci si vous utilisez Debian 10. Vous mettez à jour le système d'exploitation :

```
apt update && apt full-upgrade
```

Comme la version est récente, tous les serveurs miroirs ne sont pas à jour. C'est pourquoi nous ajoutons le dépôt sury.org pour pouvoir effectuer l'installation :

```
apt install -y lsb-release apt-transport-https ca-certificates wget
wget -0 /etc/apt/trusted.gpg.d/php.gpg https://packages.sury.org/php/apt.gpg
echo "deb https://packages.sury.org/php/ $(lsb_release -sc) main" |tee /etc/apt/sources.list.d/php.list
L'installation de PHP 8 s'effectue :
apt update
apt install php8.0 -y
apt install php8.0-fpm php8.0-common php8.0-mysql php8.0-gmp php8.0-curl php8.0-mb
string php8.0-json php8.0-xmlrpc php8.0-gd php8.0-xml php8.0-readline php8.0-cli php8.0-zip
```

Pour vérifier que tout est correctement installé :

```
php -v
```

Pour obtenir le résultat suivant :

```
PHP 8.0.0 (cli) (built: Dec  6 2020 06:56:45) ( NTS )
Copyright (c) The PHP Group
Zend Engine v4.0.0-dev, Copyright (c) Zend Technologies
    with Zend OPcache v8.0.0, Copyright (c), by Zend Technologies
```

Les différentes étapes présentées ci-dessus seront presque identiques à tous les OS.

Jérémy Jeanson
Ingénieur d'Études et Développement
Inetum

Développer des applications pour les montres Fitbit : Qui ? Quoi ? Comment ?

Se lancer dans le développement d'applications pour des montres connectées n'est pas toujours évident. On se demande souvent ce que l'on va pouvoir faire, comment, avec quels moyens. Doit-on apprendre un nouveau langage, de nouveaux outils ?

Si vous avez déjà tenté l'expérience, vous avez pu constater que cela n'était pas simple. Heureusement pour nous, Fitbit a pris les contrepieds de nombreux fabricants. Tant pour le développement d'applications que dans la distribution et la monétisation.

Fitbit ?

Pour ceux qui ne connaîtraient pas Fitbit. Il s'agit d'une société californienne fondée en 2007. Elle s'est principalement portée sur commercialisation de trackers. Au fil du temps elle a acquis plusieurs sociétés qui conservaient des applications, ou trackers à destination des sportifs. On notera surtout l'acquisition d'un précurseur dans le domaine des smartwatchs : Pebble en 2016. Malheureusement, l'aventure des montres Peeble et de leur système d'exploitation Pebble OS s'arrête là. Fitbit profite cependant de la propriété intellectuelle de Pebble et de son équipe pour créer Fitbit OS et lancer sa première véritable smartwatch en 2017 (la Ionic).

Fitbit est donc une sorte d'OVNI dans le domaine de ce que l'on appelle aujourd'hui la « smartwatch » ou « montre connectée ». Elle a l'expérience des smartwatchs et celle des trackers d'activités (très appréciés des sportifs). C'est certainement ce qui a conduit à son rachat par Google en ce début d'année 2021.

	Ionic	Versa	Versa Lite	Versa 2	Versa 3	Sense
Fitbit OS	4				5	
SDK	4				5	
Fonctionnalités Ouvertes						
Heures	Oui	Oui	Oui	Oui	Oui	Oui
Cardiaque	Oui	Oui	Oui	Oui	Oui	Oui
Podomètre	Oui	Oui	Oui	Oui	Oui	Oui
Calories brûlées	Oui	Oui	Oui	Oui	Oui	Oui
Minutes en zone d'activité	Oui	Oui	Oui	Oui	Oui	Oui
Gyroscope	Oui	Oui	Oui	Oui	Oui	Oui
Accéléromètre 3 axes	Oui	Oui	Oui	Oui	Oui	Oui
Moteur vibrant	Oui	Oui	Oui	Oui	Oui	Oui
Altimètre	Oui	Oui	Non	Oui	Oui	Oui
GPS	Oui	Non	Non	Non	Oui	Oui
Fonctionnalités fermées						
Allway On Display (AOD)	Non	Non	Non	Oui	Oui	Oui
Capteur de température cutanée	Non	Non	Non	Non	Oui	Oui
Haut-parleur	Non	Non	Non	Non	Oui	Oui
Microphone	Non	Non	Non	Non	Oui	Oui

Du développement au store (App Gallery)

Commençons par les bonnes nouvelles. Le développement et la publication d'applications sur l'App Gallery de Fitbit coûtent zéro euro (lien vers l'app Gallery https://gallery.fitbit.com/). Il n'y a donc pas de souscription à payer. Le SDK lui-même est gratuit, open source et ouvert aux contributions.

Concernant la monétisation des applications, Fitbit nous laisse libres. On peut choisir la solution que l'on souhaite et Fitbit ne prend pas de commission.

Fitbit ne fixe qu'une seule et unique contrainte pour avoir le droit de publier une application : une montre doit être associée au compte de publication. Pour une montre de dernière génération Versa 3, cela fait un investissement de 229,95 €. Pour une montre de la génération précédente, une Versa Lite coûte 159,95 € aujourd'hui.

Via l'App Gallery, Fitbit vous permet de fournir un support à vos utilisateurs. Vous pouvez donner votre mail ou un lien vers votre site. Personnellement, j'ai choisi de fournir le formulaire de contact qui se trouve sur l'un de mes sites. Cela fonctionne extrêmement bien. Toutes les semaines, j'ai des retours d'utilisateurs qui me remercient ou me demandent d'ajouter de nouvelles fonctionnalités. Je vous encourage donc à faire de même. Cette communauté d'utilisateurs à un je-ne-sais-quoi qui fait la différence. Cela est très gratifiant et encourageant.

Petit tour du propriétaire

Aujourd'hui, le développement est possible uniquement sur gamme de montres qui fonctionnent avec Fitbit OS. Voici un tableau qui résume cette gamme et les possibilités accessibles ou non : voir ci-contre.

Note au sujet du GPS : Si la montre n'a pas de GPS, la position de l'utilisateur est fournie par le téléphone.

Fitbit a invité certains développeurs à utiliser l'AOD. Il n'est pas impossible que les autres fonctionnalités soient mises à disposition de la même manière avant d'être rendues accessibles à tous.

Concernant l'ECG et la mesure du SPO2 : plusieurs montres en sont capables. Cependant il y a très peu de chances que ces fonctionnalités soient ouvertes aux développeurs un jour. Voilà pourquoi, je ne les ai pas fait figurer sur ce tableau récapitulatif.

Que peut-on véritablement faire ?

Deux types de réalisations sont possibles :

- Application : Une application que l'utilisateur pourra lancer à la demande.
- Clock Face : Une application qui prend la place de l'horloge fournie à défaut par Firbit. Cette application est lancée dès que l'utilisateur allume sa montre (il ne peut y avoir qu'une Clock Face à la fois sur une montre).

Les deux types d'applications ont les mêmes capacités :

- Affichage / Navigation entre différents affichages.
- Saisie.
- Interaction avec l'utilisateur via tap, swipe (droite/gauche et haut/bas), vibrations….
- Utilisation des capteurs (pulsations cardiaques, mouvements, GPS,…).
- Accès aux statistiques de l'utilisateur (calories brûlées, pas, étages montés, distance parcourue…).
- Accès aux informations système (batterie, modèle, taille d'écran…)
- Stockage de fichiers.

Attention : le passage de la Clock Face aux divers écrans propres à l'OS se fait par swipe. Il est donc fortement déconseillé d'utiliser ce type de mouvement quand on code un Clock Face.

Les applications ont aussi accès à une palette de contrôles : boutons, listes, vues, animations… (ce sont tous des symboles SVG personnalisables) **Figure 1**

Vous aurez peut-être remarqué que je ne fais pas allusion au Bluetooth ni au Wifi. Ces deux technologies ne sont pas ouvertes aux développeurs. Pourtant, nos applications peuvent avoir accès à internet. Ceci est rendu possible via une architecture plutôt astucieuse. Une application Fitbit peut être coupée en deux :

- Une partie client (App) : déployée sur la montre. Elle se charge de toutes les opérations d'affichage et d'exploitation des capacités de la montre.
- Une partie serveur (Companion) : déployée sur le smartphone. Elle s'exécute au sein de l'application mobile de Fitbit. Elle a accès à internet et peut donc accéder à un API tiers ou télécharger des fichiers.

Les deux composants App et Companion communiquent via sockets ou échanges de fichiers.

Le couple App + Companion a un second intérêt. Les scénarios de saisie complexes trop complexes pour la montre peuvent être gérés à partir du mobile. Pour cela, Fitbit a prévu une interface appelée Settings. Celle-ci est écrite en React (JSX ou TSX).

Architecture technique

Attaquons-nous maintenant à l'architecture technique qui nous permet de faire fonctionner une application sur Fitbit OS. Ne vous attendez pas à avoir une description détaillée de Fitbit OS. Fitbit ne communique pas sur son système d'exploitation. Nous savons cependant que celui-ci utilise le moteur JerryScript pour exécuter nos applications et leur fournir l'accès aux différentes capacités de la montre.

Le développement pour Fitbit OS passe donc par JavaScript ou TypeScript avec un soupçon de SVG, de CSS et de React. Si vous êtes habitué au développement front-end, vous êtes

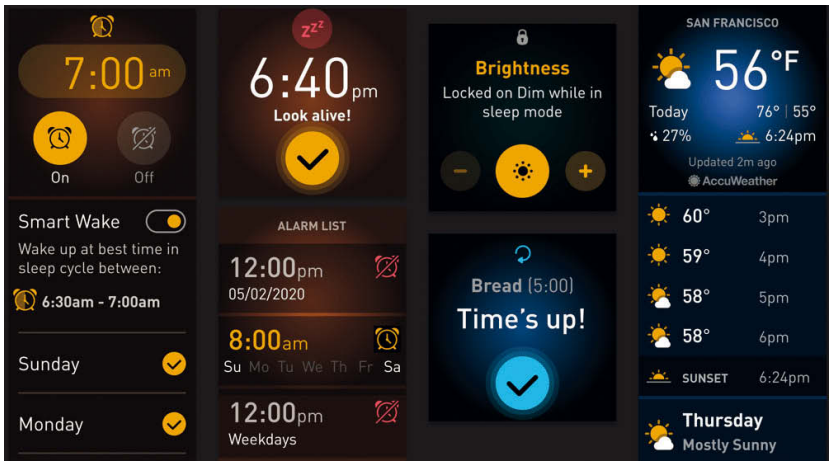


Figure 1

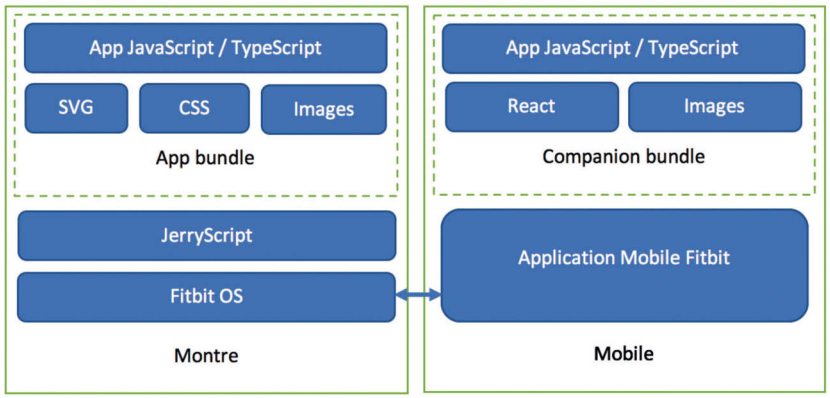


Figure 2

en territoire connu. La manipulation du SVG avec JavaScript étant similaire à celle du HTML. On applique des styles, changes des états des contrôles sans aucune difficulté. On ne peut cependant pas ajouter dynamiquement des contrôles. C'est là la seule limitation par rapport à une interface HTML.

Voici un graphique qui reprend les différentes notions que nous avons abordées : **figure 2**

Pour obtenir les bundles App et Companion, on utiliser la CLI et le SDK fourni par Fitbit. Comme pour la plupart des frameworks développent JavaScript, ceux-ci sont disponibles sous forme de modules NPM hébergée sur npmjs.com :

- @fitbit/sdk : contient les outils pour initialiser un projet, importer les modules utiles dans son JavaScript (librairies d'accès aux ressources de la montre).
- @fitbit/sdk-cli : contient plusieurs application dont un prompt qui permet la compilation, les tests et le déploiement.

Le code de ces modules est disponible sur GitHub. Si vous êtes un peu curieux, vous vous rendrez compte que le code de la CLI est inclus dans un projet nommé Developer Bridge. Il s'agit d'un service hébergé par Fitbit qui sert à faire le lien entre les outils de développement et de test :

- Fitbit Studio.
- Le prompt offert par la CLI.
- Le simulateur.
- Votre téléphone et votre montre (il y a une option Developer Bridge sur la montre et le mobile).

L'utilisation du Developer Bridge est très simple et discrète. Il suffit d'utiliser le même compte Fitbit avec l'ensemble de vos outils et matériel pour que celui-ci fonctionne.

L'environnement de développement

Le développement d'applications peut se faire via deux approches :

- Une solution web : Fitbit Studio. Ce site web permet de créer rapidement une application. Il est idéal pour se lancer, mais elle s'avère vite limitée. Il ne permet pas de gérer le versioning des applications.
- Une solution locale : Votre éditeur de code favori associé à Node.js (VS Code, NotePad++, Sublime Text…). Cette solution est la plus souple et n'a pas de limites.

La communication entre ce simulateur et votre outil de développement passe obligatoirement par internet pour atteindre le Developer Brdige. Il n'existe donc pas de solution 100% utilisable hors ligne.

Aujourd'hui, Visual Studio Code est l'outil conseillé par Fitbit. Il est aussi le mieux adapté depuis qu'il possède une extension dédiée (Fitbit SDK Extension). Celle-ci facilite l'édition des SVG, CSS via IntelliSense et la documentation. Elle détecte automatiquement les extensions de fichiers propres à Fitbit. Elle contient aussi de nombreuses snippets de code.

Je suis à l'origine de cette extension. Je ne suis peut-être plus très objectif. Pour utiliser le code qui suit, je vous conseille donc d'avoir installé VS Code, Node.js et le simulateur.

Le simulateur est disponible sur le site https://dev.fitbit.com/getting-started/. Ce simulateur fonctionne sur Windows et Mac. Linux n'est pas supporté par Fitbit, mais des développeurs indépendants le font fonctionner sur Linux au moyen de Wine.

Si l'on regarde un peu sous le capot, on se rend compte qu'il s'agit d'une application Electron. Elle ne nécessite aucune élévation de privilège pour son installation ou son utilisation. Concernant son installation, celle-ci se fait à la racine de votre profil utilisateur. Comme pour beaucoup d'application Electron, tout se fait tout seul et rien ne vous est demandé lors de l'installation. Au lancement le simulateur vous demandera de vous connecter avec votre compte Fitbit. Ceci permet de relier le simulateur au Developer Bridge. On peut ensuite choisir la montre à simuler et jouer avec ses différents capteurs. **Figure 3**

À savoir :

- Les statistiques de l'utilisateur simulées sont calculées à partir de l'heure locale (en fin de journée, elles sont donc très hautes).
- On ne peut pas interdire l'accès à un capteur ou à une statistique (il faut donc penser à tester avec une véritable montre ce qui se passe si l'utilisateur n'accepte pas de vous donner une autorisation).
- Le gyroscope ne peut pas être simulé.
- Il n'est pas possible de simuler les tâches de fond et le lancement de l'application sur le mobile.

Si l'application en cours de test comprend une partie Companion, une seconde fenêtre s'ouvre pour simuler le mobile.

Figure 4

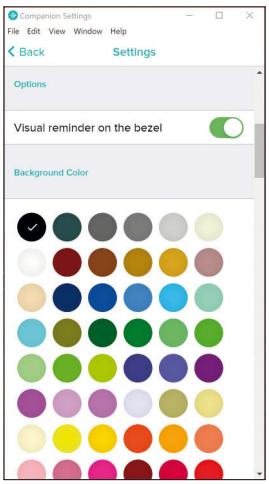


Figure 4

Figure 5

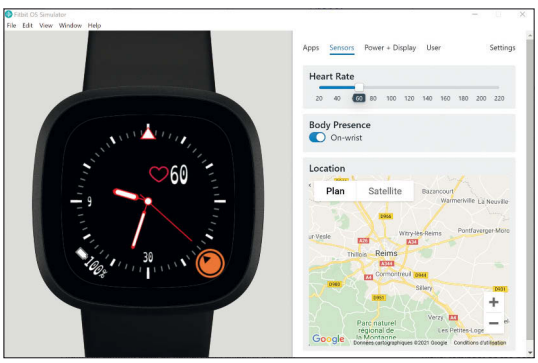


Figure 3

Créer sa première application

Pour votre première application, je vous propose de réaliser la Clock Face suivante : **Figure 5**

Celle-ci comprend :

- L'affichage de l'heure actuelle.
- L'affichage du compteur de pas.
- L'affichage du pourcentage de pas réalisés par rapport à l'objectif quotidien.

L'intégralité du code source qui suit est disponible via GitHub : https://github.com/JeremyJeanson/fitbit-demo

Pour initier notre projet, nous allons utiliser NPM. Plus précisément, sa commande "npx". Celle-ci se charge de télécharger le SDK, la CLI Fitbit, et lance la création de l'application. Exemple pour créer une application "clock1" :

```
npx create-fitbit-app clock1
```

La CLI lance alors une série de questions. La première consiste à déterminer le type d'application à créer. Pour notre exemple, je vous propose de réaliser une "Clock Face". **Figure 6**

Les questions suivantes permettent d'établir le nom de l'application et l'emploi d'une application Companion avec ou sans Settings. Ces deux options ne seront pas utilisées ici. Je les ai sectionnées afin de vous donner une idée de l'expérience compète de la CLI. **Figure 7**

La nouvelle application a alors été créée dans un dossier clock1. Il faut ensuite se déplacer vers ce dossier.

```
cd .\clock1\
```

L'application créée utilise JavaScript. Pour utiliser TypeScript, il suffit de lancer la commande suivante :

```
npx fitbit-sdk-types
```

Celle-ci télécharge les types pour TypeScript, ajoute la configuration utile et convertit les fichiers.

On dispose alors d'une arborescence similaire à celle-ci : **figure 8**

Le SDK et la CLI sont dans le dossier node_modules. Si vous créez plusieurs applications, vous aurez donc plusieurs fois ces fichiers sur votre ordinateur et vous les téléchargerez à plusieurs reprises (le déploiement et l'utilisation en global ne sont pas supportés). Pour limiter les téléchargements, Fitbit conseille l'utilisation de Yarn à la place de NPM. Personnellement, je préfère PNPM. Celui-ci a les mêmes avantages que Yarn, et ne duplique pas les fichiers sur votre ordinateur. À la place, PNPM crée des liens vers le cache des modules qu'il a téléchargé. Ce fonctionnement un peu particulier et sa manière d'organiser le dossier node_modules font qu'il vous faudra aussi installer le module tslib.

Ajoutons maintenant l'interface graphique. Pour cela, il faut éditer le fichier index.view qui se trouve dans le répertoire ressources comme ceci :

Figure 6

```
? What type of application should be created?
  app
> clockface
```

Figure 7

```
? What type of application should be created? clockface
? What should the name of this application be? Clock 1
? Should this application contain a companion component? Yes
? Should this application contain a settings component? Yes
? Which platforms should this application be built for? (Press <space> to select, <a> to toggle all, <i> to invert selection)
 (*) Fitbit Versa 3
>(*) Fitbit Sense
```

```
<svg>
 <rect id="background" />
 <svg id="steps-container">
  <arc sweep-angle="360" opacity="0.3" />
  <arc id="steps-arc" />
  <image href="images/steps_36px.png" />
  <text id="steps-text" />
 </svg>
 <text id="clock" />
</svg>
```

Une image doit aussi être ajoutée dans le répertoire ressources. Le fichier styles.css qui se trouve dans le même répertoire doit être modifié pour ajouter un peu de couleur et positionner les contrôles.

Code complet sur programmez.com & GitHub

Il ne reste plus qu'à ajouter la logique qui va permettre la mise à jour de l'interface.

Code complet sur programmez.com & GitHub

Comme vous pouvez le constater :

- L'utilisation du SDK passe par l'importation des modules utiles.
- Le rafraîchissement de l'interface ce fait en demandant à l'objet clock de déclencher son événement toutes les secondes.
- La manipulation du SVG passe par document.
- Le code n'est pas très compliqué.

Afin de demander l'accès aux statistiques d'activité de l'utilisateur et à ses objectifs, il faut aussi modifier le fichier package.json. La section requestedPermissions doit être modifiée comme ceci :

```
"requestedPermissions": [
 "access_user_profile",
 "access_activity"
],
```

Compiler et tester

Pour compiler et tester notre application, on passe encore via NPM et la commande npx :

```
npx fitbit
```

Celle-ci ouvre le navigateur internet pour que vous vous connectiez avec votre compte Fitbit. Après la connexion, vous pouvez fermer votre navigateur et revenir au terminal. Celui-ci vous ouvre l'accès à un nouveau prompt préfixé par : fitbit$. Pour lancer la compilation, il suffit d'utiliser la commande :

```
build
```

Après avoir lancé le simulateur, on procéder au déploiement via la commande :

```
install
```

Pour aller plus vite, il existe une commande qui compile build et install :

```
bi
```

Ensuite, il ne vous reste plus qu'à tester le comportement de l'application. Si des erreurs se produisent, elles seront affichées dans votre terminal. Vous pouvez y afficher vos propres messages pour tracer ce qui vous semble utile. Pour cela, il suffit d'utiliser les méthodes log, info, warn, error de console. Ceci permet de palier au fait qu'il ne soit pas possible de placer des points d'arrêt dans son code. Côté profilage, Fitbit nous permet de connaître la quantité de RAM consommée par notre application. Voici un exemple d'un code qui peut être ajouté pour loguer notre consommation toutes les secondes.

```
import { memory } from "system";
let totalUsage = 0;
let totalMeasurements = 0;
setInterval(
 function () {
  totalUsage += memory.js.used;
  totalMeasurements++;
  console.log(`JS memory: ${memory.js.used} / ${memory.js.total} (average: ${Math.round
(totalUsage / totalMeasurements)} / ${memory.js.total} )"`);
 },
 1000);
```

Pour aller plus loin ?

Si vous souhaitez en découvrir davantage sur le développement sur Fitbit, je vous conseille vivement d'aller sur le site https://dev.fitbit.com/ et https://github.com/Fitbit/ossapps/. Vous y trouverez de nombreux exemples d'application et modules MPN directement utilisables par vos applications. Mais surtout, vous y trouverez une communauté très active où se mêlent des développeurs de tous niveaux. Cerise sur le gâteau : le forum donne accès à un serveur Discord non officiel sur lequel vous trouverez presque toujours des développeurs de chez Fitbit (dont certains faisaient déjà partie de Peeble). Si vous bloquez à un moment, il y aura toujours quelqu'un pour vous aider .

Foncez ! Il y a là un beau terrain de jeux.

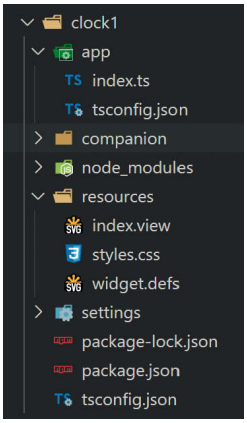
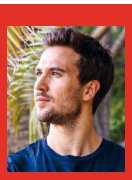


figure 8

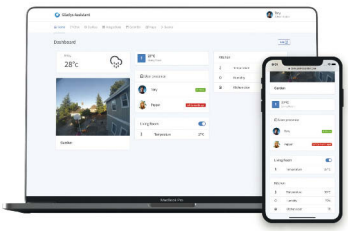


Pierre-Gilles Leymarie

Ingénieur logiciel de 26 ans. Il est le fondateur du projet open source Gladys Assistant qu'il a créé en 2013.

Un capteur température/humidité DIY avec un ESP8266, du MQTT et Gladys Assistant

Gladys Assistant est un logiciel open source de domotique qui s'installe sur n'importe quel système Linux : un Raspberry Pi, un NAS Synology, une Freebox Delta, ou n'importe quelle machine faisant tourner Linux.



La plateforme est française. Elle existe depuis 2013 et fédère une communauté active de passionnés de domotique. Depuis le début du projet en 2013, Gladys a été téléchargée plus de 40 000 fois, et le forum compte plus de 1600 membres.

En novembre 2020, le projet a sorti une nouvelle version majeure du logiciel : Gladys Assistant 4, une version réécrite de zéro, en Node.js pour le backend, SQLite pour la base de données, et preact.js pour le frontend (une version légère de React). L'accent a été mis sur la simplicité d'utilisation, la stabilité et la performance du logiciel.

Vous trouverez toutes les informations sur Gladys Assistant sur le site : https://gladysassistant.com/fr

Architecture et introduction à Gladys Assistant

Gladys Assistant est un serveur écrit en Node.js, distribué via une image Docker aux différentes plateformes supportées par Gladys : Raspberry Pi, NAS, et autres.

Le logiciel est composé d'un « core » abstrait, qui gère les appareils, les scènes, les utilisateurs, et de « services » qui, eux, font le pont entre le core et les appareils réels, un peu comme des drivers sur un ordinateur ! **Figure 1**

Le logiciel complet (core + services) est distribué via une image Docker buildée à chaque nouvelle version de Gladys. Une fois buildée, la mise à jour côté utilisateur est automatique, transparente et atomique : la mise à jour réussit, ou échoue en restant à la version précédente, mais le logiciel ne peut pas être dans un état bâtard.

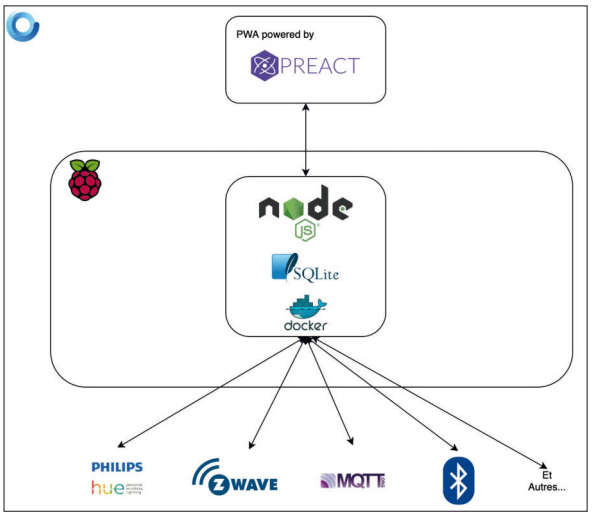


Figure 1

Contrairement à Gladys Assistant v3, et à d'autres projets open source de domotique, le choix a été fait de construire une image de Gladys comprenant autant le core que les services. Auparavant, les services devaient être installés séparément par l'utilisateur, et nous nous sommes rendu compte que cela rendait le logiciel moins stable, car il était très dur de tester de manière automatisée le comportement du logiciel avec { 1, 2, 3, … N } services installés, les combinaisons possibles de services installés étant infinies.

Aujourd'hui, nous distribuons un « bundle » clé en main, testé par une batterie de tests automatisés à chaque commit, et nous sommes capable de garantir que le logiciel dans son ensemble fonctionne comme prévu, ce que nous étions incapable de faire auparavant avec une architecture où l'utilisateur avait la responsabilité d'installer les services lui-même.

En termes d'expérience utilisateur, le logiciel est bien mieux compris car les utilisateurs peuvent configurer leurs périphériques en quelques clics après l'installation de Gladys, c'est clé en main.

Prérequis matériel

Dans cet article nous allons fabriquer un capteur température/humidité à partir d'un ESP8266, d'un capteur température/humidité pour Arduino (le fameux DHT11), et nous allons rapatrier les données vers Gladys Assistant en MQTT, afin de voir les valeurs de température/humidité sur un tableau de bord, et créer des scénarios à partir de ces valeurs.

Nous avons besoin ici :

- Un ESP8266 NodeMCU v3, trouvable pour environ 1,50€ si vous l'achetez en ligne depuis la Chine, ou 7€ depuis un vendeur en Europe. Le NodeMCU est un micro-contrôleur équipé d'une carte Wi-Fi, ainsi que de ports assez variés qui permettent d'y connecter tous types de capteurs/actionneurs. Le NodeMCU peut se programmer avec l'IDE Arduino, de la même manière qu'on programme un Arduino. Très pratique pour faire des petits projets embarqués comme ici. [Exemple lien d'achat: https://www.amazon.fr/gp/product/B06Y1ZPNMS/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1]
- Un capteur de température/humidité DHT11, trouvable selon les boutiques entre 0.60€ la pièce ou quelques euros. Attention, il existe plusieurs versions de ce capteur. Dans ce tutoriel, par simplicité nous allons utiliser le DHT11 avec résistance intégrée. Vous le reconnaîtrez facilement car il a une petite carte électronique soudée à lui et

il n'expose que 3 PINs. [Exemple lien d'achat: https://www.gotronic.fr/art-capteur-de-t-et-d-humidite-dht11-st052-26117.htm]

- 3 câbles mâle/femelle pour les branchements.
- Eventuellement une breadboard pour faire vos tests, mais c'est facultatif.

En commandant le matériel sur des sites chinois, vous devriez vous en sortir pour 3-4€ pour ce montage, et 10-12€ en achetant le matériel sur des vendeurs en Europe.

Pour la programmation de ce NodeMCU, nous la ferons en langage C via l'IDE Arduino.

Assemblage du PoC

L'assemblage du projet est plutôt simple. Vous devez brancher le pin de gauche du DHT11 sur le pin D5 du NodeMCU, le pin du milieu du DHT11 au 3V du NodeMCU, et le pin de droite au pin "G" ou "GND".

Voilà le schéma du montage : **figure 2**

Installation de Gladys Assistant sur un Raspberry Pi

Je vais vous décrire ici les étapes pour installer Gladys Assistant sur un Raspberry Pi, mais vous pouvez installer Gladys sur n'importe quelle machine Linux, du moment que Docker est supporté par la machine. Vous trouverez les instructions pour les autres machines sur le site du projet [Lien: https://gladysassistant.com/fr/docs/installation/docker/] .

Je recommande le Raspberry Pi, car c'est une formidable machine abordable (40€ environ), silencieuse (sans ventilateur), et qui ne consomme rien. C'est un outil parfait pour faire des projets DIY comme celui-ci.

Rendez-vous sur la documentation de Gladys Assistant [Lien: https://gladysassistant.com/fr/docs/], vous trouverez un lien de téléchargement vers l'image Raspberry Pi OS intégrant Gladys.

1. Téléchargez le fichier. Le fichier devrait s'appeler gladys_4.X.X -revX.img.zip ;
2. Dézippez le fichier ;
3. Branchez votre carte micro-SD à votre ordinateur ;
4. Clonez l'image sur la carte micro-SD de votre Raspberry Pi. Je vous conseille d'utiliser l'outil BalenaEtcher [Lien: https://www.balena.io/etcher/] qui vous permettra de cloner très simplement l'image ;
5. Débranchez la carte micro-SD de votre ordinateur, et insérez-la dans votre Raspberry Pi ;
6. Branchez votre Raspberry Pi en Ethernet à votre box internet, puis au secteur ;
7. Laissez le temps à votre Raspberry Pi de s'allumer. Au premier démarrage, le Pi va redimensionner sa partition afin que tout l'espace disponible sur la micro-SD soit disponible pour le système ;
8. Vous pouvez maintenant vous connecter à Gladys depuis votre navigateur. Pour cela, deux possibilités :
   - Votre box internet expose peut-être le Raspberry Pi sur le réseau par son hostname, et ainsi vous pouvez accéder à Gladys sur votre navigateur en tapant "http://gladys.local" ,
   - Si ce n'est pas le cas, il faut trouver l'adresse IP de votre Rasberry Pi sur le réseau. Je vous conseille d'utiliser une application comme Network Scanner [Lien: https://play.google.com/store/apps/details?id=com.easymobile.lan.scanner&hl=fr] sur Android, ou iNet [Lien: https://itunes.apple.com/fr/app/inet-network-scanner/id340793353?mt=8] sur iOS. Ces deux applications vous permettent de scanner le réseau, et vous montreront tous les appareils disponibles sur le réseau. Cherchez "gladys", et gardez l'adresse IP de côté. Ensuite, tapez "http://ADRESSE_IP_DU_RASPBERRY" dans votre navigateur. Par exemple : "http://192.168.1.12 ".
9. Vous devriez arriver sur une page de bienvenue ! Si c'est le cas, bravo ! Vous avez installé Gladys avec succès ;
10. Il est temps de configurer votre compte Gladys Assistant. Créez votre compte local et configurez vos préférences et votre logement. Cette configuration est uniquement locale, tout est enregistré dans votre Raspberry Pi : rien ne quitte votre machine.
11. Bravo ! Votre installation Gladys est fonctionnelle.

Configuration du périphérique MQTT dans Gladys Assistant

Il est maintenant temps de configurer le MQTT dans Gladys. Rendez-vous dans l'onglet "Intégration" => "MQTT".

Allez dans l'onglet "Configuration", et cliquez sur "Installer le broker sur Docker". Cela va installer un serveur Mosquitto via Docker. **Figure 3**

Une fois le serveur MQTT lancé, vous devriez voir des identifiants de connexion au broker MQTT: un login et un mot de passe. Gardez-les quelque part, vous en aurez besoin à la prochaine étape du tutoriel !

Maintenant, vous pouvez vous rendre dans l'onglet "Appareil" de l'intégration MQTT, et cliquer sur "Nouveau" pour définir un périphérique MQTT.

Dans notre cas, nous voulons définir un capteur température/humidité.

Remplissez nom, external_id, la pièce dans laquelle se trouve le capteur, puis ajoutez 2 fonctionnalités à ce capteur: Température et humidité.

Vous devriez avoir un capteur qui ressemble à cela : **figure 4** et **figure 5**

Gardez les deux "Topic MQTT" donnés par Gladys, vous en aurez besoin plus tard. Les topics MQTT ressemblent à : "gladys/master/device…"

Cliquez sur "sauvegarder".

Maintenant, rendez-vous sur la page d'accueil, et cliquez sur "Editer" pour éditer le tableau de bord. Ajoutez une box

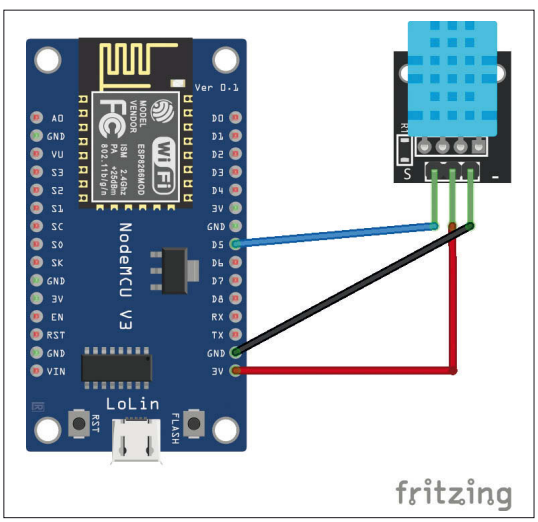


Figure 2

"Appareils de la pièce", sélectionnez la pièce de votre capteur puis sélectionnez les deux fonctionnalités que vous avez créées. Cliquez sur sauvegarder.
Vous devriez voir votre pièce, vos 2 capteurs, mais pas encore de valeurs de capteurs.

On code l'ESP8266 avec l'IDE Arduino

Maintenant que notre montage est fonctionnel, nous allons programmer le NodeMCU afin de récupérer les valeurs du capteur, et les envoyer en MQTT à Gladys Assistant.
Le NodeMCU se programme à l'aide de l'IDE Arduino que vous pouvez télécharger sur le site Arduino. [Lien: https://www.arduino.cc/en/software]

1. Installer l'IDE Arduino sur votre machine ;
2. Lancez l'IDE ;
3. Allez dans "Préférences", et dans "URL de gestionnaire de cartes supplémentaires", ajoutez "http://arduino.esp8266.com/stable/package_esp8266com_index.json" ; **figure 6**
4. Ensuite, allez dans "Outils" => "Carte"= > "Gestionnaire de cartes" ("Tools" => "Board" => "Boards manager" si vous êtes en anglais) ; **figure 7**
5. Dans le gestionnaire de carte, cherchez "esp8266" et installez la carte esp8266 ; **figure 8**
6. Avant de pouvoir brancher en USB votre ESP8266, vous devez installer les drivers pour votre système (Windows, macOS ou Linux). Le fabricant chinois WCH en fournit sur cette page : [Lien: http://www.wch-ic.com/search?q=ch340&t=downloads] ;
7. Une fois les drivers installés, vous pouvez brancher votre NodeMCU en USB à votre ordinateur. Attention, il faut nécessairement le brancher avec un câble micro-USB prévu pour transporter de la data, pas juste un câble de chargement ! Chez moi, la plupart de mes câbles micro-USB étaient des câbles de chargement uniquement, et ne fonctionnaient pas pour téléverser du code, j'ai dû trouver un câble spécial ;
8. Il faut maintenant installer 2 librairies dont nous allons avoir besoin : "PubSubClient" pour le MQTT, et "DHT Sensor library" pour pouvoir lire les valeurs du capteur. Allez dans "Croquis" => "Inclure une bibliothèque" => "Gérer les bibliothèques" (en anglais: "Sketch" => "Include library" => "Manage libraries") puis cherchez et ajoutez ces deux librairies ; **figures 9** et **10**
9. Maintenant que votre ESP8266 est branché à votre machine, vous pouvez lui téléverser le code suivant [CODE = gladys_mqtt_temperature.ino]. Pour cela, copiez-collez le code dans l'IDE Arduino, puis remplacez :
   - VOTRE_WIFI_SSID : le nom de votre WiFi ;
   - VOTRE_MOT_DE_PASSE_WIFI : le mot de passe de votre WiFi ;
   - VOTRE_IP_MQTT_SERVER: l'IP de votre instance Gladys. Exemple : 192.168.1.12 ;
   - VOTRE_UTILISATEUR_MQTT : le nom d'utilisateur de votre serveur MQTT, défini à l'étape précédente ;
   - VOTRE_MOT_DE_PASSE_MQTT : le mot de passe de votre utilisateur MQTT, défini à l'étape précédente ;
   - VOTRE_TOPIC_TEMPERATURE : le topic MQTT sur lequel Gladys écoute pour le capteur de température

Figure 3

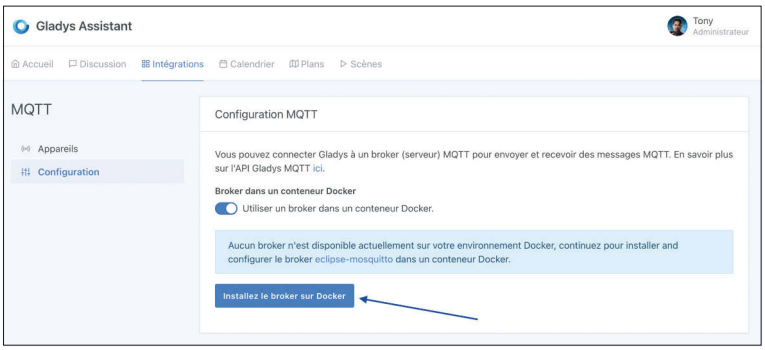


Figure 4

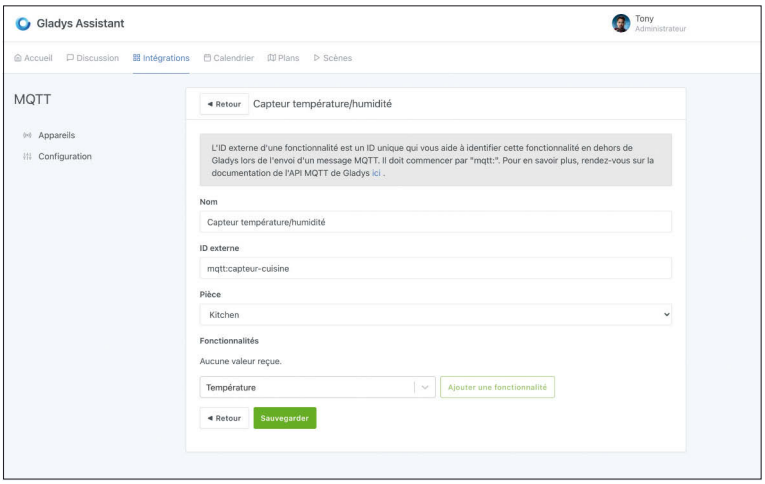


Figure 6

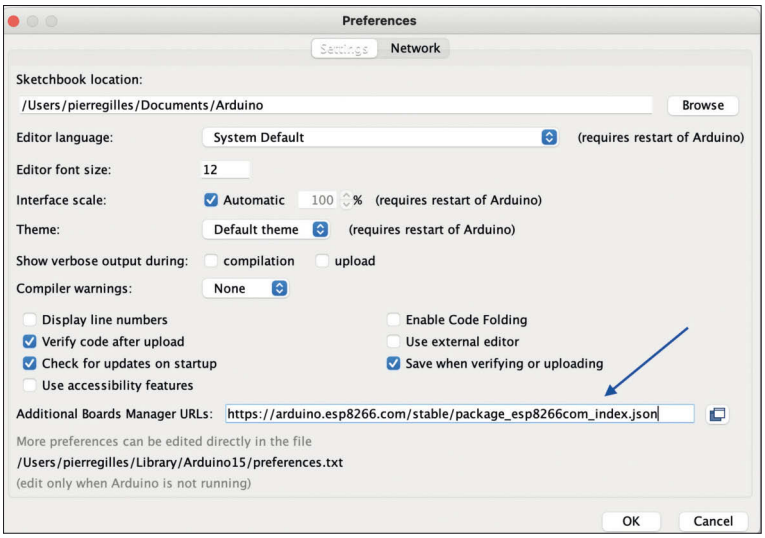
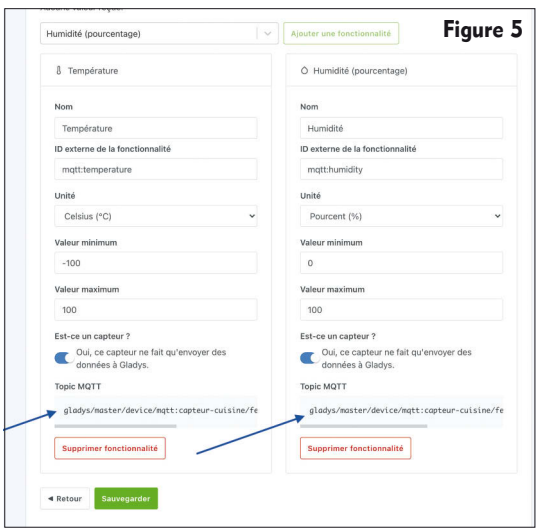


Figure 5



que vous avez créé à l'étape précédente. Exemple : gladys/master/device/mqtt:capteur/feature/mqtt:temperature/state ;

- VOTRE_TOPIC_HUMIDITY : le topic MQTT sur lequel Gladys écoute pour le capteur d'humidité que vous avez créé à l'étape précédente. Exemple : gladys/master/device/mqtt:capteur/feature/mqtt:humidity/state ;

10 Sélectionnez le type de board vers lequel vous voulez téléverser (dans "Outils" => "Carte" => "ESP8266 Boards" => "Node MCU 1.0 ESP-12E module".) ; **figure 11**

11 Puis téléversez le code en cliquant sur "Téléverser" (ou "Upload" en anglais) ;

12 Maintenant, ouvrez "Outils" => "Moniteur série" et vous devriez voir les messages de logs envoyé par l'ESP8266. Il devrait tenter de se connecter en WiFi, puis toutes les minutes, d'envoyer des valeurs de température et d'humidité ;

13 Retournez dans Gladys, vous devriez voir sur le tableau de bord vos valeurs de température et d'humidité ! Si oui, bravo ! Si ce n'est pas le cas, vous devez vérifier que vos identifiants (WiFi et MQTT) sont bons, que l'adresse IP de Gladys est bien la bonne, et qu'il est possible de se connecter en MQTT au serveur MQTT. Vous pouvez utiliser des outils comme MQTT.fx par exemple pour vérifier que le serveur MQTT est bien fonctionnel. **Figure 12**

Des alertes Telegram quand la température est trop haute ou trop basse

Certains utilisateurs Gladys utilisent un capteur de température placé dans leur réfrigérateur pour vérifier que la température est conforme à leur programmation: Ni trop haute, ni trop basse. Pratique pour vérifier que le réfrigérateur n'est pas en panne et que tous les aliments vont se périmer si rien n'est fait !

Créer une scène

Pour cela, nous allons créer une scène dans Gladys en allant dans l'onglet "Scènes" de l'interface.

Cliquez sur "Nouveau +" pour créer une nouvelle scène. Donnez-lui un nom, par exemple "Vérification température frigo", et une icône.

Une scène est découpée en 2 parties :

- **Les déclencheurs :** ce sont des règles qui vont déclencher la scène si un des déclencheurs se vérifie ;
- **Les actions :** ce sont des actions regroupées par "blocs d'actions" qui s'exécuteront quand la scène sera déclenchée. Toutes les actions dans un bloc d'actions s'exécutent en parallèle, en revanche les blocs d'actions s'exécutent eux en série les uns à la suite des autres.

Nous allons donc créer un déclencheur "Changement d'état de l'appareil", sélectionner notre capteur de température, et entrer une condition « Quand la température est supérieure à 7°C ». Nous allons cocher la case "Exécuter seulement lorsque le seuil est passé", ce qui évitera d'être spammé de message si le frigo est en panne et que vous ne pouvez pas intervenir immédiatement ! **Figure 13**

Ensuite, nous allons créer une action "envoyer un message", nous allons sélectionner notre utilisateur en destinataire, puis entrer en message "Le frigo est trop chaud !". **Figure 14**

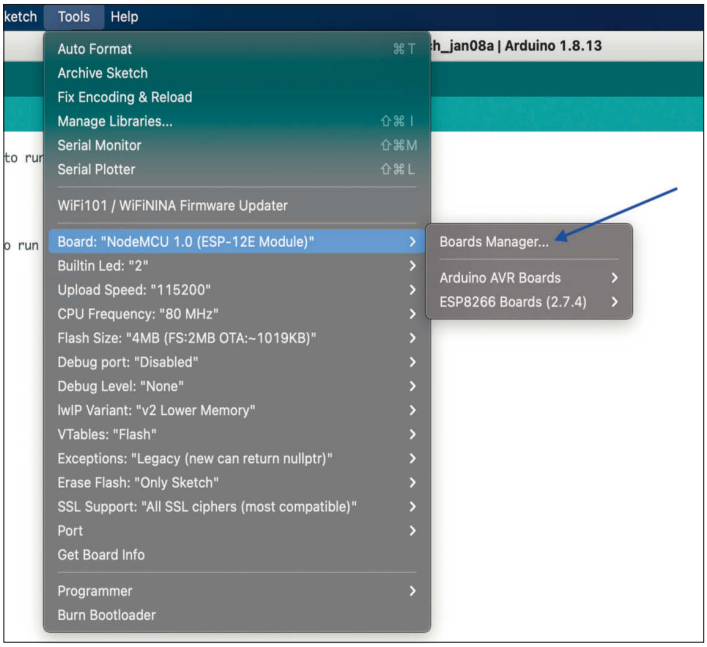


Figure 7

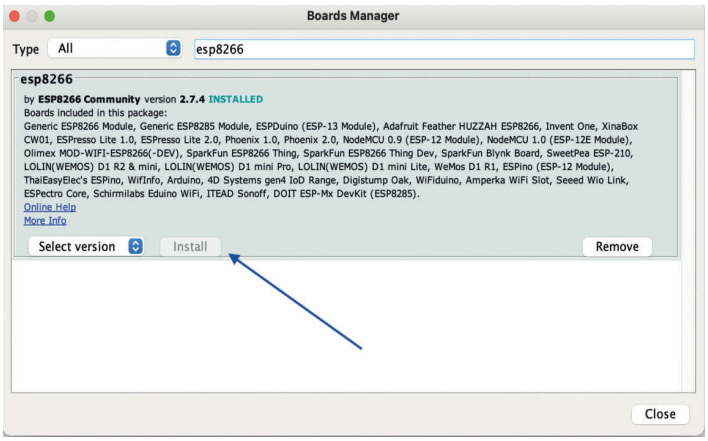


Figure 8

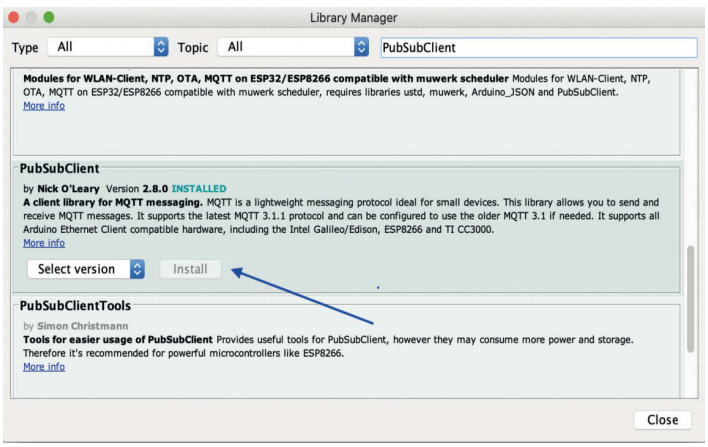


Figure 9

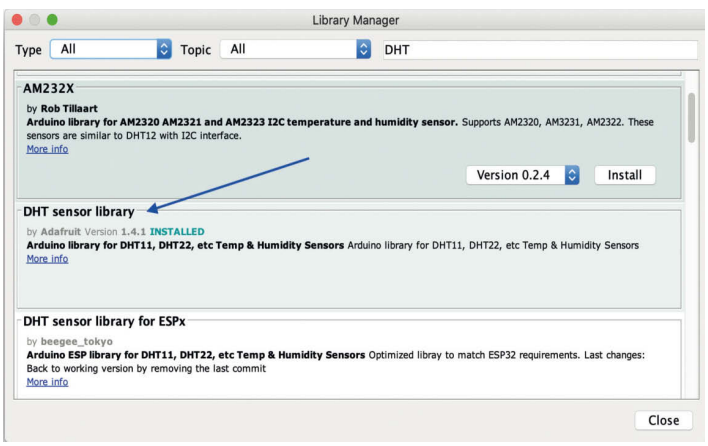


Figure 10

Cliquez sur "enregistrer le scénario" en haut à droite, puis cliquez sur "Démarrer" pour le tester. Si vous vous rendez dans l'onglet "Discussion", vous devriez voir un message "le frigo est trop chaud !"

Je suppose que vous voulez avoir des alertes sur votre téléphone, je vous recommande donc de configurer l'intégration avec Telegram.

Configurer Telegram

Vous devez avoir l'application Telegram téléchargée sur votre smartphone (disponible sur iOS et Android), c'est une application de chat similaire à WhatsApp.

Rendez-vous dans l'onglet "Intégrations" => "Telegram" et suivez les instructions.

1. Vous devez en premier lieu envoyer un message au @botfather sur Telegram pour créer un bot Telegram. Vous devriez récupérer une clé d'API.

Figure 12

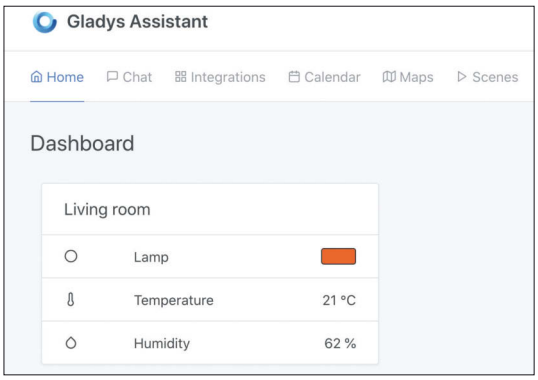
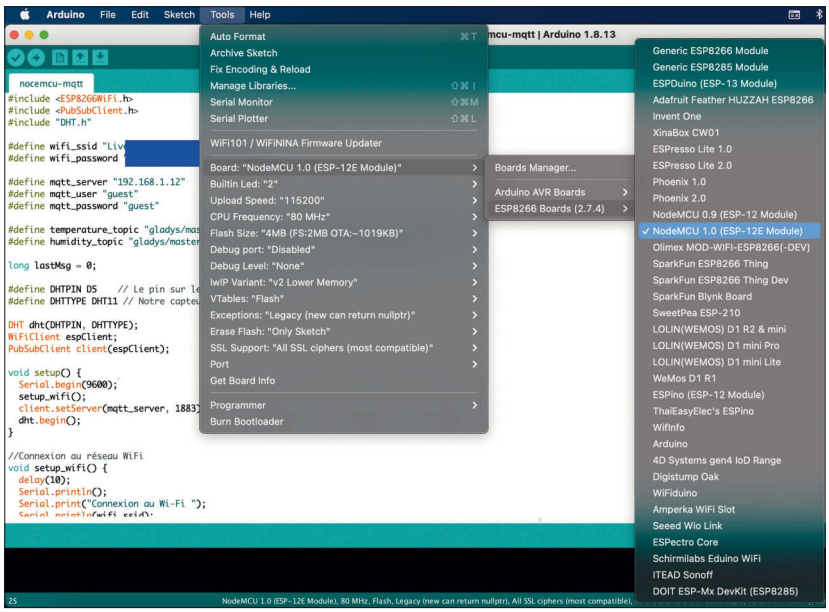


Figure 11



2. Ensuite, entrez cette clé d'API dans Gladys, et sauvegardez.
3. Vous devez maintenant cliquer sur le lien, qui devrait vous rediriger vers Telegram (web, desktop ou mobile). Cliquez sur "Start" en bas de la conversation avec votre bot, ce qui devrait lier votre bot Telegram à votre compte Gladys local.
4. Voilà, votre Gladys fonctionne maintenant avec Telegram!

Tester le scénario en entier

Maintenant que votre scène et Telegram sont en place, vous pouvez faire des tests : mettez votre capteur de température dans un endroit plus chaud (ou plus froid), et voyez si Gladys réagit à ce changement. Le capteur faisant une mesure par minute, il faut au moins attendre une minute pour voir le changement. Vous pouvez augmenter la vitesse de rafraîchissement dans le code Arduino, mais attention sur le long terme cela risque plus de remplir votre Raspberry Pi qu'autre chose si vous mettez une vitesse de rafraîchissement trop rapide !

Pour aller plus loin

Nous avons pour l'exemple utilisé un capteur humidité/température, mais vous pouvez brancher n'importe quel capteur à votre NodeMCU, et remonter les valeurs à votre instance Gladys pour les retrouver sur votre tableau de bord.

Mieux encore, il est possible de faire de votre NodeMCU un actionneur. Imaginons que vous souhaitiez contrôler la pompe de votre piscine, et que vous branchiez un relais (On/Off) à votre NodeMCU. Vous pouvez déclarer dans Gladys cet appareil comme un actionneur binaire (On/Off), et côté NodeMCU vous écoutez le topic MQTT que vous aura fourni Gladys à la configuration. A chaque fois que vous déclencherez un allumage ou une extinction dans Gladys, un ordre sera envoyé en MQTT au NodeMCU qui contrôlera le relais !

Conclusion

Ce n'est qu'un exemple de ce qu'il est possible de faire avec Gladys Assistant. Et ce n'était qu'un exemple où on mettait les mains dans le cambouis pour vous en montrer l'ouverture de l'API MQTT. Il est possible de faire bien plus, avec ou sans bricolage, selon les appareils que vous avez chez vous.

Gladys Assistant permet de gérer des ampoules Philips Hue en 3 clics, récupérer des valeurs de capteurs Xiaomi, connecter des appareils Z-Wave, afficher des caméras de surveillance, le tout sans forcément avoir de connaissances en informatique. La liste du matériel compatible est disponible sur le site du projet, que je vous invite à visiter si vous êtes intéressé par Gladys Assistant !

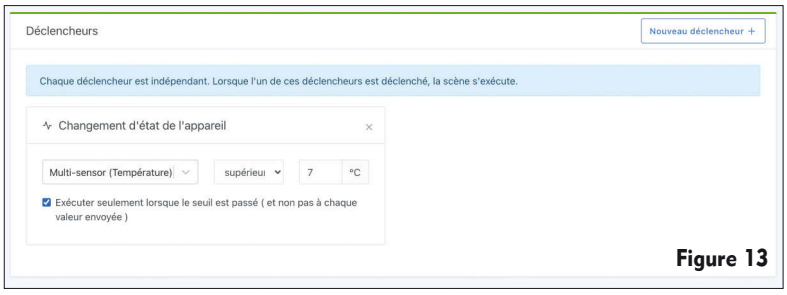


Figure 13

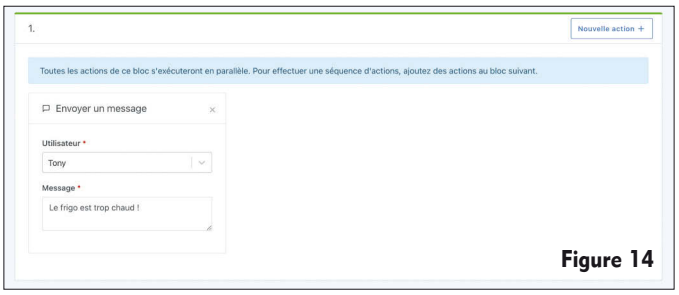


Figure 14

Qu'est-ce que Oracle GraalVM?

Plus d'un quart de siècle après sa création par Sun Microsystems, Java reste l'un des langages les plus populaires du monde comme l'attestent les différents classements de l'index TIOBE . Il est né avec la promesse d'affranchir le développeur des spécificités des plateformes d'exécution. De plus, Java est porté par le majestueux slogan WORA (Write Once, Run Anywhere) qui a fait sa force et une partie de sa popularité. Grâce à tout cela, il permet à des organisations et développeurs d'écrire du code Java, de le compiler en Bytecode. Ce dernier pouvant s'exécuter sur n'importe quelle plateforme équipée d'un outil spécial appelé la Machine Virtuelle Java. Java est quasiment implanté dans toutes les industries et organisations du monde.

Elvadas NONO

Elvadas est Principal Solution Architect chez Oracle, spécialiste de la transformation digitale et des technologies Java, Middleware et Cloud. Elvadas accompagne les entreprises françaises et internationales dans la modernisation de leur parc applicatif et la transition vers le multi cloud. Il a travaillé de nombreuses années chez Red Hat et Docker Inc. en tant que consultant spécialiste des Middlewares et des Plateformes de Containers Docker, Kubernetes.

GraalVM™

Parfois, Java a toutefois essuyé des critiques au niveau de la performance, de la verbosité du langage… En réponse, ce dernier a constamment évolué et la JVM également. A l'heure des microservices et des Cloud Native Applications, après plus d'une décennie de recherches, les preux chevaliers d'Oracle Labs ont mis sur pied une Nouvelle machine virtuelle Haute performance dénommée GraalVM qui vient en remplacement de la machine Virtuelle Hotspot. GraalVM existe en deux éditions : une Édition Community et une Édition Entreprise.

Une VM haute performance

Le package GraalVM est similaire à celui du JDK. Cependant GraalVM intègre en son sein un nouveau compilateur JIT(Just In Time) nommé Graal Compiler qui embarque une centaine d'algorithmes d'optimisation permettant de transformer plus rapidement le Bytecode issu de la compilation des sources .java en code machine optimisé.

LaRenaissance.dev, référence en matière de benchmarks Java, stipule que GraalVM accroît de l'ordre de 32% en moyenne les performances de ses benchmarks. **Figure 1**

Sur des algorithmes de la vie courante évalués par La renaissance.Dev, Graal VM est en moyenne deux fois plus rapide qu'un JDK8.

Il est donc possible pour des applications Java actuelles de maximiser leurs performances sans avoir à modifier les sources. Ceci simplement en remplaçant le Java Runtime Environment actuel par GraalVM. Cette migration transparente permet de bénéficier de toutes les optimisations induites par le compilateur JIT Graal GraalVM.

Dans cet extrait nous comparons les performances d'un bout de programme Java permettant de calculer le la somme des carrés des N premiers nombres :

```
public class Somme {

  public static void main(String[] args) {
      int n = Integer.valueOf(args[0]);
      BigInteger m=
          BigInteger.valueOf(Stream.iterate(1, i -> i+1)
              .limit(n)
              .map(j-> j*j)
              .reduce(0,Integer::sum));
      System.out.println(String.format("Somme Carrés(%d)=%d", n,m));
  }
}
```

Après avoir installé GRAALVM :

```
export GRAALVM_HOME=/Library/Java/JavaVirtualMachines/
graalvm-ee-java11-20.3.0/Contents/Home
export JAVA_HOME=$GRAALVM_HOME
```

Compilons le programme en question via l'utilitaire javac :

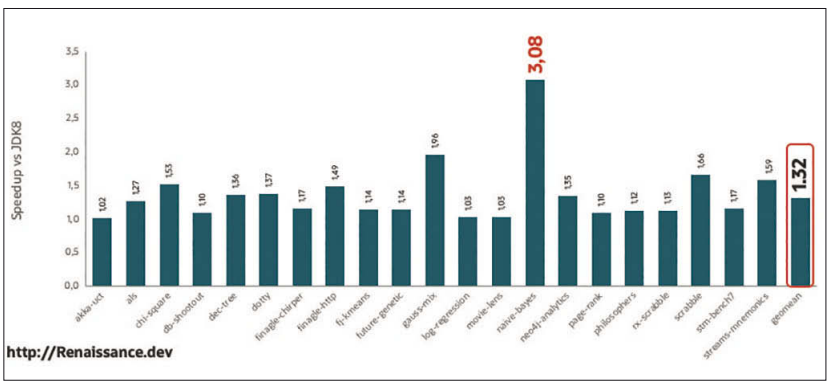
```
javac -d .  src/Somme.java
```

Évaluons les performances du programme pour la somme des carrés des 100000 premiers entiers avec le compilateur JIT Graal.

```
>  time $GRAALVM_HOME/bin/java   Somme   100000
Somme Carrés(100000)=1626540144
Somme 100000  0.14s user 0.10s system 46% cpu 0.511 total
```

Évaluons les performances du même programme avec le compilateur JIT par défaut de la JVM :

Figure 1



```
$GRAALVM_HOME/bin/java -XX:-UseJVMCICompiler Somme 100000
Somme Carrés(100000)=1626540144
-XX:-UseJVMCICompiler Somme 100000 0.17s user 0.03s system 150% cpu 0.134 total
```

l'option -XX:-UseJVMCICompiler permet de revenir au compilateur JIT par défaut de votre JVM, celui de GraalVM n'est alors plus utilisé. Notez la dégradation des performances.

En faisant tourner son service de Tweet sous GraalVM par exemple, Twitter est parvenu à augmenter les performances de l'ordre de 10% tout en réduisant de presque 20+ la latence dans le traitement des tweets.

> *"Nous économisons beaucoup d'argent et de cycles CPU"*
> **Chris Thalinger**, *Staff System Engineer, Twitter*

GraalVM native image : un packaging optimisé pour les applications à destination du cloud

GraalVM embarque en son sein un mécanisme de compilation anticipée en code machine dit AOT (Ahead-Of Time) qui permet de compiler le code java en instructions machine directement exécutables par les processeurs. Le livrable généré est appelé "native image" et inclut toutes les classes de l'application, les classes du JDK ainsi que les classes des dépendances nécessaires au fonctionnement de l'application en question.

```
$ javac HelloWorld.java
$ time java HelloWorld
user 0.070s
$ native-image HelloWorld
$ time ./helloworld
user 0.005s
```

L'AOT est rendu possible grâce à la technologie SubstrateVM.

La technologie GraalVM Native Image est particulièrement adaptée pour les applications à destination du cloud : FaaS, microservices, Serverless, les programmes en lignes de commandes qui ont plus que besoin d'un démarrage quasi instantané… Pour tous ces services, elle permet d'atteindre des vitesses de démarrage supersoniques de l'ordre de 100x fois plus rapide qu'un démarrage en compilation standard avec une empreinte mémoire 5x fois moins importante.

Les applications en native image basées sur GraalVM permettent ainsi une utilisation rationnelle des ressources CPU/Mémoire et, in fine, de réduire votre facture mensuelle chez votre Cloud Provider.

Générons une image native pour la classe Somme précédemment définie.

$

```
$GRAALVM_HOME/bin/native-image Somme
[somme:12673]   classlist:  1,479.80 ms,  0.96 GB
[somme:12673]       (cap):  10,556.30 ms,  0.96 GB
[somme:12673]     setup:  13,108.44 ms,  0.96 GB
[somme:12673]   (clinit):   237.84 ms,  1.22 GB
[somme:12673]  (typeflow):  3,801.15 ms,  1.22 GB
[somme:12673]  (objects):  3,348.80 ms,  1.22 GB
[somme:12673]  (features):   250.89 ms,  1.22 GB
[somme:12673]   analysis:  7,832.72 ms,  1.22 GB
[somme:12673]   universe:   369.13 ms,  1.22 GB
[somme:12673]   (parse):  1,329.08 ms,  1.68 GB
[somme:12673]   (inline):  1,779.36 ms,  1.68 GB
[somme:12673]  (compile):  17,715.76 ms,  3.27 GB
[somme:12673]   compile:  21,855.01 ms,  3.27 GB
[somme:12673]    image:  1,774.18 ms,  3.27 GB
[somme:12673]    write:   764.49 ms,  3.27 GB
[somme:12673]   [total]:  47,374.69 ms,  3.27 GB
```

Un nouveau binaire natif est disponible

```
(base) $ls -rtl
total 17024
-rw-r--r--  1 enonowog  staff    425 Dec 15 16:03 StreamDemo.iml
-rw-r--r--  1 enonowog  staff   2191 Dec 16 15:15 Fibonacci.class
drwxr-xr-x  3 enonowog  staff     96 Dec 16 18:36 out
drwxr-xr-x  4 enonowog  staff    128 Dec 16 18:58 src
-rw-r--r--  1 enonowog  staff   2202 Dec 16 18:58 Somme.class
-rwxr-xr-x  1 enonowog  staff  8701796 Dec 16 19:09 somme
```

Évaluons les performances du programme natif issu de la compilation AOT de la classe Somme.class

```
(base) $time ./somme 100000
Somme Carrés(100000)=1626540144
./somme 100000  0.01s user 0.00s system 3% cpu 0.322 total
```

Le gain de performance est exceptionnel.

Plusieurs frameworks Cloud Native à l'instar de SpringBoot, Hélidon Quarkus… font déjà usage de GraalVM afin de permettre à leurs utilisateurs de mettre sur pied des microservices optimisés et ultra-performants.

Partant d'une image native issue de la compilation GraalVM, il devient plus facile de transporter une application vers le cloud.

GraalVM : une Machine virtuelle Polyglotte et fédératrice des langages de programmation

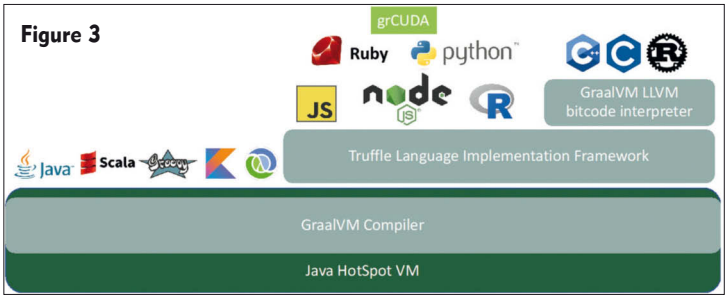
Write Everywhere, Run in GraalVM

GraalVM supporte en natif tous les langages basés sur le Bytecode (Java, Scala, Kotlin, Clojure, Groovy etc.) d'une part, mais également tous les langages pour lesquels une implémentation Truffle existe. Truffle est un framework open source permettant de mettre sur pied des interpréteurs de langages par le biais des arbres syntaxiques AST.

A ce jour, le moteur polyglotte de GraalVM supporte l'interprétation des langages JavaScript, Python, Ruby, R, ainsi que des langages bas niveau dits LLVM (C, C++, Rust). GraalVM permet de mixer du code dans ces différents langages simplement et sans overhead : les données sont échangées et partagées entre langages sans être dupliquées. Dans cette optique d'interopérabilité, grCUDA est un langage basé sur Truflle et mis sur pied par NVIDIA afin de simplifier les interactions avec le hardware GPU : ainsi il devient très facile pour des programmeurs d'interagir avec un

GPU dans leur langage de prédilection simplement en faisant des appels polyglottes en grCuda depuis GraalVM. **Figure 3**
En outre, GraalVM Supporte le débogage des applications hébergées et fournit une implémentation du protocole Chrome DevTools. Ceci permet par exemple de debugger n'importe quelle application fonctionnant sous GraalVM depuis l'extension Chrome Developer Tools de son navigateur indépendamment du langage dans lequel le programme soit écrit (C, C++…)

Figure 3



EXEMPLE PRATIQUE

Dans cet exemple, nous présentons un microservice polyglotte permettant de visualiser la courbe des hospitalisations Covid-19 dans n'importe quel département Français. L'exemple met en exergue les fonctionnalités polyglottes de GraalVM à travers l'implémentation de différentes fonctionnalités qui concourent à l'affichage de la courbe en plusieurs langages de programmation dont Java, R, Python, Javascript et LLVM(C). **Figure 3** et **4**

Installer la dernière version de GraalVM GraalVM

```
export GRAALVM_HOME=/Library/Java/JavaVirtualMachines/graalvm-ee-java11-20.3.0/Contents/Home
export JAVA_HOME=$GRAALVM_HOME
```

Installer les addons Python, R et LLVM de GraalVM

```
$GRAALVM_HOME/bin/gu install python R llvm-toolchain
```

Récupérer les sources et les compiler via Maven ou Gradle

```
git clone https://github.com/nelvadas/helidon-polyglot-demo
cd helidon-polyglot-demo
mvn clean package
```

Importer les certificats Sante Publique France dans notre JVM

```
keytool  -importcert \
-file datagouvfr.pem \
-alias datagouvfr \
-keystore $GRAALVM_HOME/lib/security/cacerts \
-storepass changeit -trustcacerts -noprompt
```

Démarrer le microservice via GraalVM

```
$GRAALVM_HOME/bin/java -jar target/helidon-polyglot-demo.jar
```

L'application expose un certain nombre de endpoints REST sur le port 8080

```
2020.12.17 14:03:02 INFO io.helidon.microprofile.server.ServerCdiExtension Thread[main,5,main]: Server started on http://localhost:8080 (and all other host addresses) in 2318 milliseconds (since JVM startup).
```

1 Télécharger le fichier des hospitalisations fourni par Santé Publique France à l'adresse https://www.data.gouv.fr/fr/datasets/ donnees-hospitalieres-relatives-a-lepidemie-de-covid-19/ via une méthode Java.

```
$ curl -X GET http://localhost:8080/covid-19/fr/download
2020.12.17 16:20:18 INFO org.graalvm.demo.CovidResource Thread[helidon-1,5,server]
: localpath =/TMP/COVID-DATA.CSV
```

2 Vérifier le statut du téléchargement (en C).
A tout instant, l'on peut vérifier la présence ou non du fichier par une routine C

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int main() {
 char *filename="/tmp/covid-data.csv";
 printf("Checking the downloaded file!\n");
 struct stat st = {0};
 int status=stat(filename, &st);

 return status;
}
```

Le code en C est premièrement compilé sous forme de bitcode LLVM

```
$LLVM_TOOLCHAIN/clang downloadstatus.c -o downloadstatus
./downloadstatus /tmp/covid-data.csv
Checking the downloaded file!
/tmp/covid-data.csv file attributes : size: 364782 bytes, last modified: 1597935473

curl -X GET http://localhost:8080/covid-19/fr/download/status
```

A l'invocation de l'uri /download/status, un appel polyglotte Java vers LLVM est effectué afin de vérifier le statut du téléchargement.

3 Obtenir les noms de département Java->Python

```
curl -X GET http://localhost:8080/covid-19/fr/department/75

2020.12.17 16:21:23 INFO org.graalvm.demo.CovidResource Thread[helidon-3,5,
server]: Departement 75=PARIS
```

4 Afficher la courbe des hospitalisations pour la région parisienne :

```
Appel Java-Javscript->Python-R

curl -X GET http://localhost:8080/covid-19/fr/trends/75
```

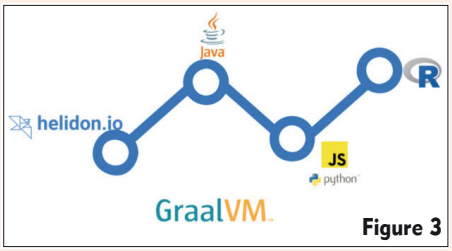


Figure 3

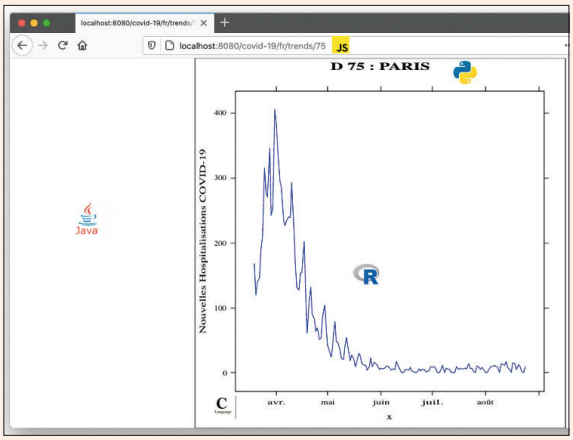


Figure 4

- Le contrôleur reçoit la requête ;
- effectue une validation du numéro de département en Javascript ;
- obtient le nom du département via un appel Python ;
- passe les paramètres au scripts R permettant de produire un graphique du format SVG qui est retourné à l'appelant.

Epilogue

GraalVM représente une fondation solide sur laquelle les entreprises peuvent s'appuyer pour maximiser les performances des applications existantes, uniformiser les machines virtuelles au sein du système d'information, mettre sur pied des microservices Java ultra-performants, adaptés pour le cloud et le serverless en particulier.
Compte tenu des temps de démarrage supersoniques, GraalVM réduit aussi considérablement les coûts d'infrastructure, tout en intégrant plus facilement les langages de programmation pour toujours plus d'innovation.

Code source https://github.com/nelvadas/helidon-polyglot-demo

Dossier codé et compilé par
Philippe BOULANGER
Manager des expertises C/C++ et Python
www.invivoo.com

Programmation & calculatrices

Partie 1

Philippe nous a codé un superbe dossier sur la programmation sur les calculatrices. Depuis plusieurs années, elles supportent le langage Python et certaines peuvent même servir à de la robotique, à interagir avec capteurs, etc. C'est un support idéal pour apprendre les bases de la programmation. La rédaction.

Au début fut le Sinclair QL

C'est à l'âge de 11 ans, déjà passionné par les mathématiques, que j'ai découvert l'informatique après la lecture d'articles dans de vieux Science & Vie : ceux sur l'ordinateur en papier de Turing et ceux sur la programmation de la TI-59. Dès lors, j'ai fait du forcing auprès de mes parents pour acheter un ordinateur familial. Mon père nous a offert un Sinclair QL qui était une machine formidable : un processeur puissant pour l'époque, 128Ko de mémoire vive et un SuperBasic qui était plus proche du Pascal que du Basic du C64. Autodidacte, j'ai appris à le programmer en utilisant le tutoriel fourni. **Figure 1**

En 3e, j'ai commencé à travailler avec une TI 57 II : machine limitée (tant sur les performances que sur la mémoire disponible), mais dont le langage proche d'un assembleur impose la rigueur. **Figure 2**

En 2d j'achetais la Casio FX 730P (calculatrice programmable en BASIC) grâce à laquelle j'ai écrit des programmes (liés aux programmes scolaires) que je revendais aux autres élèves, ce qui m'a rapidement amené à apprendre à programmer toutes les calculatrices utilisées par les autres élèves. **Figure 3**

Dans le même temps, sur mon Tandy 1000 HX, je suis passé du GW-BASIC au TURBO PASCAL, puis à l'assembleur x86. C'est durant cette période que j'ai commencé à collectionner les calculatrices…

Après des études d'ingénieur en informatique à l'ENSEEIHT à Toulouse, j'ai commencé à travailler chez C4W en mars 1998 : nous y avons développé un logiciel de CAO sous Windows. C'est à ce moment que j'ai découvert le C++ puis le Python en 99. Ensuite j'ai rejoint ESI Group pour travailler sur un préprocesseur de crash-test de voiture : c'est l'outil graphique qui permet de faire la mise en données d'une simulation numérique de crash.

Pendant l'été 2004, j'ai quitté ma zone de confort pour rejoindre le monde de l'informatique embarquée. Cela me permit de travailler sur l'optimisation de code tant pour améliorer les performances que pour diminuer la consommation mémoire.

Puis en 2007, j'ai sauté le pas pour l'univers impitoyable de la banque de finance dans une équipe Front Office et vécu la chute de Lehman Brothers en direct dans le desk qui gérait des subprimes. Quelques crises plus tard, j'évolue toujours dans ce domaine qui me permet à la fois de relever des défis techniques motivants en rejoignant Invivoo en 2017 pour gérer les expertises Python et C++. En parallèle depuis 2015, j'anime des formations tant en Mastère Banque-Finance-Assurance à Dauphine qu'en organismes de formation.

Figure 1

Figure 3

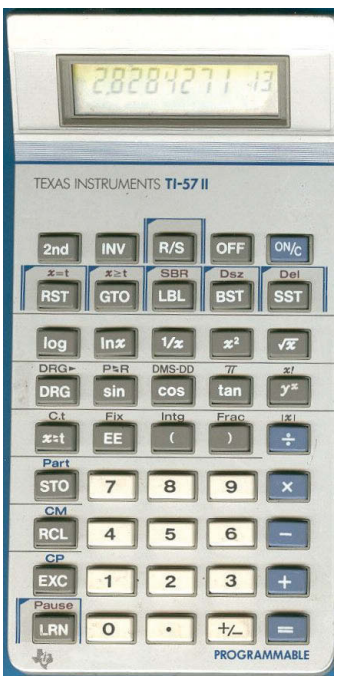


Figure 2

Calculatrices programmables en Python

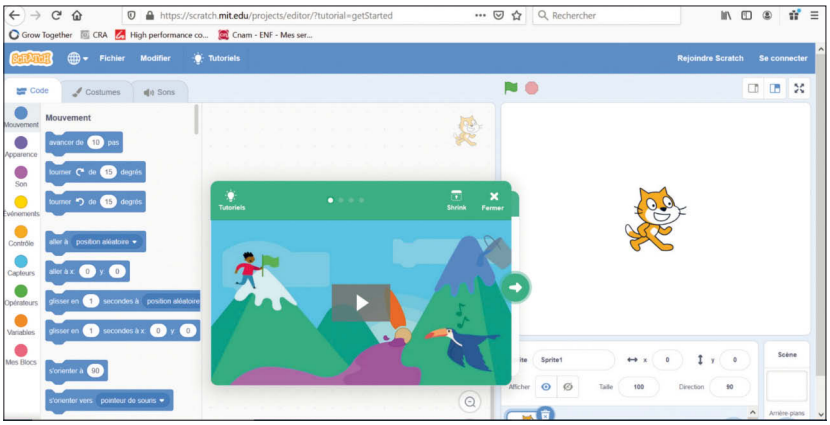
Comme je l'ai expliqué dans mon profil, je collectionne les calculatrices depuis de nombreuses années et en suis régulièrement les développements et les nouveautés. Depuis quelques années de nouveaux modèles sont apparus en France : des calculatrices programmables en Python.

La loi française

Depuis 2012, plusieurs réformes de l'enseignement des mathématiques aux collèges et lycées furent menées par les gouvernements Sarkozy et Macron : elles ont rendu l'enseignement de l'algorithmique et la programmation obligatoire et évaluable au baccalauréat.

Au collège, les élèves apprendront l'algorithmique grâce à de la programmation graphique : une des méthodes proposées est le langage Scratch (https://scratch.mit.edu/) créé par une équipe du MIT. L'avantage de Scratch c'est que l'on peut s'en servir via un navigateur web avec HTML5 et JavaScript ; de plus il permet la création d'histoires interactives, de dessins animés, de jeux, de compositions musicales, de simulations numériques et permet de les partager facilement sur le web… **Figure 1**

Figure 1



Au lycée, dès la classe de seconde, l'apprentissage de l'algorithmique passe par l'apprentissage d'un langage textuel. Plusieurs langages sont proposés dans la circulaire de 2017, mais il semble, aujourd'hui, que Python s'impose petit à petit dans les lycées. Dès lors les principaux fabricants de calculatrices ont proposé des outils adaptés : calculatrices, simulateurs et périphériques associés.

Les calculatrices

Trois marques proposent des calculatrices programmables en Python : Numworks, Texas Instruments et Casio avec deux modèles. Selon un benchmark Python disponible sur le site TI-Planet, le classement en fonction de la performance est :

1. Numworks
2. Casio Graph 90+E
3. Casio Graph 35+E II
4. TI-83 Premium CE Edition Python

Ce benchmark a été effectué avant la sortie des Texas Instruments TI-Nspire™ CX II et TI-Nspire™ CX II CAS qui sont compatibles Python.

Numworks

Cette société française fondée par Romain Goyet est la première à proposer une calculatrice programmable en Python à l'été 2017 (https://www.numworks.com/fr/). Elle est conforme au programme du lycée et possède le mode examen conformément à la loi française. Le hardware est sous licence libre : les schémas et les plans sont disponibles en téléchargement sur le site web. Le code est sous licence open source Creative Commons BY-NC-SA. **Figure 2**

Sur le plan de l'esthétique, la calculatrice est sobre et belle à regarder. À l'usage, l'écran est beau et pratique, les couleurs ressortent bien. Cependant si la luminosité ambiante n'est pas parfaite ou qu'il y a des reflets alors certains petits caractères des touches ressortent mal.

La version de Python choisie est un MicroPython 1.11 (compatible avec Python 3.4) recompilable et reconfigurable par les utilisateurs avertis. Très légère, à peine 167g, elle est équipée d'un processeur 32bits ARMv7 cadencé à 216MHz, de 8Mo de ROM ainsi que de 256Ko de mémoire vive. L'écran de 320x240 pixels permet d'afficher 262144 couleurs. Un simulateur écrit en JavaScript est disponible sur le site https://www.numworks.com/fr/simulateur/ afin de l'essayer avant de décider de l'acheter ou non. Des applications pour les téléphones portables sont aussi disponibles pour les 2 OS majoritaires :

Figure 2

- iOS : https://apps.apple.com/fr/app/numworks/id1456585807
- Android : https://play.google.com/store/apps/details?id=com.numworks.calculator

Par défaut, il n'y a que les modules suivants de disponibles :

- math : qui propose toutes les fonctions courantes sur les nombres flottants ;
- cmath : qui propose toutes les fonctions courantes sur les nombres complexes ;

- random : les générateurs de nombres aléatoires ;
- turtle : la tortue qui permet de faire du dessin (comme dans le langage LOGO) ;
- kandinsky : contient les fonctions basiques de manipulation des pixels graphiques.

Les entiers longs sont supportés par l'interpréteur, ce qui peut permettre de faire des exercices sur des grands nombres premiers ou sur des clés de chiffrement.

Texas Instruments

Texas Instruments s'est aussi adapté au nouveau programme français. Trois possibilités s'offrent aux utilisateurs depuis la rentrée 2019 : la TI83 CE Edition Python ou un adaptateur externe qui apporte Python à une TI 83 CE standard. À la rentrée 2020, de nouvelles versions de TI-Nspire™ CX II-T et TI-Nspire™ CX II-T CAS devraient être disponibles avec une application de programmation Python.

L'esthétique est au rendez-vous ; les touches sont bien lisibles, mais je regrette que certaines fonctions de calcul ne soient pas immédiatement disponibles via les touches (comme sin, cos, tan). Les couleurs de l'écran sont agréables et, par défaut, en mode édition Python, les couleurs sont similaires à ce que proposent bon nombre d'outils PC. Les textes sur le clavier sont en français contrairement à Numworks ou Casio qui ont fait le choix de l'anglais par défaut.

Le processeur de la TI83 CE Edition Python serait toujours un eZ80 8bits à 48 MHz si l'on en croit le site TI-planet (https://ti-planet.org/forum/viewtopic.php?t=22781). La calculatrice a 154Ko de mémoire vive pour 3Mo de ROM. Son écran couleur de 320x240 pixels permet l'affichage de 65000 couleurs. **Figure 3**

Tout comme la calculatrice de Numworks, les entiers longs sont supportés dans l'interpréteur Python permettant ainsi de faire des exercices sur des grands nombres premiers ou sur des clés de chiffrement.

Pour les TI-Nspire™ CX II-T nous manquons d'informations sur les caractéristiques matérielles : la version TI-Nspire™ CX II avait un ARM9 cadencé à 396MHz. Ce que l'on sait c'est qu'elle a 90Mo de stockage et 54Mo de fonctionnement. L'écran a une dimension de 320x240 points avec des couleurs sur 16 bits. Son gros avantage est le clavier alphanumérique qui facilitera la création de programmes. Cependant le choix du tri alphabétique pour le clavier est malheureux : une version qwerty ou azerty aurait été moins déroutante. **Figure 4**

Casio Graph 90+E

Cette calculatrice est sortie en avril 2017, mais la version Python est disponible depuis la rentrée 2018. Elle est équipée d'un processeur Renesas SH4-A à 117.96 MHz, de 16Mo de ROM, mais de seulement 60Ko pour l'utilisateur. Son écran a 396×224 pixels pour 65000 couleurs. En plus de Python, un SDK PrizmSDK permet de développer des add-ins C/C++.

Casio propose un émulateur sous Windows ou macOS : https://www.casio-education.fr/products/fx-cg-manager-plus. Elle pèse 272g. **Figure 5**

Casio Graph 35+E II Python intégré

Disponible depuis la rentrée 2019, cette version noir et blanc de 190g dispose d'un écran de 128x64 pixels. Équipée d'un processeur Renesas SH4 cadencé à 58MHz, elle dispose de 61Ko de mémoire vive ainsi que de 3Mo de ROM. C'est la plus abordable des calculatrices Python sur le marché français. **Figure 6**

Figure 3

Figure 4

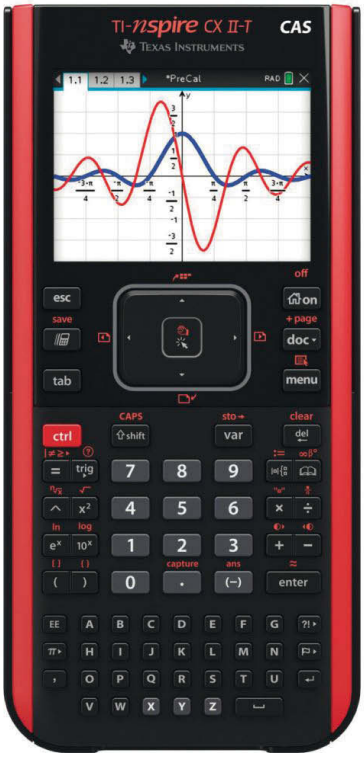


Figure 5

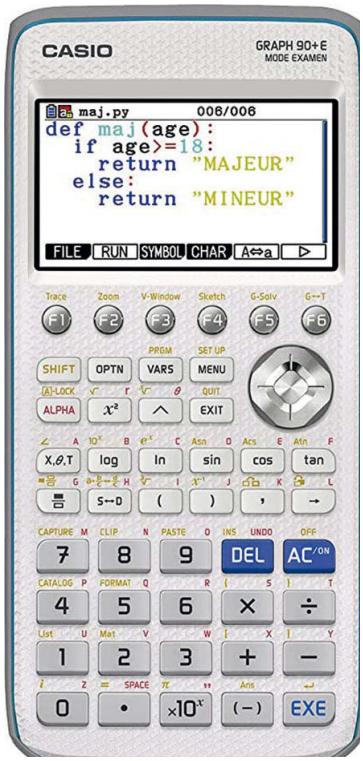


Figure 6

Algorithmes

Vouloir écrire des programmes c'est bien, mais garantir qu'ils fourniront le service attendu, c'est mieux. Un programme est l'implémentation d'un algorithme dans un langage donné. Ne reste plus qu'à savoir ce qu'est un algorithme…

Qu'est-ce qu'un algorithme ?

Définition

Un algorithme est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre un problème ou d'obtenir un résultat.

Le terme algorithme a été inventé très tardivement (au IXe siècle) par un mathématicien perse Al-Khwarizmi. Mais cela faisait bien longtemps que l'on en utilisait.

Ils sont partout

Bien que nous n'en ayons pas forcément conscience, les algorithmes sont partout autour de nous. Ils nous sont naturels. L'exemple le plus simple est la recette de cuisine : **figure 1**

Les étapes de la recette sont bien une suite finie et non ambiguë d'opérations dans le but d'obtenir des crêpes.

Lorsqu'un passant vous demande son chemin, vous lui indiquez la route en lui donnant une suite d'instructions dans le style « Continuez tout droit sur 3 intersections, puis au feu prenez à droite. Avancez sur 50m, ce sera sur votre gauche ». C'est un algorithme !

Mais nous aurions tout aussi bien pu parler du plan de montage d'un meuble : **figure 2**

Les algorithmes mathématiques

La résolution de problèmes mathématiques se prête bien à la création d'algorithmes. D'ailleurs, les plus anciens algorithmes répertoriés sont issus des mathématiques.

L'algorithme de Héron

Cette méthode permet de calculer une approximation de la racine carrée d'un nombre. Elle est encore appelée méthode babylonienne et elle a été redécouverte en 1896 dans un livre d'Héron d'Alexandrie. Ce scientifique l'a lui-même recopié : on a retrouvé une tablette d'argile datant de 1800 à 1600 avant Jésus Christ comportant une approximation de racine carrée de 2.

Calcul du PGCD par Euclide

Euclide a défini en 300 av. J.-C. environ un algorithme permettant de calculer le plus grand commun diviseur. Mais il semblerait que lui-même n'ait que compilé les résultats de mathématiciens ayant vécu avant, selon le mathématicien et historien B. L. van der Waerden.

Calcul de PI par Archimède

Archimède a créé, en 250 av. J.-C., une méthode simple qui permet de calculer une approximation de p grâce à l'aire de polygones inscrits et circonscrits d'un cercle.

Liste des ingrédients

- 250 g de farine tamisée ou fluide
- 4 oeufs
- 450 ml de lait légèrement tiède
- 2 c. à soupe de rhum ambré ou de fleur d'oranger, ou 1/2 verre de bière blonde
- 1 c. à soupe d' extrait de vanille ou 1 sachet de sucre vanillé
- 2 c. à soupe de sucre
- 1 pincée de sel
- 50 g de beurre fondu

Etapes de la recette

1. Faites fondre le beurre au micro-ondes et faites légèrement chauffer le lait qui doit être à peine tiède (ça évite les grumeaux)
2. Mélangez la farine tamisée, le sucre, le sel dans un grand bol. Vous pouvez remplacer 50 g de farine par la fécule de maïs pour plus de légèreté
3. Ajoutez les oeufs, le beurre fondu, puis progressivement le lait, en battant avec un fouet bien pour éviter la formation des grumeaux
4. Ajoutez la bière ou le rhum, ou encore l'eau de fleur d'oranger, la vanille et laissez reposer 30 minutes avant d'attaquer la cuisson
5. Faîtes chauffer une noix de beurre dans la poêle et disposez une louche de pâte
6. Faites cuire vos crêpes de chaque côté, qu'elles soient bien dorées
7. Au fur et à mesure, réservez dans une assiette en couvrant avec une feuille de papier aluminium pour les garder chaudes et moelleuse

Figure 1

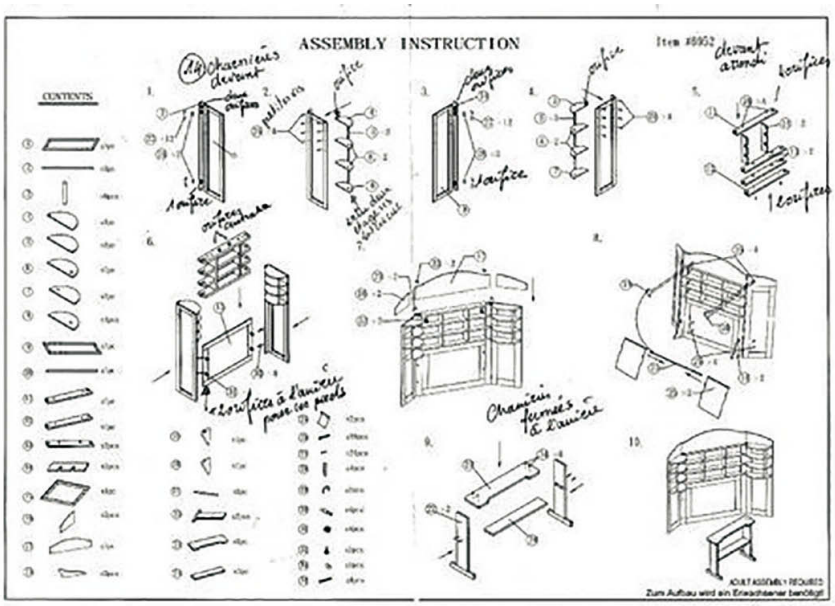


Figure 2

Crible d'Eratosthène

Eratosthène de Cyrène, qui a vécu de -276 à -194, a mis en place une méthode très simple pour calculer tous les nombres premiers entre 1 et n en utilisant que des additions. Nous la mettrons en application dans les chapitres suivants.

La notion d'état

« Un algorithme est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre un problème ou d'obtenir un résultat. »

Si nous prenons l'exemple de la recette de cuisine, chaque

instruction provoque un changement d'état des éléments initiaux : faire fondre le beurre au micro-ondes transforme le beurre solide en beurre liquide. À chaque étape de la recette, nous devons stocker le nouvel état du produit dans un contenant (bol, un saladier, etc.).
Par exemple, nous avons tous une boîte ou un pot sur lequel une étiquette est collée avec marqué dessus « farine ». Cette boîte contient une quantité définie de farine : 1kg au début puis à chaque prélèvement la quantité diminue….

L'état est variable

Dans le cadre d'un programme informatique, nous aurons exactement le même besoin : stocker le résultat de la transformation. Et pour ce faire nous allons utiliser un outil qui nous vient du monde mathématique : une variable. La variable est un peu comme ce pot de farine : c'est une boîte ayant une étiquette avec le nom de la variable indiqué dessus, on y stocke une quantité de données typées (entier ou texte, etc.).

Supposons que l'on souhaite calculer 2 fois le contenu de la variable x plus 3 et stocker le résultat dans la variable y. En langage algorithmique, suivant les conventions, on peut écrire cela de différentes façons :

- 2*x+3→y
- y ← 2*x+3
- y = 2*x+3
- y := 2*x+3

La première de la liste étant celle qui est la plus usitée dans la présentation des algorithmes. Celle avec le '=' est celle qui se rapproche de la plupart des langages de programmation (Python, C/C++, Java, C#, etc.).

Le langage algorithmique

Afin de pouvoir partager un algorithme, il faut avoir une méthode pour le représenter de façon claire et non ambiguë. Un langage textuel proche du langage parlé est très souvent utilisé : on le définit via des structures de contrôle qui ont chacune leur utilité.

Gestion de cas

La première des structures de contrôle correspond à la gestion de cas.

```
SI condition est vraie ALORS
Opération 1
SINON
Opération 2
FIN SI
```

Répétition potentiellement infinie

Répéter une action tant qu'un état n'est pas atteint est quelque chose que nous avons régulièrement besoin de faire. Par exemple, lorsque l'on fait la pâte à crêpes, ***tant qu'il reste des grumeaux, il faut fouetter le mélange***.

```
TANT QUE condition est vraie FAIRE
Opérations
FIN TANT QUE
```

Répétition finie

Lorsque l'on prépare une mousse au chocolat avec 8 œufs, il faut séparer les blancs des jaunes et on écrirait cela : « ***pour chaque œuf, séparer les blancs des jaunes et les réserver (les blancs dans un saladier et les jaunes dans un bol)***».

```
POUR CHAQUE ELEMENT e DE LA LISTE liste FAIRE
action( e )
FIN POUR
```

Une seconde variante est une itération sur des entiers (une incrémentation de 1 du compteur de boucle i est implicite) :

```
1 à s
POUR i DE 1 A n FAIRE
s * i → s
FIN POUR
```

ce qui est strictement équivalent à :

```
1 → s
1 → i
TANT QUE i <= n FAIRE
s * i → s
i + 1 → i
FIN TANT QUE
```

Fonctions

Afin de pouvoir réutiliser une suite d'actions sans avoir à la réécrire, il est utile de pouvoir isoler cet algorithme en le nommant. C'est le rôle de la fonction que de permettre d'isoler un sous-algorithme.

```
FONCTION nom_de_fonction( param_1, param_2, …, param_n )
Action1
Action2
…
Action n
RETOURNE résultat
FIN FONCTION
…
nom_de_fonction( 1, 2, 3, 4 ) → x
```

Le principe de la fonction est que le résultat ne doit dépendre que des paramètres en entrée. Une fonction peut appeler d'autres fonctions ; elle peut même s'appeler elle-même auquel cas elle est dite récursive. Un bon exemple étant la fonction « n!=1*2*3*…*n » :

```
FONCTION factorielle( n )
  SI n < 2 ALORS
    RETOURNE 1
  SINON
    RETOURNE n * factorielle( n-1 )
  FIN SI
FIN FONCTION
```

A vous de coder !

Jouons avec les entiers

Nombre de problèmes mathématiques, et les algorithmes associés, tournent autour des entiers. Bien souvent ces algorithmes sont simples et accessibles à des développeurs débutants.

PGCD

Explications

Le calcul du Plus Grand Commun Diviseur est un classique des mathématiques. L'une des méthodes pour le calculer est tirée du lemme d'Euclide :

Soit un couple d'entiers naturels non nuls (a,b), si des entiers naturels q et r, avec r ≠ 0, sont tels que a = bq + r , alors : PGCD(a,b) = PGCD(b,r).

Pratiquons quelques exemples afin d'en déduire l'algorithme :

PGCD(782, 221)

- 782 = 3 × 221 + 119
- 221 = 1 × 119 + 102
- 119 = 1 × 102 + 17
- 102 = 6 x 17 + 0

PGCD(17, 13)

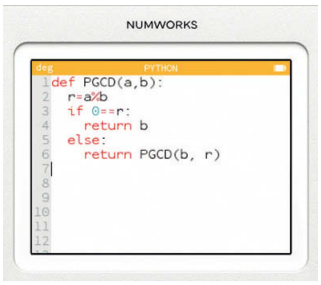
- 17 = 1 x 13 + 4
- 13 = 3 x 4 + 1
- 4 = 4 x 1 + 0

Algorithme récursif

L'algorithme que l'on en déduit est :

```
fonction PGCD(a, b)
   a modulo b → r
   si r est nul alors
      retourne a
   sinon
      retourne PGCD( b, r )
   fin si
fin fonction
```

Cet algorithme est dit récursif, car la fonction PGCD utilise dans son implémentation un appel à elle-même. Écrit en Python, cela nous donne :



Le caractère '%', qui sert à obtenir le reste de la division entière a par b, nécessite une petite astuce pour l'obtenir via le clavier de la calculatrice Numworks. Il faut presser la touche Toolbox( ) puis accéder au menu « Catalogue » pour trouver le caractère.

Algorithme non récursif

La récursivité, en programmation, est souvent inefficace en termes de performance. Heureusement, nous pouvons « dérécursifier » l'algorithme grâce à une boucle ***tant que*** :

```
fonction PGCD(a, b)
   tant que b n'est pas nul
      a modulo b → r
      b → a
      r → b
   fin tant que
   retourne a
fin fonction.
```

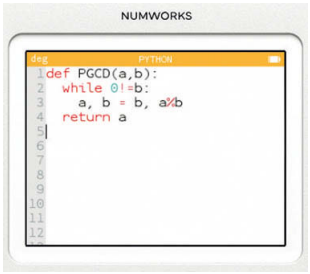
Le packing, au sens des tuples, consiste à construire un tuple à partir de plusieurs valeurs (empaqueter), ce qui se traduit par les exemples suivants :

- t=(2, 3, 4)
- u=2, 3, 4

L'unpacking ou la sortie du paquetage consiste à extraire les données du tuple en les affectant à des variables comme dans l'exemple suivant :

- x, y, z = t
- a, b, c = 2, 6, 7

En profitant de cette astuce sur les tuples ainsi que sur l'évaluation des calculs, nous pouvons écrire en Python :

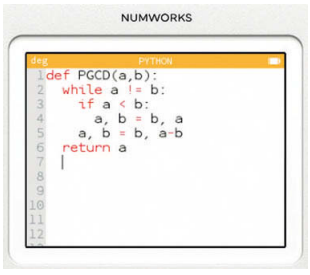


Algorithme sans modulo

Une variante de l'algorithme existe et permet d'éviter les modulos au profit de soustractions dont voici l'algorithme :

```
fonction PGCD(a, b)
   tant que a différent de b
      si a < b alors
            on échange les contenus de a et de b
      fin si
      b-a → r
      b → a
      r → b
   fin tant que
   retourne a
fin fonction
```

Ce qui nous donne en Python :



Le crible d'Eratosthène

Un peu d'histoire

Eratosthène de Cyrène est un grand scientifique grec né en -276 et mort en -194 à Alexandrie, dont il fut directeur de la grande bibliothèque. Il fut le premier à calculer le diamètre de la Terre. Il inventa et nomma une discipline : la géographie (souvent difficile pour les élèves).

Il a travaillé notamment sur les nombres premiers, succédant ainsi à Pythagore. Voici comment on définit les nombres premiers :

Un nombre premier est un entier naturel non nul qui est seulement divisible par 2 entiers distincts : 1 et lui-même.

Il a décrit l'algorithme du crible qui porte son nom et qui permet, facilement, de trouver tous les nombres premiers entre 1 et n. Cet algorithme ne nécessite pas de compétence particulière en mathématiques pour être mis en œuvre : il ne fait appel qu'aux additions sur les entiers.

Exemple

Nous allons mettre en application le crible pour trouver les nombres premiers compris entre 1 et n=99 inclus. Pour cela nous aurons besoin d'une grille :

unités / dizaines	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

Cette grille représente une case pour chaque nombre de 0 à 99 inclus, nous y avons placé les unités en colonnes et les dizaines en lignes. Au démarrage de l'algorithme, toutes les cases sont vides et les cases auront 2 états :

- Vide (fond blanc) : indique un nombre premier
- Pleine (fond gris) : indique un nombre multiple d'un nombre premier

Étape 1 :

Éliminez 0 et 1 qui ne sont pas des nombres premiers en grisant les cases correspondantes. La case pour '0' est sur la ligne pour 0 dizaine et sur la colonne '0' unité.

unités / dizaines	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

Étape 2 :

Nous allons parcourir les cases dans l'ordre à partir de 2. La case '2' est vide : c'est donc un nombre premier. Il nous faut donc éliminer tous les multiples de 2.

unités / dizaines	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

Étape 3 :

Nous passons à la case suivante '3'. La case est vide c'est donc un nombre premier ! Éliminons tous les multiples.

unités / dizaines	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

Étape 4 :

On passe à la case suivante : '4'. Elle est grisée : c'est donc un multiple…

Étape 5 :

On passe à la case '5', elle est vide : c'est donc un nombre premier. Éliminons tous les multiples !

unités / dizaines	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

Étape 6 :

La case '6' est grisée, c'est donc un multiple.

Étape 7 :
La case '7' est vide : c'est donc un nombre premier. Éliminons tous les multiples !

unités / dizaines	0	1	2	3	4	5	6	7	8	9
0	■	■	□	□	■	□	■	□	■	■
1	■	□	■	□	■	■	■	□	■	□
2	■	■	■	□	■	■	■	■	■	□
3	■	□	■	■	■	■	■	□	■	■
4	■	□	■	□	■	■	■	□	■	■
5	■	■	■	□	■	■	■	■	■	□
6	■	□	■	■	■	■	■	□	■	■
7	■	□	■	□	■	■	■	■	■	□
8	■	■	■	□	■	■	■	■	■	□
9	■	■	■	■	■	■	■	□	■	■

Étape 8 :
On passe à la prochaine case non grisée : '11'. Éliminons les multiples.

unités / dizaines	0	1	2	3	4	5	6	7	8	9
0	■	■	□	□	■	□	■	□	■	■
1	■	□	■	□	■	■	■	□	■	□
2	■	■	■	□	■	■	■	■	■	□
3	■	□	■	■	■	■	■	□	■	■
4	■	□	■	□	■	■	■	□	■	■
5	■	■	■	□	■	■	■	■	■	□
6	■	□	■	■	■	■	■	□	■	■
7	■	□	■	□	■	■	■	■	■	□
8	■	■	■	□	■	■	■	■	■	□
9	■	■	■	■	■	■	■	□	■	■

Mais réfléchissons un peu, les multiples de 11 sont :
- 22 = 2 x 11
- 33 = 3 x 11
- 44 = 4 x 11 = 2 x 2 x 11
- 55 = 5 x 11
- 66 = 6 x 11 = 2 x 3 x 11
- 77 = 7 x 11
- 88 = 8 x 11 = 2 x 2 x 2 x 11
- 99 = 9 x 11 = 3 x 3 x 11

Nous avions déjà éliminé les multiples de 2, 3, 5 et 7. En fait le premier multiple de 11 que nous n'avons pas encore éliminé est 11x11 qui est en dehors de notre espace de recherche. Lorsque l'on est en train de travailler sur le nombre premier *p*, partant du principe que nous avons traité tous les nombres premiers plus petits que *p*, le premier multiple à éliminer est donc p^2. On peut donc arrêter l'algorithme dès lors que l'on a passé la case correspondante à la racine carrée de *n* (n=99). Donc, une fois passée la case 10, nous avons déjà éliminé tous les multiples entre 1 et 99 !

Choix des structures de données

Bien choisir les types des données ainsi que les structures dont nous aurons besoin fera une grosse différence entre un algorithme simple à écrire ou un algorithme (inutilement) complexe. Notre grille est, en fait, un tableau numéroté de 0 à 99 inclus : la forme de la grille (2 dimensions) était plus pour faciliter la représentation dans un article (les feuilles ne sont pas très larges).
Les cases ont 2 états : vide (blanche) ou cochée (grisée). En gros, elles sont soit vraies (le nombre est premier) soit fausses (le nombre n'est pas premier). Le type adéquat en Python est donc le type '*bool*' pour les valeurs booléennes (*True* et *False*).

L'algorithme

Voici l'algorithme

```
fonction ListePremiers ( n )
    # initialisation
    n + 1 → N
    tableau de N éléments initialisés à Vrai → t
    Faux → t[ 0 ]
    Faux → t[ 1 ]
    Racine carrée de N convertit en entier → last

    # boucle d'élimination
    pour i de 2 à last
        si t[ i ] est vrai alors
            pour j de i*i à N avec un pas de i
                faux → t[ j ]
            fin pour
        fin si
    fin Pour

    # récupération du résultat
    tableau vide → resultat
    pour i de 2 à N
        si t[ i ] est vrai alors
            ajoute i dans resultat
        fin si
    fin pour
    retourne resultat
fin fonction
```

Cela peut sembler long et compliqué, mais, en Python, on peut l'écrire de façon plus concise grâce aux astuces syntaxiques du langage. En effet, la création d'un tableau de n valeurs initialisé à une valeur simple x (booléen, entier, flottants, par exemple) s'écrit en Python : [x]*n. Cela permet d'éviter une boucle d'initialisation.
De même, une fois les éliminations des multiples effectuées, le filtrage pour récupérer les nombres premiers est simplifié par l'usage d'une liste par compréhension [i **for** i **in** range(2, N) **if** t[i]].

```
from math import sqrt

def ListePremiers( n ):
    # initialisation
    N=n+1
    last=int(sqrt(N))
    t=[True]*N

    # boucle d'elimination
    for i in range(2, last):
        if t[i]:
            for j in range(i*i, N, i):
```

```
            t[j]=False

    # retourne le resultat
    return [i for i in range(2, N) if t[i]]
```

Décomposer en produit de nombres premiers

Tout nombre entier peut se décomposer en un produit de nombres premiers. Si Euclide en avait jeté les bases, c'est Carl Friedrich Gauss qui en finalisa la démonstration. Par une méthode de force brute (on va parcourir tous les entiers inférieurs au nombre à décomposer), il est assez facile d'extraire les facteurs. En premier lieu, il nous faut savoir si un nombre entier est divisible, au sens euclidien du terme, par un autre entier.

Théorème de la division euclidienne

Le théorème de la division euclidienne dans les entiers naturels (les nombres entiers pris à partir de 0) s'énonce ainsi :

À deux entiers naturels *a* et *b*, $b \neq 0$, on associe de façon unique deux entiers naturels, le quotient *q* et le reste *r*, qui vérifient :

- $a = bq + r$
- $r < b$

L'unicité du reste se déduit de ce que le seul multiple de *b* strictement inférieur à *b* est 0, or la différence de deux restes vérifiant les deux conditions est un tel entier. L'unicité du quotient se déduit de celle du reste.

Est divisible par

Etant donné deux entiers *a* et *b*, en Python les résultats de la division euclidienne peuvent s'obtenir de différentes façons :

- ***q* = *a* // *b*** retourne le quotient de la division euclidienne
- ***r* = *a*%*b*** retourne le reste de la division euclidienne
- ***q*, *r* = *divmod(a, b)*** retourne en même temps le quotient et le reste

Déterminer si *a* est divisible par *b* consiste donc à vérifier que le reste *r* de la division euclidienne de *a* par *b* est nul. Ce qui se traduit en Python par : ***a* % *b* == 0**.

Diviser pour régner

La décomposition va se faire par la méthode de la force brute : nous essayerons tous les nombres premiers dans l'ordre croissant en partant de 2. Pour des raisons pratiques, nous ne décomposerons que des nombres entiers positifs ; nous mettons une sécurité pour éliminer les nombres négatifs en retournant ***None***. Nous allons tester toutes les valeurs comprises entre 2 et $\sqrt{n}$ si n est supérieur ou égal à 2. A chaque fois que l'on trouve un diviseur, on l'ajoutera dans le tableau résultat puis on mettra à jour la valeur de ***n*** et cela « déplacera » mécaniquement la condition d'arrêt de l'algorithme.

```
from math import sqrt

def decompose( n ):
    if n < 2:
        return None

    result = []
    i    = 2
    while i <= sqrt( n ):
        if n % i == 0: # on a un diviseur
            result.append( i )
            n //= i
        elif i >= 3: # pour iterer sur les premiers impairs
            i += 2
        else: # pour passer à 3
            i += 1

    if n > 1: # il reste une valeur qui sera un nombre premier
        result.append( n )
    return result
```

La dernière condition « si *n* est strictement supérieur à 1 » est là pour gérer le cas $n=3$ et donc $\sqrt{n}=1.73...$ Lorsque $i=2$ on sort de la boucle « tant que », mais « *n* » est toujours égal à 3. Vu que la valeur n'a pas été divisée par les nombres premiers plus petits que *n*, c'est que celle-ci est un nombre premier. D'où le ramasse-miette avec le test « si *n* est strictement plus grand que 1 » pour ajouter la dernière valeur à la liste résultat.

Triangle de Pascal

Explication

Bien qu'associé en occident aux travaux de Pascal, il fut étudié bien avant par d'autres mathématiciens notamment en Chine où il est appelé triangle de Yang Hui. Il permet de calculer facilement les coefficients de $(x+1)^n$. Voici un exemple des premières lignes de ce triangle :

		x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8
0	$(x+1)^0$	1								
1	$(x+1)^1$	1	1							
2	$(x+1)^2$	1	2	1						
3	$(x+1)^3$	1	3	3	1					
4	$(x+1)^4$	1	4	6	4	1				
5	$(x+1)^5$	1	5	10	10	5	1			
6	$(x+1)^6$	1	6	15	20	15	6	1		
7	$(x+1)^7$	1	7	21	35	35	21	7	1	
8	$(x+1)^8$	1	8	28	56	70	56	28	8	1

Donc, comment lire ce tableau, si je veux calculer $(x+1)^4$, je vais lire la ligne commençant par un 4 et en déduire que : $(x+1)^4 = 1.x^0 + 4.x^1 + 6.x^2 + 4.x^3 + x^4 = 1 + 4.x + 6x^2 + 4x^3 + x^4$

Dans sa démonstration (celle de Pascal), si nous appelons $C_{n,k}$ les coefficients de la ligne *n* et de la colonne *k*, ils se déduisent en suivant les règles suivantes :

- $C_{n,0} = 1$
- $C_{n,n} = 1$
- $C_{n,k} = C_{n-1,k-1} + C_{n-1,k}$

Algorithme de calcul

Nous allons créer une fonction qui calculera tous les coefficients pour une valeur de *n* donnée. Et pour ce faire, nous cal-

culerons toutes les lignes depuis la ligne 0 jusqu'à la ligne *n*. Dans cet algorithme t sera un tableau qui contiendra les $C_{n,k}$ déjà calculés et newT servira à calculer les $C_{n+1,\,k}$.

```
fonction PascalCoefficients( n )
   [1] → t
   pour i de 1 à n
      tableau de i+1 valeurs initialisées à 1 → newT
      on ajoute 0 à la fin de t
      pour j de 1 à i-1
         t[ j – 1 ] + t[ j ] → newT[ j ]
      fin pour
      newT → t
   fin pour
   retourne t
end fonction
```

En transformant cet algorithme sans se poser de question, on obtiendrait :

```
def PascalCoefficients0( n ):
   t = [ 1 ]
   for i in range( 1, n + 1 ):
      newT = [ 1 ]
      t.append( 0 )
      for j in range( 1, i ):
         newT.append( t[ j - 1 ] + t[ j ] )
      newT.append( 1 )
      t = newT
   return t
```

L'un des intérêts de Python est sa syntaxe évoluée et « très » compacte. Nous allons grâce aux fonctionnalités des listes (et notamment des listes par compréhension et de la concaténation des listes) écrire l'algorithme de la manière la plus compacte possible :

```
def PascalCoefficients( n ):
   t = [ 1 ]
   for i in range( 1, n + 1 ):
      t = [ 1 ] + [ t[ j - 1 ] + t[ j ] for j in range( 1, i ) ] + [ 1 ]
   return t
```

Les constructeurs misent sur l'ouverture et Python

Python s'est imposé comme le langage utilisé dans les écoles pour la programmation et l'utilisation des capteurs externes. Pour 2021, les constructeurs préparent de belles nouveautés, certaines sont d'ores et déjà disponibles.

Texas Instruments

Pour septembre 2021, le constructeur va proposer une TI-82 Advanced Python. La TI-82 est la dernière nouveauté du constructeur. Elle se destine à un public large et plus polyvalent. Elle supporte Python 3.x. TI a fait un gros effort pour généraliser l'écran couleur 320x240, avoir un meilleur support graphique et apporter plus de confort pour coder en Python.

Depuis la rentrée 2020, TI propose sur certaines calculatrices le support de la carte BBC Micro:bit ! La Micro:bit est une carte anglaise dédiée au monde de l'éducation et aux jeunes. Elle embarque des capteurs et elle se programme facilement notamment en Python ! On code sur la calculatrice et on interagit en temps réel avec la carte pour exécuter le code. L'avantage de la micro:bit est son prix et qu'elle soit présente dans de nombreuses écoles. On connecte directement la calculatrice à la micro:bit. TI a développé plusieurs modules / librairies pour pouvoir coder les capteurs et les usages. Actuellement, la partie communication sans fil comme le Bluetooth n'est pas supportée. Toutes les librairies ne sont pas compatibles avec les TI. Il faut utiliser celles livrées par TI. Compatibilité micro:bit : TI-83 Premium CE Edition Python, TI-Nspire CX II-T & TI-Nspire CX II-T CAS

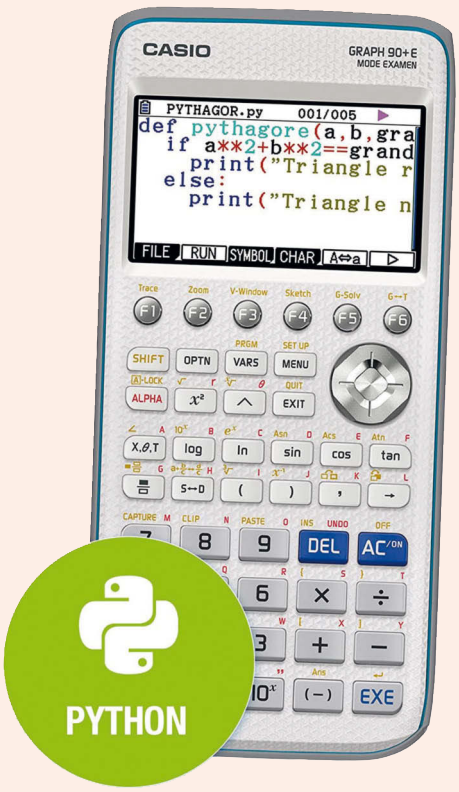
Est-ce la mort de l'excellent TI-innovator Hub et du support des capteurs Grove ? Comme nous l'a précisé Carlos Coelho (Texas Instruments France), le support de la micro:bit ne remplacera ni le hub, ni le rover. C'est un usage complémentaire. À noter que le TI-innovator Hub et le rover sont maintenant programmables en Python !

Est-ce que d'autres cartes seront supportées ? Pour le moment, non. Ce sera une question d'opportunité et d'usages.

Pour aller plus loin :
https://education.ti.com/fr/ressources-et-formations/microbit

Les nouveautés côté Casio

Le constructeur supporte MicroPython 1.9.4. Deux modèles supportent le langage : Graph 35+E II et Graph 90+E. La Graph 90+E propose un écran couleur. Plusieurs modules ont été développés par Casio : Matplotlib, Turtle. La connexion avec les calculatrices se fait par un simple câble USB fourni dans la boîte. Par défaut, les librairies math et random sont installées. Pour le moment, il faut installer les autres librairies via une mise à jour gratuite disponible sur notre site web CASIO-Education.fr.



Les codes ne doivent pas dépasser 300 lignes. Il faut aussi que les noms des fichiers ne soient pas réservés, ni commencer par un chiffre. Il est possible d'utiliser n'importe quel éditeur puis on transfère les fichiers .py sur la calculatrice. Pour étendre les usages, notamment en Physique / Chimie, il est possible d'interfacer des capteurs CLAB.

Stéphane HEUZÉ

Architecte Web chez Kaliop, conçoit des systèmes de données avec Drupal. Son but, offrir à ses clients les refontes, migrations, spécifications et conceptions les plus folles, des expériences Web merveilleuses, et à ses développeurs des migraines d'autant plus monstrueuses.

Drupal : une goutte d'eau apparue il y a 20 ans

L'accès au Web est partout. Les moyens de communication actuels permettent d'accéder à une information très riche extrêmement aisément. Fournir une vitrine à ses activités professionnelles, associatives, commerciales ou même à de simples notes de blogs sur ses passions est devenu relativement incontournable pour se faire connaître et trouver.

La démocratisation de l'accès au contenu a découlé de la nécessité d'en produire toujours plus facilement. Les systèmes de gestion de contenu (CMS) sont une réponse à cette demande : mettre en place un site web complet - autant l'affichage des informations aux utilisateurs, le Front Office, que leur administration, le Back Office - est devenu une tâche aisée. On trouve un nom de domaine, on trouve un serveur gratuit ou payant avec sa base de données, et il ne reste plus qu'à télécharger une archive et souvent, le tour est joué. On suit l'assistant qui nous guide tranquillement et quelques minutes plus tard on peut créer ses premiers contenus publiables aux yeux de tous.

D'après w3techs.com, début 2021, 62% du Web est composé de sites web propulsés par des CMS. Si le plus célèbre d'entre eux, Wordpress, accapare 64,3% des parts de marché du CMS, un autre a obtenu ses lettres de noblesse au cours des années passées.

Drupal, sorti il y a 20 ans, le 15 janvier 2021, est aujourd'hui dans le top 5 des CMS les plus utilisés.

Au-delà de sa gratuité, il a su s'imposer comme une solution fiable et performante, qui sait s'adapter aux évolutions technologiques. En 2015, la sortie de sa version 8 avait marqué sa communauté en passant à un modèle de programmation Objet, basé sur des briques du Framework PHP Symfony et du moteur de template TWIG. En juin dernier, la version 9 arrive.

Sans un lot de nouveautés flagrantes par rapport à la précédente release, la 8.9, c'est toute une philosophie consistant à inscrire le gestionnaire de contenu dans un calendrier global basé sur les sorties des nouvelles versions des technologies standards du Web : Apache, Twig, la JSON:API ou Symfony pour ne citer qu'elles.

Dans ce dossier, nous verrons ce qui rend Drupal si particulier. De son ancrage dans une communauté libre d'appartenance à une quelconque entreprise à une évolution sur le long terme de ses concepts phares. Après nous être attardés sur les spécificités techniques qui différencient Drupal de ses concurrents et qui le rendent si adaptable pour répondre à des problématiques métier complexes, nous verrons comment la solution a mis en place des structures avancées de caches pour répondre aux problématiques déterminantes de performances de chargement des pages. Nous évoquerons également comment Drupal offre des approches radicalement différentes pour répondre aux façons de consommer les données sur Internet, notamment grâce à un système de découplage de l'interface et de l'administration.

Drupal 9 : l'âge de la maturité

Beaucoup d'utilisateurs et propriétaires de sites et plateformes Drupal se souviennent avec un goût amer du passage de la version 7 à la version 8. En effet, comme nous l'évoquions précédemment, le changement de paradigme de programmation, couplé au changement de moteur de template, avait forcé à une refonte totale des sites, sans pouvoir conserver, dans l'immense majorité des cas, ni les contenus, ni les modules ou thèmes développés. Désormais, tout ceci est loin derrière nous. Nous allons voir comment Drupal a changé sa philosophie pour qu'à partir de Drupal 8, passer à une version supérieure ne soit qu'une formalité.

Stéphane HEUZÉ

Architecte Web chez Kaliop, conçoit des systèmes de données avec Drupal. Son but, offrir à ses clients les refontes, migrations, spécifications et conceptions les plus folles, des expériences Web merveilleuses, et à ses développeurs des migraines d'autant plus monstrueuses.

Une nouvelle approche « Lean »

Dans un souci d'offrir une approche « lean », la Communauté a opté pour une méthode itérative : on garde le meilleur, on enlève ce qui ne correspond plus aux normes. Le core de la mouture la plus récente embarque les services les plus déterminants apparus dans la version précédente.

Cette approche assure également la compatibilité avec les dernières versions des librairies externes (jQuery, Ckeditor, Symfony) et solutions logicielles (PHP, Composer, Guzzle), mais aussi un nettoyage du code en dépréciant les instructions désuètes. **Figure 1**

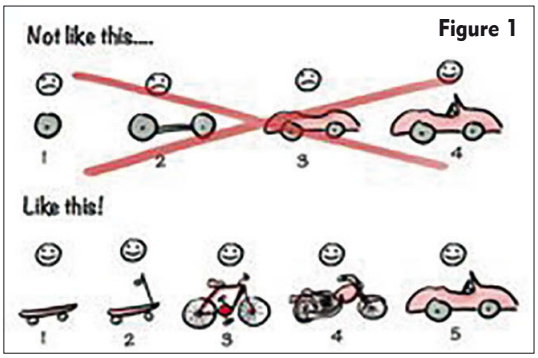


Figure 1

Henrik Kniberg https://blog.crisp.se/2014/10/08/henrikkniberg/what-is-scrum CC

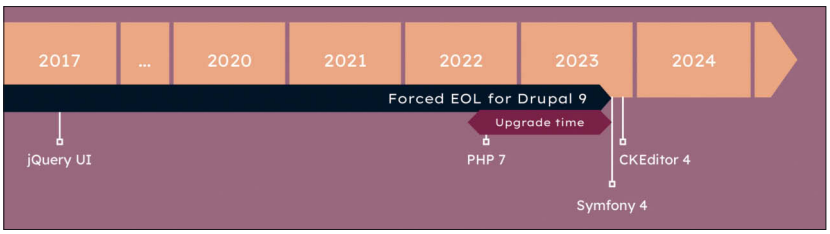
Une itération régulière pour mettre en avant le meilleur et déprécier le désuet.

Une « end of life » maîtrisée

Une promesse : plus de refonte nécessaire, juste des mises à jour régulières sans douleur, complètement embarquées dans la maintenance des sites. Grâce à un planning fixé sur plusieurs années, plus de surprises de dernière minute, de changements majeurs impromptus ou montées de version inattendues.

Dès maintenant, il est établi que, pour des raisons de sécurité, les administrateurs de sites Drupal devront passer à Drupal 10 d'ici le 10 novembre 2023. Soit plus d'un an après la sortie de ce dernier (annoncée pour juin 2022). D'ici là, des releases mineures seront publiées tous les six mois.

Tout est anticipé dès aujourd'hui. L'installation du CMS suit les standards du Web, sans être obligé de repenser son site. Des standards du Web, Drupal en consomme. D9 s'appuie sur Symfony 4 et PHP 7.3 à minima, ce qui renforce la sécurité et la stabilité des instances. Demain, les nouvelles versions de ces solutions seront gérées. **Figure 2**



Drupal 9 end of life. State of Drupal July 2020 CC

Figure 2

Le meilleur de Drupal 8 préservé

Drupal 8 a fait découvrir trois grandes fonctionnalités devenues incontournables pour proposer à l'ensemble des groupes (côté utilisation, administration et développement) plus de simplicité d'usage et une interface toujours plus flexible. En back-office, le module Media déploie une bibliothèque de médias claire et performante, avec une intégration à 99 % du Wysiwyg CKeditor, pour une organisation efficace de vos documents. **Figure 3**

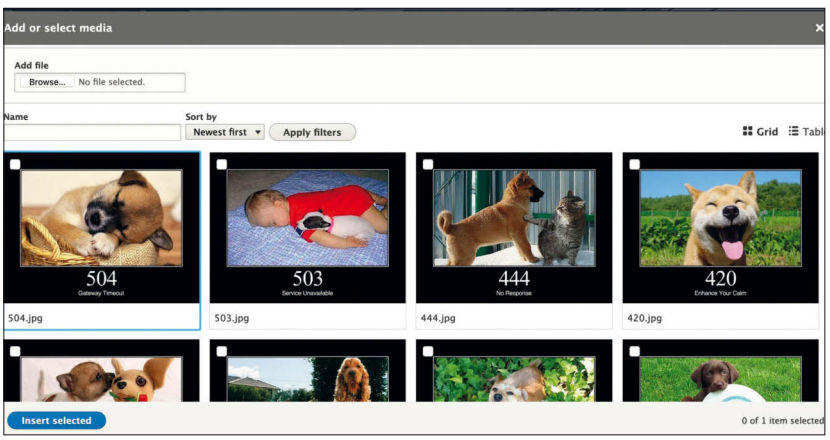
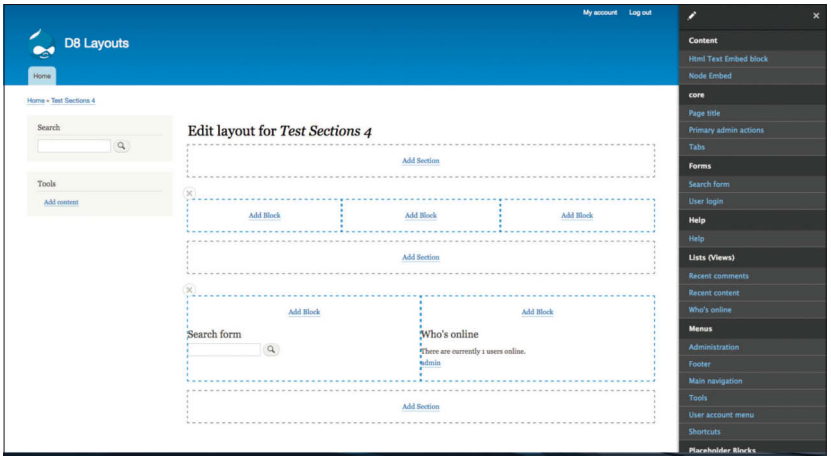


Figure 3

Pour offrir une expérience éditoriale aux éditeurs de contenu, le gestionnaire de modèle de pages « Layout Builder » leur permet de créer aisément des structures d'éléments et de positionner à la souris leurs blocs de manière ergonomique et rapide. **Figure 4**

Pour que l'environnement devienne un carrefour de communication fortement connecté vers l'extérieur, Drupal 9 est pensé API-first. Les développeur-euse-s pourront toujours plus facilement recevoir et transmettre les données depuis et vers tout type de plateformes externes (CRM, ERP, Interfaces graphiques Node.Js ou React…) depuis votre usine à sites, votre intranet ou votre vitrine.

Figure 4



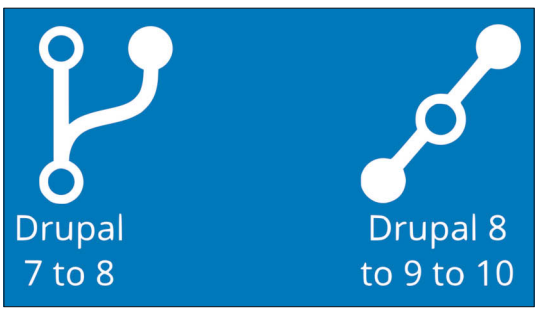


Figure 5

De D7 à D10 https://slides.com/gaborhojtsy/state-of-drupal9, *Gábor Hojtsy CC*

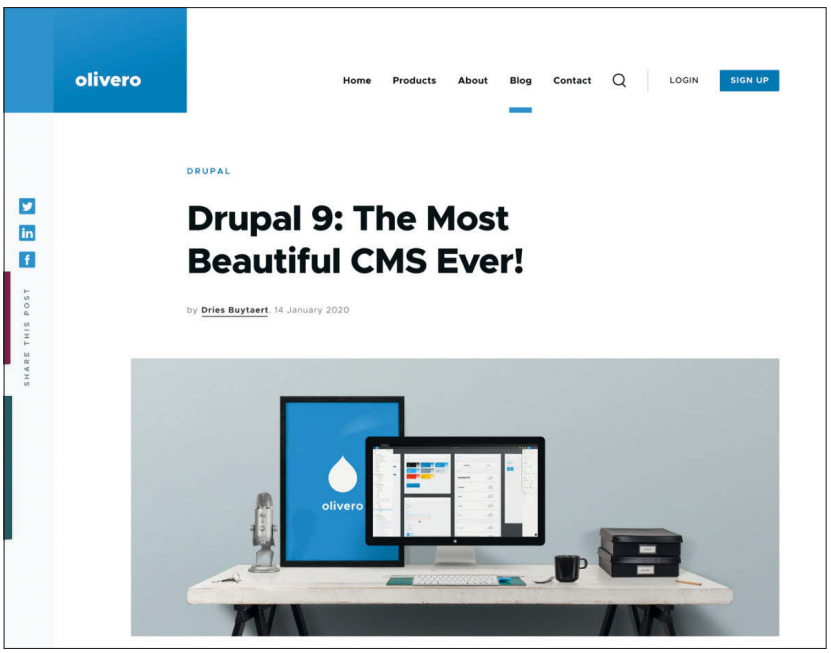


Figure 6 *Le thème Olivero State of Drupal July 2020*

L'ascension vers la pérennité

Pour reprendre les propos de Dries Buytaert, créateur du système, la montée de version de la 8 à la 9 devient la plus aisée des quinze dernières années d'existence du CMS. Un audit a démontré qu'un tiers des modules contribués ont passé le test de compatibilité à Drupal 9. Un autre tiers ne nécessite que la modification d'une ligne de code (dans le. info.yml pour celles et ceux nourris dans le sérail). **Figure 5**

Dans l'objectif avoué de préserver « la pertinence et l'impact de Drupal », l'accent a été mis sur l'aisance d'utilisation pour les Site Builders et les contributeurs. Mais également sur la facilité de prise en main, grâce à un système "out of the box" intégrant la migration de contenus.

Des solutions techniques comme Composer, le gestionnaire de configuration et les éléments déjà évoqués précédemment tels que le plan de releases biannuel permettent de réduire les coûts de maintenance pour les personnes qui développent ou détiennent des sites.

Le futur déjà entre vos mains

Dorénavant, il n'y aura à réaliser régulièrement que les montées de version des composants installés et le passage à Drupal 10 est déjà assuré. Des modules assistent déjà à la tenue à jour du code source. Ils s'assurent qu'aucune application bientôt dépréciée n'est utilisée (Upgrade Status) et même "conseillent" sur les modifications à apporter (Upgrade Rector).

D'ici la sortie de Drupal 10, la Communauté s'engage à ce que les plateformes intègrent de nouvelles fonctionnalités majeures. **Figure 6**

- Drupal pourra être complètement découplé et présenté dans une interface Vue.Js ou React, avec un menu entièrement intégrable.
- Pour une expérience UX/UI toujours plus fluide, un nouveau thème d'administration sera créé.
- Le front ne sera pas en reste avec le thème Olivero, également inclus nativement.
- Et le clou du spectacle viendra des mises à jour de sécurité et de modules automatiques.

CONCLUSION

Si vous avez déjà franchi le pas vers Drupal 8, alors n'hésitez plus. En accédant à Drupal 9, vous entrez dans une nouvelle phase de vie de votre projet. Embarquez dans une roadmap parfaitement jalonnée, vous serez dès à présent engagé dans une solution moderne, avançant au rythme des librairies et logiciels qui font le web. Votre plateforme, dernier cri et multicanal, avec l'ergonomie, tant Back que Front, digne des plus grands de l'Internet, aura le bon goût de vous faire oublier le mauvais souvenir des mises à jour coûteuses.

Quelques mots sur la Communauté et l'Open Source

Stéphane HEUZÉ

Architecte Web chez Kaliop, conçoit des systèmes de données avec Drupal. Son but, offrir à ses clients les refontes, migrations, spécifications et conceptions les plus folles, des expériences Web merveilleuses, et à ses développeurs des migraines d'autant plus monstrueuses.

Drupal, comme la majorité des CMS, est gratuit. Mais il a une particularité bien à lui, c'est qu'il est totalement libre. Contrairement à d'autres solutions, il n'est pas maintenu par une entreprise au business model plus ou moins engagé dans de la facturation freemium. C'est sa communauté qui fait avancer Drupal au fil de ses versions majeures.
Des comités ouverts se réunissent pour discuter des sujets d'avenir, des comités fermés pour les sujets sensibles comme la sécurité, comités que l'on peut rejoindre sur candidature… Tout le reste se passe dans les évènements officiels, les réunions d'associations des différents pays… Et force est de constater que le système fonctionne. Nous vous proposons un petit tour d'horizon du succès d'un des plus grands projets Open Source au monde.

Drupal est profondément inscrit dans la philosophie libriste. Il n'y a pas de hiérarchie ou de grand décisionnaire. La Communauté fait ses choix. Les modules qui sont intégrés au Core de la solution (sa base technologique) à chaque version majeure, les orientations technologiques (par exemple le choix de la technologie pour les API), sont décidés par des discussions où chacun peut intervenir et argumenter. Tout se passe sur le site drupal.org. Bien que la société Acquia, fondée par Jay Batson et Dries Buytaert, concepteur de Drupal, soit très engagée dans le maintien de la solution, elle n'a pas une voix prépondérante.
Les avancées se font autant par les grosses entreprises contributrices que par les individus engagés. D'après les informations publiées par Dries Buytaert sur son blog (dri.es), des informations intéressantes sont à noter sur la bonne santé de ce modèle Open Source.

Une contribution active

En premier lieu, bien que la majorité des contributeurs soit encore des individus volontaires, on constate que de plus en plus de contributions sont sponsorisées par des entreprises. Ces dernières saisissent l'intérêt qu'elles peuvent avoir à ne pas être simple consommateur de l'Open Source, mais bien à associer leur nom aux nombreux développements qu'elles entreprennent, en tant qu'intégrateur de la solution, au nom de leur client final ou par les équipes internes de ces entreprises. Voici quelques chiffres qui permettent de juger de la bonne santé de la solution et de son modèle Open Source.
Ces chiffres sont issus des relevés sur la période de 12 mois, du 1er juillet 2019 au 30 juin 2020 sur drupal.org et présentés sur le blog dries.es
Durant cette période Drupal, c'est :

- 31 153 *issues* fermées ou résolues, soit une augmentation de 13% par rapport à l'année d'avant.
- Soit 85 tickets traités par jour.
- 4 195 projets différents ont reçu des contributions (+20% par rapport à l'année précédente), dont 66% sur des modules contribués.
- On recense 8 303 individus qui contribuent et 1 216 sociétés.
- Le Top 30 des contributeurs produisent 20% du volume total.
- 50% des individus n'ont qu'un seul *commit* à leur actif.
- 15% des contributions sont réalisées par des volontaires, 69% sont annoncées comme sponsorisées.

À travers ces chiffres dont on ne peut que constater la croissance, signe de santé, on note un modèle nécessitant un petit nombre de gros contributeurs, mais gagnant beaucoup des nombreuses petites contributions.
Pour gérer l'ensemble de la production de code du système modulaire qu'est Drupal, le site drupal.org est l'alpha et l'omega. En plus de la documentation, on y trouve un système de ticketing très similaire à celui de GitHub. C'est ainsi que tout peut être suivi de manière asynchrone par les développeurs et contributeurs. Y sont évoquées dans des pages indépendantes les anomalies sur le Core ou les modules contribués (versés à la Communauté), les demandes d'ajout de fonctionnalités ou les nouvelles idées. On y trouve également les tâches à effectuer. Un système simple, bien connu des développeurs de tous horizons, mais qui s'avère particulièrement efficace pour retrouver l'information correctement nomenclaturée. La Communauté a mis en place un nouveau système pour inciter les utilisateurs de Drupal à contribuer. Ces derniers peuvent, pour chaque développement qu'ils poussent, chaque module ou patch proposé, spécifier s'ils le font en leur nom propre ou bien s'ils le font à travers la société qui les embauche et préciser si c'est pour un client final (**Figure 1**). Pour permettre à chacun de se jauger et aux entreprises contributrices de mettre en avant leur implication libre, un système de score est désormais en place, avec un classement des contributeurs sponsorisés ou non. Ainsi, pour chaque contribution validée par la Communauté et répondant aux critères de qualité de drupal.org (vérifié par des tests unitaires puis une approbation par un pair) son contributeur et la société qui sponsorise gagnent chacun un point. Ce score est ensuite pondéré par le poids attribué au module ciblé par le patch. Si le développement touche le coeur, le point reçu sera multiplié par 10. Si c'est un module contribué fortement téléchargé, il gagnera 1 point par tranche de 100 000 installations référencées du module. Par exemple, un patch approuvé sur le module de gestion de formulaire Webform, installé 470 000 fois rapporte 4,7 à son contributeur.
Ainsi, la rétribution que recevront contributeurs et entreprises sera la popularité et l'exposition. Sans contre-partie pecuniaire, chacun peut trouver de l'intérêt à enrichir Drupal.
Bien évidemment, le système n'est pas parfait. Il est possible de tricher légèrement en faisant des petits *comits*, pauvres en complexité ou en qualité, mais comme il n'y a pas de retour financier direct, l'intérêt est limité et la triche, naturellement régulée.
Drupal, c'est donc un exemple réussi d'un système horizontal et libre, qui s'enrichit de la variété de ses contributeurs et qui a réussi à mettre en avant l'intérêt de ne pas être simple consommateur de l'Open Source.

Figure 1

Extrait du blog de dri.es. Creative commons/. On voit sur cette image que le développeur Jamadar à réaliser ce patch, en tant qu'employé de TATA Consultancy Services alors qu'il travaillait sur le projet de leur client Pfizer.

Romain DALVERNY

Spécialisé en développement web chez Kaliop, plus particulièrement avec le logiciel Drupal et champion du Puissance 4, je sais aligner les modules comme personne afin de répondre aux besoins de mes clients.

Les concepts de programmation dans Drupal

Durant mes années d'expérience, j'ai pu constater une certaine défiance envers Drupal de la part de développeurs utilisant d'autres technologies. Ce logiciel a une image de CMS click click ou de style de programmation PHP procédurale. Avec Drupal 8, de nombreux paradigmes comme la programmation orientée objet (POO) et les design patterns ont été introduits par des librairies externes. Ces derniers permettent à Drupal d'utiliser les mêmes standards que l'ensemble des frameworks modernes.

Module

Lorsqu'on l'installe Drupal, un ensemble de **modules l'accompagne automatiquement.** Ces modules regroupent un ensemble de logiques de code pouvant apporter une nouvelle fonctionnalité ou en étendre une déjà existante. Ils ont évolué, passant d'une programmation de type procédurale dans Drupal 7 à une approche objet afin de se rapprocher des standards actuels. C'est similaire au concept de *bundle* dans Symfony. Prenons le temps d'aborder quelques concepts disponibles au sein d'un module en analysant le module « block" du cœur de Drupal.

```
block
|—— block.api.php
|—— block.info.yml
|—— block.install
|—— block.libraries.yml
|—— block.links.contextual.yml
|—— block.links.menu.yml
|—— block.links.task.yml
|—— block.module
|—— block.permissions.yml
|—— block.post_update.php
|—— block.routing.yml
|—— block.services.yml
|—— config/
|—— css/
|—— js/
|—— migrations/
|—— src
|       |—— BlockAccessControlHandler.php
|       |—— BlockForm.php
|       |—— BlockInterface.php
|       |—— BlockListBuilder.php
|       |—— BlockPluginCollection.php
|       |—— BlockRepositoryInterface.php
|       |—— BlockRepository.php
|       |—— BlockViewBuilder.php
|       |—— Controller/
|       |—— Entity/
|       |—— EventSubscriber/
|       |—— Form/
|       |—— Plugin/
|       |—— Tests/
|       |—— Theme/
|—— templates/
|—— tests/
```

On peut voir des contrôleurs « block/src/Controller/", un système de routing "block/block.routing.yml", la définition de services "block/block.services.yml", des entités "block/src/Entity/", des assets "block/css/" "block/js/"et même des dépendances à des librairies externes.

Une des notions fondamentales de Drupal est le **HOOK**. De type procédural, il va permettre aux modules d'interagir. Lorsqu'un module va mettre à disposition un hook, les autres modules vont pouvoir l'implémenter pour y ajouter de nouvelles fonctionnalités ou modifier les existantes.
Concrètement, lorsqu'un hook est déclenché, il va parcourir l'ensemble des modules et thèmes qui l'ont surchargé. Pour ce faire, il faut respecter une certaine nomenclature 'my_module_hook_name ()'. "my_module" correspond au nom du module qui l'a implémenté. « hook_name" correspond au nom du hook à surcharger.

Un des plus connus est 'hook_preprocess ()' qui permet de prétraiter les variables avant qu'elles ne soient rendues. On le retrouve la plupart du temps dans le fichier *.theme, et 'hook_xxx_alter()' qui permet d'altérer des informations avant qu'elles ne soient rendues.

Avec l'arrivée de la programmation orientée objet et des composants de Symfony, deux nouveaux concepts sont voués à remplacer les « hooks" dans le futur.

- **Les Plugins** : c'est une idée intégrée dans Drupal pour étendre la configuration de fonctionnalités graphiques. De façon *objet*, un module "parent" va mettre à disposition une classe abstraite pour définir un modèle à suivre. Les autres modules pourront alors créer une classe qui étend ce "plugin" en lui définissant un comportement spécifique. Comme exemples de plugins, nous avons les "FieldFormatter", "Block", "EntityType".

- **Les Events** : ce concept reprend le composant « EventDIspatcher" introduit par Symfony. Cela va permettre à des composants de communiquer entre eux en envoyant des événements et

en les écoutant. Dans Drupal, on va définir une classe qui implémente "EventSusbcriber Interface". Elle nous force à redéfinir la fonction "get SubscribedEvents". Cette fonction retourne un tableau de noms d'événements que l'abonné souhaite écouter. Pour chaque nom, on lui associe une ou des fonctions qui seront appelées lorsque celui-ci sera exécuté.

```
class BlockPageDisplayVariantSubscriber implements EventSubscriberInterface {

 /**
  * Selects the block page display variant.
  *
  * @param \Drupal\Core\Render\PageDisplayVariantSelectionEvent $event
  *   The event to process.
  */
 public function onSelectPageDisplayVariant(PageDisplayVariantSelectionEvent $event) {
  $event->setPluginId('block_page');
 }

 /**
  * {@inheritdoc}
  */
 public static function getSubscribedEvents() {
  $events[RenderEvents::SELECT_PAGE_DISPLAY_VARIANT][] = ['onSelectPage
DisplayVariant'];
  return $events;
 }

}
```

Derniers concepts que j'aimerais aborder dans cette partie : l'apparition des **Services** et de l'**injection de dépendance**. Ce concept est repris de la POO. Drupal met à disposition des centaines de fonctionnalités découplées dans le but qu'elles soient le moins dépendantes possible les unes des autres. Le *service container* va s'occuper d'instancier les objets dont on a besoin. Voici un exemple de déclaration de service avec Drupal. On a le fichier mon_module.services.yml à la racine de notre module.

On peut le déclarer comme ceci :

```
services:
 mon_module.mon_service:
  class:[mon_namespace]\ma_classe
  arguments: []
```

Ensuite, on va utiliser le service container afin d'instancier nos services par injection de dépendance dans nos contrôleurs.

```
// MonSuperController.php

class MonSuperController extends ControllerBase {

  protected $entityQuery ;
  protected $maClasse ;

  public function __construct(QueryFactory $entity_query, MaClasse
$ma_classe) {
    $this->entityQuery = $entity_query ;
    $this->maClasse = $ma_classe ;
  }

  public static function create(ContainerInterface $container) {
    return new static(
      $container-> get('entity.query'),
      $container-> get('mon_module.mon_service')
    ) ;
  }
}
```

Voici les principaux concepts présents dans Drupal lors de la création de modules dont je voulais vous parler. Si vous avez l'occasion de créer un module, il est important d'anticiper la conception d'une fonctionnalité générique et une instanciation spécifique pour répondre précisément au besoin. Ainsi, la partie générique pourra être partagée avec d'autres projets...

Entity API

Dans son histoire, Drupal a évolué par son expérience. Dans Drupal 6, tout était nœud, puis dans Drupal 7, tout est devenu entité. Drupal 8 a ajouté la couche objet aux entités. Deux types d'entités sont présents :

- Des entités de configuration que nous aborderons un peu plus loin dans cet article ;
- Des entités de contenu.

Concernant ces dernières, il est important de connaître celles qui sont disponibles par défaut dans Drupal pour modéliser correctement ses données et son contenu. Les plus connues sont les nœuds (les pages par exemple), les utilisateurs, les taxonomies. On peut également créer nos propres types d'entités. Le but de ces entités est de stocker en base de données les contenus générés par les utilisateurs. Vos entités, en fonction de leur configuration, mettent à disposition une personnalisation des champs, gèrent le multilingue et les révisions.

Si on s'intéresse de plus près à la définition d'une entité, comme le type d'entité nœud, on s'aperçoit qu'elle est définie dans le dossier " node/src/entity/Node.php". La classe implémente l'annotation "ContentEntityType" qui étend le plugin de type "EntityType".

```
* @ContentEntityType(
*   id = "node",
*   label = @Translation("Content"),
*   label_collection = @Translation (« Content"),
*   label_singular = @Translation (« content item"),
*   label_plural = @Translation (« content items"),
*   label_count = @PluralTranslation(
*     singular = "@count content item",
* plural = « @count content items"
*],
*   bundle_label = @Translation("Content type"),
*   handlers = {
*     "storage" = "Drupal\node\NodeStorage",
*     "storage_schema" = "Drupal\node\NodeStorageSchema",
```

```
*   "view_builder" = "Drupal\node\NodeViewBuilder",
*   "access" = "Drupal\node\NodeAccessControlHandler",
*   "views_data" = "Drupal\node\NodeViewsData",
*   "form" = {
*    "default" = "Drupal\node\NodeForm",
*    "delete" = "Drupal\node\Form\NodeDeleteForm",
*    "edit" = "Drupal\node\NodeForm",
*    "delete-multiple-confirm" = "Drupal\node\Form\DeleteMultiple"
*   },
*   "route_provider" = {
*    "html" = "Drupal\node\Entity\NodeRouteProvider",
*   },
*   "list_builder" = "Drupal\node\NodeListBuilder",
*   "translation" = "Drupal\node\NodeTranslationHandler"
* },
* base_table = « node",
* data_table = "node_field_data",
* revision_table = "node_revision",
* revision_data_table = "node_field_revision",
* show_revision_ui = TRUE,
* translatable = TRUE,
* list_cache_contexts = { "user.node_grants:view" },
* entity_keys = {
*   "id" = "nid",
*   "revision" = "vid",
*   "bundle" = "type",
*   "label" = "title",
*   "langcode" = "langcode",
*   "uuid" = "uuid",
*   "status" = "status",
*   "published" = "status",
*   "uid" = "uid",
*   "owner" = "uid",
* },
* revision_metadata_keys = {
*   "revision_user" = "revision_uid",
*   "revision_created" = "revision_timestamp",
*   "revision_log_message" = "revision_log"
* },
* bundle_entity_type = « node_type",
* field_ui_base_route = "entity.node_type.edit_form",
* common_reference_target = TRUE,
* permission_granularity = "bundle",
* links = {
*   "canonical" = "/node/{node}",
*   "delete-form" = "/node/{node}/delete",
*   "delete-multiple-form" = "/admin/content/node/delete",
*   "edit-form" = "/node/{node}/edit",
*   "version-history" = "/node/{node}/revisions",
*   "revision" = "/node/{node}/revisions/{node_revision}/view",
*   "create" = "/node",
* }
*)
```

Dans l'annotation, on détermine l'ensemble des données qui vont définir notre entité. C'est également ici que l'on va définir si notre entité est traduisible et mettre en place le système de gestion de versions, appelées *révisions*.

Gestion de la configuration

Que ce soit pour avoir de la configuration réutilisable, ou pour déployer nos modifications de configuration sur nos différents environnements, Drupal 7 a engendré une importante frustration chez une partie de ses développeurs. Avec Drupal 8, le projet « Content Management Initiative » (CMI) – mettant à disposition une nouvelle API – est né dans le but d'avoir de la configuration exportable et importable facilement. Cela permet de séparer la configuration spécifique du site dans des fichiers YAML. Remarque importante, par défaut CMI exporte l'ensemble de la configuration. Il n'est pas possible, par défaut, d'isoler une partie de celle-ci.

Deux autres notions majeures sont à comprendre avec CMI :

• **Active directory / Staging Storage**

Après l'installation de notre site, on va pouvoir exporter la configuration qui lui est propre. L'ensemble des fichiers est exporté dans un répertoire du système de fichier. C'est ce qu'on appelle le « Staging storage". L'avantage ici, c'est qu'avec un système de version comme GIT nous avons un suivi de l'évolution des configurations.

Lorsque l'on modifie un fichier de la configuration de notre site et qu'on l'importe pour la mettre à jour, cela va mettre à jour notre "**Active Storage**". C'est l'endroit où est stockée la configuration courante de notre site.

• **Configuration de données simple / Entité de configuration**

Deux types de configurations, la "Donnée simple" et l'"Entité de configuration".

Le type de configuration "donnée simple" entrepose des données de type basique. On l'utilise pour des cas spécifiques comme des configurations propres à notre module. Pour les manipuler, je vous laisse regarder du côté du service "config.factory".

Quant au type de configuration "Entité de configuration", il entrepose la configuration plus complexe. Cela va permettre à des types d'entités de sauvegarder leurs données de configuration dans des fichiers YML. Par exemple, voici quelques types d'entités de configurations : les vues, les rôles, les menus, les styles d'images. En analysant les exemples précédents, on s'aperçoit que ce sont des types d'entités qui peuvent avoir de multiples instances qui leur sont propres.

Je vous conseille de jeter un œil aux modules "config_ignore" et "config_split". Ils vous permettront d'ignorer certains fichiers de configuration que vous ne voulez pas écraser et avoir une gestion plus fine de configurations par environnement.

Multilingue / internationalisation

Drupal 8 a repris le meilleur de son passé et s'est transformé pour arriver avec 4 modules par défaut qui structurent l'ensemble du multilingue en son cœur. Il va gérer les langues, mais aussi tous les concepts d'internationalisation qui lui sont liés (les dates, le sens des textes de gauche ou de droite).

Voici une présentation des 4 modules majeurs du cœur :

Langage module

Ce module vous permet de choisir la langue de base à l'installation de votre site. À tout moment après l'installation de votre site, vous pouvez ajouter une nouvelle langue dans l'interface d'administration.

Locale (Interface Translation) module

Ce module vous permet, grâce à son interface, de traduire l'ensemble des textes affichés en *front*. Pour vous aider avec les traductions du cœur qui en compte plus de dix mille actuellement dans cette version 9. La communauté complète le site localize.drupal.org auquel il se connecte automatiquement pour mettre à jour les traductions.

Content Translation module

Ce module est celui qui va vous permettre de traduire le contenu. Avec son interface d'administration on va pouvoir choisir de traduire une entité, plus finement un de ces "bundle ", ou encore plus précisément le champ d'un "bundle".

Configuration Translation

Ce module vous permet de traduire vos valeurs de configuration. Lorsque vous exportez vos fichiers de configuration, vous aurez alors un dossier de langue qui contient les données de votre configuration dans une langue spécifique.

D'après mon expérience, deux concepts de programmation sont importants à connaître.

Tout ce qui s'affiche en front est traduisible.

Drupal contient un Trait "StringTranslationTrait" qui va vous permette de traduire ce que vous souhaitez dans une classe. La fonction que vous utiliserez le plus souvent est celle-ci :

```
$this->t('J'adore les %food.', [
 '%food » => $my_food,
['context » => 'Mon site']
])
```

Elle comprend trois paramètres : une chaîne de caractères, les variables dynamiques liées, et, enfin, ce que nous rajoutons par défaut à chaque fois, un contexte. En effet, il est possible d'avoir deux fois la même chaîne de caractères que vous souhaitez différente en fonction du contexte.

Il existe également une fonction qui vous permet de gérer les singuliers et pluriels.

Une fonction de traduction fournie par Drupal est disponible en Javascript et dans Twig.

Il faut charger nos entités dans la bonne langue.

Maintenant que nous avons traduit l'ensemble de notre front, qu'en est-il de nos contenus traduits dans notre back-office ? Si vous souhaitez charger un contenu en fonction du langage courant ou du navigateur ou autre, Drupal met à disposition le LanguageManager service.

Ce service va vous permettre entre autres choses de charger votre contenu d'entité en fonction de la langue passée en paramètre.

```
if ($item->hasTranslation ($langcode)) {
 $item = $ item->getTranslation($langcode);
}
```

L'exemple ci-dessus permet de vérifier que votre contenu existe dans la langue voulue avant de vouloir le changer. Dernier conseil avant de clôturer cette partie, suite à nos retours d'expérience, et ce même si votre site n'est pas multilingue, implémentez ces méthodes.

Sécurité

Drupal 8 est populaire et reste un des CMS privilégiés du marché. Il est préconisé pour les sites à fort trafic. Cette popularité attire des personnes à la recherche de la moindre faille à exploiter. Heureusement, depuis sa création, Dries Buytaert, son concepteur, a pris très au sérieux la partie sécurité. Drupal a sa propre équipe de sécurité recherchant les vulnérabilités qui peuvent être présentes dans le cœur mais aussi dans les modules de la communauté. Des mises à jour de sécurité sont alors disponibles dans de brefs délais. Il ne vous reste plus qu'à mettre à jour vos modules.

J'ai trouvé des données statistiques des failles de sécurité majeures que Drupal a recensées ces dernières années. Voici une petite présentation des types de failles qui reviennent le plus souvent et des outils mis en place pour y remédier.

Cross Site Scripting (XSS)

Drupal 8 a fait un grand pas en avant dans ce domaine avec l'utilisation du moteur de template Twig. Afin de sécuriser le système contre les attaques de type XSS, il vérifie automatiquement toutes les variables alimentées par Twig. Cela permet d'avoir une protection contre les attaques XSS en utilisant correctement les API de rendu fournies.
Drupal dans son API a mis à disposition un certain nombre de fonctions, afin de garantir la sûreté de vos chaînes. Voici la liste :

- t() pour les chaînes traduisibles ;
- Html::escape() pour les textes simples ;
- Xss:filter() pour les chaînes avec du HTML dedans ;
- UrlHelper::stripDangerousProtocols() pour checker les URL.

ByPass something

Cette faille permet à un attaquant de provoquer une atteinte à la confidentialité des données. Pensez à mettre en place des règles de droit d'accès rigoureuses sur vos pages. Drupal met à disposition un système avancé de permission que vous pouvez faire évoluer.

Exécution de code

Cela consiste à exploiter une faille dans le CMS qui pourrait permettre de l'exécution de code à distance. Souvent lié aux formulaires, pensez à toujours bien vérifier toutes les données envoyées depuis vos formulaires.

CSRF

C'est une faille qui permet aux attaquants de transmettre des commandes non autorisées d'un site tiers au site qui fait confiance à l'utilisateur donné. Ses résultats peuvent être similaires à ceux de XSS, mais son fonctionnement est légèrement différent.

Drupal 8 protège ses ressources REST contre les attaques CSRF en exigeant l'envoi d'un en-tête de requête X-CSRF-Token lorsqu'il utilise une méthode non sécurisée. Ainsi, lors de requêtes non en lecture seule, ce jeton est requis.

La recommandation de Drupal est de ne jamais utiliser de formulaires HTML personnalisés et d'utiliser plutôt l'API de formulaire.

Tests

Dans sa version 8, Drupal a adopté l'utilisation du framework PHPUnit, toujours dans le but d'utiliser les standards de la communauté PHP. Voici un exemple de structure d'implémentation des tests dans un module Drupal

```
text
|—— config/
|—— migrations/
|—— src/
|—— tests
|       |—— src
|               |—— Functional
|               |       |—— TextFieldTest.php
|               |       |—— TextRequiredSummaryUpdateTest.php
|               |—— FunctionalJavascript
|               |       |—— TextareaWithSummaryTest.php
|               |—— Kernel
|               |       |—— Migrate
|               |       |—— TextFormatterTest.php
|               |       |—— TextSummaryTest.php
|               |       |—— TextWithSummaryItemTest.php
|               |—— Unit
|                       |—— Migrate/
|                       |—— Plugin/
|—— text.es6.js
|—— text.info.yml
|—— text.js
|—— text.libraries.yml
|—— text.module
|—— text.post_update.php
```

Comme on peut le voir dans la capture ci-contre, un répertoire "tests » se situe à la racine de notre module. Voici le chemin des fichiers :

```
/modules/[mon_module]/tests/src/[TypeDeTest]
```

Il existe un répertoire type pour chacun de nos quatre types de tests :

- **Unit** : cela correspond à nos tests unitaires afin de tester des méthodes de classe individuellement. Ce sont les tests les plus rapides à exécuter ;
- **Kernel** : test avec le chargement du noyau, accès à la base de données et à une liste limitée de modules activés ;
- **Functional** : test avec une instance complète de Drupal, possibilité d'ajouter des modules complémentaires ;
- **FunctionalJavascript** : comme les tests fonctionnels mais avec un navigateur émulé afin de tester le JavaScript.

Lorsque vous créez un test, celui-ci doit respecter cette structure de l'espace de nom :

```
Drupal\Tests\[mon_module]\[TypeDeTest]
```

Les classes de tests doivent toujours finir par le terme "MaClassTest » et pour finir, les fonctions commencent par le mot "testDeMaFonction".

https://www.drupal.org/docs/testing/types-of-tests

On arrive au terme de la présentation des concepts que je souhaitais vous présenter. J'espère vous avoir davantage donné envie de découvrir ce CMS. Drupal depuis sa version 8 n'a pas peur de se remettre en question et se modernise continuellement, N'hésitez pas à venir faire un tour sur le slack de Drupal France drupalfrance.slack.com si vous avez des questions, et, pourquoi pas, un jour, participer aux issues de la communauté pour proposer vos évolutions.

Rafraîchir le front de Drupal : une approche Headless du CMS

Stéphane HEUZÉ

Architecte Web chez Kaliop, conçoit des systèmes de données avec Drupal. Son but, offrir à ses clients les refontes, migrations, spécifications et conceptions les plus folles, des expériences Web merveilleuses, et à ses développeurs des migraines d'autant plus monstrueuses.

Le Web a bien changé depuis sa création en 1989 par Tim Berners Lee. Le HTML s'est imposé. Les traitements sur son rendu, réalisés du côté du serveur avec des langages comme PHP sont devenus monnaie courante et ont amené l'essor des CMS, solution simple à mettre en œuvre pour déployer des sites de contenus riches en un minimum d'étapes.

Aujourd'hui, le Web se consomme sur de nouveaux terminaux comme les téléphones mobiles ou les systèmes embarqués. Les systèmes actuels doivent évoluer pour s'adapter aux modes d'accès à leurs informations. On n'a pas les mêmes besoins quand on surfe sur un ordinateur fixe ou un frigo connecté.
C'est de ce constat qu'a découlé l'apparition de la notion d'architecture web découplée, appelée "headless" en anglais. Qu'est-ce que ce terme aux consonances antimonarchistes signifie ? Qu'est-ce que cela peut nous permettre de gagner ? Comment cela fonctionne-t-il avec Drupal et comment le mettre en œuvre facilement ? C'est ce que nous verrons dans cet article.

Qu'est ce que le Headless ?

C'est une approche où la partie responsable du modèle et des contrôleurs d'un modèle MVC - l'ensemble des données et leur traitement - est séparée de la vue - l'affichage des données.
Dans notre cas, le CMS fera office de back-end, dans lequel les contenus sont créés. Le front-end est géré par une autre technologie. Les cas les plus fréquents sont l'utilisation d'un framework JavaScript tel que Vue.js ou React. **Figures 1 et 2**

Cette approche présente des avantages certains. Elle permet d'utiliser une technologie prévue à l'origine pour faire du rendu, telle que le React. Ces technologies offrent plus de souplesse et peuvent être plus performantes en fonction des cas d'utilisation.
La rupture entre le code et sa présentation facilite la maintenance. Il devient évident de savoir quelle partie gère quoi et donc quelle équipe doit être mobilisée.
On peut constater des améliorations des performances, car l'API va limiter le nombre d'appels directs à la base de données, point de friction clef d'une architecture web.

Dans quel cas doit-on envisager cette architecture ?
- Besoin spécifique de la puissance d'une technologie Front ;
- Plusieurs plateformes vont afficher les données : mobile, télévision, différentes interfaces web, etc. ;
- Si l'affichage doit être particulièrement adaptable à l'information qu'il affiche ;
- Si le contenu est consommé par de nombreuses cibles, le back-office centralisant les données est un moyen d'éviter la duplication des contenus.

Quelles pourraient être les mauvaises raisons de la choisir ?
- "Headless" est un buzzword. On en entend beaucoup parler. Mais il ne convient pas à tous les projets ;
- Choisir de déléguer le front à d'autres personnes sur une autre technologie parce que personne ne sait le faire en Drupal/Twig ;

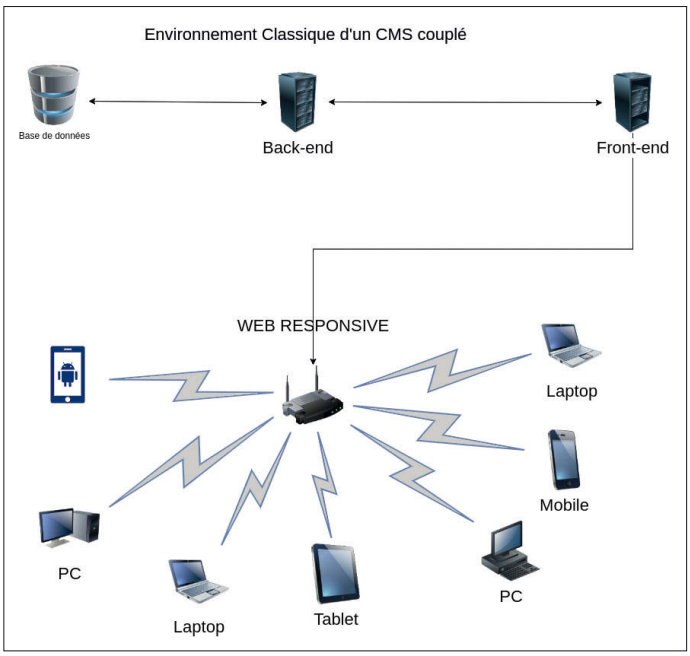


Figure 1

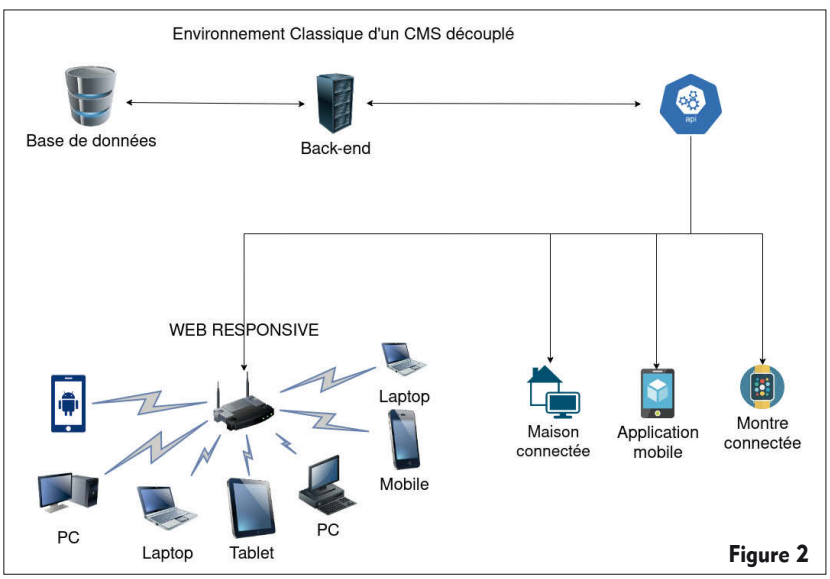


Figure 2

- Pour réduire les coûts. Ce n'est pas le cas. Les économies ne viennent qu'avec une mise à l'échelle : plusieurs récepteurs des données fournies par le back, réutilisation des sources… Une configuration spécifique du serveur, pour ouvrir les flux de communication, l'ajout de la gestion des frameworks Front… autant d'éléments qui peuvent tirer les coûts de mise en œuvre à la hausse.

L'approche découplée offre des avantages tangibles qu'il ne faut pas négliger.

En premier lieu, la flexibilité. L'affichage s'adapte dorénavant uniquement à la donnée reçue.

La mise sur le marché de votre solution est plus rapide. La structuration des données n'étant plus nécessaire, il y a moins de temps à consacrer à designer l'interface. De plus, l'agglomération des données peut permettre de générer des pages sans intervention de développeurs. Pour illustrer ce cas, on peut penser à un espace connecté de banque ou d'assurance : beaucoup de blocs qui se retrouvent d'une page à l'autre, en fonction des paramètres.

Le contenu peut s'adapter à n'importe quel support, puisqu'il n'est pas formaté lorsqu'il est envoyé par le back-office. Ce dernier propose une donnée agnostique qui ne "connaît" pas le support qu'elle sert.

La sécurité est facilitée à certains niveaux. Notamment, il est plus difficile de programmer une attaque par déni de service (DDoS), car il n'y a plus d'appel direct du front vers la base de données. C'est au niveau de l'API que l'on peut juguler le trafic.

L'évolutivité des plateformes Web est optimisée. Une modification du back-office ne mobilise pas le front-office et réciproquement. Cette séparation peut offrir la possibilité d'avoir deux équipes distinctes, deux prestataires différents, chacun spécialiste de sa partie.

Nous ne voulons pas dresser un portrait trop idyllique sans mettre en garde sur quelques surprises que les administrateurs d'une architecture Headless pourraient rencontrer.

La plus grande peut être que lors de la contribution du contenu, il n'est plus natif de voir le rendu du contenu nouvellement créé. Cela peut perturber de travailler "à l'aveugle" et peut amener à devoir concevoir un système palliatif en back-office.

L'autre grande problématique est que le parcours de l'utilisateur n'est plus aussi intuitif que dans une navigation couplée : les cookies et les données de sessions ne remontent plus automatiquement au serveur. Offrir une expérience basée sur le parcours devient alors plus difficile.
Il en va de même pour le SEO. Il faut y prendre particulièrement garde pour du contenu référencé. Une anticipation rigoureuse est nécessaire pour les éléments tels que la construction des URL, les balises canonical, les metadonnées, la hiérarchisation du contenu par les balises titres, les sitemaps XML ou encore les CDN.

Bien que les DDoS soient rendues plus complexes, il faut cependant veiller à correctement protéger les données qui transitent entre le Back et le Front. L'encryptage doit être particulièrement soigné, surtout si les données sont sensibles.

N'oublions pas non plus qu'il faut s'assurer des compétences de ses développeurs lorsque les équipes sont en interne. Une personne qui maîtrise l'intégration du Twig/HTML/CSS ne saura pas forcément faire du Vue.js.

Nous l'aurons compris, le découplage peut transformer votre site web en une véritable expérience. Mais Drupal sait-il vraiment le faire ?

Pour divulgâcher (« spoiler ») immédiatement la réponse à cette question, nous pouvons vous assurer que la réponse est Oui !

La technologie a pris le virage du Headless dès la sortie de la version 8.0 à la fin de 2015. Depuis, la Communauté n'a eu de cesse d'amener le CMS à la Petite Goutte vers une place majeure dans ce microcosme. Avec une philosophie "API First" et non "d'API Only", selon les propres mots de son créateur, Drupal permet de gérer aussi bien le couplé que le "sans-tête".
Ainsi, le concepteur du site peut déployer son B.O. tandis que les équipes Front sortent les interfaces utilisateurs indépendamment les unes des autres au fil de l'eau.

Aujourd'hui, le service web de communication RESTful est profondément ancré dans Drupal. Avec un module dédié et un pendant RestUI, pour permettre aux administrateurs de créer des endpoints directement dans le Back-office, sans intervention des développeurs.

Le CMS a su utiliser au mieux ses forces pour permettre une puissance de développement native pour interagir avec les données de manière fluide avant leur envoi vers l'interface.
- Gestion des permissions d'accès avec un contrôle des accès au contenu ;
- Preprocessing et Hooks pour intervenir à plusieurs niveaux sur la donnée durant sa génération et son traitement ;
- La gestion des événements pour augmenter l'expérience utilisateur.

Drupal va même jusqu'à dépasser de nombreux concurrents dont le headless est la seule raison d'être grâce à sa bonne intégration de la JSON:API, schéma de données permettant de sérialiser et d'exposer des structures complexes.

Après de nombreux débats, essais et avancées, la Communauté a fait le choix de concentrer ses efforts sur la JSON:API, qui a été jugée meilleure que RestFul.
Le cahier des charges était strict :
- Le moins de requêtes possibles ;
- API compréhensible et documentée ;

• Solution fiable, sécurisée, facile à mettre en œuvre ;
• Écrire de la data doit être aussi simple que de la lire.

Bien que Rest soit très bien intégré dans Drupal 8, la technologie fait qu'il est souvent nécessaire d'enchaîner les requêtes au serveur pour cibler précisément l'information. Ses requêtes trop verbeuses sont également rebutantes lorsque l'on parle performance ou même impact écologique.

La solution GraphQL a également longtemps été dans la balance. Elle aussi répond à de nombreux points. Avec JSON:API, les deux offrent des réponses à la taille limitée, les requêtes et leurs réponses peuvent être mises en cache. Ces mêmes requêtes peuvent être composées côté client. C'est le client qui contrôle également les données. Ce qui allège d'autant plus la charge du serveur.

Cependant, c'est JSON:API qui a gagné "la bataille" grâce à ses arguments indiscutables :

• Pas besoin d'une architecture et de bibliothèques client ;
• Très simple à mettre en place dans Drupal, puis à utiliser ;
• Les fonctionnalités de lecture et d'écriture natives ;
• L'approche de suivre les standards du Web ;
• Son modèle de sécurité.

Les trois technologies sont exploitables, mais le futur s'inscrit dans la JSON:API.

Fort de son architecture et de ses avancées technologiques, il est à noter que la Communauté ne se repose pas sur ses lauriers, en allant chercher une nouvelle solution pour découpler, plus que le contenu, la structure des menus du CMS.

Nous voilà plus éclairés sur l'approche technologique choisie par Drupal, les tenants et les aboutissants, les pour et les contre… Mais comment décider si l'on a besoin d'une plateforme découplée ou non ?

Si, à ce moment-là de l'article, le lecteur ne sait pas comment trancher, alors peut-être que c'est la voie médiane que propose également le CMS qui va pouvoir emporter les suffrages : l'adaptation du progressive headless ou semi-découplage.

Un petit schéma plutôt qu'un grand discours saura éclairer ce concept. **Figure 3**

• Parfait si on a un seul élément très interactif ;
• Ou des sources de données à enrichir ;
• Laisser au BO la gestion du HTML. On envoie des markup déjà prêts (du HTML déjà formaté) ;
• Permet de préserver les interfaces contextuelles et le workflow ;
• On n'envoie que ce que l'on veut. C'est Drupal qui gère ;
• On évite l'architecture plus complexe du découplé ;
• Dernier point et non des moindres, on conserve les puissantes capacités de caches de la technologie.

La joie, autant que les concepts abstraits, nous envahit devant cette pléthore d'approches convaincantes. Mais, cédons à la curiosité de la mise en œuvre d'un Drupal découplé.

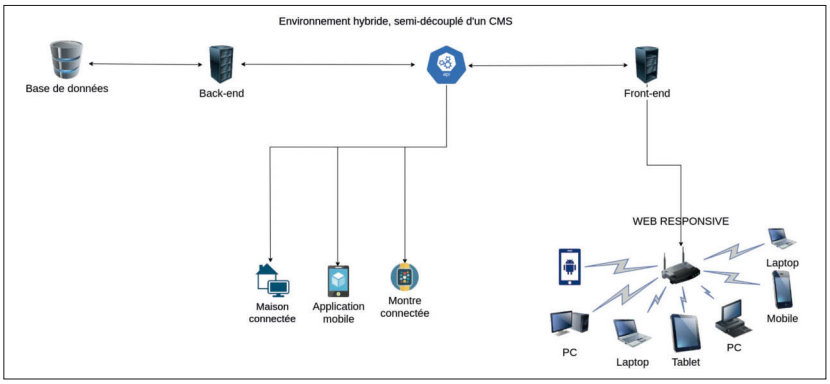


Figure 3

Comment faire ?

En tout premier lieu, il est fortement conseillé de travailler efficacement pour anticiper ce qui doit être découplé : qu'est-ce qui sera dans l'API, qu'est-ce qui sera traité par Drupal et surtout pourquoi ?

Une fois votre projet parfaitement spécifié, il ne reste plus qu'à créer un nouveau projet Drupal, avec Composer.

```
composer create-project drupal/recommended-project drupal
```

Il convient ensuite d'ajouter le module suivant :
Jsonapi_extras

Le module de la Json:API, désormais inclus dans le Core de Drupal, n'a pas besoin d'une configuration particulière. Vous pouvez choisir cependant si vous souhaitez que votre API n'accepte que les demandes de lecture ou si vous ouvrez aux CRUD (create, read, update, delete).

Pour augmenter un peu le niveau de sécurité de votre API, allez dans la configuration de Json_api:extras (/admin/config/services/jsonapi/extras). Cochez la case pour "Include count in collection queries" et définissez un préfixe de chemin personnalisé. L'exemple trivial est "/api/json/".

Il ne vous reste plus qu'à consulter, avec Postman par exemple, l'URL de votre API, en mode GET, pour obtenir le fichier JSON ccomprenant la liste des entités objets de vos requêtes :
http://localhost/api/json/node/article

Avec le JSON obtenu, votre Front pourra récupérer chaque noeud, chaque champ et les afficher de la manière qui s'adapte à vos besoins et vos envies.

La procédure que nous venons de décrire est bien évidemment très succincte. Nous vous conseillons de mettre en place un système d'authentification tel que OAuth 2.0 couplé au système de permissions de Drupal. Pour vos premiers tests, vous pouvez essayer le module "simple_oauth".

La variété de modules de Drupal permet d'avoir déjà un grand nombre de besoins couverts. Vous n'aurez aucun mal à trouver des tutoriels sur la mise en place avancée d'un Drupal semi-découplé ou découplé. Vous avez désormais la majorité des concepts qui permettront d'orienter vos recherches.

La gestion du cache avec Drupal

Afin de rester compétitif et performant face aux autres CMS, Drupal a dû mettre en place une politique de cache performante. La gestion du cache a pour but d'améliorer le temps de réponse du site internet et d'éviter la surcharge du serveur. Ce temps de réponse est devenu un levier direct d'augmentation du trafic, notamment par le ranking réalisé par Google. Une gestion de cache performante permet également de réduire les requêtes et les calculs pour les serveurs. Il s'agit donc d'une démarche d'écoconception des solutions web modernes.

Benjamin HOQUY

Développeur PHP depuis 6 ans spécialisé dans du Drupal chez Kaliop. Spéléologue par la force des choses, me promener dans les entrailles de ce CMS est devenu mon quotidien.

La politique de gestion de cache Drupal s'établit grâce à la Cache API. Il s'agit de l'ensemble du code fourni par Drupal servant à la gestion du cache. Cet ensemble s'articule autour de trois axes majeurs : les cache ***tags*** (qui ciblent les éléments dont le cache d'une page dépend), les cache ***contexts*** (qui font varier le cache en fonction d'un élément lié à la navigation) et le cache ***max-age*** (qui donne la durée de vie maximale du cache de la page).

Quand une page est construite en Drupal, une combinaison de cache tags, de *contexts* et d'un max est associée. Dès lors que l'un de ces critères devient obsolète, ou qu'un tag est invalidé, le cache de la page demandée est régénéré puis servi à l'utilisateur.

Le jeu consiste, via la combinaison de ces trois systèmes, à obtenir le cache le plus performant possible. Pour que la politique du cache d'une page soit performante, il faut que le cache réponde aux deux règles suivantes : il doit rester flexible et le système doit pouvoir invalider le cache seulement si un ou plusieurs éléments ont été modifiés. La même version de la page en cache doit servir le plus d'utilisateurs possible. Ces deux règles ne sont pas absolues, mais donnent une bonne base pour un site building classique ([construction d'un site en utilisant seulement le back office et des modules ne fournissant pas la communauté]).

Il est très rarement nécessaire d'interagir avec la cache API. En effet, Drupal a une bonne gestion native de son cache. Mais lors de développements spécifiques, ou d'un développement de module pour la communauté, il est parfois nécessaire d'interagir avec la cache API pour répondre à un besoin et à des contraintes particulières, qui ne peuvent être gérées nativement.

Les cache tags

Les *cache tags* permettent de cibler un élément précis dont le cache de la page dépend. Mais le cache de l'élément ne dépend pas de sa durée de vie, d'un contexte utilisateur ou du système. Sur une même page, une multitude de *cache tags* coexistent, chacun ciblant un élément de la page. Drupal fournit par défaut une liste de *cache tags* [{entity_type} : {id}, {entity_type}_list, config:{ma_config}, library_info]. Mais il est possible de créer ses propres cache tags afin de cibler des éléments et d'ajouter des règles d'invalidation custom. Il est également possible de surcharger la liste de *cache tags* d'une page afin d'affiner la gestion du cache de la page.

Plus concrètement, les cache tags permettent d'invalider le

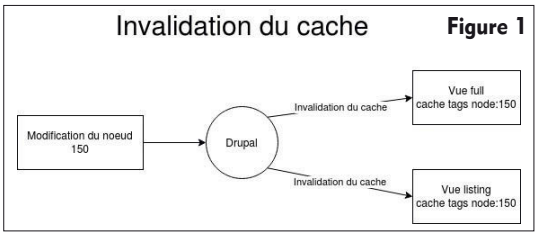


Figure 1

cache d'une page. Par exemple, dans le cas d'un nœud [contenu], le cache tag de la page sera « node:{identifiant_du_noeud}". Lors de la modification du nœud, Drupal invalidera le cache tag "node:{identifiant_du_noeud}", ce qui aura pour effet d'invalider toutes les pages du site qui contiennent ce cache tag. **Figure 1**

Il est possible d'ajouter des cache tags dès que le système par défaut n'est plus suffisant. Par exemple, pour un listing d'articles où l'on souhaite invalider le cache pour chaque modification d'un même type de contenu (articles), l'injection du cache tag custom peut s'effectuer dans un hook ou dans un contrôleur, selon le mode de rendu de la page qui a besoin de ce cache tag.

Ici, le cache tag qui sera ajouté pour le listing sera :

```
$variables[« #cache"]["tags"][] = 'all_articles_listing';
```

Pour invalider ce cache tag, il faut d'abord définir à quel moment le cache doit être invalidé. Dans le cas du listing, c'est à chaque mise à jour d'un article, afin que le contenu mis à jour ou que le nouveau contenu s'affiche. Le code correspondant sera le suivant :

```
function module_node_presave [EntityInterface $entity) {
if ($entity-> getType() === 'articles') {
   Cache : : invalidateTags('all_articles_listing');
}
}
```

Grâce à ce code, dès qu'un article sera enregistré (création ou mise à jour), toutes les pages qui contiennent le cache tag « all_articles_listing" verront leur cache vidé.

Certains modules contrib (fournis par la communauté) ajoutent des *cache tags*. Handy cache tags est couramment utilisé afin d'ajouter des cache tags qui s'avèrent souvent manquants dans des développements custom. Ce module permet d'ajouter de la finesse dans la gestion des cache tags des entités. Grâce à lui, il est en effet possible d'ajouter des tags pour n'importe quel type d'entité, et même de spécifier le bundle de l'entité [ex. *handy_cache_tags: [entity_type]: [entity_bundle]]*, ce qui n'est pas possible avec les tags fournis par le core Drupal.

Les cache contexts

Le *cache context* permet de placer en cache plusieurs versions d'une page en fonction d'un *contexte* lié à la navigation. Mais, il ne dépend pas de la durée de vie de la page ni d'un élément précis de cette page. Drupal fournit par défaut une liste de cache *contexts* assez complète. Comme pour les cache tags, il reste toujours possible d'en ajouter si nécessaire.

```
cookies
 :name
headers
 :name
ip
languages
 :type
protocol_version// Available in 8.9.x or higher.
request_format
route
 .book_navigation
 .menu_active_trails
  :menu_name
 .name
session
 .exists
theme
timezone
url
 .path
  .is_front// Available in 8.3.x or higher.
  . parent
 .query_args
  :key
  .pagers
   :pager_id
 .site
user
 .is_super_user
 .node_grants
  :operation
 .permissions
 .roles
  :role
```

Les cas d'usage des cache *contexts* sont nombreux. Le cas d'usage le plus fréquent est dans la gestion du fil d'Ariane pour qu'il soit différent sur toutes les pages. Dans ce cas, le cache *context* « 'url.path'" est utilisé. Le cache *context* est aussi utilisé pour le multilingue : une même page sera mise en cache pour chacune de ses langues, et le système renverra la langue correspondante à l'utilisateur en fonction de son contexte.

Le cache max-age

Le cache max-age permet de définir la durée de validité maximale du cache de la page. Par défaut, il est préconisé d'y mettre une valeur élevée, de l'ordre d'un an pour éviter des régénérations de cache inutiles. Sa valeur peut être surchargée sur chaque page afin de désactiver le cache en y affectant la valeur 0. Il est aussi possible de programmer l'expiration du cache d'une page à une date précise, ou à intervalle régulier.

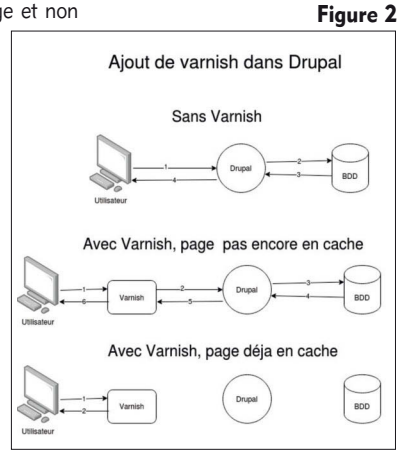
BigPipe

BigPipe n'est pas à proprement parler impliqué dans la gestion du cache, mais contribue à améliorer les performances du site. BigPipe est originellement un module intégré au corps de Drupal 8.1. Il permet de donner une impression de rapidité au site. Grâce à BigPipe, la page n'est pas livrée à l'utilisateur en une fois, mais en plusieurs : à l'arrivée sur le site web, au lieu d'une page blanche pendant X secondes, c'est une version allégée de celle-ci qui s'affiche, avec quelques blocs manquants, lesquels s'afficheront de manière différée. La page complète ne sera pas affichée plus rapidement, mais l'utilisateur pourra accéder plus tôt à une partie ciblée de la page.

Pour aller plus loin

Pour aller plus loin dans la gestion du cache, des outils complémentaires existent, comme Varnish ou Redis, qui peuvent s'avérer très performants. Varnish est un reverse proxy qui répond à la place du serveur web. Il renvoie au client des pages qu'il aura mises en cache préalablement, ce qui évite de devoir passer par Drupal qui est plus lent à répondre. Pour pouvoir utiliser Varnish sur Drupal, les modules "Purger" et "Varnish purger" sont nécessaires afin que le site puisse invalider du cache Varnish. La purge de cache Varnish s'effectue par défaut via le cron Drupal, mais il est plutôt préconisé d'utiliser les crons systèmes. La cache API Drupal est toujours compatible avec Varnish, donc les politiques de caches mises en place grâce aux *cache tags*, *contexts* ou max-age seront toujours valides. Mais au contraire, BigPipe n'est pas compatible. Malheureusement les ESI, permettant de découper la page en blocs avec une gestion de caches différente, ne sont toujours pas implémentés. Dans ce cas, le cache est alors géré de manière globale sur la page et non atomique. **Figure 2**

Figure 2



La gestion de cache en base de données de Drupal peut être un frein aux performances. Pour pallier ceci, il est possible d'utiliser Redis (via le module « Redis »), grâce auquel le cache est déplacé dans la RAM, au lieu d'être stocké en base de données. Ce mouvement améliore grandement les vitesses de chargement, la lecture de tableau stocké en RAM étant plus rapide qu'une requête à la base de données. Memcached (via le module « Memcache") peut aussi être une solution aux problèmes de performances des caches de bases de données. Memcached et Redis ont des fonctionnements assez similaires.

Conclusion

Cet article permet de mieux comprendre les outils externes et les systèmes mis en place par Drupal afin d'obtenir les meilleures performances possibles en gestion des caches. Il ne faut cependant pas oublier que, la plupart du temps, il n'est pas nécessaire d'affiner les politiques de gestion du cache d'une application web. Il est facile de tomber dans le piège de vouloir trop en faire, de rechercher une gestion du cache beaucoup trop fine, et de facto le rendre moins performant. Il faut rester vigilant pour ne pas casser les mécaniques natives de Drupal, ce qui rendrait son cache inopérant. Le risque étant de ne plus avoir de cache du tout, ou des pages dont le cache est trop souvent invalidé.

PROGRAMMEZ!

Programmez! n°245
mars - avril 2021

Directeur de la publication & rédacteur en chef
François Tonic
ftonic@programmez.com

Secrétaire de rédaction
Olivier Pavie

Contacter la rédaction
redaction@programmez.com

Ont collaboré à ce numéro
la rédaction de ZDnet.fr
Commitstrip

Les contributeurs techniques
Maël Morel
Alexis « Horgix » Chotard
Franck Cussac
Gildas Morel des Vallons
Franck Dubois
Christophe Villeneuve
Jérémy Jeanson
Pierre-Gilles Leynarie
Elvadas Nono
Philippe Boulanger
Stéphane Heuzé
Romain Dalverny
Benjamin Hoquy

Couverture
© Mordolff

Maquette
Pierre Sandré

Marketing – promotion des ventes
Agence BOCONSEIL - Analyse Media Etude
Directeur : Otto BORSCHA
oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Publicité
Nefer-IT
Tél. : 09 86 73 61 08
ftonic@programmez.com

Impression
SIB Imprimerie, France

Dépôt légal
A parution

Commission paritaire
1220K78366 (en cours de renouvellement)

ISSN
1627-0908

Abonnement
Abonnement (tarifs France) : 49 € pour 1 an, 79 € pour 2 ans. Etudiants : 39 €. Europe et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € - Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 €.
Autres pays : consultez les tarifs sur www.programmez.com.
Pour toute question sur l'abonnement :
abonnements@programmez.com

Abonnement PDF
monde entier : 39 € pour 1 an.
Accès aux archives : 19 €.

Nefer-IT
57 rue de Gisors, 95300 Pontoise France
redaction@programmez.com
Tél. : 09 86 73 61 08

Tangente 198 — *Tangente* HS 77

Les deux numéros du magazine de la culture mathématique en vente chez votre marchand de journaux et en ligne sur tangente-mag.com

Mieux ! Pour le recevoir, abonnez-vous sur infinimath.com/librairie

Actuellement en librairie ou disponible avec l'abonnement SUPERPLUS

Bib Tangente 73

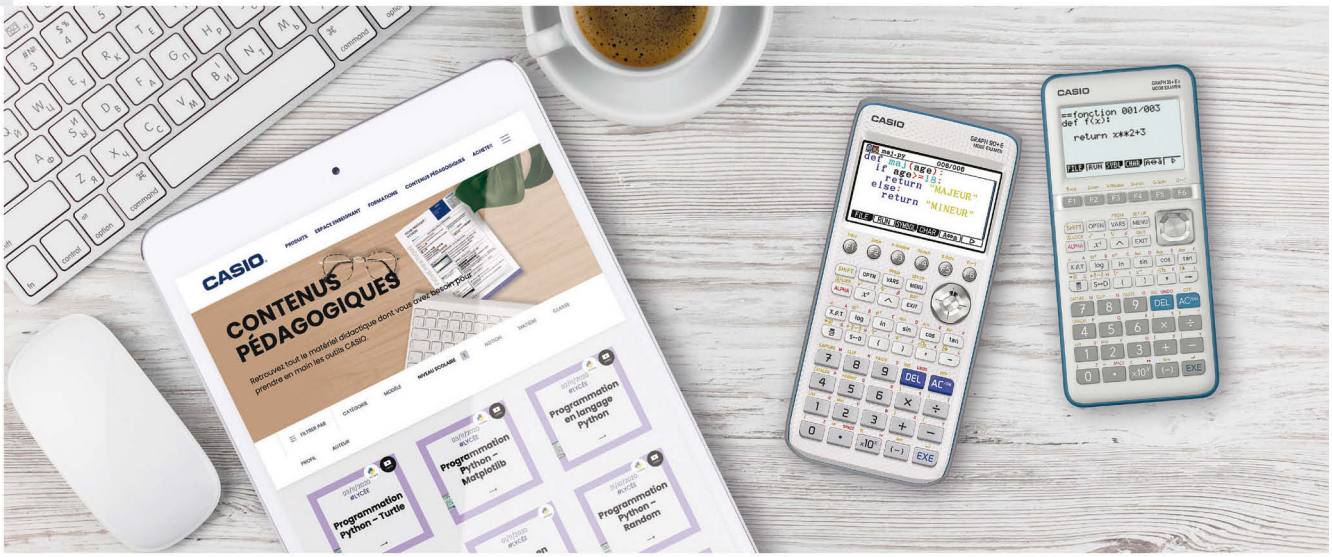
Maths et emploi

160 pages mettant en valeur l'importance des formations mathématiques dans le monde du travail

CASIO

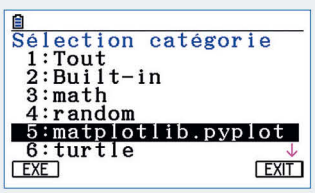
PROGRAMMEZ VOTRE RENTRÉE

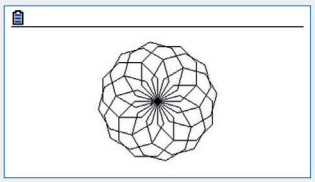
CALCULATRICES SCOLAIRES



CALCULATRICES GRAPHIQUES ÉQUIPÉES DU **LANGAGE PYTHON**

- Version de **MicroPython 1.9.4** adaptée pour la calculatrice.
- Bibliothèques disponibles : math, random, matplotlib et turtle.
- Transfert des fichiers en .py directement de la calculatrice à l'ordinateur et inversement.





GRAPH 35+E II **GRAPH 90+E**

Tutoriels et scripts .py clés en main disponibles sur notre site **www.casio-education.fr**