PROGRAMEZ!

Le magazine des dévs

N°254
09/10
2022

M 04319 - 254 - F: 6,99 € - RD

Compilateur Roslyn ... Gladys ... Figma Net7.0
... Gradient ... NextJS ... VPN ... Hydra ... Anthos ...

Printed in EU - Imprimé en UE - BELGIQUE 7,50 € - Canada 10,55 $ CAN - SUISSE 14,10 FS - DOM Surf 8,10 € - TOM 1100 XPF - MAROC 59 DH

L'innovation, une question de point de vue?

- Eh ! Rex (le dinosaure aimait donner des surnoms), as tu vu la météorique ce matin ?
- Oui Rex (bon après on n'a jamais dit que les surnoms étaient variés), une de plus.

Et non, celle de trop ! Bye bye Dino T-rex, hello Moby-dick.

Vers 3300 av. J.-C., invention de l'écriture avec en même temps la phobie administrative, puis invention de l'écriture minuscule (dite Caroline) au 8e siècle sous l'école Palatine (mais non pas Palpatine, rien à voir… *soupir*).
Depuis on écrit plus vite, mais les micro-ordinateurs ont mis plusieurs années à gérer à la fois la majuscule ET la minuscule (il fallait réécrire le code de l'OS et beaucoup plus de mémoire)

Vers 105 après J.-C. découverte en Chine de la pâte à papier et de l'imprimerie au XVe siècle. Le résultat est sans appel, elle conduit à la grève des copistes et des fabricants de papyrus, mais pour le plus grand bonheur de ce qu'on appellera bien plus tard les bibliophiles, cela enclenche la diffusion "massive" des écrits.

Au même siècle (1487-1490), visionnaire, De Vinci conçoit les plans de la vis aérienne, il faudra attendre 3 siècles pour voir les premiers hommes "voler"... En Avi…Ah pas encore…Restons à la montgolfière (les frères Montgolfier).

XVIIe siècle, Blaise Pascal invente la machine à calculer (dite Pascaline), les fabricants de bouliers font grève (ils étaient déprimés, comment on le sait ? Le voyage dans le temps…).

XIXe siècle, l'une des premières machines programmables voit le jour, basée sur des cartes perforées, Jacquard a révolutionné le métier à tisser. Se développent alors le langage binaire et la première calculatrice universelle (oui, enfin, ne vous imaginez pas la calculatrice d'aujourd'hui) ; Babache et Lovelace, ont travaillé en duo pour produire un des premiers programmes exécutables sur leur machine.

Après il y a eu boum, boom et boum, en d'autres termes la Seconde Guerre Mondiale, qui a engendré une révolution sur le plan technique : carburant de synthèse, naissance de l'informatique au sens moderne du terme, naissance des langages de programmation (Conrad Zuse), technologie de guidage et des fusées, cryptographie, naissance du métier de hacker, projet Manhattan, Roswell (la zone 51 pas la série), …

Dans les années 60, Apollo 11 permet de mieux définir le métier de programmeur, on entend parler de la miniaturisation, on voit la naissance d'Intel et on commence à expliquer partiellement la notion d'ordinateur « portable ».

Et puis, un beau jour au XXIe siècle, arrivent l'iPhone et la révolution mobile !
BlackBerry, Nokia, Microsoft prédisent l'échec du "truc" d'Apple. Visiblement à trop critiquer, ils ont un peu loupé le coche et perdent le marché du mobile. Google va rebooter Android (une obscure start-up rachetée) en OS mobile… Création de l'App Store, dématérialisation de la notion de logiciels.

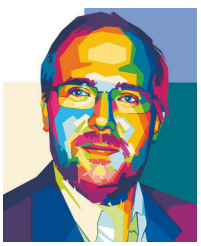
Une autre manière de consommer l'informatique
L'innovation est une notion complexe et difficile. Vous avez la technologie, la technique, les utilisateurs et l'impact sociétal. Mais au cœur de ces innovations, vous avez toujours des inventeurs, créateurs, ingénieurs, penseurs, visionnaires.

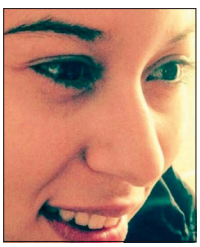
Repensez à Steve Jobs et son discours de Stanford : restez affamés, restez dingues !

Aujourd'hui, le développeur est un peu tout à la fois. Il est au cœur de l'innovation et le sera de plus en plus. IA, cloud computing, téléphonie mobile, robotique, IoT, derrière, il y a du code, donc des développeurs.

Même si Ballmer s'est totalement planté avec l'iPhone en 2007, reconnaissons-lui son célèbre discours (plutôt un mot) endiablé : développeurs, développeurs, développeurs !

```
prog@Programmez-Ubuntu :~$ gcc draft_jinane_mehdad.c -o script_francois_tonic
prog@Programmez-Ubuntu :~$ ./script_francois_tonic
```





PROCHAIN NUMÉRO

PROGRAMMEZ! N°255

Tensorflow, Java, Python, Secure by Design

Disponible le 2 décembre 2022

Contenus

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

Toujours disponible !

PROGRAMEZ

Le magazine des dévs

SPÉCIAL ÉTÉ 2022

SÉCURITÉ & QUANTIQUE

HACKER UN DRONE

SECURE BY DESIGN

INJECTION SQL

BUG BOUNTY

DEVSECOPS

PENTEST

OWASP

100% SÉCURITÉ

M 01642 - 8 - F: 6,99 € - RD

Numéro compilé en partenariat avec yogosha

Printed in EU - Imprimé en UE - BELGIQUE 7,50 € - Canada 10,55 $ CAN - SUISSE 14,10 FS - DOM Surf 8,10 € - TOM 1100 XPF - MAROC 59 DH

Les événements Programmez!

Meetups Programmez!

18 octobre
8 novembre
13 décembre

Où : Scaleway
11b rue Roquépine, Paris
A partir de 18h30

Conférence DevCon #15 100 % Kotlin
3 novembre 2022

Où : Campus de l'école ESGI - 242 rue du Faubourg Saint-Antoine, Paris
Transport : Nation (RER A, Métro 1, 2, 6, 9) - A partir de 13h30

Conférence DevCon #16 100 % Gaming
24 novembre 2022

Gaming Campus, école de gaming à Paris La Défense - à partir de 13h30

Conférence DevCon #17
.Net & technologies Microsoft 3e édition

15 décembre 2022

Informations & inscription : programmez.com

Octobre

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
			Volcamp (Clermond Ferrand)			
			Forum PHP (Paris)			
17	18	19	20	21	22	23
	Flow Con		DevFest Nantes			
			EveryDay AI par			
24	25	26	27	28	29	30
			Agile Tour (Bordeaux)			
31						

Décembre

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
			1	2	3	4
			DevOps DDay / Marseille	DevFest Dijon		
				BDX I/O / Bordeaux		
5	6	7	8	9	10	11
12	13	14	15	16	17	18
		API Day Paris	API Day Paris			
			DevCon			
19	20	21	22	23	21	25
26	27	28	29	30		

novembre

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
	1	2	3	4	5	6
		Paris Game Weeks / Paris				
			Programmez DevCon	ScalaIO		
7	8	9	10	11	12	13
	Open Source Experience / Paris					Forum PHP
14	15	16	17	18	19	20
Forum PHP	ParisTestConf / Paris		Codeurs en Seine / Rouen	DevFest / Strasbourg		
	Agile Tour Toulouse			GreHack / Grenoble		
				Capitole du Libre / Toulouse		
21	22	23	24	25	26	27
			Programmez DevCon			
28	29	30				

A VENIR

2023

- SnowCamp : 25-28 janvier 2023 / Grenoble
- FOSDEM : février 2023 / Bruxelles
- Devoxx : 12-14 avril / Paris
- Best of web : juin / Paris

Merci à Aurélie Vache pour la liste 2022, consultable sur son GitHub : https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

Les partenaires 2022 de

PROGRAMMEZ!

Le magazine des dévs

Niveau maître Jedi

salesforce

Scaleway

The cloud that makes sense

Niveau padawan

OSAXiS

Vous voulez soutenir activement Programmez! ?
Devenir partenaires de nos dossiers en ligne et de nos événements ?

Contactez-nous dès maintenant :

ftonic@programmez.com

Thomas Gentilhomme

Thomas est expert Node.js. Développeur indépendant, il a plus de 300 projets en tous genres à son actif : 50 à 100 en production, 5 ou 6 gros projets représentant des milliers d'heures de développement. Il est également le mainteneur principal de SlimIO, contributeur de Node-Secure, et membre du Node.js Security Working Group.

LinkedIn :
https://www.linkedin.com/in/thomas-gentilhomme

Github :
https://github.com/fraxken

SlimIO :
https://github.com/SlimIO

Ecosystem Security Working Group :
https://github.com/Nodejs/security-wg

nSecure :
https://github.com/ES-Community/nSecure

Node.js et la sécurité

PARTIE 2

La sécurité dans les écosystèmes JavaScript et Node.js est un enjeu primordial. Pendant la rédaction de l'article "Comment sécuriser son application JavaScript en 2022 ?", je me suis intéressée à Node Secure entre autres initiatives. J'ai rencontré Thomas Gentilhomme dans le cadre d'un atelier technique individuel qu'il proposait sur LinkedIn. À cette occasion, j'ai rejoint le serveur Discord de l'ES Community.

Interview menée par Romy Alula

R.A. : Qu'est-ce qui t'a amené à créer Node Secure ?

T.G. : Sur SlimIO, on a une centaine de projets. On avait du mal à comprendre les interactions entre eux. L'ancêtre de Node Secure, c'est le Dependency Analyzer, (https://github.com/SlimIO/Dependency-Analyser). Il permettait d'analyser les liens entre les projets. Node Secure, lui, donne plus de visibilité sur le graphe des dépendances. **Figure A**

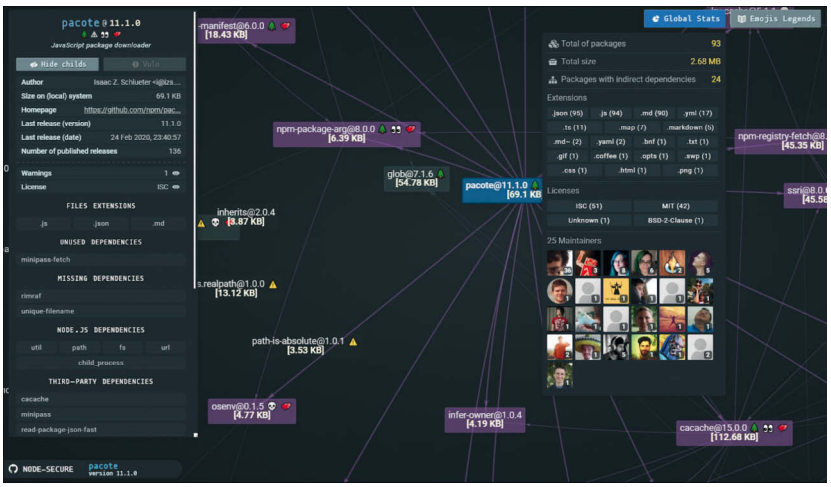
C'est une super idée venue de notre vécu. On était très confiants dans la qualité et l'intérêt du projet dès le début. Il répondait à des cas d'usage très précis et concrets. Son impact a été confirmé dès les premiers prototypes. Depuis 2 ans, Node Secure a continué à évoluer.

Concernant les dépendances de SlimIO, avec une centaine de projets, on a 2665 packages open source, ce qui est peu, grâce à une belle optimisation. Chez Myunisoft, mon client actuel, on a 2960 dépendances pour une dizaine de projets. Pour donner un ordre de grandeur, quand il y a du front, on peut facilement dépasser les 3000 dépendances pour un projet donné. De nombreux facteurs entrent en jeu, mais c'est une moyenne. Généralement, le front engendre beaucoup de dépendances. Ça monte très vite. Vu qu'on les installe très facilement, on ne se rend pas compte qu'on utilise une grande variété de packages. Ce qui est assez incroyable.

R.A. : Comment ça fonctionne "sous le capot" ?

T.G. : Node Secure utilise la même API que npm audit. La commande `npm audit` va chercher les vulnérabilités connues dans le npm advisatory. Elle remonte la plupart du temps des CVE - Common Vulnerabilities Exposure. La fonctionnalité est disponible sur la branche principale. Mais elle n'est pas encore publiée.

Figure A



R.A. : Quelles sont les prochaines étapes de développement pour livrer la v 1.0.0 ?

T.G. : Depuis 2 ans, rester en v.0.x a été un choix délibéré, car j'étais encore débutant en termes de sécurité. On aura une v1 très aboutie lorsqu'on aura franchi toutes ces étapes :

- Reconstruction du graphe avec la lib D3.js
- Réduction des lags : actuellement, on est dans l'incapacité de pousser davantage
- Améliorations sur l'UX, la lisibilité et les performances
- Amélioration interface et du code Vanilla JS
- Éventuellement passer aux web components sans compromis. On a l'idée d'intégration lit-element (basé sur le projet projet Polymer), c'est un mix entre Vanilla JS et les web components
- Segmentation du projet pour la simplification et l'amélioration de la maintenance : la CLI est le cœur de Node Secure (5-7 packages que j'ai personnellement créés, spécifiquement à cet effet). Tony (_tonygo sur YouTube) est l'un des contributeurs les plus actifs. Aujourd'hui, les trois quarts des libs sont dédiées à la CLI, donc dispensables si on utilise l'API de Node Secure.
- Amélioration globale de la qualité
- Stabilisation de toutes les fonctionnalités en place (faux positifs, bugs)

js-x-ray (JavaScript & Node.js open source SAST - Static Application Security Testing - scanner, https://github.com/fraxken/js-x-ray, version actuelle 3.1.1)

R.A. : Est-ce que tu peux nous présenter JS X Ray dans les grandes lignes ?

T.G. : C'est un outil capable de prendre un code source sans l'exécuter et d'y trouver des patterns et codes malicieux. Notre travail est basé sur des attaques précédentes dans l'écosystème pour une détection instantanée.

C'est un outil d'analyse. C'est aux devs de se documenter. Le public visé est intermédiaire et senior.

R.A. : Qu'est-ce que cette lib apporte en plus de Node Secure ?

T.G. : C'est un projet né d'un code qui était dans Node Secure. Il a été mis à part pour de meilleures optimisations, maintenance, évolution. Dans notre démarche de segmentation de Node Secure, JS-x-ray est devenue une lib indépendante. Avec cet outil, un de mes kifs serait d'analyser tout l'écosystème NPM, passer régulièrement à travers tous les packages JavaScript et en sortir des statistiques. Améliorer les faux positifs, découvrir de nouveaux positifs, ce serait le Graal pour moi.

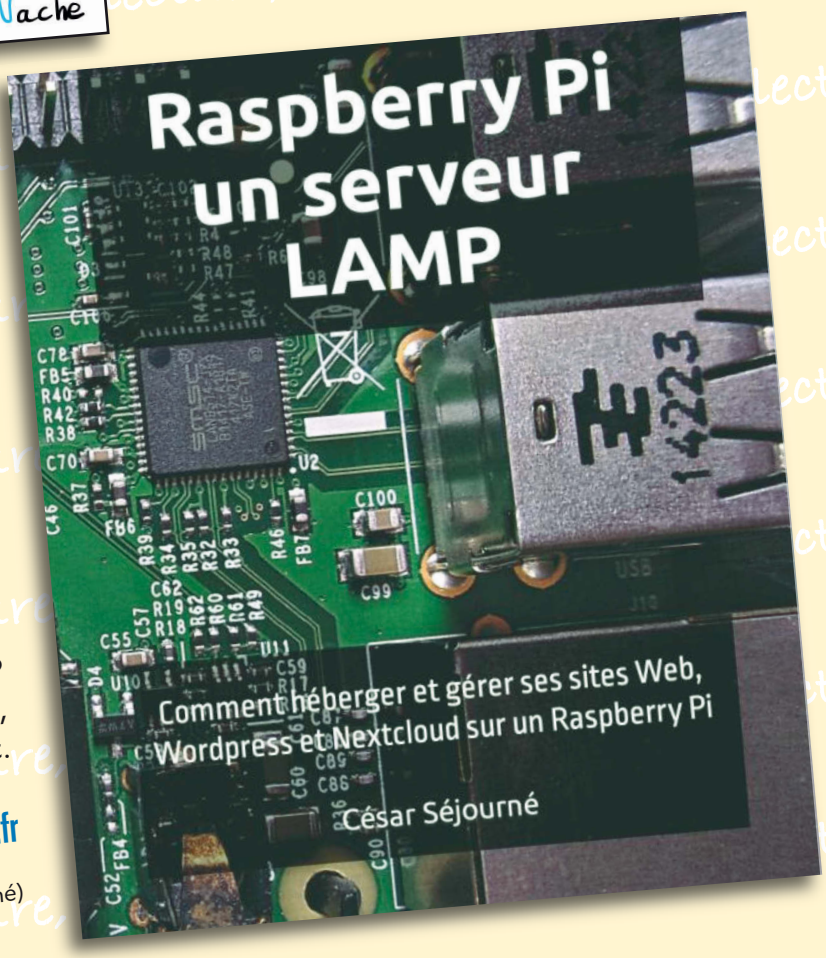
R.A. : Tu dis ne pas avoir d'expériences / compétences en sécurité, mais une passion grandissante pour l'analyse de

Understanding Kubernetes in a visual way

Comprendre Kubernetes, voilà le défi d'**Aurélie Vache**. Avec son approche ludique et BD, Aurélie nous plonge dans l'univers Kubernetes.

Disponible sur amazon.fr

25 € (format broché)



Rapsberry Pi : un serveur LAMP

Et si on passait à la pratique avec sa Pi ?

Comment installer LAMP ?
Comment le configurer ?
Comment déployer son site web ?
Le livre aborde la sécurité, Wordpress, Nextcloud, etc.

Disponible sur amazon.fr

20 € (format broché)

code statique. Comment tu t'y prends pour répertorier un maximum de code malicieux ? Quelles sont tes sources ?

T.G. : Dans la sécurité, j'ai un parcours atypique. Je ne fais pas de pen testing (test d'intrusion). Je ne participe pas beaucoup aux concours. Pour moi, c'est un investissement personnel dans l'écosystème qui me passionne.

L'analyse statique (AST - Abstract Syntax Tree) m'a permis de faire un lien avec la sécurité. Je suis très axé sur le code, sa compréhension, son analyse. Ma zone de compétences est bien délimitée dans la sécurité.

Dans l'analyse de code, il y a peu de monde et peu d'expertise. En France, je ne connais quasiment personne d'autre qui s'investisse dans l'analyse. Pas dans l'open source, en tout cas.

À la lecture d'un article, j'ai découvert que des hackers avaient encodé du code JS en morse pour le rendre illisible. Ça m'a fait ultra rire. Je me suis dit que c'était trop beau pour ne pas le gérer. Ça m'a permis de me replonger dans le morse. Aujourd'hui, JS-x-ray est capable de le détecter (en théorie) ! Dans l'informatique, tu finis toujours par apprendre un truc. Mes sources :

- awesome Node security qui référence l'existant, en termes d'outils et autres ressources.
- badjs.org (https://github.com/jsoverson/badjs.org) liste des attaques de manière détaillée. En cas d'attaque, NPM retire les codes et ne les publie pas. Ce qui est problématique pour l'analyse de code : impossibilité de travailler.

Snyk, avec la puissance qu'ils ont pourrait collaborer avec NPM en demandant l'accès à ces codes malicieux, essayer d'utiliser les post-mortem. Contrairement aux contributeurs open source, on n'a pas la même légitimité.

Face à un vecteur d'attaque fort, un remède simple peut être préconisé : si tu peux faire un code JS avec des boucles et conditions, c'est recommandé plutôt qu'une regex. Dans le cas où une regex est indispensable, il vaudrait mieux utiliser une lib spécialisée comme validator ou safe-regex (https://github.com/goldbergyoni/Nodebestpractices#-616-prevent-evil-regex-from-overloading-your-single-thread-execution).

R.A. : Si aujourd'hui, je me lance dans le développement d'une application full JS, application manipulant des données sensibles (open banking). Comment est-ce que je pourrais la blinder dès le premier commit ?

T.G. : Sujet très vaste. La sécurité regroupe énormément de choses :

- Définition du besoin compris et maîtrisé
- Fonctionnalités très simples : je t'envoie A, tu me retournes B
- Évolutions maîtrisées
- Architecture, qualité du code : features plus dédiées
- Infrastructure
- Réseau

Tout ça représente un large scope de métiers et demande une rigueur d'ensemble. La sécurité, c'est aussi être conscient des risques et toujours se remettre en question.

Éventuellement, dans des entreprises comme Thalès, ils savent faire preuve d'une grande rigueur. Après, pour une personne qui débute, ça va être très difficile, voire impossible, de sécuriser son projet dès le début. Je considère avoir 10% des compétences. Appliquer les bonnes pratiques (cf awesome Node security) peut permettre une sécurisation à 99%. Pour le dernier pour cent, la consultation d'experts et un audit externe sont fortement recommandés.

R.A. : Des conseils pour monter en compétences en sécurité dans l'écosystème JavaScript ?

T.G. : Quelques ressources intéressantes :

- Awesome Node Security, la section Educational
- la section Sécurité de "Devenir un(e) développeur(se) Node.js", mon document en constante évolution
- CTF (Capture The Flag), un concours d'intrusion

Tout ça, ce sont des premiers pas. Après, il faut se lancer, lire et s'y intéresser. Il existe également des listes "awesome security" généralistes et leur version par langage. Les repos sont dispos en ligne.

R.A. : Est-ce que tu voudrais ajouter quelque chose ?

T.G. : Face à Express* et sa grande popularité, Fastify, c'est l'apothéose de ce qui se fait de mieux dans l'écosystème Node.js : la sécurité y est optimale. C'est maintenu par des contributeurs core de Node, avec une communauté d'experts très actifs.

Node n'est pas dépassé. C'est aux devs d'aller chercher les modules les plus récents et optimaux. Dans 2 ou 3 ans, Express sera déprécié. Il faudrait faire une analyse des dépendances, découper le métier pour le séparer du serveur HTTP. Beaucoup de packages sont en train de mourir. Souvent ce sont des middlewares. Express, c'est beaucoup de middlewares qui ne sont plus maintenus. Le cœur d'Express sera maintenu, mais pas les packages spécifiques.

*Passer d'Express à Fastify n'est pas forcément très compliqué. Ce sont tous deux des frameworks low scope. Il existe un mode de compatibilité pour migrer plus facilement. Un package de huit ans sans mise à jour, c'est un vecteur d'attaque, d'instabilité. Il faut assurer la maintenance soi-même, ça demande beaucoup de travail. NSecure peut être installé à la racine du projet pour analyser les dépendances : est-ce qu'elles sont à jour ?

Je lutte contre les répercussions de ce framework bloqué en 2015 avec des callbacks, un système de middlewares dépassé et pourtant choisi par les entreprises et utilisé dans de nombreux tutos destinés aux juniors. Il entraîne des dégâts d'image, alors qu'il y a des solutions stables existant depuis des années. Notre manière de coder dans Node.js n'a plus rien à voir avec celle d'il y cinq ou six ans. L'écosystème, c'est les packages. Les libs en coulisse sont souvent les mêmes. Comprendre l'historique permet d'éviter les dégâts causés par certaines libs. Le travail d'écriture permet de toucher un autre public.

Sources

Awesome Node.js security :
https://github.com/lirantal/awesome-Nodejs-security
Awesome cross platform Node.js :
https://github.com/bcoe/awesome-cross-platform-Nodejs
Les projets open-source de Thomas :
SlimIO : https://github.com/SlimIO
Node.js security working group :
https://github.com/Nodejs/security-wg

Mise à jour “Over the Air” avec .NET nanoFramework et Azure IoT

Over the Air (OTA) est l’équivalent de Windows Update ou des mises à jour de package sous Windows ou Linux, mais dans le monde des Micro Controller Unit (MCU). Même si le terme OTA peut être utilisé dans beaucoup d’autres circonstances, il est surtout dans l’Internet des objets, Internet of Things (IoT). Vous trouverez également le terme Field Update.

Laurent Ellerbach
Principal Engineer Manager
Microsoft
laurelle@microsoft.com
https://github.com/Ellerbach

Le but est de pouvoir mettre à jour le code de façon sécurisée, sans avoir à récupérer le périphérique pour le reflasher. Il peut être possible de se connecter au périphérique à distance ou physiquement.

En général, OTA prend la forme de mise à jour de firmware et de mise à jour de code. Le firmware est souvent l’équivalent d’un Real Time OS (système d’exploitation temps réel) avec les couches basses permettant son fonctionnement alors que le code est la partie qui fonctionne dessus. Les deux peuvent être totalement indépendants. Le mécanisme en place permet de sauvegarder dans un stockage interne la nouvelle version en parallèle de l’ancienne. Ensuite, le changement est appliqué. Si quelque chose ne fonctionne pas bien, l’ancienne version est rechargée. Cela permet d’éviter de bloquer complètement le périphérique en cas d’échec. Le système fonctionne avec un watchdog (chien de garde) qui vérifie que tout fonctionne.

En termes de sécurité et d’intégrité, il faut être sûr que la nouvelle mise à jour est bien signée, qu’elle est intègre lors de son téléchargement. Sans ces deux mécanismes simples, il est impossible de garantir qu’un code non voulu soit déployé sur le périphérique.

Et bien sûr, il faut un mécanisme qui soit facile à mettre en place pour télécharger ces données de façon sûre sans avoir besoin de déplacer le périphérique. Quand une connectivité réseau existe, il est possible d’utiliser celle-ci. Quand il n’y en a pas, un réseau local Bluetooth, Wi-Fi ou même brancher un câble est alors nécessaire.

Dans l’exemple que je vais donner avec .NET nanoFramework, nous allons nous focaliser sur l’update du code exécutable et pas celui des couches basses. Donc cela ne couvrira pas la mise à jour du firmware, mais le code qui s’exécute sur le périphérique et qui est écrit en C#. Pour cela, nous allons utiliser une connectivité Wi-Fi ou Ethernet avec Azure IoT (https://azure.microsoft.com/fr-fr/overview/iot).

L’ensemble du code est disponible dans les exemples de .NET nanoFramweork : https://github.com/nanoframework/Samples/tree/main/samples/AzureSDK/AzureEdgeOta.

Le modèle que je vais décrire en détail dans cet article permet de télécharger un nouveau code depuis un stockage blob dans le Cloud, utilisant les avantages de Azure IoT avec la notion de jumeaux numériques (twins) comme un mécanisme de distribution de l’information. Pour ajouter à la magie, .NET nanoFramework, permet d’utiliser la réflexion et de charger dynamiquement du code. Voilà pour les points principaux. Fin de la théorie, regardons le code ! **Figure A**

.NET nanoFramework (https://github.com/nanoframework) apporte le support de .NET sur des petits MCU comme des ESP32, STM32, TI et NXP. Comme la mémoire, le stockage et l’environnement d’exécution sont très contraints, vous pouvez imaginer que ce qui tourne sur ces MCU est une version allégée de .NET. Il y a une compatibilité bien entendu et il est possible de réutiliser du code avec le grand frère. .NET nanoFramework est open source et un effort communautaire. Je suis un des principaux contributeurs à .NET nanoFramework.

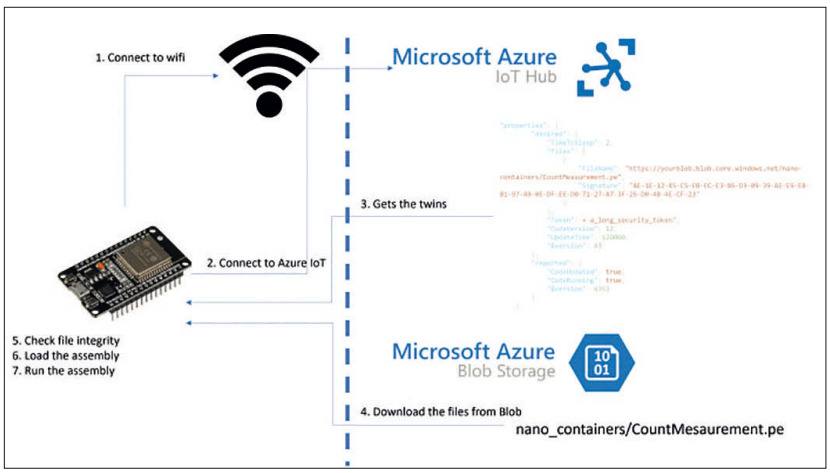
À noter que ce que je vais présenter fonctionne également parfaitement avec Azure IoT Plug & Play (https://docs.microsoft.com/azure/iot-develop/overview-iot-plug-and-play). Je ne l’utilise pas ici pour des questions de simplification.

Azure IoT : connectivité cloud sécurisé et fiable

Avec les MCU récents, il est désormais possible de les connecter directement à un réseau filaire ou sans fil. Ils supportent également les communications basées utilisant Transport Layer Security (TLS) et équivalent garantissant des communications sécurisées. Prenez un simple ESP32 pour quelques euros avec une connexion Wi-Fi, certains possèdent aussi de l’Ethernet. C’est ce que j’utiliserais dans mon exemple, car ils sont simples à trouver.

Mais pourquoi connecter de tels périphériques directement au Cloud ? Et bien, parce que l’on peut en toute sécurité maintenant ! Et dans de plus en plus de cas, il y a un réel intérêt à les connecter directement et non pas à travers une passerelle (gateway). Travaillant avec des clients dans le domaine de l’automobile et de l’industrie, je vois de plus en

Figure A



plus d'opportunités de donner une vie numérique à de vieux instruments à un coût extrêmement réduit à travers une connexion série entre le MCU et l'ancienne machine. Pour prendre un autre exemple, l'utilisation sur batterie avec un simple panneau solaire et une connexion à travers une carte SIM sont aussi des scénarios que l'on voit de plus en plus.
J'ai souvent la question de pourquoi ne pas utiliser un périphérique comme un Raspberry Pi à la place. La raison est simple : le coût et la consommation énergétique. L'utilisation d'un mini ordinateur fonctionnant sur batterie a un coût de 5 à 100 fois plus élevé. Ce choix doit donc être réservé à une utilisation où le calcul nécessaire est vraiment très important et justifie le surcoût.
.NET nanoFramework offre la possibilité de se connecter de façon sécurisée à un réseau en utilisant TLS et des certificats pour l'authentification dans Azure IoT. Quelques lignes suffisent pour se connecter grâce à la librairie Azure (https://github.com/nanoframework/nanoFramework.Azure.Devices):

```
// Nous stockons le certificat dans les ressources.
X509Certificate azureRootCACert = new X509Certificate(Resource.GetBytes(Resource
.BinaryResources.AzureRoot));
azure = new(Secrets.IotBrokerAddress, Secrets.DeviceID, Secrets.SasKey, MqttQoS
Level.AtLeastOnce, azureRootCACert);
```

Dans cet exemple, nous utilisons un token SAS pour simplifier un peu. Si vous utilisez des certificats, il y a la possibilité de les stocker dans un store interne sans avoir besoin de reflasher le périphérique.

Azure Twins : dis-moi ce que je dois télécharger

Le concept d'Azure Twins (https://azure.microsoft.com/services/digital-twins/), jumeaux numériques, utilisé dans Azure IoT se base sur des propriétés. Vous avez d'un côté les *desired* (désirées) et les *reported* (reportées). Leur représentation prend la forme d'un JSON et vous pouvez les utiliser comme vous le souhaitez. Voici un exemple :

```
{
  "properties": {
    "desired": {
      "TimeToSleep": 2,
      "Files": [
        {
          "FileName": "https://blob_name.blob.core.windows.net/
nano-containers/CountMeasurement.pe",
          "Signature": "4E-1E-12-45-C5-EB-EC-E3-86-D3-09-39-AE-E9-
E8-81-97-A9-0E-DF-EE-D0-71-27-A7-3F-26-D0-4B-4E-CF-23"
        }
      ],
      "Token": "A SAS token to connect to the blob storage",
      "CodeVersion": 12,
      "UpdateTime": 120000,
      "$version": 43
    },
    "reported": {
      "CodeUpdated": true,
      "CodeRunning": true,
    },
    "$version": 53
  }
}
```

Le mécanisme des twins permet de délivrer des informations, mais pas de larges fichiers. Cela doit rester dans la limite du raisonnable (quelques kilo-octets). Si besoin, un pointeur sur un blob ou équivalent permet de charger une configuration plus large ou dans notre cas, directement un binaire.
Nous allons donc utiliser ce modèle avec une version de code, un token et une liste d'URL.
Une fois connecté, il est possible de demander les twins et aussi de souscrire à un changement de twins. La librairie dans .NET nanoFramework permet de le faire très simplement :

```
var twins = azure.GetTwin(new CancellationTokenSource(10000).Token);
if (twins == null)
{
 // Ceci est une erreur!
}
// Souscrire aux changements de twins
azure.TwinUpated += TwinUpdated;
void TwinUpdated(object sender, TwinUpdateEventArgs e)
{
 // Les twins ont été mises à jour, nous pouvons les traiter
}
```

Quand vous utilisez la fonction *GetTwins*, vous obtenez à la fois les désirées et les reportées. Vous pouvez ajuster les reportées et les publier avec vos changements. Par contre, vous ne pouvez pas modifier les désirées sur les périphériques. Une notion de version gérée automatiquement est disponible également.

Blob Storage : du versioning, accès simple et sécurisé

Le stockage sous forme de blob dans Azure permet d'accéder à des données non structurées. Il y a également la possibilité d'avoir un versioning des fichiers et de gérer les accès de façon très fine. Le coût est extrêmement faible. Ce type de stockage est ce qui est utilisé pour les data lake par exemple.
Nous allons utiliser les blobs pour y stocker notre code qui sera ensuite téléchargé par notre périphérique. Lorsque vous créer un programme ou une librairie pour .NET nanoFramework, vous utilisez la chaîne de compilation de .NET, msbuild. Cela crée un exe ou une dll contenant ce que l'on appelle un Intermediate Language (IL), langage intermédiaire qui est ensuite compilé et exécuté. Pour .NET nanoFramework, nous avons un processus complémentaire qui est exécuté et qui va extraire, simplifier et transformer cet IL dans un IL capable de s'exécuter sur un MCU et qui est spécifique au Common Language Runtime (CLR) de .NET nanoFramework. Tout cela s'effectue de façon transparente lorsque vous utilisez l'extension pour .NET nanoFramework avec Visual Studio. L'extension ainsi générée est un fichier Platform Executable (PE).
L'accès au blob requiert un token qui est généré par contai-

ner (équivalent d'un répertoire) ou par fichier. La durée de validité du token peut être longue, mais on préférera des durées courtes et renouveler les tokens régulièrement si besoin. Les tokens sont dérivés d'un des tokens principaux et permettent de gérer l'accès en lecture seule ou en lecture et écriture. Cela en fait un mécanisme simple et sécurisé avec un degré de granularité suffisant.

Dans notre cas, nous fournirons dans le twin, les fichiers à télécharger et le token que l'on générera du côté Cloud. Cette génération peut-être effectuée automatiquement très simplement. Pour simplifier et pour tester, vous pouvez le générer directement depuis le portail.

L'accès à un stockage blob s'effectue avec HTTPS. Voici schématiquement comment le faire avec .NET nanoFramework, pour télécharger depuis le blob et sauver le fichier sur une carte SD ou l'espace de stockage Internet. L'exemple complet vous permettra d'avoir quelques éléments complémentaires :

```
string fileName = url.Substring(url.LastIndexOf('/') + 1);
// Si nous sommes connectés à Azure, nous nous déconnectons, car les petits MCU ne
 peuvent avoir qu'une seule connexion TLS à la fois !
if (azure.IsConnected)
{
   azure.Close();
}

httpClient.DefaultRequestHeaders.Add("x-ms-blob-type", "BlockBlob");
// Nous utilisons Tls 1.2 avec Azure
httpClient.SslProtocols = System.Net.Security.SslProtocols.Tls12;
// Nous utilisons le certificat Azure
httpClient.HttpsAuthentCert = new X509Certificate(Resource.GetBytes(Resource.
BinaryResources.azurePEMCertBaltimore));
HttpResponseMessage response = httpClient.Get($"{url}?{sas}");
response.EnsureSuccessStatusCode();

using FileStream fs = new FileStream($"{RootPath}{fileName}", FileMode.Create,
 FileAccess.Write);
response.Content.ReadAsStream().CopyTo(fs);
fs.Flush();
fs.Close();
response.Dispose();
```

Le code est simple, vous remarquerez au passage que .NET nanoFramework a des classes de type HttpClient pour faciliter la vie, des streams et bien plus encore ! La seule chose vraiment importante lorsque vous accéder à un stockage blob dans Azure, c'est de bien penser à ajouter l'en-tête *x-ms-blob-type*. Pour la sauvegarde, ici, nous utilisons une simple copie de stream. Dans l'exemple complet, il sauvegarde par morceau pour éviter de saturer la mémoire.

Chaque fichier vient avec une signature SHA256. Cela nous permet de vérifier que le fichier téléchargé est bien intégré. Nous sommes ainsi sûrs que ce qui est téléchargé est bien de ce qui est demandé, que rien n'a été modifié entre temps. Cette vérification va s'effectuer avant de charger dynamiquement le code.

Assembly.Load : chargez votre code dynamiquement en .NET

Il y a quelque chose qui m'a toujours fasciné avec .NET c'est la réflexion et le chargement dynamique des assemblies. Cela a toujours été un problème complexe à résoudre dans l'industrie. Cette capacité native et simple en .NET est clairement un avantage comparé à beaucoup d'autres plateformes. La capacité de pouvoir appeler des méthodes, des propriétés, d'avoir leurs noms, est quelque chose de vraiment important.

C'est cette capacité que nous allons utiliser, la réflexion et l'appel dynamique de code. Le tout depuis le fichier que nous aurons téléchargé au préalable. Voici à quoi peut ressembler le code :

Code complet sur programmez.com & github

Dans ce code, nous allons d'abord lire l'intégralité du fichier puis le passer à *Assembly.Load* qui va le charger en mémoire. À ce moment, aucun code n'est encore exécuté. Comme discuté précédemment, nous vérifions bien sûr l'intégrité du fichier. S'il y a un problème à ce moment-là, nous le saurons et nous pourrons prendre une action.

Assembly dynamiques : trouver le bon point d'entrée

Dans ce modèle, comme dans tout code nécessitant de charger dynamiquement du code, il est important de savoir par quel endroit rentrer dans le code. Le fameux *static void main* est ce point d'entrée dans les programmes exécutables. Ici, nous allons utiliser une fonction *Start* et utiliserons également deux autres fonctions un peu plus tard. Une première *Stop* qui permettra d'arrêter proprement le code et une autre en cas d'update des twins :

```
namespace CountMeasurement
{
  public static class OtaRunner
  {
     public static void Start(DeviceClient azureIot)
     public static void Stop()
     public static void TwinUpdated(TwinCollection twins)
  }
}
```

Ici, j'ai fait le choix d'utiliser des fonctions statiques, car ce code n'est pas destiné à être instancié plusieurs fois. Cela implique d'avoir également les variables privées statiques.

```
// Nous chargeons les assemblies qui ont été sauvées sur le disque
if (!isRunning)
{
  LoadAssemblies();

  isRunning = false;
  if (toRun != null)
  {
     Type typeToRun = toRun.GetType(OtaRunnerName);
     var start = typeToRun.GetMethod("Start");
     stop = typeToRun.GetMethod("Stop");
```

```
    twinUpated = typeToRun.GetMethod("TwinUpdated");

    if (start != null)
    {
      try
      {
        // Nous appelons notre function et passons l'objet azure en paramètre
        start.Invoke(null, new object[] { azure });
        isRunning = true;
      }
      catch (Exception)
      {
        isRunning = false;
      }
    }
  }
}
```

Dans un premier temps, nous cherchons nos trois méthodes. Si nous avons trouvé la méthode *Start* alors nous l'appelons et passons l'objet azure qui est notre connexion à Azure IoT. Cela permettra à la classe chargée dynamiquement de pouvoir envoyer de la télémétrie par exemple.
Pour la fonction TwinUptaded, elle sera appelée en cas de changement des twins. L'exemple complet montre comment récupérer des informations permettant de régler la fréquence de publication du compteur.
En plus de ces fonctions d'autres ont été ajoutées permettant un appel direct à travers le mécanisme des Direct Methods (https://docs.microsoft.com/azure/iot-hub/iot-hub-devguide-direct-methods) offertes par Azure IoT.
On ajoute la ligne `_azurelot.AddMethodCallback(GetCount);` dans notre fonction *Start*. Cela permet d'avoir directement un appel à la fonction *GetCount* :

```
private static string GetCount(int rid, string payload)
{
  Debug.WriteLine("GetCount called");
  // Nous ignorons ici le payload et retournons simplement le compteur
  return $"{{\"Counts\":{_count}}}";
}
```

Ce code est ici extrêmement simple, mais il peut bien sûr être beaucoup plus complexe. Vous pouvez accéder à des GPIO, I2C, SPI, des capteurs, utiliser d'autres nugets. L'ensemble des fichiers nécessaires et non présents sur le périphérique seront ainsi téléchargés et chargés dynamiquement.
Le résultat ressemble à cela quand on regarde la fenêtre de debug de Visual Studio :

```
Step into: Stepping over non-user code 'Program.'
Program Started.
Connecting to wifi.
Date and time is now 02/28/2022 10:02:10
The thread '<No Name>' (0x4) has exited with code 0 (0x0).
The thread '<No Name>' (0x6) has exited with code 0 (0x0).
The thread '<No Name>' (0x3) has exited with code 0 (0x0).
The thread '<No Name>' (0x5) has exited with code 0 (0x0).
Can seek: False, Lengh: 2300
Total bytes read: 2300
The nanoDevice runtime is loading the application assemblies and starting execution.
I:\CountMeasurement.pe: 2300
  Assembly: CountMeasurement (1.0.0.0)   (468 RAM - 2300 ROM - 820 METADATA)

  AssemblyRef   =      12 bytes (      3 elements)
  TypeRef       =      60 bytes (     15 elements)
  FieldRef      =       0 bytes (      0 elements)
  MethodRef     =      84 bytes (     21 elements)
  TypeDef       =      24 bytes (      3 elements)
  FieldDef      =      20 bytes (      9 elements)
  MethodDef     =      28 bytes (     14 elements)
  StaticFields  =      84 bytes (      7 elements)

  Attributes    =       0 bytes (      0 elements)
  TypeSpec      =       0 bytes (      0 elements)
  Resources     =       0 bytes (      0 elements)
  Resources Files =       0 bytes (      0 elements)
  Resources Data  =       0 bytes
  Strings       =     858 bytes
  Signatures    =     128 bytes
  ByteCode      =     496 bytes



Total: (22704 RAM - 245512 ROM - 97249 METADATA)

  AssemblyRef   =     248 bytes (     62 elements)
  TypeRef       =    2236 bytes (    559 elements)
  FieldRef      =      60 bytes (     15 elements)
  MethodRef     =    3556 bytes (    889 elements)
  TypeDef       =    4088 bytes (    511 elements)
  FieldDef      =    1888 bytes (    937 elements)
  MethodDef     =    6404 bytes (   3190 elements)
  StaticFields  =    1284 bytes (    107 elements)

  DebuggingInfo =    3232 bytes

  Attributes    =     120 bytes (     15 elements)
  TypeSpec      =      36 bytes (      9 elements)
  Resources Files =      24 bytes (      1 elements)
  Resources     =      24 bytes (      3 elements)
  Resources Data  =    1782 bytes
  Strings       =   48684 bytes
  Signatures    =   13845 bytes
  ByteCode      =   95062 bytes


*** No debugging symbols available for 'CountMeasurement'. This assembly won't be loaded in the current debug session. ***
The thread '<No Name>' (0xb) has exited with code 0 (0x0).
Start called
Getting twins
Having twins
Update time: 120000
Running...
In the thread
```

```
Sending telemetry... Counts: 1
Sending telemetry... Counts: 2
Sending telemetry... Counts: 3
```

Ce qui est intéressant à voir ici c'est que lorsque le CLR charge un assembly, on a le détail de ce qui est associé et qui peut être nécessaire pour son exécution. Cela va se passer à chaque fois qu'une assembly sera chargée. Cela facilite la mise au point bien entendu. Debugger ce type de solution est en effet complexe sur un MCU!

Une fois que la télémétrie a commencé, vous pouvez voir les messages dans le debug. Vous pouvez également utiliser Azure IoT Explorer (https://docs.microsoft.com/azure/iot-fundamentals/howto-use-iot-explorer) pour vérifier que tout fonctionne comme vous le souhaitez. Vous verrez des messages de ce type :

```
Mon Feb 28 2022 08:08:17 GMT+0100 (Central European Standard Time):
{
 "body": {
  "Counts": 2
 },
 "enqueuedTime": "Mon Feb 28 2022 08:07:17 GMT+0100 (Central
 European Standard Time)"
}
Mon Feb 28 2022 08:07:17 GMT+0100 (Central European Standard
 Time):
{
 "body": {
  "Counts": 1
 },
 "enqueuedTime": "Mon Feb 28 2022 08:07:17 GMT+0100 (Central
 European Standard Time)"
}
```

Vous pouvez vérifier également les propriétés des twins et bien-sûr envoyer des messages *GetCount*, dans notre exemple, et recevoir la réponse.

Limitations et possibles améliorations

Avant de rentrer dans les limitations, pour récapituler, le modèle mis en place avec cette solution permet, sur un microprocesseur de télécharger de façon sécurisé, intègre du code, de pouvoir mettre à jour ce code, de passer dynamiquement des propriétés, de recevoir un état de ces propriétés, d'envoyer de la télémétrie et tout cela sans avoir besoin de reflasher physiquement le périphérique. Les mécanismes en place sont complexes, mais cette complexité est entièrement masquée pour le développeur grâce à .NET et plus particulièrement .NET nanoFramework et Azure IoT.

Un des axes d'amélioration est d'implémenter un mécanisme de sûreté. Quand nous téléchargeons le code, nous supprimons les anciennes versions. Pour être plus certain, il faudrait stocker la nouvelle version en parallèle de l'ancienne et en cas de problème pouvoir retourner sur la précédente. Cela doit se faire aussi bien sur le périphérique qui doit être capable de vérifier si le code est chargé, mais aussi dans le cloud pour vérifier que la télémétrie ou les twins sont corrects. L'utilisation dans le Cloud des Azure Function (https://docs.microsoft.com/azure/azure-functions/functions-overview) et Azure Monitor (https://azure.microsoft.com/services/monitor/#overview) font des miracles dans ce domaine.

J'ai implémenté ce code et testé ce code sur des ESP32 limités en ressources mémoires. C'est la raison pour laquelle il reboote quand il a chargé une nouvelle version, se déconnecte d'Azure IoT Hub lorsqu'il doit charger les fichiers du stockage blob. Avec un périphérique disposant de plus de mémoire, il est bien sûr possible d'optimiser cette partie également.

La taille compte: plus c'est petit, mieux c'est ! Comme les assembly doivent être entièrement chargés en mémoire pour les charger dynamiquement, il vaut mieux créer plusieurs petits assembly qu'un seul gros. 4 assembly de 5 K bytes chacun seront mieux gérés qu'un seul de 20 K. Cela réduit la pression sur la mémoire surtout pour ceux qui ont une mémoire faible et partitionnée telle que les ESP32. Il est possible aussi d'améliorer ce modèle pour ne charger qu'un seul des assembly à mettre à jour. Cela sera particulièrement intéressant sur les réseaux à faible débit.

Où commencer ?

Il vous faut d'abord un périphérique tel qu'un ESP32, installer une des dernières versions de Visual Studio, l'extension .NET nanoFramework qui va avec, flasher l'ESP32 et écrire votre « hello world » en C#, mettre un point d'arrêt et appuyer sur F5. Le reste viendra tout seul ! Et évidemment, nous avons tout prévu pour vous aider avec un guide détaillé pour commencer (https://docs.nanoframework.net/content/getting-started-guides/getting-started-managed.html).

Ensuite, pour commencer avec Azure IoT et apprendre comment l'utiliser avec .NET nanoFramework, suivez les instructions dans le repository GitHub de la librairie Azure(https://github.com/nanoframework/nanoFramework.Azure.Devices)

Et bien sûr, vous avez de très nombreux exemples dans le repository à cet effet (https://github.com/nanoframework/Samples), ils vous permettrons de rapidement prendre en main .NET nanoFramework et d'en apprécier toute sa magie !

L'exemple complet est lui disponible également dans le repository des exemples (https://github.com/nanoframework/Samples/tree/main/samples/AzureSDK/AzureEdgeOta). Il comprend également un outil permettant de publier automatiquement les fichiers nécessaires dans le stockage blob et de préparer les éléments du twin à copier.

A vous donc la possibilité de flasher avec un minimum d'éléments votre MCU avec .NET nanoFramework et de bénéficier de la mise à jour de code OTA de façon sécurisée, intègre et efficace.

Une nouvelle approche d'analyse du code au sein d'un environnement DevSecOps avec Checkmarx

Pour identifier, corriger et prévenir les failles de sécurité sans ralentir le développement, les entreprises misent sur le DevSecOps, ou l'intégration de la sécurité au sein d'une approche DevOps à toutes les étapes du cycle de développement. Ce concept implique de penser la sécurité des applications et de l'infrastructure comme partie intégrante du développement applicatif. Il repose sur la conviction que les équipes de sécurité et de développement en sont co-responsables, fédérant ensemble le développement, la sécurité et les processus opérationnels. Si d'un point de vue purement fonctionnel, les organisations devront toujours effectuer une analyse incrémentielle de code source à la recherche de vulnérabilités, la clé est d'intégrer ces fonctionnalités directement dans les outils utilisés par les développeurs lors de l'écriture, l'extraction, la fusion et l'intégration de lignes de code.

Dans le cas contraire, cette analyse ne pourra pas s'intégrer efficacement dans les initiatives DevSecOps et ralentira la livraison et le déploiement logiciel. L'utilisation de composants open source et de bibliothèques tierces devenant courante dans les pratiques modernes, les organisations doivent également effectuer une analyse de composition logicielle SCA pour dresser l'inventaire des Open Source utilisés dans une application. Si cette approche est effectuée avec l'analyse de code statique, elle fonctionnera mieux dans DevSecOps, car elle ne ralentira pas non plus la livraison et le déploiement. Enfin, les organisations doivent également effectuer une analyse interactive de la sécurité pendant l'exécution des builds. L'approche optimale est d'appliquer cette analyse durant les tests fonctionnels pour éviter les délais supplémentaires inattendus.

Pour résumer, adopter et implémenter de nouvelles approches d'analyse du code adaptées aux objectifs DevSecOps implique de fédérer et de comprendre les fonctionnalités de chaque solution - SAST (analyse du code spécifique), SCA (analyse du code open source) et IAST (analyse temps réel des applications en cours d'exécution) – et de les appliquer exactement au moment et à l'endroit où elles font le plus de sens pour les développeurs. C'est l'approche DevSecOps de Checkmarx.

Exploit LOG4J, exemples de variantes et les mises à jour corrélées

Par Alex Livshiz, chercheur Checkmarx

EXÉCUTION DE CODE À DISTANCE APACHE LOG4J – CVE-2021-44228 (https://checkmarx.com/blog/apache-log4j-remote-code-execution-cve-2021-44228/)

En 2021, l'exploit zero-day le plus critique de ces dernières années a affecté la plupart des plus grandes entreprises. Découvert (https://logging.apache.org/log4j/2.x/security.html) dans la très populaire bibliothèque de logs Java log4j (https://mvnrepository.com/artifact/log4j/log4j), il a permis de lancer des attaques par exécution de code à distance RCE (Remote code execution) en enregistrant une charge utile.

La vulnérabilité a reçu le surnom de « Log4Shell », obtenu un score CVSS (Common Vulnerability Scoring System) de 10 - le risque le plus élevé - et a été publiée par GitHub advisory (https://github.com/advisories/GHSA-jfh8-c2jp-5v3q) avec un niveau de sévérité critique.

PORTÉE DE L'EXPLOIT

Log4Shell a été exploité pendant quelques jours avant sa divulgation publique. De plus, des tentatives de scan de log4shell ont été découvertes jusqu'à deux semaines auparavant. Les attaquants ont pu installer des cryptomineurs, créer des botnets et voler des données sensibles et des informations d'identification système. À ce jour, on estime qu'il a affecté plus d'un million de machines.

CVES PERTINENTS

Depuis sa divulgation, cinq CVE (Common Vulnerabilities and Exposures) concernant Log4j2 et Log4j1 ont été publiées :

LOG4J2 : CVE-2021-44228

Les versions de 2.0-beta9 à 2.15.0 de Log4j2 (excluant 2.12.x après la version 2.12.1) sont vulnérables à l'exécution de code à distance via l'analyseur JNDI LDAP (Lightweight Directory Access Protocol). Un attaquant capable de contrôler les messages de log ou les paramètres de message de logs peut exécuter du code arbitraire chargé à partir de serveurs LDAP lorsque la substitution de recherche de message est activée. La vulnérabilité initiale désignée CVE-2021-44228 (https://nvd.nist.gov/vuln/detail/CVE-2021-44228) aurait été corrigée dans les versions 2.12.2 et 2.15.0. Le correctif inclut la désactivation de JNDI par défaut et la restriction de l'accès LDAP via le JNDI dans la recherche d'objets et le gestionnaire JNDI de log4j2.

Cette vulnérabilité a reçu le score CVSS de 10 le plus élevé (https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H&version=3.1) et affecte les packages suivants, disponibles via Maven Package Manager :

- org.apache.logging.log4j:log4j-api
- org.apache.logging.log4j:log4j-core

Notes - *La vulnérabilité se situe dans log4j-core. Logger, utilisé pour déclencher l'exploit, est défini dans log4j-api. Pour détecter une telle utilisation et le chemin exploité, Checkmarx a ajouté les méthodes de Logger comme vulnérables (finissant par déclencher la vulnérabilité selon l'étude). Cette approche est décrite dans la page Avis de Github pour cette vulnérabilité.*

LOG4J2 : CVE-2021-45046

Le 11 décembre 2021, Checkmarx a découvert que le correctif de CVE-2021-44228 était incomplet dans certaines configurations autres que celles par défaut. Cela permettait aux attaquants disposant de données d'entrée

malveillantes conçues à l'aide d'un modèle de recherche JNDI d'engendrer une fuite d'informations, d'exécuter du code à distance dans certains environnements ainsi que du LCE (Local Code Execution) dans l'ensemble des environnements. Le RCE était également possible dans certains environnements macOS. Cette vulnérabilité complémentaire a été désignée sous le nom de CVE-2021-45046 (https://nvd.nist.gov/vuln/detail/CVE-2021-45046) et corrigée dans les versions 2.12.2 et 2.16.0 en désactivant JNDI par défaut et en supprimant la recherche de message.
Comme cette vulnérabilité était initialement considérée comme n'autorisant que les attaques DOS (Déni de Service), le score CVSS attribué initialement était de 3,7. Son impact plus menaçant qui a été découvert plus tard lui a valu le nouveau score CVSS de 9,0 (https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:H/A:H&version=3.1).
Cette vulnérabilité étant une extension de CVE-2021-44228, les packages affectés sont les mêmes.

Options d'atténuation
pour CVE-2021-44228 et CVE-2021-45046 :

- Les utilisateurs Java 8 (ou version ultérieure) doivent effectuer une mise à niveau vers la version 2.16.0 ou plus.
- Les utilisateurs Java 7 doivent effectuer une mise à niveau vers la version 2.12.2 ou plus.
- Supprimer la classe JndiLookup du chemin de classe :

zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class

LOG4J2 : CVE-2021-45105

Checkmarx a découvert le 15 décembre 2021 que les versions 2.0-alpha1 à 2.16.0 (à l'exclusion de 2.12.x de 2.12.3 : https://github.com/apache/logging-log4j2/commit/ff844c0a4d8eb4afe260494be1c2dc1b52cbf50d) log4j2 étaient vulnérables aux attaques DOS, car non protégées de la récursivité incontrôlée des recherches autoréférentielles. Ces dernières entraînant à leur tour une erreur Stack Overflow mettant fin au processus. Cette vulnérabilité a été publiée sur NVD le 18 décembre 2021, sous l'appellation CVE-2021-45105 (https://nvd.nist.gov/vuln/detail/CVE-2021-45105) et a été corrigée dans la version 2.17.0 en corrigeant la récursivité de substitution de chaîne et en limitant JNDI au seul protocole Java. Selon GitHub Advisory et les correctifs précédents pour les variantes de log4Shell, le correctif pour les utilisateurs Java7 devrait être publié dans la prochaine version 2.12.3 (https://github.com/apache/logging-log4j2/commit/ff844c0a4d8eb4afe260494be1c2dc1b52cbf50d). Le score CVSS qui lui est attribué par Apache est de 5,9 (https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H&version=3.1).

Options d'atténuation pour CVE-2021-45105

- Les utilisateurs Java 8 (ou version ultérieure) doivent effectuer une mise à niveau vers la version 2.17.0 ou plus.
- Les utilisateurs Java 7 doivent effectuer une mise à niveau vers la version 2.12.3 ou plus.
- Les utilisateurs Java 6 doivent effectuer une mise à niveau vers la version 2.3.1 ou plus.
- Au sein de PatternLayout dans la configuration de journalisation, remplacer les recherches contextuelles, telles que ${ctx:loginId} ou $${ctx:loginId}, par des modèles de mapping contextuels de menaces (%X, %mdc ou %MDC).
- Dans la configuration, supprimer les références aux recherches contextuelles, telles que ${ctx:loginId} ou $${ctx:loginId}, lorsqu'elles proviennent de sources externes à l'application, telles que les en-têtes HTTP ou les entrées utilisateur.

LOG4J2 : CVE-2021-44832

L'équipe de recherche Checkmarx a révélé une nouvelle vulnérabilité découverte le 28 décembre 2021 (https://checkmarx.com/blog/cve-2021-44832-apache-log4j-2-17-0-arbitrary-code-execution-via-jdbcappender-datasource-element/). Cette vulnérabilité permet l'exécution arbitraire de code (ACE) dans les versions 2.0-beta7 à 2.17.0 (à l'exclusion des versions de correctifs de sécurité 2.3.2 et 2.12.4).
Lorsqu'un attaquant prend le contrôle de la configuration de log (via une attaque MITM - https://en.wikipedia.org/wiki/Man-in-the-middle_attack - car il existe une fonctionnalité permettant de charger un fichier de configuration distant dans log4j), il peut construire une configuration malveillante à l'aide de JDBC Appender avec une source de données référençant un URI JNDI, qui peut ensuite exécuter du code distant.
Cette vulnérabilité a été corrigée dans la version 2.17.1 en limitant les noms de sources de données JNDI au protocole Java et le score CVSS de 6,6 lui a été attribué (https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H&version=3.1), un score de criticité légèrement inférieur, car il est plus complexe à exploiter que les variantes log4Shell précédentes. CVE-2021-44832 affecte uniquement le package log4j-core.

Options d'atténuation pour CVE-2021-44832

- Les utilisateurs Java 8 (ou version ultérieure) doivent effectuer une mise à niveau vers la version 2.17.1 ou plus.
- Les utilisateurs Java 7 doivent effectuer une mise à niveau vers la version 2.12.4 ou plus.
- Les utilisateurs Java 6 doivent effectuer une mise à niveau vers la version 2.3.2 ou plus.
- Dans les versions antérieures, confirmer que l'appendice JDBC n'est pas configuré pour utiliser un protocole autre que Java.

Remarque importante concernant les vulnérabilités log4j2 : seul le fichier JAR log4j-core est impacté par ces vulnérabilités. Les applications utilisant uniquement le fichier JAR log4j-api sans le fichier JAR log4j-core ne sont pas affectées. D'autres projets comme Log4net et Log4cxx ne sont pas touchés par ces vulnérabilités.

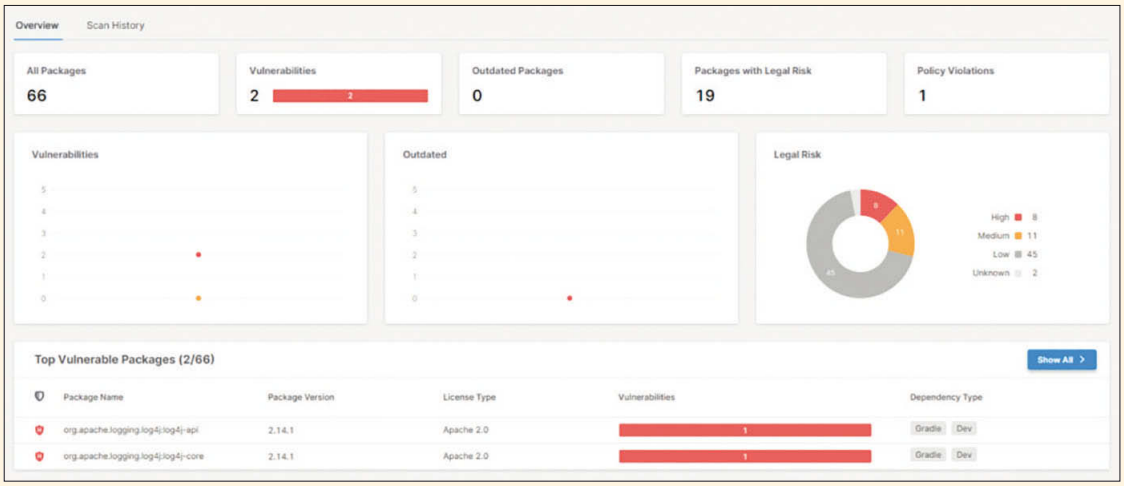
LOG4J1 : CVE-2021-4104

Divulgué le 13 décembre 2021 et publié le 14 décembre 2021 sur NVD sous l'appellation CVE-2021-4104 (https://nvd.nist.gov/vuln/detail/CVE-2021-4104), les équipes ont découvert que log4j1 était également vulnérable à la vulnérabilité log4Shell - auparavant censée affecter uniquement log4j2.
La cause racine de cette vulnérabilité se trouve dans la classe org.apache.log4j.net.JMSAppender, vulnérable à la désérialisation de données non fiables lorsque l'attaquant dispose *d'un accès en écriture* à la configuration Log4j. L'attaquant peut fournir des payloads malveillants aux paramètres de configuration, ce qui oblige JMSAppender à effectuer des requêtes JNDI entraînant l'exécution de code à distance. Cela affecte les configurations autres que celles par défaut de Log4j 1.2 puisque la configuration de JMSAppender est désactivée par défaut.
Le score CVSS attribué à cette vulnérabilité est de 6,6 (https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H&version=3.1), ce qui est inférieur à CVE-2021-44228, car l'attaquant doit disposer d'un accès en écriture à la configuration log4j pour l'exploiter.
Cette vulnérabilité affecte le package **log4j:log4j**, disponible via le Gestionnaire de packages Maven :

Options d'atténuation pour CVE-2021-4104 :

- Les utilisateurs doivent effectuer une mise à niveau vers Log4j2, soit 2.12.4, 2.3.2, 2.17.1 ou supérieure, car il résout de nombreux autres problèmes des versions précédentes.
- Ne pas exposer à des appelants non approuvés tout mécanisme susceptible d'autoriser l'accès à la classe JMSAppender ou d'apporter des modifications et une configuration à des instances de cette dernière.

Figure 1 *Présentation SCA, avec la liste des packages et des risques détectés*



- Commenter ou supprimer le JMSAppender dans la configuration log4j s'il est utilisé
- Supprimer le chemin de classe JMSAppender :

zip -q -d log4j-*.jar org/apache/log4j/net/JMSAppender.class

Remarque importante : Apache Log4j 1.2 a atteint sa fin de vie en août 2015.

Détection de Log4Shell avec Checkmarx SCA

Checkmarx SCA (https://checkmarx.com/product/cxsca-open-source-scanning/) fournit une détection rapide et facile des vulnérabilités Log4Shell mentionnées ci-dessus dans les dépendances open source. Les captures d'écran suivantes montrent les résultats de l'analyse SCA du code avec des dépendances tiers vulnérables. **Figure 1**

Les solutions d'analyse de composition logicielle (SCA) sont déterminantes pour les organisations qui utilisent des logiciels open source. Checkmarx SCA permet de résoudre les problèmes de sécurité open source au plus tôt dans le cycle complet de développement SDLC afin d'identifier et de gérer les risques de manière plus efficace.

Comment corriger la vulnérabilité Log4j RCE ?

Le moyen le plus simple et le plus recommandé consiste à effectuer la mise à jour vers log4j version 2.15.0 ou ultérieure (https://logging.apache.org/log4j/2.x/download.html). En cas de non possibilité de mise à jour du package, ce comportement exploitable peut être atténué dans les versions précédentes 2.10.0 à 2.15.0 en définissant la propriété système sur :

log4j2.formatMsgNoLookups=true

En outre, une variable d'environnement peut être définie pour ces mêmes versions affectées :

LOG4J_FORMAT_MSG_NO_LOOKUPS=true

Pour les versions de 2.0-beta9 à 2.10.0, la solution est de supprimer JndiLookup du chemin de classe. La commande est la suivante :

zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class

Plus de détails dans la validation GitHub (https://github.com/apache/logging-log4j2/commit/c77b3cb39312b83b053d23a2158b99ac7de44dd3).

Pourquoi est-ce si critique ?

Selon le service de surveillance néo-zélandais CERT (Computer Emergency Response Team - https://www.cert.govt.nz/it-specialists/advisories/log4j-rce-0-day-actively-exploited/) et de Greynoise (https://twitter.com/_mattata/status/1469144854672379905), les attaquants recherchent activement des serveurs vulnérables pour exploiter cette attaque. Il existe plus de 100 hôtes distincts (https://twitter.com/GreyNoiseIO/status/1469326260803416073) qui analysent Internet afin de trouver des moyens

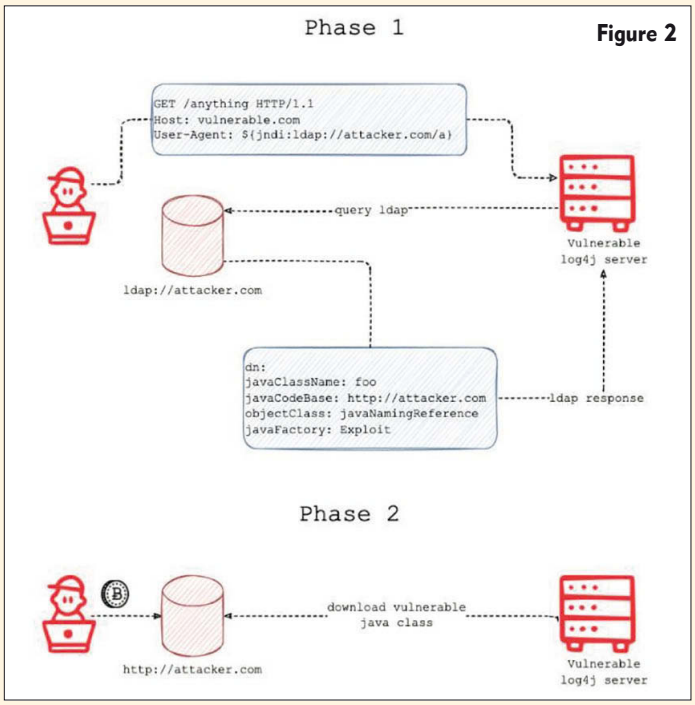


Figure 2

Source: Fastly https://www.fastly.com/blog/digging-deeper-into-log4shell-0day-rce-exploit-found-in-log4j

d'exploiter cette vulnérabilité. L'impact est d'envergure car **log4j** est une bibliothèque de logs **extrêmement** courante, utilisée dans la plupart des applications Java, y compris dans les systèmes d'entreprise pour enregistrer les informations de logs. Moins de 24 heures après la publication de la vulnérabilité, un crypto-mineur (https://twitter.com/GossiTheDog/status/1469322120840708100) était déjà déployé.

Suis-je vulnérable ?

Il est possible de vérifier librement si votre domaine est vulnérable à la CVE-2021-44228 à l'aide d'outils de test open source, comme huntresslabs/log4shell-tester (https://github.com/huntresslabs/log4shell-tester) par exemple.

En outre, si votre application utilise une version de log4j inférieure à 2.15.0 comme **package direct ou transitif**, vous êtes vulnérable. Vérifiez également la présence de ces hachages dans votre inventaire logiciel. Si vous les trouvez c'est que vous utilisez un **log4j** vulnérable dans vos systèmes :

GitHub - mubix/CVE-2021-44228-Log4Shell-Hashes (https://github.com/mubix/CVE-2021-44228-Log4Shell-Hashes) : hachages pour les versions VULNÉRABLES de LOG4J.

Pour contourner le paramètre trustURLCodebase=false, une solution a été trouvée (https://twitter.com/marcioalm/status/1470361495405875200). Pour être protégé, nous vous recommandons de mettre à jour votre bibliothèque log4j. **Figure 2**

WebAuthn : sécurisez votre connexion

Christophe Villeneuve

Consultant open source 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…

L'authentification d'aujourd'hui ne sera pas celle de demain. Car WebAuthn va aider les utilisateurs à s'identifier plus facilement aux applications et sites Web, sans nécessairement être obligés de mémoriser des mots de passe complexes. Tout le monde est gagnant : l'utilisateur n'a plus de mots de passe à connaître et pour le développeur, l'implémentation et la gestion sont plus simples.

WebAuthn signifie Web Authentication. Il s'agit d'un standard du Web Consortium (W3C) avec la contribution de la FIDO Alliance qui propose une interface d'authentification des utilisateurs aux applications Web à l'aide de clés asymétriques. Cela se traduit par une API Web Authentication dédiée.

NDLR : depuis 2-3 ans, les géants de la tech parlent de la fin des mots de passe. Ils travaillent avec l'alliance FIDO. Durant la dernière WWDC, Apple a présenté son système Passkeys qui repose justement sur WebAuthn. Mais dans le cas d'Apple, le matériel par défaut n'est pas une clé USB mais plutôt FaceID ou TouchID.

C'est pourquoi une extension de l'API « credential Management » est prise en charge par l'ensemble des environnements (Linux, Windows, Mac), Android, navigateurs (Chrome, Edge, Firefox…). Cette API d'authentification Web est disponible en version stable et peut être déployée et mise en œuvre par des sites Web sous sa forme actuelle, sans crainte de modifications futures.

Ainsi, WebAuthn permet aux utilisateurs de s'enregistrer et de s'authentifier sur des sites Web ou des applications mobiles en utilisant un "authentificateur" au lieu d'un mot de passe à partir d'une interface d'authentification des utilisateurs aux applications Web à l'aide de clés asymétriques.

L' "authentificateur" peut être une clé de sécurité matérielle que l'utilisateur a connecté à son ordinateur ou un ID biométrique qui peut être utilisé à partir des capteurs du PC ou du smartphone, tels que les empreintes digitales, les scans faciaux, les scans iris et autres.

Standard

Les experts en sécurité considèrent que cette API est un système d'authentification sans mot de passe, qui peut être considéré comme l'avenir de la sécurité des comptes d'utilisateurs. C'est pourquoi ce standard viserait à éliminer la nécessité de saisir des mots de passe lorsque les utilisateurs se connectent à Internet.

Ainsi, son but va vous permettre de sécuriser l'accès aux applications web, pour empêcher les attaques de phishing, ce qui est utile pour les applications en B2B/B2C.

Matériel

Pour utiliser WebAuthn, il faut acquérir une clé USB sécurisée de type FIDO U2F pour stocker l'ensemble des clés d'authentification. Par exemple, une clé Winkeo FIDO U2F coûte 20 €. U2F (second facteur universel) est une norme d'authentification ouverte qui vise à renforcer et à simplifier l'authentification à deux facteurs en utilisant des périphériques USB ou à communication en champ proche (NFC). **Figure 1**

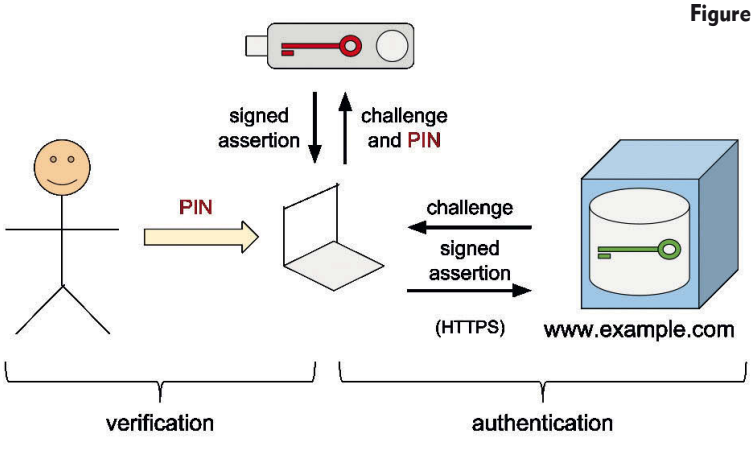
L'utilisation consiste à placer cette clé sur le port USB de votre choix de votre ordinateur et lorsque vous souhaitez l'utiliser, vous devez saisir un mot de passe que vous avez déterminé au préalable.

Fonctionnement **Figure 2**

La mise en place d'un workshow avec WebAuthn dans une application web se déroule en plusieurs étapes.

La première étape consiste à enregistrer un nouvel utilisateur qui pourra se connecter par la suite. Celui-ci envoie d'abord un nom d'utilisateur et un mot de passe de son choix (même si la définition d'un mot de passe n'est pas obligatoire, il est recommandé d'avoir une méthode d'authentification de secours). Ensuite, le serveur renvoie une réponse qui sera utilisée avec l'authentificateur. Ce dernier génère une assertion qui est ensuite reçue par le serveur. Les informations d'iden-

Figure 1



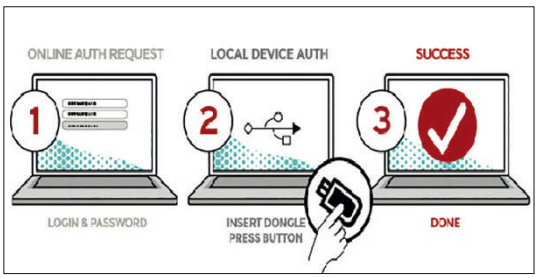


Figure 2

tification créées auparavant seront attribuées à l'assertion générée afin de reconnaître l'utilisateur en cas de besoin.
Lorsqu'il s'agit de se connecter, le workflow est assez similaire. L'utilisateur saisit d'abord son nom d'utilisateur et le RP (Relying Party) répond automatiquement avec un challenge qui demande à l'utilisateur de s'authentifier. Pour cela, il doit utiliser l'identifiant attribué auparavant. Cela permet au serveur d'identifier l'utilisateur. Si l'authentification s'effectue avec succès, l'utilisateur peut se connecter.
L'ensemble de ces opérations peut se représenter comme l'image **Figure 3**
Si vous avez un authentificateur supporté, vous pouvez tester ce processus ici.

En pratique

Au niveau du code, l'opération d'enregistrement se décompose de la manière suivante.

La partie création :

```
navigator.credentials
 .create({
  publicKey: {

   challenge: base64url.decode("<%= challenge %>"),

   rp: {
    name: "NomCodeApplication"
   },
   user: {
    id: base64url.decode("<%= id %>"),
    name: "<%= name %>",
    displayName: "<%= displayName %>"
   },
   authenticatorSelection: { userVerification: "preferred" },
   attestation: "direct",
   pubKeyCredParams: [
    {
     type: "public-key",
     alg: -7 // "ES256" IANA COSE Algorithms registry
    }
   ]
 }
})
.then(res => {
 var json = publicKeyCredentialToJSON(res);

 post("/webauthn/register", {
  state: "<%= state %>",
  provider: "<%= provider %>",
  res: JSON.stringify(json)
 });
})
.catch(console.error);
```

Le code se compose de plusieurs parties avec des arguments à prendre en compte :

- PublicKey = Générer une clé aléatoire (16 caractères)
- RP = Nom de l'objet décrivant la partie dépendante lors de la création
- User = Partie utilisateur
- Res = Réponse au format JSON, renvoyée par le serveur

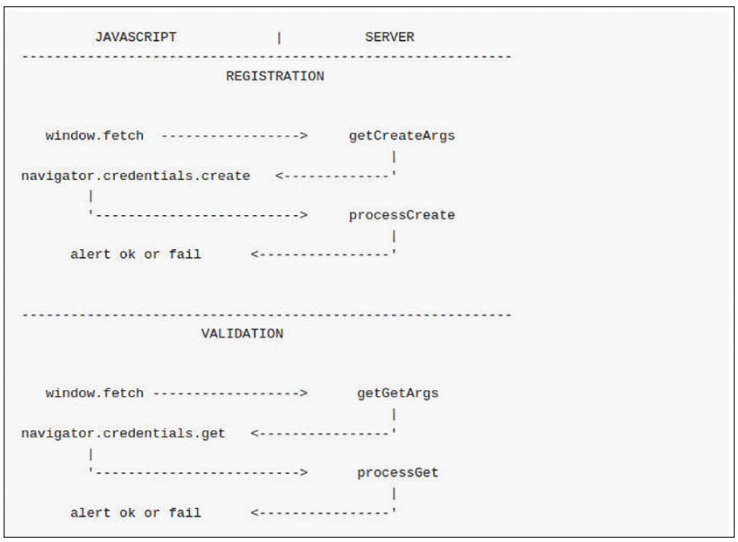
La partie Authentification :

```
navigator.credentials
 .get({
  publicKey: {

   challenge: base64url.decode("<%= challenge %>"),
   allowCredentials: [
    {
     id: base64url.decode("<%= id %>"),
     type: "public-key"
    }
   ],
   timeout: 15000,
   authenticatorSelection: { userVerification: "preferred" }
  }
})
.then(res => {
 var json = publicKeyCredentialToJSON(res);

 post("/webauthn/authenticate", {
  state: "<%= state %>",
  provider: "<%= provider %>",
  res: JSON.stringify(json)
 });
})
.catch(err => {
 alert("Invalid FIDO device");
});
```

Ici, nous avons :

- PublicKey = Générer une clé aléatoire (16 caractères)
- Res = Réponse au format JSON, renvoyée par le serveur

Figure 3



Une application d'authentification

La mise en place d'une authentification WebAuthn dans un projet web peut se construire de la manière suivante : nous créons une page d'accueil et vérifions que le navigateur est bien compatible avec l'API WebAuthn.

Fichier index.html

```
<!DOCTYPE html>
<html>

<head>
 <meta charset="utf-8">
 <title>WebAuthn Demo</title>
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.
min.js"></script>
</head>

<body>

 Username:
 <br>
 <input type="text" name="username" id="email" placeholder="votre
email">
 <br>
 <br>
 <button onclick="registerUser()">Register</button>
 <button onclick="loginUser()">Login</button>

 <script>

  $(document).ready(function () {
      // Verifie si le navigateur est compatible WebAuthn
     if (!window.PublicKeyCredential) {
        alert("Error: le navigateur n'est pas compatible avec WebAuthn");
        return;
     }
  });

 </script>
</body>
</html>
```

Ce formulaire propose 2 boutons : création et la connexion

l'enregistrement

Le processus d'enregistrement entre le serveur et le formulaire d'authentification se déroule en plusieurs étapes.

1. Le serveur met à disposition différentes informations, exposé au JavaScript
2. Le navigateur relaie ces informations d'identité
3. L'authentificateur vérifie l'utilisateur et la nouvelle paire de clés
4. Il renvoie une attestation de cette nouvelle clé
5. Le JavaScript renvoie au serveur l'attestation
6. Le serveur valide la réponse

Au niveau du code, lors de l'enregistrement (register) WebAuthn indique qu'un authentificateur génère une paire de clés publiques/privées qui est liée aux informations fournies par le serveur (c'est-à-dire les informations sur l'utilisateur, l'organisation).

```
</script>
function registerUser() {
```

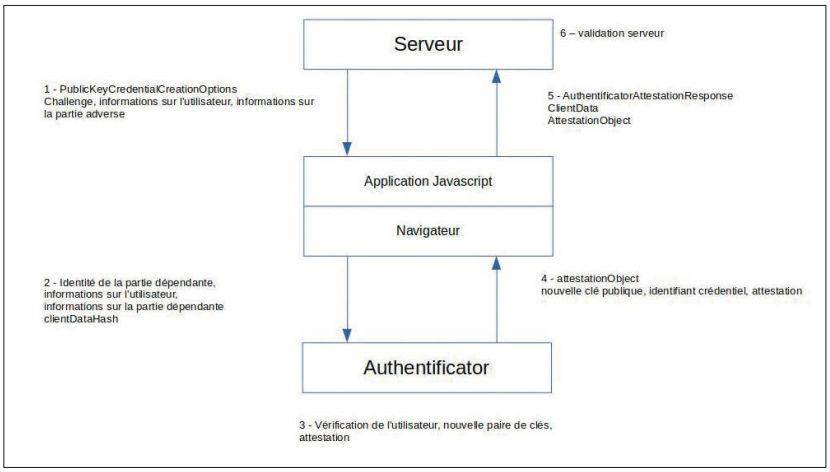


Figure 4

```
  username = $("#email").val()
  if (username === "") {
   alert("Saisir un email");
   return;
  }

  $.get(
   '/register/begin/' + username,
   null,
   function (data) {
     return data
   },
   'json')
   .then((credentialCreationOptions) => {
       // Execution
   });
}

</script>
```

Lors de la création, un appel sera effectué vers le serveur, en appelant le fichier server.go contenant :

```
package main

import (
    "log"
    "net/http"

    "github.com/duo-labs/webauthn.io/session"
    "github.com/duo-labs/webauthn/webauthn"
    "github.com/gorilla/mux"
)

var webAuthn *webauthn.WebAuthn
var sessionStore *session.Store
var userDB *userdb

func main() {

    var err error
    webAuthn, err = webauthn.New(&webauthn.Config{
        RPDisplayName: "Foobar Corp.",    // display name for your site
        RPID:      "localhost",      // generally the domain name for your site
```

```
    })

    if err != nil {
        log.Fatal("failed to create WebAuthn from config:", err)
    }

    userDB = DB()

    sessionStore, err = session.NewStore()
    if err != nil {
        log.Fatal("failed to create session store:", err)
    }

    r := mux.NewRouter()

    r.HandleFunc("/register/begin/{username}", BeginRegistration).
Methods("GET")

    r.PathPrefix("/").Handler(http.FileServer(http.Dir("./")))

    serverAddress := ":8080"
    log.Println("starting server at", serverAddress)
    log.Fatal(http.ListenAndServe(serverAddress, r))
}
```

Le fichier a pour but d'initialiser WebAuthn, nous fournissons une configuration avec deux champs (il y a d'autres champs optionnels), tous sont liés à la Relying Party (RP) qui est l'application/entité web qui enregistre et authentifie l'utilisateur.
Les champs sont *RPDisplayName* : le nom d'affichage du site du RP (par exemple Foobar Corp.) et *RPID* : basé sur le nom de domaine du RP (par exemple foobar.com).
Nous stockons les utilisateurs enregistrés qui seront gérés par le fichier userdb.go suivant :

```
package main

import (
    "fmt"
    "sync"
)

type userdb struct {
    users map[string]*User
    mu    sync.RWMutex
}

var db *userdb

// DB returns a userdb singleton
func DB() *userdb {

    if db == nil {
        db = &userdb{
            users: make(map[string]*User),
        }
    }

    return db
}

// GetUser returns a *User by the user's username
func (db *userdb) GetUser(name string) (*User, error) {

    db.mu.Lock()
    defer db.mu.Unlock()
    user, ok := db.users[name]
    if !ok {
        return &User{}, fmt.Errorf("error getting user '%s': does not exist", name)
    }

    return user, nil
}

// PutUser stores a new user by the user's username
func (db *userdb) PutUser(user *User) {

    db.mu.Lock()
    defer db.mu.Unlock()
    db.users[user.name] = user
}
```

L'implémentation User se fait par l'interface utilisateur de duo-lab. :
Code complet sur programmez.com & github

Après dans le fichier server.go, nous implémentons le gestionnaire BeginRegistration.

```
func BeginRegistration(w http.ResponseWriter, r *http.Request) {

    // get username
    vars := mux.Vars(r)
    username, ok := vars["username"]
    if !ok {
        jsonResponse(w, fmt.Errorf("must supply a valid username i.e. foo@
bar.com"), http.StatusBadRequest)
        return
    }
}
```

Dans le fichier JS où se trouve la fonction registerUser(), les options de création d'accréditation retournées par le serveur, ressemblent à ceci :

```
{
  "publicKey": {
   "challenge": "FsxBWwUb1jOFRA3ILdkCsPdCZkzohvd3JrCNeDqWpJQ=",
   "rp": {
    "name": "Foobar Corp.",
    "id": "localhost"
   },
   "user": {
    "name": "foo@bar.com",
    "displayName": "foo",
    "id": "x0ywiL6f3q9EAA=="
   },
   "pubKeyCredParams": [
    {
```

```
    "type": "public-key",
    "alg": -7
   },
   {
    "type": "public-key",
    "alg": -35
   },
   ...
   {
    "type": "public-key",
    "alg": -8
   }
  ],
  "authenticatorSelection": {
   "userVerification": "preferred"
  },
  "timeout": 60000,
  "attestation": "direct"
 }
}
```

Connexion

La partie connexion de notre formulaire est un peu différente de la partie d'enregistrement : **Figure 4**

1. Le serveur met à disposition différentes informations, exposées au JavaScript
2. Le navigateur relaie ces informations hashées du client
3. L'authentificateur vérifie l'utilisateur et confirme la clé
4. Il renvoie une attestation de signature
5. le JavaScript renvoie l'attestation au serveur au format JSON
6. le serveur valide la réponse

Au niveau du code, l'appel est effectué à partir du bouton loginUser() de la page index.html en récupérant le nom d'utilisateur et en effectuant un appel GET vers /login/begin/{username}.

```
function loginUser() {

 username = $("#email").val()
 if (username === "") {
  alert("please enter a username");
  return;
 }

 $.get(
  '/login/begin/' + username,
  null,
  function (data) {
   return data
  },
  'json')
  .then((credentialRequestOptions) => {
    // TODO
  })
}
```

L'appel de la fonction « BeginLogin » est très similaire à « BeginRegister ». C'est à dire, nous appelons duo-labs/webAuthns BeginLogin() avec un utilisateur (et des LoginOptions optionnelles) et nous obtenons les options retournées (implémentation de PublicKeyCredentialRequest Options discutées ci-dessous), sessionData (utilisée pour vérifier l'assertion retournée), et une erreur si quelque chose ne va pas. Nous stockons les données de session et retournons les options à l'utilisateur.

Code complet sur programmez.com & github

Le credentialRequestOptions se présente comme suit :

```
{
 "credentialRequestOptions": {
  "publicKey": {
   "challenge": "AZeYdiLc1hGDU0wo9NOZmZG9vu+aPnff2aPFgJEw1HI=",
   "timeout": 60000,
   "rpId": "localhost",
   "allowCredentials": [
    {
     "type": "public-key",
     "id": "ABJcniVJwrH45aueEJJsD0LFTGMUxot1sHC0ttym/p7rNPF72Zc
/NcGo05j3KzDkU5fWELqRz7h9"
    }
   ]
  }
 }
}
```

Les autres étapes de vérifications sont traitées sur le même principe.

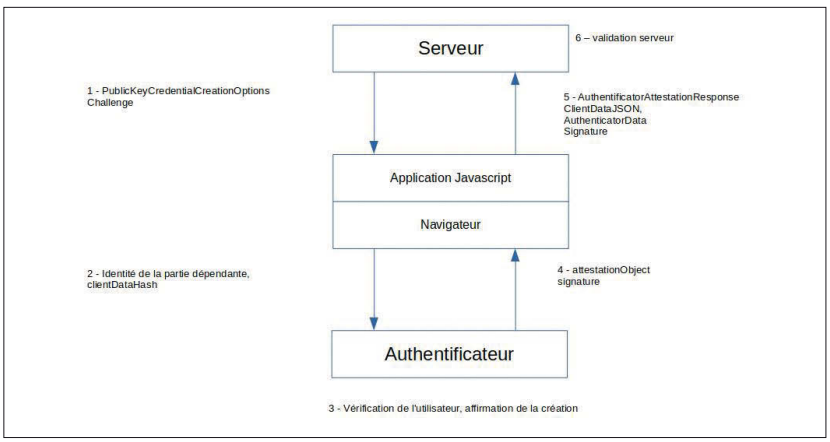
Au final

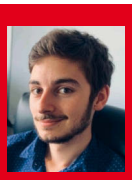
Comme le montre l'exemple, la mise en place de webauthn est assez simple. L'ensemble du code est disponible sur le dépôt de l'auteur de l'extension que vous pouvez retrouver : https://github.com/hbolimovsky/webauthn-example

Conclusion

WebAuthn fait évoluer la gestion de l'authentification, mais est dépendant aussi du matériel, qui commence à être intégré dans les différents terminaux. Même si la route peut sembler longue, son fonctionnement est plutôt simple.

Figure 4





Pierre Bouillon
Ingénieur logiciel Full Stack développant principalement des applications web avec .NET et Angular.
https://pbouillon.github.io

.NET 7: nouveautés et améliorations

Dernière version en date du framework cross-platform de Microsoft, .NET 7 apporte son lot de nouveautés et d'améliorations à de nombreux niveaux et sur divers aspects de son écosystème (Minimal API, System.Text.JSON, etc.). A cela se rajoute une nouvelle version de C#.

.Net 7 n'est pas une version LTS comme l'est .Net 6 et la future v8. Si vous cherchez une version à support long, soit vous restez sur la 6 ou attendez la 8.

Installation et mise à jour

L'installation de .NET 7 se fait de la même manière que les versions précédentes, à savoir via le site de Microsoft https://dotnet.microsoft.com.

Plusieurs images docker sont également disponibles sur le .NET du docker hub de Microsoft : https://hub.docker.com/_/microsoft-dotnet.

Localement, pour utiliser la dernière version du framework, il vous faudra simplement changer l'attribut TargetFramework du csproj de votre projet cible pour net7.0.

Certaines fonctionnalités accompagnant la dernière version de C# ne sont pas encore immédiatement utilisables. Aussi, pour activer celles qui vous manqueront, vous pouvez manuellement ajouter la propriété LangVersion et l'assigner à preview.

Voici un exemple du type de csproj que vous pourriez avoir pour un projet console paramétrée pour .NET 7 :

```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net7.0</TargetFramework>
  <LangVersion>preview</LangVersion>
 </PropertyGroup>
</Project>
```

Étant une mise à jour majeure, cette version introduit des changements qui rendent incompatibles ou obsolètes certains appels effectués avec des versions antérieures du framework.

Par exemple, la logique de comparaison des types NaN a été altérée, le package Microsoft.Data.SqlClient a été mis à jour, certains types d'exceptions levé par certaines API ont été modifié, et plus encore.

Pour consulter ces dernières, Microsoft les a listées, ainsi que les éventuelles actions de remédiation à entreprendre, dans une page de documentation dédiée : https://docs.microsoft.com/fr-fr/dotnet/core/compatibility/7.0.

.NET
Gestion du polymorphisme dans System.Text.Json

Jusqu'alors la sérialisation et la désérialisation d'instances avec System.Text.Json ne permettaient pas de gérer facilement le polymorphisme et nécessitaient de coder des contournements pour réussir à le prendre partiellement en charge. Dans le cadre d'un système réagissant à des événements extérieurs par exemple, il n'est pas rare qu'un message soit un dérivé d'un type de base.

Prenons par exemple une entité Customer et un événement CustomerCreated associé à sa création :

```
record Customer(int Id, string Name);
record CustomerCreated(int Id, string Name, DateTime CreatedOn)
  : Customer(Id, Name);
```

Si nous souhaitons émettre des événements relatifs aux clients, le système peut les sérialiser et désérialiser en tant que Customer pour ne pas réaliser une implémentation pour chaque classe héritant de la classe mère.

Dans notre cas, la sérialisation d'un événement CustomerCreated et sa désérialisation pourrait ressembler à ceci :

```
Customer customer = new CustomerCreated(1, "John Doe", DateTime.Now);

var serialized = JsonSerializer.Serialize(customer);
// { "Id": 1, "Name": "John Doe" }

var deserialized = JsonSerializer.Deserialize<CustomerCreated>(serialized);
// {CustomerCreated { Id = 1, Name = John Doe, CreatedOn = 01/01/0001 00:00:00 }}
```

Pourtant, avec cette approche, deux problèmes majeurs sont immédiatement visibles. Tout d'abord, la valeur sérialisée ne contient aucune information qui n'est pas dans le type de base et la date de création est ainsi perdue. En réalité, le sérialiseur ne prend pas en compte le type réel du paramètre, mais simplement le type sous lequel il est utilisé. Ici, comme notre instance de CustomerCreated est manipulée sous le type Customer, tout le contexte rajouté par la classe fille est perdu. Ensuite, cause directe du problème précédent, la désérialisation en le type fille résulte en la complétion des propriétés manquantes par leurs valeurs par défaut.

Notre événement, une fois la sérialisation/désérialisation faite, contient ainsi des valeurs totalement erronées (1er janvier de l'an 1 à minuit au lieu de la date courante).

C'est à ce type de problème que .NET 7 apporte une solution en permettant aux développeurs de définir, pour une classe donnée, les types des classes héritant de cette dernière.

Dans notre exemple, il suffit alors de spécifier que Customer est héritée par CustomerCreated avec l'attribut JsonDerivedTypeAttribute :

```
[JsonDerivedType(typeof(CustomerCreated))]
record Customer(int Id, string Name);

record CustomerCreated(int Id, string Name, DateTime CreatedOn)
  : Customer(Id, Name);
```

En décorant la classe Customer de cette manière, on indique alors au sérialiseur qu'il doit traiter la désérialisation de notre instance en gardant le contexte associé.
Ainsi, le même code nous donne alors les résultats suivants dans lesquels le contexte a pu être préservé :

```
Customer customer = new CustomerCreated(1, "John Doe", DateTime.Now);

var serialized = JsonSerializer.Serialize(customer);
// {"CreatedOn":"2022-07-10T11:50:41.9534943+02:00","Id":1,"Name":"John Doe"}

var deserialized = JsonSerializer.Deserialize<CustomerCreated>(serialized);
// {CustomerCreated { Id = 1, Name = John Doe, CreatedOn = 10/07/2022 11:50:41 }}
```

Cette approche répond à notre problème dans le cadre où toutes les instances sont manipulées au sein même du runtime.
En revanche, si l'objet sérialisé provient d'un système tiers, System.Text.Json manquera alors de contexte pour savoir comment désérialiser le message vers le bon type.
Là encore, une solution est apportée au travers des autres constructeurs disponibles :

```
public JsonDerivedTypeAttribute(Type derivedType);
public JsonDerivedTypeAttribute(Type derivedType, int typeDiscriminator);
public JsonDerivedTypeAttribute(Type derivedType, string typeDiscriminator);
```

Il est possible de spécifier un discriminateur sous forme d'un entier ou d'une chaîne de caractère.

Concrètement, il s'agit d'une sorte de métadonnée ajoutée à l'instance sérialisée afin d'aider le sérialiseur à inférer le type qu'il doit traiter.
En spécifiant un discriminant, il apparaît alors sous la forme suivante :

```
var serialized = JsonSerializer.Serialize(customer);
// {"$type":"CustomDiscriminator","CreatedOn":"2022-07-10T16:59:43.038819+02:00",
"Id":1,"Name":"John Doe"}

var deserialized = JsonSerializer.Deserialize<CustomerCreated>(serialized);
// {CustomerCreated { Id = 1, Name = John Doe, CreatedOn = 10/07/2022 11:50:41 }}

[JsonDerivedType(typeof(CustomerCreated), "CustomDiscriminator")]
record Customer(int Id, string Name);

record CustomerCreated(int Id, string Name, DateTime CreatedOn)
  : Customer(Id, Name);
```

Le sérialiseur pourra alors, sur base de cette valeur, désérialiser le bon type même si la valeur sérialisée provient d'un système tiers.
Bien que représentant déjà une avancée facilitant grandement la gestion de la sérialisation/désérialisation des messages, on peut cependant espérer voir apparaître prochainement une version un peu plus claire faisant usage de la généricité nouvellement possible avec les attributs pour pouvoir fortement les typer au lieu d'utiliser typeof.

ASP.NET

Limiter les requêtes entrantes

Fonctionnalité attendue depuis longtemps : la limitation de requêtes HTTP entrantes arrive enfin avec .NET 7.

Les types de limitateurs

Conscients des problématiques propres à chaque équipe et projet, le package NuGet System.Threading.RateLimiting expose divers limitateurs, chacun fonctionnant selon une logique lui étant propre.
Bien que le type de limitateur choisi puisse différer, la mise en place de l'un d'entre eux se fait au travers de la méthode d'extension UseRateLimiter. Cette dernière prend en paramètre une instance de RateLimiterOption qui définira au travers de sa propriété Limiter le limitateur utilisé.
La structure de la mise en place d'un middleware de limitation de requêtes HTTP s'articule ainsi de la manière suivante :

```
app.UseRateLimiter(new RateLimiterOptions
{
  Limiter = PartitionedRateLimiter.Create<HttpContext, string>(context =>
  {
    return /* ... */;
  }),
});
```

La méthode statique PartitionedRateLimiter.Create<,> permet de définir une lambda afin de créer ce limitateur. Via les deux types génériques, il est possible, au sein de cette lambda, d'accéder au HttpContext courant afin de récupérer des informations relatives à la session courante.

Concurrent

Le premier des limitateurs est celui correspondant le plus au comportement auquel on peut s'attendre lorsque l'on ajoute ce mécanisme à son API : les requêtes HTTP entrantes sont placées dans une file et progressivement traitées au fil du temps.
À partir de la méthode de création vue ci-dessus, nous pouvons le définir de la manière suivante :

```
app.UseRateLimiter(new RateLimiterOptions
{
  Limiter = PartitionedRateLimiter.Create<HttpContext, string>(context =>
  {
    return RateLimitPartition.CreateConcurrencyLimiter(
      "ConcurrencyLimiter",
      _ => new ConcurrencyLimiterOptions(1, QueueProcessingOrder.NewestFirst, 0));
  }),
});
```

Déconstruisons sa création au fil de ses paramètres. Le premier est son nom, utilisé pour nommer et définir le limitateur et le second est une méthode lambda de construction du limitateur, définissant son comportement.
Dans cette méthode le premier paramètre est le nombre de requêtes que le système pourra traiter à un instant donné (ici 1). Le second paramètre est l'ordre dans lequel les requêtes placées dans la file seront traitées. Deux valeurs sont possibles, à savoir les plus anciennes ou les plus récentes d'abord (LIFO ou FIFO).
Enfin, le dernier paramètre est le nombre de requêtes pouvant être placées dans cette file. Ces dernières seront alors dans un état d'attente où elles ne seront ni traitées ni rejetées. Dans notre exemple aucune requête ne peut être dans la file ce qui se manifestera en pratique par le rejet de toutes

les requêtes dès lors que le système sera actuellement en train d'en traiter une et une seule.

Illimité

Le second (plus explicite) est simplement celui qui n'a aucune action limitante et laisse passer l'entièreté des requêtes.
Sa création est également la plus directe puisqu'elle ne nécessite pas de logique spécifique du fait de sa simplicité. Le seul paramètre attendu est son nom :

```
app.UseRateLimiter(new RateLimiterOptions
{
  Limiter = PartitionedRateLimiter.Create<HttpContext, string>(context =>
  {
    return RateLimitPartition.CreateNoLimiter("NoLimiter");
  }),
});
```

Tel quel, il peut être difficile de se représenter des cas d'usages où un limitateur qui ne l'est pas peut se révéler pertinent.
En réalité, il est possible de multiplier les limitateurs utilisés par l'application et de les conditionner à une route spécifique. De cette façon, certaines routes peuvent ne pas être soumises à une limitation contrairement à d'autres. Nous verrons par la suite comment définir un tel comportement.

A l'aide de jetons

Le dernier des limitateurs est à la fois le plus complexe à construire, mais également le plus intéressant à mon sens d'un point de vue pratique et se définit comme ceci :

```
app.UseRateLimiter(new RateLimiterOptions
{
  Limiter = PartitionedRateLimiter.Create<HttpContext, string>(context =>
  {
    return RateLimitPartition.CreateTokenBucketLimiter(
      "TokenBucketLimiter",
      _ => new TokenBucketRateLimiterOptions(
        10, QueueProcessingOrder.NewestFirst, 0,
        TimeSpan.FromMinutes(5), 5, true));
  }),
});
```

Contrairement aux limitateurs précédents, le sens des paramètres de celui-ci est moins équivoque et nécessite de comprendre la logique encapsulée par le TokenBucketRate Limiter. Pour se représenter son fonctionnement, il faut se figurer une machine qui prendrait un jeton en entrée pour fonctionner et un utilisateur disposant d'un nombre donné de ces jetons. Chaque jeton permet de faire exécuter une requête HTTP par le système qui périodiquement, au bout d'une période définie, en redonne un certain nombre à l'utilisateur, qui peut alors les réutiliser pour exécuter d'autres requêtes.
Chacun de ces paramètres est alors, dans l'ordre :

- Le nombre de jetons dont dispose l'utilisateur (10 dans notre cas)
- L'ordre dans lequel les requêtes en attente seront traitées
- Le nombre de requêtes pouvant être mises en attente (ici aucune)
- Le temps à attendre avant que le système ne redonne un certain nombre de jetons (5 minutes dans l'exemple)
- Le nombre de jetons redonnés au bout de cette période (5 également)

Si le système redonne automatiquement ou non ce nombre de jetons au bout de la période définie
Il n'est pas possible de récupérer plus de jetons que l'utilisateur n'en dispose initialement. Par exemple, même s'il ne fait qu'une requête en 5 minutes et dispose encore de 9 jetons au bout de la période où le système en redonne, il ne disposera alors que de 10 jetons à nouveau et non de 14.
Un cas d'utilisation particulièrement adapté à ce genre de fonctionnement pourrait être la limitation du nombre d'appels que peut réaliser un client à une API payante. Il serait alors possible de donner aux utilisateurs un nombre de jetons plus ou moins grand selon leur type d'abonnement.
Cependant, un inconvénient majeur qu'il porte également est le fait qu'il ne puisse pas être distribué. Ainsi, si vous avez deux instances d'une même API, alors deux limitateurs avec chacun leur propre compteur interne de jetons seront exposés. Aussi, lors du traitement d'une requête, seul l'un de ces deux limitateurs consommera un jeton de l'utilisateur.

Affiner la limitation

Ces trois types de limitateurs représentent en eux-mêmes un ajout particulièrement intéressant de .NET 7, mais il est également possible d'affiner leurs utilisations en se reposant sur le contexte les accompagnants.

Selon les routes

Comme mentionné plus tôt, un limitateur qui ne jugule pas le flux de requêtes entrantes n'est pas toujours pertinent. Cependant, il n'est pas plus pertinent de brider l'accès à certaines routes spécifiques, telles que la version par exemple.
Grâce au contexte HTTP passé dans la lambda définissant le rate limiter, il est possible de récupérer la requête, donc la route courante, pour agir différemment selon sa valeur et ainsi implémenter ce comportement :

```
app.UseRateLimiter(new RateLimiterOptions
{
  Limiter = PartitionedRateLimiter.Create<HttpContext, string>(context =>
  {
    if (context.Request.Path == "/version")
    {
      return RateLimitPartition.CreateNoLimiter("NoLimiter");
    }

    return RateLimitPartition.CreateConcurrencyLimiter(
      "ConcurrencyLimiter",
      _ => new ConcurrencyLimiterOptions(1, QueueProcessingOrder.NewestFirst, 0));
  }),
});
```

Selon l'authentification

Au-delà des routes, c'est parfois selon certains rôles qu'il est nécessaire d'adapter le middleware.
Par exemple, il est envisageable que les utilisateurs classiques voient leurs requêtes limitées, là où les administrateurs ne devraient pas l'être.
Là encore, grâce au HttpContext, récupérer les autorisations

ou le statut de l'authentification de l'utilisateur émettant la requête est tout à fait réalisable :

```
app.UseRateLimiter(new RateLimiterOptions
{
  Limiter = PartitionedRateLimiter.Create<HttpContext, string>(context =>
  {
    var isAdmin = context.User.Claims.Any(claim => claim.Value == "admin");

    if (isAdmin)
    {
      return RateLimitPartition.CreateNoLimiter("NoLimiter");
    }

    return RateLimitPartition.CreateConcurrencyLimiter(
      "ConcurrencyLimiter",
      _ => new ConcurrencyLimiterOptions(1, QueueProcessingOrder.NewestFirst, 0));
  }),
});
```

Définir les actions engendrées

Plus que le limitateur lui-même, il est parfois désirable de réaliser certaines actions en réponse à une requête rejetée (journalisation, statistiques, etc.).

C'est cette fois-ci au travers des propriétés OnRejected et DefaultRejectionStatusCode de l'objet englobant RateLimiterOptions qu'il sera possible d'influer sur ces cas de figure :

```
app.UseRateLimiter(new RateLimiterOptions
{
  Limiter = PartitionedRateLimiter.Create<HttpContext, string>(context
=> /* ... */),
  OnRejected = (context, lease) =>
  {
    return Task.CompletedTask;
  },
  DefaultRejectionStatusCode = StatusCodes.Status429TooManyRequests,
});
```

La première propriété est une lambda définissant une action à réaliser lors de la rejection de la requête et la seconde, comme son nom l'indique, est le code HTTP retourné lorsque la requête est rejetée. Il est important de noter que par défaut ce dernier est HTTP 503 (service indisponible).

Les minimal API

Depuis .NET 6, les minimal API sont devenues une nouvelle manière légère et plus "fonctionnelle" de développer ses API web. Microsoft en fait une des fonctionnalités recevant la plus grande attention : groupement d'endpoints, enrichissement de la documentation de vos endpoints avec OpenAPI, et bien plus !

Filtrer les requêtes entrantes

Pour filtrer les requêtes entrantes d'une API web, .NET mettait à notre disposition un ensemble de filtres de type qu'il était possible de peupler à la configuration.

Cependant, avec la popularisation des minimal API, l'équipe de développement du framework s'est retrouvée dans la nécessité de porter un mécanisme de filtrage similaire aux endpoints définis dans ces dernières.

Avec .NET 7, les filtres arrivent enfin sur les endpoints définis pas les RouteHandlerBuilders avec plusieurs méthodes d'extensions AddFilter.

Ils peuvent être créés de nombreuses façons, mais retournent systématiquement un RouteHandlerBuilder, permettant de chaîner l'appel d'AddFilter à d'autres méthodes d'extension et ainsi de conserver la définition en cascade des endpoints.

La première des manières de créer un filtre est en lui passant directement un délégué créé à partir du RouteHandler InvocationContext, donnant des précisions sur les paramètres et la nature de l'appel, et d'un RouteHandler FilterDelegate, délégué vers l'appel suivant.

Comme pour les autres méthodes d'extension AddFilter, le délégué retournera un ValueTask<object?>. Il est ainsi possible de court-circuiter la chaîne d'appel et de renvoyer au plus tôt une valeur synchrone au lieu de continuer le flux d'exécution asynchrone en cas de condition bloquante.

Voici une simple implémentation d'un filtre avec un délégué, rejetant les appels pour une liste donnée de noms :

```
var blockList = new[] { "John", "Doe" };

app.MapGet("hello/{name}", (string name) => $"Hello {name} !")
  .AddFilter(async (routeHandlerInvocationContext, next) =>
  {
    var name = routeHandlerInvocationContext.GetArgument<string>(0);

    if (blockList.Contains(name))
    {
      return Results.Problem($"{name} is not allowed");
    }

    return await next(routeHandlerInvocationContext);
  });
```

Si, à la définition du filtre, vous souhaitez le créer selon un contexte particulier à capturer lors de sa création, la seconde méthode d'extension sera alors sans doute plus pertinente, utilisant non pas un délégué, mais une factory.

Cette dernière retourne un délégué retournant une ValueTask<object?> à partir d'un RouteHandlerInvocation Context, à partir du RouteHandlerContext et d'un RouteHandler FilterDelegate, là encore le délégué vers l'appel suivant.

Un aspect intéressant de cette approche est qu'elle permet de capturer et d'agir en fonction des métadonnées de la route actuellement configurée : via le RouteHandlerContext il est possible d'accéder à la propriété MethodInfo qui permettra alors d'accéder aux paramètres, à leurs types, etc.

Voici le même filtre avec cette seconde méthode :

```
app.MapGet("hello/{name}", (string name) => $"Hello {name} !")
  .AddFilter((routeHandlerContext, next) =>
  {
    var blockList = new[] { "John", "Doe" };

    return async (routeHandlerInvocationContext) =>
    {
      var name = routeHandlerInvocationContext.GetArgument<string>(0);

      if (blockList.Contains(name))
```

```
    {
        return Results.Problem($"{name} is not allowed");
    }

    return await next(routeHandlerInvocationContext);
  };
});
```

Enfin, il est possible de passer non pas un bloc de code, mais une classe spécifique, implémentant l'interface IRouteHandlerFilter. Cette dernière ne définit qu'une méthode à implémenter qui est la suivante :

```
ValueTask<object?> InvokeAsync(RouteHandlerInvocationContext context, RouteHandlerFilterDelegate next);
```

Rien d'inconnu ici qui viendrait rompre avec les précédentes définitions : le type de retour reste celui de la chaîne d'appel pouvant être interrompue et les paramètres permettent à nouveau d'accéder au contexte et à l'appel suivant.
Cette troisième manière peut se révéler particulièrement pratique pour limiter l'explosion du nombre de lignes de code dans le fichier définition des routes. En extrayant les définitions des filtres dans leurs classes respectives, le paramétrage de l'endpoint devient alors plus court et concis.
Avoir une classe dédiée permet également de bénéficier de l'injection de dépendance et donc d'enrichir facilement et sans introduire une complexité notable une logique plus poussée dans le filtre.
Enfin, isoler cette classe permet de la rendre plus facilement testable unitairement, ce qui est nettement plus difficile avec les lambdas précédemment définies.

Grouper ses endpoints avec RouteGroups

Usuellement, pour définir les endpoints, de nombreux projets utilisaient une méthode d'extension pour réaliser le mapping à partir de l'instance de WebApplication :

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();
app.MapProductEndpoints();

app.Run();

internal record Product(Guid Id, string Name);

internal static class ProductsEndpoints
{
  private static readonly IEnumerable<Product> _products = new Product[] { /* ... */ };

  internal static WebApplication MapProductEndpoints(this WebApplication app)
  {
    app.MapGet("api/products/", () => Results.Ok(_products)).AllowAnonymous();
    app.MapPost("api/products/", (Product product) => { /* ... */ }).AllowAnonymous();
    app.MapGet("api/products/{id}", (Guid id) => { /* ... */ }).AllowAnonymous();
    app.MapDelete("api/products/{id}", (Guid id) => { /* ... */ }).AllowAnonymous();

    return app;
  }
}
```

On note alors deux points redondants qui apportent une certaine verbosité aux définitions des endpoints: la répétition du préfixe de la route et la spécification du contrôle d'accès.
Dans .NET 7 il est possible, via la nouvelle classe RouteGroupBuilder et sa méthode d'extension, de définir un ensemble d'endpoint pour une route donnée et d'y spécifier un ensemble de caractéristiques qui seront communes à tous les endpoints ainsi définis :

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();
app.MapProductEndpoints();

app.MapGroup("api/products").MapProductEndpoints().AllowAnonymous();

internal record Product(Guid Id, string Name);

internal static class ProductsNewEndpoints
{
  private static readonly IEnumerable<Product> _products = new Product[] { /* ... */ };

  internal static RouteGroupBuilder MapProductEndpoints(this RouteGroupBuilder group)
  {
    group.MapGet("/", () => Results.Ok(_products));
    group.MapPost("/", (Product product) => { /* ... */ });
    group.MapGet("/{id}", (Guid id) => { /* ... */ });
    group.MapDelete("/{id}", (Guid id) => { /* ... */ });

    return group;
  }
}
```

Lors de la réalisation de certains contrôleurs propres à des types de ressources bien définis, il devient ainsi bien plus facile de configurer de la même manière les endpoints associés tout en s'assurant de les enregistrer avec la bonne route.
Initialement, aucun support n'était encore assuré pour prendre en charge la méthode d'extension AddFilter. Le code remanié grâce à cette classe gagnait donc en lisibilité, mais malheureusement, uniquement sur certains aspects bien définis et la généralisation de la logique de filtrage n'est pas encore possible.
Heureusement depuis, l'ancien objet GroupRouteBuilder a été remplacé par un RouteGroupBuilder qui, lui, permet d'ajouter des filtres aux routes qu'il regroupe via la méthode AddRouteHandlerFilter<TFilter>.
Son utilisation est plus intéressante pour l'uniformisation des endpoints qu'il regroupe.

Amélioration de la documentation de l'API avec OpenAPI

OpenAPI est une spécification agnostique de tout langage qui permet de décrire des API.
On peut ainsi apporter certaines précisions à l'API elle-même

(version, description, etc.), mais également aux endpoints qui la compose (type de retour, routes, codes HTTP de retour, etc.) Avec OpenAPI et son intégration facilitée par des librairies tierces telles que Swashbuckle ou encore NSwag, leur documentation est devenue à la fois un incontournable du développement d'API, mais également un standard au point que la mise en place d'un Swagger avec Swashbuckle peut être générée dès la création d'un nouveau projet.
Avec .NET 7, un nouvelle librairie Microsoft.AspNetCore.OpenApi va être disponible afin de documenter ses endpoints. Cette dernière devrait permettre de générer la spécification OpenAPI d'un endpoint d'une minimal API à partir de ses métadonnées et de la route.
L'utilisation la plus simple de la méthode exposée et un simple appel à la méthode d'extension WithOpenApi() :

```
app.MapGet("api/products/{id}", (Guid id) =>
  {
    var product = products.SingleOrDefault(product => product.Id == id);
    return Results.Ok(product);
  })
  .WithOpenApi();
```

Il est cependant possible d'enrichir la description de cette opération en précisant de nombreux aspects de l'endpoint ainsi décoré. On peut alors documenter à la fois l'opération elle-même, mais aussi ses paramètres, s'ils doivent respecter certaines conditions, ou bien encore les types de retour et leur format :

```
app.MapGet("api/products/{id}", (Guid id) =>
  {
    var product = products.SingleOrDefault(product => product.Id == id);
    return Results.Ok(product);
  })
  .WithOpenApi(operation =>
  {
    operation.Summary = "Retrieve a specific product given its ID";

    operation.Parameters[0].Description = "The ID of the product to retrieve";
    operation.Parameters[0].AllowEmptyValue = false;

    return operation;
  });
```

C'est dans cette seconde utilisation que la librairie montre son potentiel intérêt, en permettant d'apporter le même degré d'information que Swashbuckle pourrait le faire via la xlmdoc ou bien des attributs, mais cette fois-ci directement en bénéficiant du C# dans un bloc dédié.
Reste à suivre l'évolution de ce package qui pour le moment pourrait être perfectible en ce qui concerne l'accès à certains paramètres. Par exemple, la récupération d'un paramètre particulier est actuellement faite via un index sur Operation.Parameters et peut alors être sujet à problème si la signature d'un endpoint vient à changer.
Une autre limitation à cette méthode d'extension précédemment mentionnée est qu'elle n'est actuellement pas compatible avec les RouteGroupBuilder et il n'est donc pas possible de l'utiliser au niveau d'un groupement d'endpoint pour le moment, bien que Microsoft ait annoncé qu'il s'agit là d'une fonctionnalité déjà planifiée.

Amélioration des types de retour
Amélioration de la testabilité des types retournés

Avec .NET 6, les endpoints retournant une réponse en plus de données particulières pouvaient jouir de l'interface nouvellement introduite IResponse.
Pour retourner des instances de cette dernière, la classe statique Results exposaient un certain nombre de méthodes symbolisant tant bien la redirection, qu'un échec ou un succès de la requête.
Un endpoint typique avait alors la forme suivante :

```
app.MapGet("api/products", GetProducts);

IResult GetProducts()
{
  var products = new Product[] { /* ... */ };
  return Results.Ok(products);
}
```

Pourtant, l'inconvénient majeur de IResult était que les classes qui l'implémentait n'étaient pas publiques. Impossible donc de manipuler le type de réponse effectivement renvoyé, ce qui est particulièrement gênant si l'on souhaite tester les endpoints et leurs retours, car la réponse, ainsi que les données qu'elle englobe sont inaccessibles programmatiquement.
Avec .NET 7, deux changements majeurs ont été réalisés pour solutionner ce désagrément.
Dans un premier temps, les types implémentant IResult ont été rendus publics. Il est donc maintenant possible d'accéder à une instance de la classe Ok<TResponse> dans ses tests, pour s'assurer à la fois du type de retour, mais également au type des données retournées.
Cependant, seulement avec ce changement, le type de retour de notre endpoint précédemment défini n'est pas Ok<Product[]> comme on pourrait s'y attendre, mais Ok<object>. La raison vient du fait que Results.Ok prend en paramètre un type object?, masquant ainsi le type d'origine de la réponse. Pour pallier à ce problème, Microsoft a publié une nouvelle classe statique, équivalente à l'existante Microsoft.AspNet Core.Http.Results, mais qui, elle, préserve les informations relatives au type d'origine: Microsoft.AspNet Core.Http. TypedResults.
Cette dernière s'utilise sensiblement de la même manière et notre endpoint ainsi modifié n'a pas besoin de changement autre que le changement de Results en TypedResults :

```
app.MapGet("api/products", GetProducts);

IResult GetProducts()
{
  var products = new Product[] { /* ... */ };
  return TypedResults.Ok(products);
}
```

Le type des données TResponse du retour Ok<TResponse> sera alors bien Product[] et non plus object comme précédemment.
L'avantage de cette nouvelle classe, reposant sur les mêmes définitions que .NET 6, est qu'elle permet de transitionner

vers des retours où le type est préservé et plus pertinent sans nécessiter un remaniement immédiat de l'existant.
De la même manière que les nullables pouvaient être chirurgicalement appliqués, avec cette stratégie il est possible de progressivement utiliser la nouvelle classe TypedResults sans risquer une régression des endpoints déjà fonctionnels. La seule différence sera que d'éventuels tests, s'assurant que le retour était bien du type Ok<object> échoueront à présent.

Amélioration de la signature des retours des endpoints

Bien que par la programmation il soit maintenant possible de récupérer le type sous-jacent de la réponse grâce à TypedResults, le type de retour reste bien IResult et ne donne alors que peu d'informations sur la réponse à la seule lecture de la signature de l'endpoint :

```
app.MapGet("api/products/{id}",
  IResult (Guid id) =>
  {
    var products = new Product[] { /* ... */ };
    var product = products.SingleOrDefault(product => product.Id == id);

    return product is null
      ? TypedResults.NotFound()
      : TypedResults.Ok(product);
  });
```

De la même manière, la découvrabilité de l'endpoint, dans le but que sa documentation soit la plus précise possible dans un Swagger par exemple, s'en trouve limitée. Pour répondre à ce problème, une autre classe a été ajoutée : Results <TResult1, TResultN>. Cette dernière permet de retourner un ensemble de réponses de type IResult, possiblement retournés par l'endpoint. En bénéficiant de ce type, il nous est alors possible d'affiner la définition de notre endpoint :

```
app.MapGet("api/products/{id}",
  Results<NotFound, Ok<Product>> (Guid id) =>
  {
    var products = new Product[] { /* ... */ };
    var product = products.SingleOrDefault(product => product.Id == id);

    return product is null
      ? TypedResults.NotFound()
      : TypedResults.Ok(product);
  });
```

La documentation résultant dans la Swagger UI en est ainsi d'autant plus précise.
Bien que déjà intéressant en soit pour la lisibilité et l'enrichissement de la Swagger UI, l'affinage des retours des endpoints ne se limite pas à ces seuls avantages.

Figure 2

```
app.MapGet("api/products/{id}",
    Results<NotFound, Ok<Product>> (Guid id) =>
    {
        var products = new Product[] { /* ... */ };
        var product = products.SingleOrDefault(product => product.Id == id);

        return product is null
            ? TypedResults.Unauthorized()
            : TypedResults.Ok(product);
    });
```

Le fait de bénéficier d'une union typée permet également de remonter des erreurs dès la compilation d'une éventuelle dissonance dans l'implémentation de la logique par rapport à la signature:

C# 11 : Figure 2

De multiples changements sont à noter, et en particulier en ce qui concerne les patterns ainsi que la gestion des strings.

Prise en charge du multi-lignes pour les strings interpolées

Jusqu'à C# 10, l'interpolation ne prenait pas en charge plusieurs lignes de C# et l'entièreté devait donc être écrit en une seule fois sans retours. Aussi, de longues expressions (un calcul avec LINQ par exemple) devenaient rapidement très verbeux et peu lisible à la simple lecture du code.
Avec C# 11, il est maintenant possible de séparer en plusieurs ligne ce genre d'expressions :

```
// Avant C# 11
_ = $"'0' in text is {Enumerable.Range(0, 10).Select(index => index + 1).Sum()}";

// Avec C# 11
_ = $"'0' in text is {Enumerable.Range(0, 10)
  .Select(index => index + 1)
  .Sum()}";
```

La prise en charge est d'autant plus efficace qu'il ne s'agit pas juste de retour à la ligne, mais bien de prendre en charge n'importe quel code C# valide, par exemple une switch expression :

```
// Avant C# 11
var binaryDigitToString = (int number) => number switch
{
  0 => "zero",
  1 => "one",
  _ => "?",
};
_ = $"'0' in text is {binaryDigitToString(0)}";

// Avec C# 11
_ = $"'0' in text is {0 switch
{
  0 => "zero",
  1 => "one",
  _ => "?",
}}";
```

Bien qu'intéressante, cette fonctionnalité laisse cependant dubitatif et pourrait inciter à produire du code moins lisible. Il faudra donc rester vigilant à l'utiliser avec parcimonie.

Introduction des raw strings literals

Les raw string literals sont un nouveau type de strings introduites avec C# 11.
Ces dernières ont la particularité de commencer par trois doubles quotes et de pouvoir contenir n'importe quel texte, y compris les espaces, sauts de ligne, quotes ou encore des caractères spéciaux, sans nécessiter de les échapper :

```
_ = """
  This is a string
```

```
    "literal"
  """;
```

Dans le cas où l'on souhaiterait y incorporer un texte contenant autant de quotes que le délimiteur, il est possible de multiplier ces dernières au début et à la fin pour échapper le texte :

```
_ = """"
  This is a string
    """ literal """
  """";
// ^ Le délimiteur est 5 doubles quotes au lieu de 3
```

De la même manière que les strings, ce type permet d'interpoler des valeurs dans le corps. Il est possible de multiplier le symbole $ qui marque l'interpolation de la même manière qu'il est possible d'étendre le délimiteur afin de pouvoir afficher les accolades en les échappant :

```
var name = "John";
_ = $$"""
  Hi Mary! Here's my JSON: {"name": "{{name}}"}
                  - Regards, {{name}}
  """;
// Hi Mary! Here's my JSON: {"name": "John"}
//                  - Regards, John
```

Il est à noter que les espaces se situant à gauche du délimiteur fermant du raw string literal seront occultés sur toutes les lignes.

Amélioration de l'initialisation des structs

C# 11 apporte également des nouveautés quant à la création de structures. Toutes les structures ont un constructeur par défaut sans paramètre. Ce dernier est soit explicitement implémenté, soit généré par le constructeur.
Un code où le constructeur n'initialiserait pas explicitement chacune des propriétés peut maintenant compiler et tous ces champs seront alors initialisés à leur valeur par défaut si une valeur ne leur a pas été explicitement assignée :

```
Console.WriteLine(new Product(5));
Console.WriteLine(new Product());
Console.WriteLine(default(Product));
Console.WriteLine(string.Join(", ", new Product[3]));
// #5: Not set
// #0:
// #0:
// #0: , #0: , #0:

struct Product
{
  public int Id { get; set; }
  public string Name { get; set; } = "Not set";

  public Product(int id) => Id = id;
  public Product(int id, string name) => (Id, Name) = (id, name);

  public override string ToString() => $"#{Id}: {Name}";
}
```

Toujours au sujet de la construction des structures, si une de ces dernières ne possède que des propriétés accessibles, il est alors possible de la créer sans l'opérateur new :

```
Person p;
p.Id = 1;
p.Name = "John";

struct Person
{
  public int Id;
  public string Name;
}
```

Amélioration du pattern matching

Switch expressions étendues

De la même manière qu'il était possible de faire de la reconnaissance de pattern sur des strings, il est maintenant également possible d'en faire sur les types Span<char> et ReadonlySpan<T> pour tester leur valeur contre une constante donnée.

List patterns

Les patterns appliqués aux listes sont une nouveauté très puissante qui vient s'ajouter à la longue liste des améliorations venant avec cette mise à jour.
Dans la continuité des switch expressions, il sera dorénavant possible de tester un tableau de valeurs grâce à l'opérateur switch. Un nouveau pattern est pour cela disponible : le range pattern (..). Ce dernier identifiera toute séquence de zéro ou plusieurs éléments.
Fait intéressant, il est possible de réaliser un list pattern au sein d'un autre list pattern :

```
var numbers = new[]
{
  new[] { 0, 1, 2 },
  new[] { 3, 4, 5 },
  new[] { 6, 7, 8 },
};

var extracted = numbers switch
{
  [_, [.. int[] beginning, _],_] => $"Beginning 2nd line: {string.Join(',',
beginning)}",
  _ => "Nothing matched",
};

Console.WriteLine(extracted);
// Beginning 2nd line: 3,4
```

La fonctionnalité peut donc se révéler très puissante dans certains cas de figure, mais également rapidement devenir cryptique. Là encore, il faudra peut-être être vigilant quant à son utilisation et éventuellement la combiner à LINQ pour garder un code lisible.

Généricité pour les entités mathématiques

La généricité est un concept largement couvert et utilisé au sein du framework. Pourtant, bien qu'il soit possible d'ajouter des contraintes sur les types génériques afin de s'assurer

qu'ils respectent une certaine forme, il n'était jusqu'alors pas possible de contraindre un type générique à être un nombre. Avant .NET 7, rendre générique une méthode mathématique nécessitait sa réécriture et résultait souvent en l'écriture de ce genre de code :

```
int ComputeSumInt(params int[] numbers)
{
  var result = 0;
  foreach (var number in numbers) result += number;
  return result;
}

double ComputeSumDouble(params double[] numbers)
{
  var result = .0;
  foreach (var number in numbers) result += number;
  return result;
}
```

Dans cette mise à jour un nouveau type INumber<T> a été ajouté, ce dernier étant implémenté par tous les types numériques natifs (int, float, double, etc.).
En plus de définir le comportement des opérateurs, il expose aussi d'autres concepts tels que la notion de 0 ou de 1 par exemple.
Ainsi, le code que l'on pouvait trouver dupliqué précédemment peut être remanié de manière nettement plus claire grâce à l'application de la généricité nouvellement possible.
Au-delà de la perspective de générifier certaines méthodes mathématiques, la possibilité de pouvoir générer ses propres types de nombres est également intéressante.

Propriétés statiques abstraites

Les propriétés statiques abstraites sont une fonctionnalité peu mise en avant pour cette mise à jour bien qu'étonnement efficace dans certains contextes. Il s'agit également de la fonctionnalité qui a rendu possible de rendre générique les types mathématiques.
Le fonctionnement des propriétés statiques abstraites est très simple et fonctionne exactement comme son nom l'indique : dans une classe ou interface, il est possible de spécifier une propriété avec les modificateurs abstract et static. Les types en héritant ou les implémentant devront alors explicitement les définir :

```
Console.WriteLine($"{nameof(Service)} has for public name {Service.PublicName}");
// Service has for public name ServicePublicName

interface IHasPublicName
{
  public abstract static string PublicName { get; }
}

class Service : IHasPublicName
{
  public static string PublicName => "ServicePublicName";
}
```

Cela peut se révéler particulièrement pratique pour exposer de manière statique des attributs particuliers d'une classe.

Attributs génériques

Dans le même contexte d'amélioration de la généricité, les attributs souffraient également d'un manque de prise en charge des types génériques.
Historiquement, afin de récupérer un type depuis un attribut, ce dernier devait le prendre sous la forme d'un attribut de type Type :

```
[AttributeUsage(AttributeTargets.Class)]
class FruitAttribute : Attribute
{
  private readonly Type _fruitType;

  public FruitAttribute(Type fruitType)
    => _fruitType = fruitType;
}

[Fruit(typeof(Apple))]
class Golden { }
```

Ce qui impliquait une certaine quantité de code afin de récupérer un type, et encore plus si l'on souhaitait s'assurer de sa validité par exemple, dans notre cas, vérifier que le type passé est bien un fruit.
Dorénavant, il est possible d'utiliser la généricité avec les attributs afin de fortement les typer, mais également de bénéficier du système de contraintes pour s'assurer qu'il convient à nos besoins :

```
[AttributeUsage(AttributeTargets.Class)]
class FruitAttribute<T> : Attribute where T : IFruit { }

[Fruit<Apple>]
class Granny-smith { }
```

Pour conclure

Les nombreuses améliorations apportées à .NET à l'occasion de cette mise à jour sont très intéressantes et de nombreux développeurs sont déjà excités par certaines d'entre elles.
Parmi celles-ci, l'attention particulière portée aux minimal API souligne une volonté de la part de Microsoft de proposer une alternative aux API plus traditionnelles, qui se veut plus légère, moderne et flexible que celles basées sur des contrôleurs.
Pourtant certaines de ces fonctionnalités, bien que prometteuses, laissent parfois une impression d'inachevée. C'est par exemple le cas pour les nouveaux limitateurs de requêtes, qui ne présentent à l'heure actuelle aucun moyen de fonctionner de manière distribuée.
D'autres améliorations peuvent laisser dubitatif. Il est difficile de se représenter l'utilisation de blocs de code entiers dans une string interpolée ou l'utilisation d'un grand nombre de $ et " pour les délimiter qui puissent rendre une expression plus lisible.
Dans l'ensemble le dynamisme qu'apporte .NET 7 à l'écosystème .NET est le bienvenu et certains changements étaient attendus. En revanche, les mises à jour mineures qui lui seront apportées seront à suivre, dans la mesure où elles apporteront peut-être plus d'éclaircissements et/ou de maturité sur certains aspects.

Principales nouveautés détaillées de JavaScript de ES2015 (ES6) à ES2021 (ES12)

PARTIE 2.2 : LES NOUVEAUTÉS DE JAVASCRIPT 2017

Nous allons explorer au travers de cet article les principales évolutions du langage JavaScript de 2015 à 2021 accompagnées d'exemples illustrant les nouveautés apportées par les différentes versions. JavaScript repose sur les standards d'ECMAScript qui lui se charge de définir un ensemble de normes concernant les langages de programmation de type script et standardisées par Ecma International dans le cadre de la spécification ECMA-262. Dans cette partie, nous allons finir avec les nouveautés JavaScript 2017.

Sylvain Cuenca

Je suis Architecte Système basé en région toulousaine et tout simplement passionné d'informatique depuis mon plus jeune âge, avec bientôt une vingtaine d'années d'expérience dans le monde du logiciel. Les thématiques autour de l'architecture logicielle en général, les performances, la fiabilité, la robustesse, la maintenabilité, l'amélioration continue et l'empreinte mémoire des solutions logicielles sont des sujets que j'aborde pleinement au quotidien. En parallèle, j'effectue beaucoup de veille technologique, m'occupe de la montée en compétence des équipiers et pratique les méthodes japonaises Kaizen sur l'amélioration continue autant sur le plan personnel que professionnel.

Ajout de la fonction Object#values(myObject)

La fonction « Object#values() » renvoie un tableau dont les éléments sont les valeurs des propriétés énumérables directement rattachées à l'objet passé en argument. L'ordre du tableau est le même que celui obtenu lorsqu'on parcourt les propriétés manuellement.

- **Syntaxe :**

  Object.values(myObject)

  - Le paramètre « myObject » représente l'objet dont on souhaite connaître les valeurs de ses propriétés énumérables.

- **Exemples :**

Obtention des valeurs des propriétés énumérables d'un objet

```
const MY_CAR = {
  brand: 'Acura',
  model: 'NSX',
  color: 'yellow',
  year: 1992
};

console.log(Object.values(MY_CAR));
```

Affichage dans la console :

```
[ 'Acura', 'NSX', 'yellow', 1992 ]
```

Obtention des valeurs des propriétés énumérables d'un objet semblable à un tableau

```
const TOYOTA_GROUP = {
  0: 'Lexus',
  1: 'Toyota',
  2: 'Daihatsu',
  3: 'Scion'
}

console.log(Object.values(TOYOTA_GROUP));
```

Affichage dans la console :

```
[ 'Lexus', 'Toyota', 'Daihatsu', 'Scion' ]
```

Obtention des valeurs des propriétés énumérables d'un objet semblable à un tableau, mais dont les index sont ordonnés de manière aléatoire

```
const TOYOTA_GROUP = {
  3: 'Scion',
  1: 'Toyota',
  0: 'Lexus',
  2: 'Daihatsu'
}

console.log(Object.values(TOYOTA_GROUP));
```

Affichage dans la console :

```
[ 'Lexus', 'Toyota', 'Daihatsu', 'Scion' ]
```

Dans cet exemple, on s'aperçoit que la fonction « Object#values » trie par ordre croissant les propriétés selon leur index avant de retourner les valeurs associées aux propriétés de l'objet « TOYOTA_GROUP ».

Obtention des valeurs des propriétés d'un objet contenant à la fois des valeurs énumérables et non énumérables

```
const CAR = {
  brand: 'Acura',
  model: 'NSX',
  color: 'yellow',
  year: 1992
};

Object.defineProperty(CAR, 'wheelNumber', { value: 4, enumerable: false });

console.log('Wheel number:', CAR.wheelNumber);

console.log('The CAR values are:', Object.values(CAR));
```

Affichage dans la console :

```
Wheel number: 4
The CAR values are: [ 'Acura', 'NSX', 'yellow', 1992 ]
```

Ici l'objet « CAR » comporte quatre propriétés énumérables :

« brand », « model », « color » et « year » et une propriété non énumérable « wheelNumber ». Au travers de cet exemple, on s'aperçoit que la valeur « 4 » de la propriété non énumérable « wheelNumber » n'est pas listée lorsqu'on liste les valeurs des propriétés définissant l'objet « CAR ».

Un argument de type primitif, ayant des propriétés, passé à la fonction « Object#values » sera converti en un objet avec des propriétés, avant extraction des valeurs des propriétés énumérables

```
// Un argument de type primitif ayant
// des propriétés sera converti en un
// objet via la fonction "Object#values"
console.log(Object.values('String example'));
```

Affichage dans la console :

```
[
 'S', 't', 'r', 'i',
 'n', 'g', ' ', 'e',
 'x', 'a', 'm', 'p',
 'l', 'e'
]
```

Un argument de type primitif, n'ayant pas de propriété, passé à la fonction « Object#values » sera converti en un objet vide, avant extraction des valeurs des propriétés énumérables

```
// Un argument de type primitif n'ayant
// pas de propriété sera converti en un
// objet vide via la fonction "Object#values"
console.log(Object.values(1000));
```

Affichage dans la console :

```
[]
```

Extraction des valeurs des propriétés d'un objet vide

```
console.log(Object.values({}));
```

Affichage dans la console :

```
[]
```

Extraction des valeurs des propriétés d'un tableau non vide

```
console.log(Object.values(['Toyota', 'Lexus', 'Daihatsu', 'Scion']));
```

Affichage dans la console :

```
[ 'Toyota', 'Lexus', 'Daihatsu', 'Scion' ]
```

Extraction des valeurs des propriétés d'un tableau vide

```
console.log(Object.values([]));
```

Affichage dans la console :

```
[]
```

Avis : *C'est pratique seulement dans les cas où on souhaite extraire uniquement les valeurs des propriétés des objets.*

Ajout de la fonction Object#getOwnPropertyDescriptors (myObject)

Cette fonction permet d'examiner de façon précise les différentes propriétés directement rattachées à un objet.
Elle retourne un objet contenant les propriétés de l'objet en y associant pour chacune d'entre elles sa description sous forme d'objet. Une propriété JavaScript se définit par un nom (une chaîne de caractères) ou un symbole (Symbol) et un descripteur. Un descripteur de propriété est un enregistrement qui possède les attributs suivants :

- **value** : Valeur associée à la propriété (uniquement pour les descripteurs de données).
- **writable** : Indicateur booléen permettant de savoir si la valeur associée à la propriété peut être changée (uniquement pour les descripteurs de données).
- **get** : Fonction utilisée comme accesseur pour la propriété ou « undefined » s'il n'existe pas de tel accesseur (uniquement pour les descripteurs d'accesseur/mutateur).
- **set** : Fonction utilisée comme mutateur pour la propriété ou « undefined » s'il n'existe pas de tel mutateur (uniquement pour les descripteurs d'accesseur/mutateur).
- **configurable** : Indicateur booléen permettant de savoir si le type de descripteur peut être changé et si la propriété peut être supprimée de l'objet correspondant.
- **enumerable** : Indicateur booléen permettant de savoir si cette propriété est listée lorsqu'on énumère les propriétés de l'objet correspondant.
- **Syntaxe :**
  Object#getOwnPropertyDescriptors(myObject)
  - Le paramètre « myObject » définit l'objet dont on souhaite connaître les descripteurs des propriétés

- **Exemples :**

Affichage des descripteurs de propriétés d'un objet

```
const CAR = {
  brand: 'Toyota',
  model: 'RAV4',
  color: 'blue',
  year: 2000,
  displayInformation: () =>
    console.log('Special edition')
};

console.log(Object.getOwnPropertyDescriptors(CAR));
```

Affichage dans la console :

```
{
 brand: {
  value: 'Toyota',
  writable: true,
  enumerable: true,
  configurable: true
 },
 model: {
  value: 'RAV4',
  writable: true,
  enumerable: true,
  configurable: true
 },
 color: {
  value: 'blue',
  writable: true,
  enumerable: true,
```

```
  configurable: true
},
year: { value: 2000, writable: true, enumerable: true, configurable: true },
displayInformation: {
  value: [Function: displayInformation],
  writable: true,
  enumerable: true,
  configurable: true
}
}
```

Avis : *Fonction intéressante lorsqu'on souhaite copier ou cloner un objet.*

Apparitions des « Trailing commas » ou Virgules finales dans la liste des paramètres d'une fonction

Les virgules finales existantes déjà dans la définition d'objets ont été étendues aux paramètres des fonctions.

Le fait de rajouter une virgule à la fin des paramètres d'une fonction n'a aucun effet secondaire et il s'agit simplement d'un changement de style de codage qui n'ajoute pas de paramètre à la fonction. Les virgules finales peuvent également être utilisées lors des définitions de méthodes dans les objets et les classes. L'intérêt de rajouter une virgule à la fin des paramètres d'une fonction est d'éviter d'oublier d'en rajouter une lorsqu'on décide d'ajouter un nouveau paramètre. Un autre intérêt est de pouvoir sauter des lignes après une virgule et ainsi d'aérer le code.

Restrictions : Les définitions de paramètres ou les appels de fonctions qui ne contiennent qu'une seule virgule lèveront une exception « SyntaxError ». Par ailleurs, on ne peut pas utiliser de virgule finale avec les paramètres du reste ("Rest parameters").

• **Exemples :**

Définition de deux fonctions équivalentes

```
const logWithoutComma = (message) => console.log(message);

const logWithComma = (message,) => console.log(message);

logWithoutComma('Without comma');

logWithComma('With comma');
```

Affichage dans la console :

```
Without comma
With comma
```

Définition de méthodes de classe, l'une statique et l'autre non statique avec chacune une virgule finale au niveau de leurs paramètres

```
class MathUtils {
  static staticAdd(operand1 = 0, operand2 = 0,) {
    return operand1 + operand2;
  }

  add(operand1 = 0, operand2 = 0,) {
    return MathUtils.staticAdd(operand1, operand2);
  }
}

const operand1 = 1;
const operand2 = 4;

const mathUtils = new MathUtils();

console.log(`The sum of ${operand1} and ${operand2} is`,
  mathUtils.add(operand1, operand2));
```

Affichage dans la console :

```
The sum of 1 and 4 is 5
```

Dans l'exemple précédent, on constate bien que les virgules finales présentes dans les paramètres des méthodes statiques et non statiques de classe n'ont aucun effet de bord lors de leurs exécutions.

Echecs lors de la définition de paramètres ou lors d'appels de fonctions qui ne contiennent qu'une seule virgule

```
function myFunction(,) { } // SyntaxError: missing formal parameter

(,) => { }; // SyntaxError: expected expression, got ','

myFunction2(,) // SyntaxError: expected expression, got ','

function myFunction3(...p,) { } // SyntaxError: parameter after rest parameter

(...p,) => { } // SyntaxError: expected closing parenthesis, got ','
```

Échec lors de l'utilisation d'une virgule finale avec les paramètres du reste (« Rest parameters »)

```
const [a, ...b,] = [1, 2, 3]; // SyntaxError: rest element
                              // may not have a trailing comma
```

Avis : *Je ne recommande pas cette nouveauté, car cela peut perturber inutilement la compréhension du code. Suivant comment on règle les préférences de son linter de code, on peut être amené à devoir rajouter des virgules.*

Mémoire partagée et atomique

L'objet de type « Atomics » a été créé pour fournir des opérations atomiques sous la forme de méthodes statiques et donc il n'a pas de constructeur. Celles-ci sont utilisées avec les objets de type « SharedArrayBuffer ».

L'objet de type « SharedArrayBuffer » a été créé afin de représenter un buffer de données de longueur fixe qui est partageable et accessible en parallèle par plusieurs threads.

Lorsque la mémoire est partagée, plusieurs threads peuvent lire et écrire sur les mêmes données en mémoire. Les opérations atomiques permettent de s'assurer que des valeurs prévisibles sont écrites et lues, que les opérations sont finies avant que la prochaine débute et que les opérations ne soient pas interrompues.

En effet, pour les navigateurs, « SharedArrayBuffer » et « Atomics » sont destinés à être utilisés avec des « WebWorkers », ce qui est le moyen naturel de faire exécuter du code dans des threads concurrents dans le contexte d'un navigateur.

Avis : *Utile quand on souhaite partager des données entre des threads concurrents.*

Dans la partie 3, nous explorerons les nouveautés de JavaScript 2018 et 2019.

Jean-Christophe Riat

Passionné depuis le lycée par la programmation, j'enseigne l'intelligence artificielle à EPSI Paris, 1ère école d'informatique créée en France par des professionnels [4]. J'aime partager mes expériences sur les résultats récents en « machine learning », avec les ressources accessibles pour tous sur Internet (contact : jcr.formation@gmail.com)

LG AI DD : une IA dans votre lave-linge !

Avec les assistants personnels ou la publicité ciblée sur Internet, les applications de l'intelligence artificielle font désormais partie de notre quotidien. Cette technologie n'est pas exclusive de puissants serveurs puisqu'elle peut également améliorer le fonctionnement de produits électro-ménager, comme sur les derniers lave-linges commercialisés par la marque coréenne LG.

Démocratisées depuis la fin des années 1950, les machines à laver le linge équipent maintenant plus de 95 % des ménages français. Si leurs performances en bilan énergétique et qualité de lavage s'améliorent régulièrement des progrès restent encore nécessaires sur l'expérience utilisateur. La diversité des réglages possibles (multitude de cycles pré-programmés, température de l'eau, vitesse d'essorage, option pour le linge délicat, …) peut laisser perplexe alors que l'unique besoin est de laver le linge placé dans le tambour !

L'idéal serait une façade de commande avec un seul bouton « démarrage » et un appareil qui choisit automatiquement les réglages adaptés en fonction du linge. Tous les fabricants d'électro-ménager travaillent pour trouver des solutions, la principale difficulté étant de réussir à déterminer la nature des vêtements présents dans le tambour. Dans sa communication sur ses nouveaux lave-linges, LG annonce « *optimiser de manière autonome les cycles de lavage en fonction des textiles* » [1]. Nous avons cherché à comprendre quelles étaient les technologies utilisées et leurs conditions de fonctionnement.

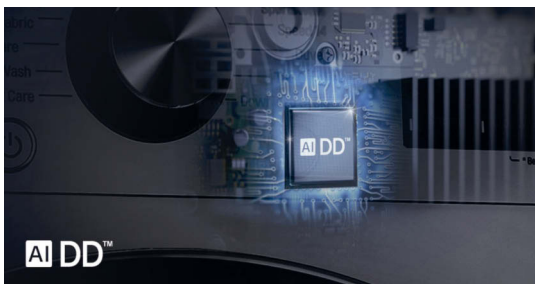
Informations disponibles grâce aux brevets

En général les entreprises qui communiquent sur le contenu technologique de leurs produits mettent en avant les avantages de la technologie, mais restent discrètes sur les détails de sa mise en œuvre. Pour se protéger des contrefaçons les industriels déposent des brevets, ce qui interdit l'usage à titre commercial du dispositif protégé, pour toute autre entreprise que la société détentrice [2]. Il faut savoir qu'un brevet ne reste confidentiel que pendant une durée limitée : 18 mois après son dépôt, il devient accessible gratuitement sur Internet à toute personne qui souhaite le consulter.
Une recherche avec le moteur « patents.google », en utilisant les mots clé sur la société dépositaire et le type d'application, permet de trouver facilement les brevets déposés sur les lave-linges par la société LG [3]. Le plus intéressant porte la référence « EP3617370A1 » qui correspond à un dépôt au niveau européen **(Figure n°2)**. Il existe également un dépôt mondial, mais le brevet est rédigé en coréen !

Découverte des principes utilisés

La lecture d'un brevet n'est pas forcément aisée, car il s'agit d'un document à la fois technique et juridique. Mais avec

Figure n°1: *technologie AI DD™ utilisée sur les lave-linges de la marque LG*



l'utilisation des figures toujours présentes, il est généralement possible pour une personne familière du domaine technique considéré de comprendre le fonctionnement protégé.

La solution développée par LG consiste à optimiser les modes de commande pour la rotation du tambour (vitesse, accélération, …). Le but est de diminuer l'usure du linge, avec une réduction annoncée de 18 %. Les cycles de fonctionnement du tambour ne sont plus uniquement liés au programme de lavage sélectionné, mais sont adaptés automatiquement en fonction du linge présent dans le tambour. La machine est capable de caractériser la quantité et la nature des vêtements, et de choisir à partir de ces deux résultats le fonctionnement le mieux adapté **(Figure n°3)**.

La caractérisation des textiles est réalisée avant le début du lavage : pendant quelques secondes, un profil de vitesses est imposé au moteur du tambour avec une mesure de l'intensité du courant résultant. Etant donné que, dans un moteur électrique, le courant est directement lié au couple du moteur, la valeur du signal mesuré contient des informations sur le linge. En premier lieu sa quantité, car plus le poids est élevé,

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 3 617 370 A1**

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
04.03.2020 Bulletin 2020/10

(51) Int Cl.:
D06F 35/00 *(2006.01)* *D06F 33/02* *(2006.01)*
D06F 39/00 *(2020.01)*

(21) Application number: **19194651.6**

(22) Date of filing: **30.08.2019**

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR
Designated Extension States:
BA ME
Designated Validation States:
KH MA MD TN

(30) Priority: **30.08.2018 KR 20180103081**
07.08.2019 KR 20190096301

(71) Applicant: **LG Electronics Inc.**
Seoul 07336 (KR)

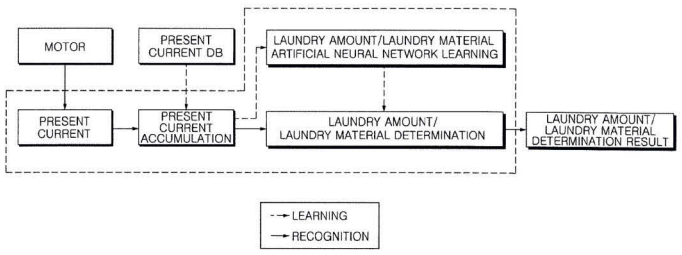
(72) Inventors:
- **IM, Myunghun**
  Seoul 08592 (KR)
- **JUNG, Hwanjin**
  Seoul 08592 (KR)
- **YU, Seonhwa**
  Seoul 08592 (KR)
- **KIM, Sunggyun**
  Seoul 08592 (KR)
- **WOO, Kyungchul**
  Seoul 08592 (KR)

(74) Representative: **Vossius & Partner**
Patentanwälte Rechtsanwälte mbB
Siebertstrasse 3
81675 München (DE)

(54) **ARTIFICIAL INTELLIGENCE WASHING MACHINE AND CONTROLLING METHOD THEREFOR**

(57) The present invention relates to a washing machine and a control method of washing machine. The control method of washing machine includes: a first detection step of acquiring amount of laundry accommodated in a washing tub (4); a first washing step of performing washing based on a first laundry amount, when the first laundry amount acquired in the first detection step is equal to or larger than a preset first threshold value; a second detection step of, after the first washing step, acquiring the laundry amount accommodated in the washing tub by an output of an output layer of an artificial neural network while using a current value input to a motor (9) for rotating the washing tub during an accelerated rotation of the washing tub as an input data of an input layer of the artificial neural network previously learned by machine learning; and a second washing step of performing washing based on a second laundry amount, when the second laundry amount acquired in the second detection step is smaller than the first threshold value.

FIG. 8



Printed by Jouve, 75001 PARIS (FR)

EP 3 617 370 A1

Figure n°2: *brevet déposé en Europe par la société LG*

plus le moteur aura un couple important à fournir, surtout au démarrage. Les ingénieurs de LG ont également observé expérimentalement que le signal comporte des caractéristiques associées à la nature du linge : un textile fragile, généralement souple, provoquera une résistance à la rotation du tambour différente d'un vêtement plus grossier qui sera plus rigide. A partir de la forme du signal, il est donc possible de déduire un indicateur sur un niveau de douceur du linge.

Détails sur l'utilisation de l'IA

Même si on sait qu'un signal contient certaines informations, il n'est pas toujours facile de définir les algorithmes de calculs pour les extraire. L'originalité de la solution développée par LG consiste à utiliser une technique d'intelligence artificielle à base de réseaux de neurones **(Figure n°4)**. Cette approche est reconnue comme très efficace pour les problèmes de classification. Ici le but n'est pas de définir précisément le poids du linge ou le pourcentage exact de répartition entre les différents types de textile, mais de définir des catégories, ce qui revient à faire une classification. 4 catégories sont définies pour le poids (0 à1 kg, 1 à 3 kg, 3 à 5 kg et 5 à 7kg) et 5 groupes sont retenus pour le type de linge selon la proportion de linge délicat (0 %, 25 %, 50 % 75 % et 100 %).
Ces catégories correspondent aux sorties des réseaux de neu-

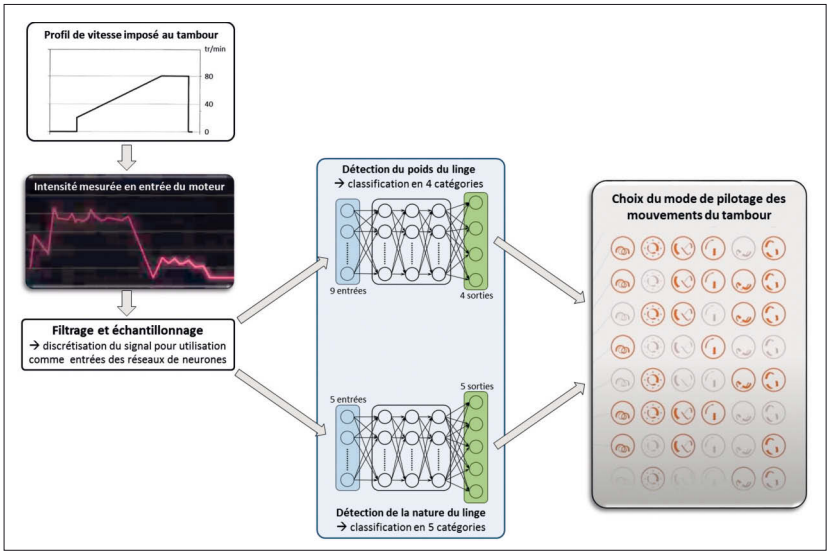


Figure n°3: *synoptique du système embarqué dans le lave-linge*

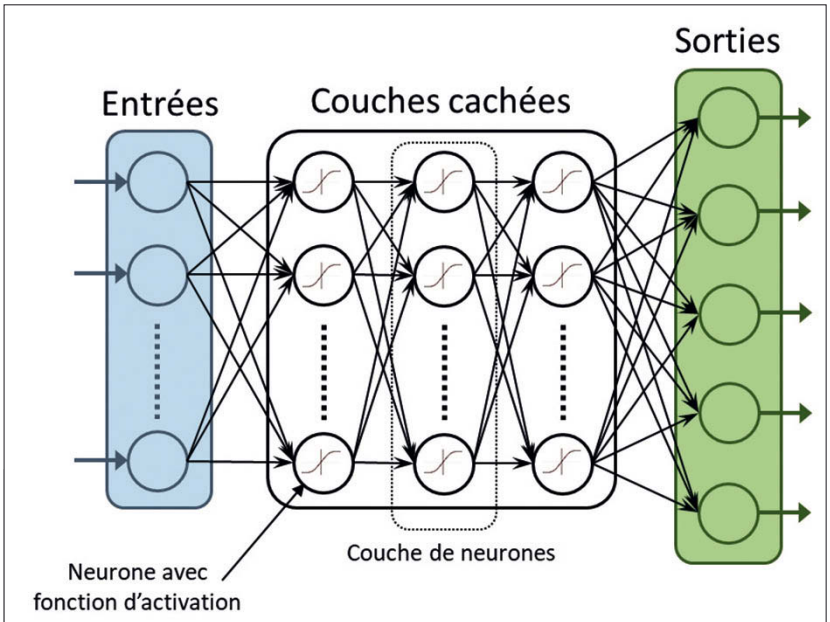


Figure n°4: *réseaux de neurones*

Un réseau de neurones de type perceptron multicouche, permet de modéliser un comportement fortement non linéaire entre un vecteur de valeurs présenté en entrées et des valeurs calculées en sorties. Le fonctionnement du réseau se décompose au niveau de chaque neurone qui calcule individuellement un résultat en fonction des valeurs retournées par la couche précédente. Ainsi les valeurs en entrées sont propagées progressivement entre les couches de neurones, jusqu'à la dernière couche dont les valeurs correspondent aux sorties du réseau.

A chaque connexion entre neurones est associé un poids pour moduler l'importance de la relation entre ces neurones. L'ensemble de ces poids forment les paramètres du réseau, dont les valeurs sont calculées par algorithme, à partir d'exemples de comportements entrées/sorties souhaités. Cette capacité « d'apprentissage » permet, pour un problème donné, de définir une logique de calculs entre des variables uniquement à partir d'exemples, ce qui autorise des applications dans des domaines très variés.

rones, c'est-à-dire aux résultats calculés **(Figure n°3)**. Les informations utilisées en entrée proviennent du signal mesuré sur l'intensité du courant qui alimente le moteur. Après filtrage, toutes les valeurs échantillonnées ne sont pas utilisées. Pour limiter la taille des réseaux, elles sont moyennées par intervalles, ce qui conduit à retenir 9 valeurs pour déterminer le poids et 5 valeurs pour le type de linge **(Figure n°3)**.

Après la définition du réseau de neurones il est nécessaire de procéder à une phase d'apprentissage pour obtenir le fonctionnement souhaité. Cette phase nécessite de disposer d'exemples qui correspondent ici à des formes de signal du courant associées à des contenus de linge connus (poids et proportion de linge délicat). Sur son site internet, LG indique que sa solution AI DD est « *basée sur plus de 20 000 expériences de lavage* ». On peut supposer que ces données ont été utilisées, par les ingénieurs, pour « faire apprendre » le réseau durant les étapes de mise au point.

Après l'apprentissage, le réseau de neurones est entièrement défini. Son fonctionnement, qui nécessite très peu de puissance de calcul, est programmé dans le microcontrôleur qui pilote le lave-linge. Ainsi, à chaque début de cycle, l'IA caractérise les vêtements présents dans le tambour. Les informations ainsi obtenues sont ensuite exploitées pour optimiser le cycle de lavage.

Conclusion

Cet exemple illustre une mise œuvre concrète de la technique des réseaux de neurones dans un appareil du quotidien pour simplifier son utilisation, avec une automatisation de certains réglages. A l'avenir, ce type d'applications, va certainement se développer sur d'autres produits électroménagers, car il apporte un vrai bénéfice client avec une amélioration des performances et une simplification de l'expérience utilisateur.

Références

[1] Vidéos de la société LG, sur le fonctionnement des lave-linges avec la technologie AI DD :
https://www.youtube.com/watch?v=6fobogcrnYM
https://www.youtube.com/watch?v=0rSHCURntv8
[2] Site français sur la propriété intellectuelle (INPI) : https://www.inpi.fr/fr
[3] Recherche de brevets sur le site de Google : https://patents.google.com/
(référence EP3617370A1 pour le brevet associé au lave-Linge de LG)
[4] Ecole d'ingénierie informatique EPSI (membre de HEP Education) : https://www.epsi.fr/

XGBoost : principe et implémentation

XGBoost est probablement l'une des bibliothèques les plus utilisées en science des données. De nombreux data scientists du monde entier l'utilisent. C'est un algorithme très polyvalent qui peut être utilisé pour effectuer une classification aussi bien qu'une régression.

Il se retrouve régulièrement en tête dans les concours de machine learning tels que ceux proposés par Kaggle. XGBoost, LightGBM ou encore CatBoost sont ainsi des librairies python qui implémentent le Gradient Boosting, et qui se retrouvent dans ces palmarès.

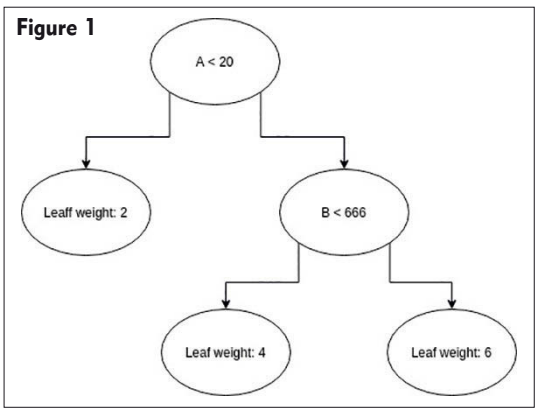
Vous pourriez penser qu'un algorithme d'apprentissage automatique aussi performant que XGBoost utilise des mathématiques très complexes et avancées. Vous imaginez probablement qu'il s'agit d'un chef-d'œuvre d'ingénierie logicielle. Et vous auriez en partie raison. La bibliothèque XGBoost est assez complexe, mais si vous ne considérez que la formulation mathématique du boosting de gradient appliquée aux arbres de décision, ce n'est pas si compliqué.

Cet article vous propose d'en comprendre réellement les principes sous-jacents. Vous verrez ci-dessous en détail comment construire des arbres de décision pour la régression en utilisant la méthode de gradient boosting avec moins de 200 lignes de code.

Arbre de décision

Avant d'entrer dans les détails mathématiques, rafraîchissons notre mémoire concernant les arbres de décision. Le principe est assez simple : associer une valeur à un ensemble donné de caractéristiques en parcourant un arbre binaire. Chaque nœud de l'arbre binaire est attaché à une condition ; les feuilles contiennent des valeurs. Si la condition est vraie, on continue la traversée en utilisant le nœud de gauche, sinon, on utilise le nœud de droite. Une fois qu'une feuille est atteinte, nous avons notre prédiction. Comme souvent, une image vaut mille mots : **Figure 1.** La figure 1 présente un arbre de décision à trois niveaux. La condition attachée à un nœud peut être considérée comme une décision, d'où le nom d'**arbre de décision**.

Figure 1

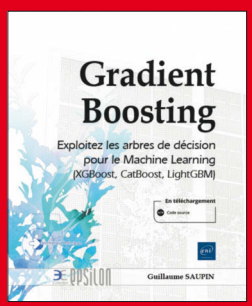


Ce type de structure est assez ancien dans l'histoire de l'informatique et est utilisé depuis des décennies avec succès. Une implémentation de base est donnée par les lignes suivantes :

```
class DecisionNode:
    """
    Node decision class.
    This is a simple binary node, with potentially two children: left and right
    Left node is returned when condition is true
    False node is returned when condition is false
    """
    def __init__(self, name, condition, value=None):
        self.name = name
        self.condition = condition
        self.value = value
        self.left = None
        self.right = None

    def add_left_node(self, left):
        self.left = left

    def add_right_node(self, right):
        self.right = right

    def is_leaf(self):
        """
        Node is a leaf if it has no child
        """
        return (not self.left) and (not self.right)

    def next(self, data):
        """
        Return next node depending on data and node condition
        """
        cond = self.condition(data)
        if cond:
            return self.left
        else:
            return self.right


class DecisionTree:
    """
    A DecisionTree is a model that provides predictions depending on input.
    A Prediction is the sum of the leaves' values, for those leaves that were activated
    by the input
    """
    def __init__(self, root):
        """
        A DecisionTree is defined by an objective, a number of estimators and a
        max depth.
        """
        self.root = root

    def predict(self, data):
        child = root
```


Guillaume Saupin

Ingénieur généraliste et docteur en mathématiques appliquées, passionné de mathématiques et du langage Lisp, Guillaume a travaillé une dizaine d'années comme chercheur au CEA . Il rejoint le monde de l'Intelligence artificielle et des start ups en 2010. Il a enseigné le Computer Graphics et le HPC en master à l'Université Paris 12 et à Epitech. Actuellement CTO de Verteego, il est également auteur de plus d'une vingtaine d'articles et publie régulièrement sur la Data Science dans Toward Data Science. Il a publié un livre sur les méthodes de Gradient Boosting pour le Machine Learning aux éditions ENI.



```
    while child and not child.is_leaf():
      child = child.next(data)
    return child.value


root = DecisionNode('root', lambda d : d['A'] > 2.0)
root_left = DecisionNode('root_left', lambda d : d['B'] > 10.0, None)
root_right = DecisionNode('root_right', None, 1)
left_left = DecisionNode('left_left', None, 2)
left_right = DecisionNode('left_right', None, 3)

root.add_left_node(root_left)
root.add_right_node(root_right)

root_left.add_left_node(left_left)
root_left.add_right_node(left_right)

tree = DecisionTree(root)
print(tree.predict({'A' : 1, 'B' : 1})) # 1
print(tree.predict({'A' : 1, 'B' : 10})) # 1
print(tree.predict({'A' : 3, 'B' : 11})) # 2
print(tree.predict({'A' : 3, 'B' : 9})) # 3
```

Combinaison de plusieurs arbres

Même si les arbres de décision ont été utilisés avec un certain succès dans un certain nombre d'applications, comme les systèmes experts (avant l'hiver de l'IA), ils restent un modèle très basique qui ne peut pas gérer la complexité habituellement rencontrée dans les données du monde réel. Nous appelons généralement ce type d'estimateurs des modèles faibles.

Pour surmonter cette limitation, l'idée est apparue dans les années 90 de combiner plusieurs modèles faibles pour créer un modèle fort : l'approche ensembliste.

Cette méthode peut être appliquée à n'importe quel type de modèle, mais comme les arbres décisionnels sont des modèles simples, rapides, génériques et facilement interprétables, ils sont couramment utilisés.

Diverses stratégies peuvent être déployées pour combiner les modèles. Nous pouvons par exemple utiliser une somme pondérée de la prédiction de chaque modèle. Ou encore mieux, utiliser une approche bayésienne pour les combiner sur la base des données d'apprentissage.

XGBoost et toutes les méthodes de boosting utilisent une autre approche : chaque nouveau modèle tente de compenser les erreurs du modèle précédent. Il s'agit donc bien de boosting, les nouveaux arbres venant booster les performances des précédents.

Optimisation des arbres de décision

Comme nous l'avons vu ci-dessus, faire des prédictions en utilisant un arbre de décision est simple. La tâche n'est guère plus compliquée lorsqu'on utilise l'approche ensembliste : il suffit de faire la somme des contributions de chaque modèle. Ce qui est vraiment compliqué, c'est de construire l'arbre lui-même ! Comment trouver la meilleure condition à appliquer à chaque nœud de notre ensemble de données de formation ? C'est là que les mathématiques nous aident. Une dérivation complète peut être trouvée dans la documentation de XGBoost. Ici, nous nous concentrerons uniquement sur les formules pertinentes pour cet article.

Comme toujours en apprentissage automatique, nous voulons définir les paramètres de notre modèle de sorte que les prédictions de celui-ci sur l'ensemble d'apprentissage minimisent un objectif donné :

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

Veuillez noter que cet objectif est composé de deux termes : L'un pour mesurer l'erreur faite par la prédiction. C'est la fameuse fonction de perte l(y, y_hat).

L'autre, oméga, pour contrôler la complexité du modèle.

Comme indiqué dans la documentation de XGBoost, la complexité est une partie très importante de l'objectif qui nous permet d'ajuster le compromis biais/variance. De nombreuses fonctions différentes peuvent être utilisées pour définir ce terme de régularisation. XGBoost utilise par exemple :

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Ici, T est le nombre total de feuilles, tandis que w_j sont les poids attachés à chaque feuille. Cela signifie qu'un grand poids et un grand nombre de feuilles seront pénalisés.

Comme l'erreur est souvent une fonction complexe et non linéaire, la méthode du gradient boosting la linéarise, en utilisant une expansion de Taylor du second ordre :

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}$$

où :

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

Le premier terme g_i est le gradient de l'erreur, tandis que h_i est sa hessienne. La linéarisation est calculée par rapport au terme de prédiction, car nous voulons estimer comment l'erreur change lorsque la prédiction change. Cette linéarisation est essentielle, car elle facilite la minimisation de l'erreur. Ce que nous voulons réaliser avec l'optimisation du gradient, c'est trouver le y_hat_i optimal qui minimisera la fonction de perte, c'est-à-dire que nous voulons trouver comment modifier l'arbre existant pour que la modification améliore la prédiction.

Lorsque l'on traite des modèles d'arbres, deux types de paramètres sont à prendre en compte :

- Ceux qui définissent l'arbre en lui-même : la condition pour chaque nœud, la profondeur de l'arbre.
- Les valeurs attachées à chaque feuille de l'arbre. Ces valeurs sont les prédictions elles-mêmes.

L'exploration de toutes les configurations de l'arbre serait trop complexe, c'est pourquoi les méthodes de Gradient Boosting ne prennent en compte que la division d'un nœud en deux feuilles. Cela signifie que nous devons optimiser trois paramètres :

1. La valeur de séparation : à quelle condition nous séparons les données.
2. La valeur attachée à la feuille de gauche
3. La valeur attachée à la feuille de droite

Dans la documentation de XGBoost, nous apprenons que la meilleure valeur pour une feuille j par rapport à l'objectif est donnée par :

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

où G_j est la somme du gradient des points d'apprentissage attachés au nœud j, et H_j est la somme du hessian des points d'apprentissage attachés au nœud j.

La minimisation de la fonction objectif obtenue avec ce paramètre optimal est :

$$\text{obj}^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

Le choix de la bonne valeur de séparation se fait par brute force : on calcule l'amélioration pour chaque valeur de fractionnement et on garde la meilleure.

Nous avons maintenant toutes les informations mathématiques nécessaires pour améliorer les performances d'un arbre initial par rapport à un objectif donné en ajoutant de nouvelles feuilles.

Avant de le faire concrètement, prenons le temps de comprendre la signification de ces formules.

Comprendre le boosting par gradient

Essayons de comprendre comment le poids est calculé, et à quoi correspondent G_j et H_i. Comme il s'agit respectivement du gradient et du hessian de la fonction de perte par rapport à la prédiction, nous devons choisir une fonction de perte. Nous allons nous concentrer sur l'erreur quadratique, qui est couramment utilisée et constitue l'objectif par défaut de XGBoost :

$$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

C'est une formule assez simple, dont le gradient est :

$$-2(y_i - \hat{y}_i)$$

et la hessienne simplement :

$$2$$

Par conséquent, si nous nous rappelons les formules pour le poids optimal qui maximise la réduction de l'erreur :

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

Nous nous rendons compte que le poids optimal, c'est-à-dire la valeur que nous ajoutons à la prédiction précédente, est l'opposé de l'erreur moyenne entre la prédiction précédente et la valeur réelle (lorsque la régularisation est désactivée, c'est-à-dire lambda = 0). L'utilisation de l'erreur au carré pour entraîner les arbres décisionnels avec la méthode du Gradient Bosting se résume à mettre à jour la prédiction avec l'erreur moyenne dans chaque nouveau nœud.

Nous constatons également que lambda a l'effet escompté, c'est-à-dire qu'il garantit que les poids ne sont pas trop importants, car le poids est inversement proportionnel à lambda.

Construction des arbres de décision

Voici maintenant la partie facile. Supposons que nous disposons d'un arbre de décision existant qui assure des prédictions avec une erreur donnée. Nous voulons réduire l'erreur et améliorer l'objectif joint en scindant un nœud.

L'algorithme pour y parvenir est assez simple :

- choisissez une caractéristique d'intérêt
- ordonnez les points de données attachés au nœud actuel en utilisant les valeurs de la caractéristique sélectionnée
- choisissez une valeur de séparation possible
- placez les points de données situés en dessous de cette valeur de division dans le nœud de droite, et ceux situés au-dessus dans le nœud de gauche.
- calculez la minimisation de l'objectif pour le nœud parent, le nœud de droite et le nœud de gauche.
- si la somme des réductions d'objectif pour les nœuds de gauche et de droite est supérieure à celle du nœud parent, conservez la valeur de division comme la meilleure.
- itérez pour chaque valeur de fractionnement
- itilisez la meilleure valeur de division, le cas échéant, et ajoutez les deux nouveaux nœuds.
- si aucun partitionnement n'améliore l'objectif, n'ajoutez pas de nœuds enfants.

Le code résultant crée une classe DecisionTree, qui est configurée par un objectif, un nombre d'estimateurs, c'est-à-dire le nombre d'arbres, et une profondeur maximale.

Comme promis, le code prend moins de 200 lignes :

Code complet sur programmez.com & github

Le cœur de la construction est codé dans la fonction _find_best_split. Elle suit essentiellement les étapes détaillées ci-dessus. Notez que pour supporter tout type d'objectif, sans avoir la peine de calculer manuellement le gradient et le hessian, nous utilisons la différenciation automatique et la bibliothèque jax pour automatiser les calculs. Initialement, nous commençons par un arbre avec un seul nœud dont la valeur de la feuille est nulle. Comme nous avons imité XGBoost, nous utilisons également un score de base, que nous définissons comme étant la moyenne de la valeur à prédire. Notez également qu'à la ligne 126, nous arrêtons la construction de l'arbre si nous atteignons la profondeur maximale définie lors de l'initialisation de l'arbre. Nous aurions pu utiliser d'autres conditions comme un nombre minimum d'échantillons pour chaque feuille ou une valeur minimum pour le nouveau poids. Un autre point très important est le choix de la caractéristique utilisée pour le découpage. Ici, pour des raisons de simplicité, les caractéristiques sont choisies aléatoirement, mais nous aurions pu utiliser des stratégies plus intelligentes, comme utiliser la caractéristique ayant la plus grande variance.

Conclusion

Dans cet article, nous avons vu comment le boosting de gradient fonctionne pour former des arbres de décision. Pour améliorer encore notre compréhension, nous avons écrit le nombre minimal de lignes nécessaires pour entraîner un ensemble d'arbres de décision et les utiliser pour la prédiction. Il est absolument essentiel de bien comprendre les algorithmes que nous utilisons pour l'apprentissage automatique. Cela nous aide non seulement à construire de meilleurs modèles, mais surtout à adapter ces modèles à nos besoins. Par exemple, dans le cas du boosting de gradient, jouer avec les fonctions de perte est un excellent moyen d'augmenter la précision des modèles.

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

PROGRAMMEZ!

Le magazine des développeurs

NOS CLASSIQUES

1 an → 10 numéros (6 numéros + 4 hors séries) — 55€ *(1)

2 ans → 20 numéros (12 numéros + 8 hors séries) — 90€ *(1)

Etudiant
1 an → 10 numéros (6 numéros + 4 hors séries) — 45€ *

Option : accès aux archives — 20€

** Tarifs France métropolitaine*

(1) Au lieu de 69,90 € ou 139,80 € selon l'abonnement, par rapport au prix facial.

ABONNEMENT NUMÉRIQUE

PDF ..... 45€
1 an

Souscription directement sur
www.programmez.com

Offres spéciales 2022

1 an Programmez!

+ 1 mois d'accès à la bibliothèque numérique ENI — **56€**

1 mois d'accès offert à la bibliothèque numérique ENI, la plus grande bibliothèque informatique française. Valeur : 49 €

2 ans Programmez! — 109€*

+ 1 mois d'accès à la bibliothèque numérique ENI
+ 1 an de Technosaures (2 numéros)

1 mois d'accès offert à la bibliothèque numérique ENI, la plus grande bibliothèque informatique française. Valeur : 49 €. Prix abonnement Technosaure : 29,90 €

* au lieu de 119,9 €

Sous réserve d'erreurs typographiques

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT à retourner avec votre règlement à :
PROGRAMMEZ, Service Abonnements
57 Rue de Gisors, 95300 Pontoise

- ☐ Abonnement 1 an : 55 €
- ☐ Abonnement 2 ans : 90 €
- ☐ Abonnement 1 an Etudiant : 45 €
  Photocopie de la carte d'étudiant à joindre
- ☐ Option : accès aux archives 20 €

- ☐ 1 an Programmez! : 56 €
  + 1 mois d'accès à la bibliothèque numérique ENI
- ☐ 2 ans Programmez! : 109 €
  + 1 mois d'accès à la bibliothèque numérique ENI
  + 1 an de Technosaures (2 numéros)

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

Adresse email indispensable pour la gestion de votre abonnement

E-mail : @

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

☐ Je souhaite régler à réception de facture

*** Tarifs France métropolitaine**

Offres pouvant s'arrêter à tout moment, sans préavis **PROG 254**

Les anciens numéros de PROGRAMMEZ!

Le magazine des dévs

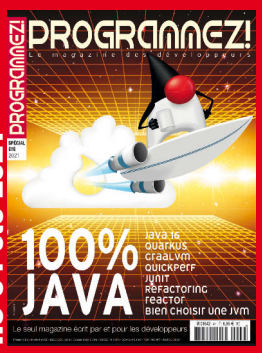
241

242

243

HS 02

246



HS 04 été 2021

249

250

251

252

253

HS 07

Tarif unitaire 6,5 € (frais postaux inclus)

☐ *241*	: __ ex	☐ *246*	: __ ex	☐ *251*	: __ ex
☐ *242*	: __ ex	☐ *HS4 été 2021*	: __ ex	☐ *252*	: __ ex
☐ *243*	: __ ex	☐ *249*	: __ ex	☐ *253*	: __ ex
☐ *HS2 automne 2020*	: __ ex	☐ *250*	: __ ex	☐ *HS7*	: __ ex

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

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

☐ M. ☐ Mme 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 254**

Pierre-Gilles Leymarie

Pierre-Gilles est un ingénieur logiciel de 28 ans. Il est le fondateur du projet de domotique open source Gladys Assistant qu'il a créé en 2013.

Une caméra de surveillance DIY avec un Raspberry Pi Zero W, le camera module & 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 / Unraid, une Freebox Delta, ou n'importe quelle machine faisant tourner Linux. **Figure 1**

Ce logiciel français existe depuis 2013. En novembre 2020, le projet a sorti une nouvelle version majeure du logiciel : Gladys Assistant 4, une version réécrite de zéro. L'accent a été mis sur la simplicité d'utilisation, la stabilité et la performance du logiciel.

Depuis la sortie de la v4, le projet n'a cessé de proposer des nouvelles fonctionnalités et intégrations : intégrations avec les appareils Zigbee, compatibilité avec les assistants vocaux Google Home et Alexa, affichage super simple de courbes de capteurs sur le tableau de bord, détection de la présence à la maison grâce au GPS du téléphone ou à des porte-clés Bluetooth, et plein d'autres…

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

Figure 2

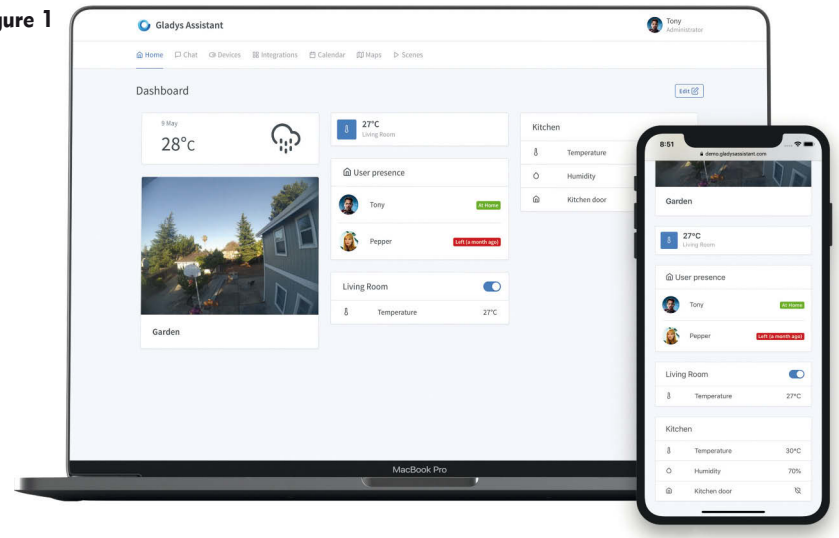
Le matériel nécessair

Nous allons fabriquer une caméra de surveillance DIY, en utilisant un Raspberry Pi Zero W et le caméra module proposé par la fondation.

Le Raspberry Pi Zero W est la version « légère » du Pi. Il fait la taille d'une carte de crédit, et est moins puissant que l'original (un seul cœur à 1Ghz et 512Mb de RAM), mais est surtout sans-fil (Wi-Fi et Bluetooth), ce qui le rend parfait pour des petits projets comme celui-ci.

En domotique, le Pi Zero W est souvent utilisé comme scanneur Bluetooth déporté pour ceux qui veulent utiliser des porte-clés Bluetooth comme capteur de présence, et veulent pouvoir être détectés à plusieurs points de leur maison.

Figure 1



Assemblage du module caméra

L'assemblage du module caméra est très simple, il suffit de brancher la nappe de la caméra au Raspberry Pi Zéro W.

Je vous conseille de mettre le Raspberry Pi Zéro W et le module caméra dans un boitier compatible pour que le montage tienne sur la durée. **Figure 2**

Installation de Gladys Assistant

Je vais décrire 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. Je pense à un NAS Synology, un NAS Unraid, un serveur, un petit VPS en ligne : tout est possible.

Vous trouverez les instructions pour les autres machines sur le site du projet [Lien: https://gladysassistant.com/fr/docs/installation/docker/]

Aussi, ici je ne parle pas d'installer Gladys sur le Raspberry Pi Zéro, mais sur un autre appareil. Le Raspberry Pi Zéro ne sera que la caméra de ce montage, il vous faut un système domotique fonctionnel à côté.

Je recommande le Raspberry Pi, car c'est une formidable machine, silencieuse (sans ventilateur), et qui consomme peu. 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_assistant_xxx.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 officiel de la fondation Raspberry Pi, Raspberry Pi Imager [Lien: https://www.raspberrypi.com/software/] qui vous permettra de cloner très simplement l'image.
5. Débranchez la carte micro-SD de votre ordinateur, et insérez là 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 :

1. 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"
2. Si ce n'est pas le cas, il faut trouver l'adresse IP de votre Raspberry Pi sur le réseau. Je vous conseille d'utiliser une application comme Network Scanner [Lien: https://play.google.com/store/apps/details?id=com.easy-mobile.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 Raspberry Pi Zéro W

Nous allons maintenant installer le Raspberry Pi Zéro W pour qu'il expose un flux vidéo RTSP diffusant l'image de la caméra. Ce flux sera ensuite consommé par Gladys Assistant qui pourra l'afficher dans l'interface de Gladys.

La configuration :

1. Sortez la micro-SD du Raspberry Pi Zéro W et insérez-la dans votre ordinateur
2. Vérifiez que vous avez bien l'outil Raspberry Pi Imager sur votre ordinateur, nous allons l'utiliser pour cloner une image sur la carte micro-SD.
3. Dans l'outil, sélectionnez votre carte micro-SD, puis sélectionnez l'image officielle de la fondation Raspberry Pi. **Figure 3**
4. Cliquez sur la roue dentée pour configurer le nom de votre Pi sur le réseau (vous pouvez mettre « gladys-camera », ou « camera », comme vous voulez), et pour donner les informations de connexion à votre réseau Wi-Fi : SSID et mot de passe. **Figure 4**
5. Cliquez sur « Write » pour écrire la micro-SD.
6. Une fois la micro-SD écrite, vous pouvez l'enlever de votre ordinateur, et la mettre dans votre Pi Zero W.
7. Branchez votre Pi Zero W au courant, et attendez qu'il démarre. Je vous conseille d'utiliser la même application recommandée au chapitre précédent afin de scanner votre réseau et trouver l'IP de votre Pi Zero W sur le Wi-Fi. Cela peut prendre quelques minutes avant que votre Pi soit prêt, soyez patient !
8. Dès que vous avez l'IP, connectez-vous en SSH à votre Raspberry Pi depuis votre ordinateur, soit en ligne de commande si vous êtes sous MacOS ou Linux, en faisant : ssh pi@192.168.1.155 (remplacez par l'IP de votre Pi Zero W), ou via Putty si vous êtes sous Windows. Un mot de passe vous sera demandé, c'est « raspberry » si vous n'avez rien touché, ou le mot de passe que vous aurez défini à l'étape 4 si vous avez modifié le mot de passe par défaut.
9. Nous allons maintenant activer la caméra de votre Pi Zero W, en tapant « sudo raspi-config », puis en allant sur « Interfaces options », « Legacy Camera », puis « Entrée ». L'utilitaire va nous demander de redémarrer le Pi. **Figure 5**
10. Il faut maintenant se reconnecter en SSH comme à l'étape 8, et vous pouvez tester si la caméra fonctionne en prenant une photo avec la commande :

```
raspistill -v -o test.jpg
```

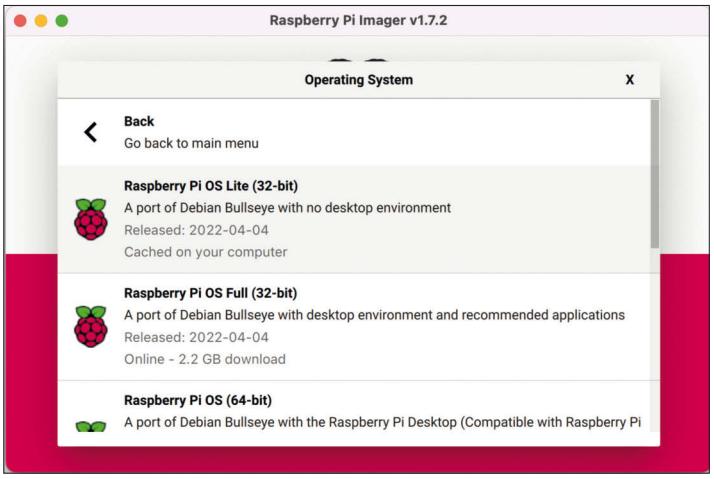


Figure 3

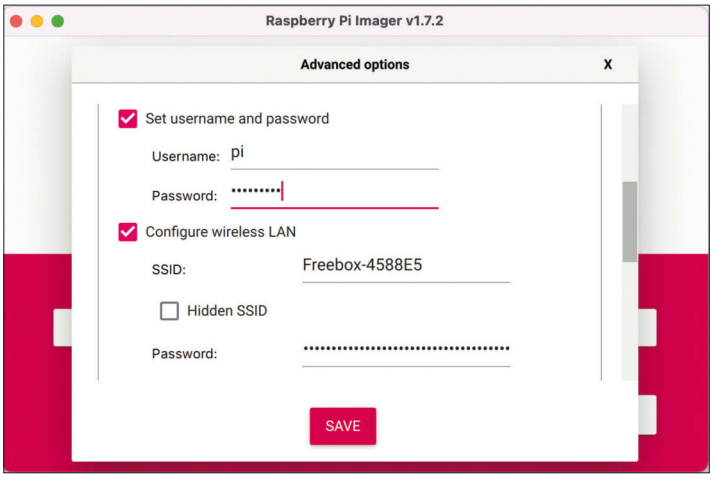


Figure 4

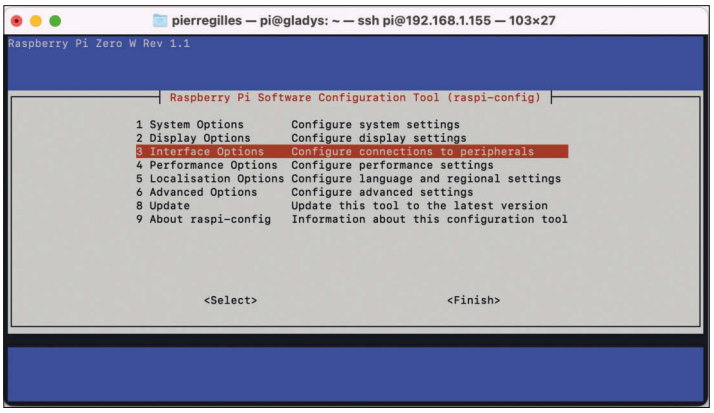


Figure 5

Si vous voulez visualiser cette photo dans votre terminal, il existe un outil, très mal nommé pour les francophones, mais très pratique, « cacaview ».
Pour l'installer :

```
sudo apt-get update
sudo apt-get install caca-utils -y
cacaview test.jpg
```

Vous devriez voir l'image prise, façon ASCII-art !
Figure 6

Figure 6

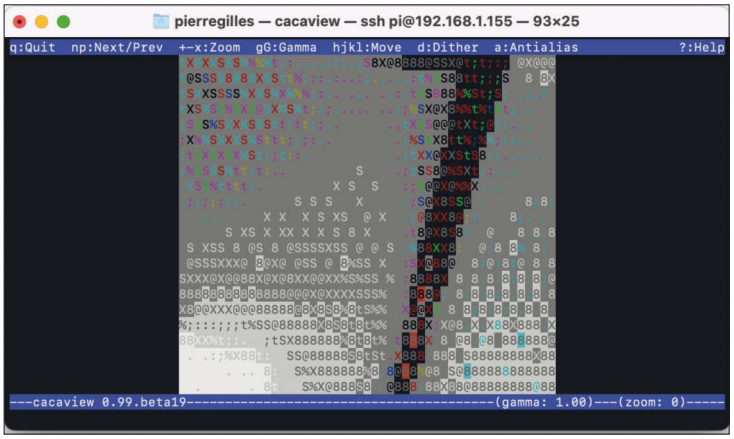


Figure 7

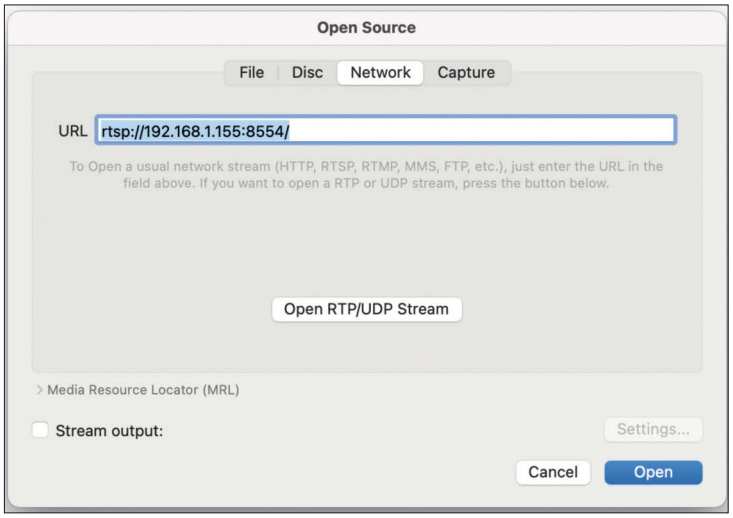


Figure 8

Maintenant que nous sommes capables de prendre une photo avec la caméra, il faut que nous puissions exposer un flux RTSP sur le réseau afin que Gladys puisse récupérer ce flux sur une autre machine :
Pour cela, vous devez installer le package « vlc » :

```
sudo apt-get install vlc
```

Puis vous pouvez lancer la diffusion du flux de cette manière :

```
sudo raspivid -o - -t 0 -n -w 600 -h 400 | cvlc -vvv stream:///dev/stdin —sout '#rtp{sdp=rtsp://:8554/}' :demux=h264
```

Ensuite, sur un ordinateur sur le même réseau, vous pouvez récupérer le flux de cette façon suivante, par exemple depuis VLC, en faisant « Fichier » => « Ouvrir réseau », puis en rentrant comme URL de flux :

```
rtsp://IP_DE_VOTRE_RASPBERRY_PI_ZERO_W:8554/
```

Figure 7

Vous devriez voir l'image de votre caméra en direct !
Figure 8

Vous pouvez couper le flux en faisant Ctrl + X dans le terminal. Maintenant que nous sommes capables de récupérer le flux, il faut automatiser tout ça et faire que le stream se lance au démarrage du Raspberry Pi Zero W, car actuellement, si nous redémarrons le Pi, le flux s'arrête.
Pour cela, nous allons créer un service systemd.
Tout d'abord, créez un fichier camera-stream.sh en faisant :

```
sudo nano /usr/bin/camera-stream.sh
```

Copiez-collez dans ce fichier le code suivant :

```
[FICHIER camera-stream.sh]
```

Ensuite, donnez les droits d'exécution sur ce fichier :

```
sudo chmod +x /usr/bin/camera-stream.sh
```

Ensuite, déclarez un service systemd :

```
sudo nano /etc/systemd/system/camera.service
```

Remplissez ce fichier avec [FICHIER: camera.service]
Ensuite, nous allons recharger le système pour qu'il ait connaissance de ce service.

```
sudo systemctl daemon-reload
```

Puis pour tester le service, vous pouvez faire :

```
sudo systemctl start camera
```

Vérifiez avec VLC que le stream fonctionne toujours.
Si c'est bien le cas, vous pouvez faire :
sudo systemctl enable camera
Pour que ce service soit lancé au démarrage du Pi.

Ajoutez ce flux RTSP dans Gladys Assistant

Maintenant que notre caméra IP DIY fonctionne, nous pouvons l'ajouter à Gladys afin de pouvoir l'afficher sur notre tableau de bord.

Allez dans « Intégrations » => « Caméras », puis cliquez sur « Nouveau + ».
Renseignez le nom de la caméra, la pièce dans laquelle la caméra se trouve chez vous, puis entrez l'URL du flux RTSP qui je le rappelle est toujours :

```
rtsp://IP_DE_VOTRE_RASPBERRY_PI_ZERO_W:8554/
```

Cliquez sur « Enregistrer », puis sur « Tester » si vous voulez vérifier que la caméra fonctionne bien. **Figure 9**
Ensuite, allez sur votre tableau de bord, et cliquez sur « Editer » pour ajouter une box caméra. **Figure 10**
Choisissez la caméra que vous venez de créer et enregistrez. Vous devriez voir votre image de caméra sur le tableau de bord ! **Figure 11**
Maintenant que votre caméra est configurée, vous pouvez l'utiliser partout dans Gladys.
Par exemple, il devient possible de demander à Gladys par message :
« Montre-moi la caméra du salon », et elle vous répondra avec une image de la caméra du salon : **Figure 12**

Configurer Telegram pour discuter avec Gladys à distance

Maintenant que nous sommes capables de voir une image de caméra sur notre dashboard Gladys, il devient intéressant d'utiliser Telegram pour pouvoir discuter avec Gladys à distance.
Vous devez avoir l'application Telegram téléchargée sur votre smartphone (disponible sur iOS et Android), c'est une application de chat similaire à What's App.
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.
2. Ensuite, entrez cette clé d'API dans Gladys, 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 !

Conclusion

Ce n'est qu'un simple exemple de ce qu'il est possible de faire avec Gladys Assistant, vous pouvez aller bien plus loin, en ajoutant des capteurs : température, humidité, ouverture de portes, mouvement. En connectant vos lumières avec des Philips Hue ou des ampoules Zigbee. En automatisant des appareils existants avec des prises télécommandées Zigbee.
N'hésitez pas à passer sur le forum Gladys (https://community.gladysassistant.com) si vous avez des questions/retours sur ce tutoriel !

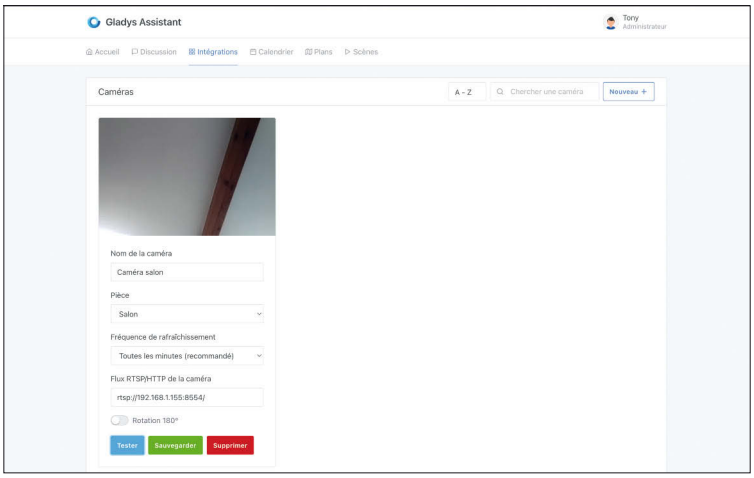


Figure 9

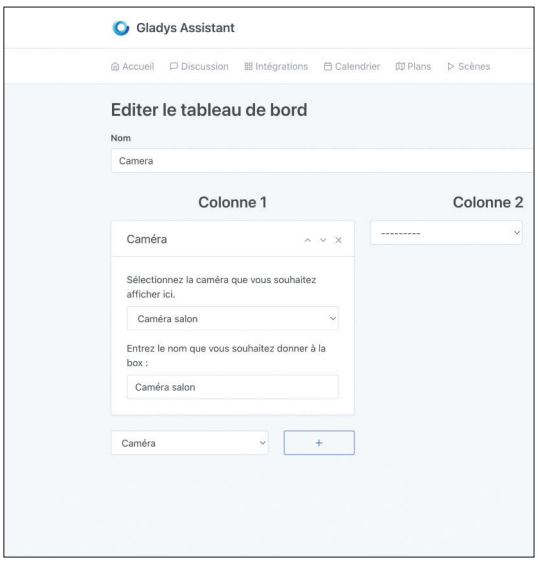


Figure 10

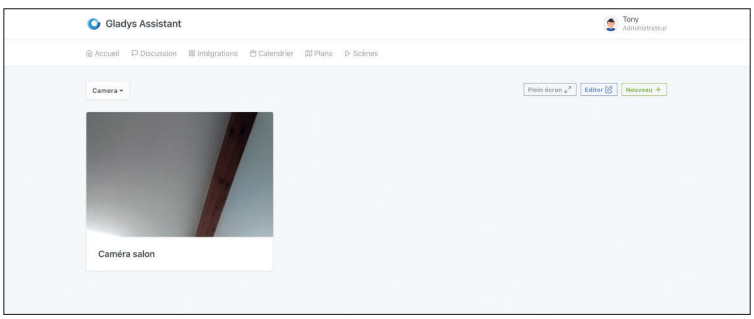


Figure 11

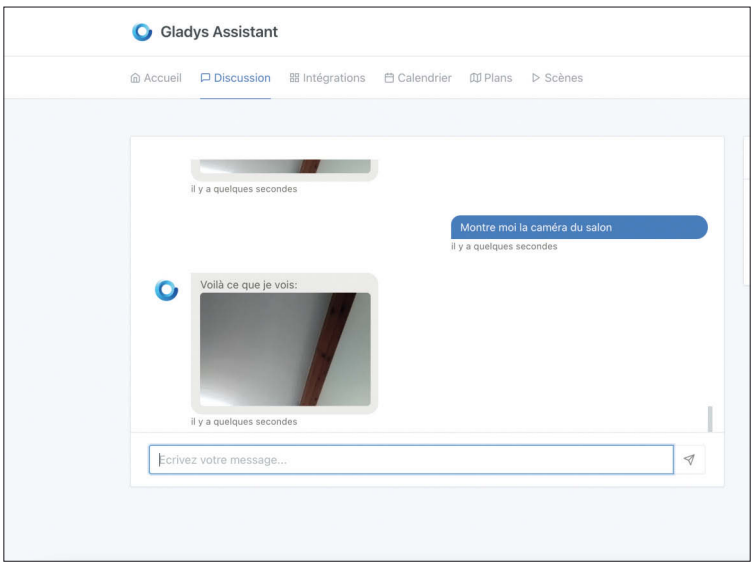


Figure 12

Loïc Mathieu
Consultant Formateur à Zenika Lille.
GCP Google Developer Expert.
Contributeur Quarkus.
https://www.loicmathieu.fr
https://twitter.com/loicmathieu

Java 19 : quoi de neuf ?

Java 19, sorti le 20 septembre, comporte 7 JEP - JDK Enhancement Proposal dont la JEP 405 qui introduit la déconstruction en Java via les Record Patterns et la très attendue JEP 425 qui implémente des threads légers en Java via les Virtual Threads du projet Loom.

JEP 405 : Record Patterns (Preview)

Cette JEP vise à enrichir le pattern matching de Java avec les record patterns qui permettent de déconstruire un record en ses attributs. Un record est un nouveau type dont le but est d'être un conteneur de données, il a été introduit en Java 14.
Avant la JEP 405, si vous vouliez faire du pattern matching sur un record puis accéder à ses attributs, vous auriez écrit du code comme celui-ci :

```
record Point(int x, int y) {}

static void printSum(Object o) {
  if (o instanceof Point p) {
    int x = p.x();
    int y = p.y();
    System.out.println(x + y);
  }
}
```

La JEP 405 permet de déconstruire le record et de directement accéder à ses attributs quand le pattern match, ce qui simplifie le code comme suit :

```
record Point(int x, int y) {}

void printSum(Object o) {
  if (o instanceof Point(int x, int y)) {
    System.out.println(x + y);
  }
}
```

Au lieu, dans le bloc de code exécuté au match du pattern, d'avoir accès à la variable `Point p`; vous avez accès directement aux attributs `int x`, `int y` du record `Point`. C'est la première étape dans la déconstruction de type en Java, on peut espérer voir prochainement la déconstruction arriver pour toutes les classes et pas uniquement les records. Plus d'informations dans la JEP 405 : https://openjdk.java.net/jeps/405.

JEP 422 : Linux/RISC-V Port

RISC-V est une architecture de jeu d'instructions RISC (Reduced Instruction Set Computer) gratuite et open source conçue à l'origine à l'Université de Californie à Berkeley, et maintenant développée en collaboration sous le parrainage de RISC-V International : https://riscv.org.
RISC-V définit plusieurs ISA, seul le jeu d'instruction RV64GV (general purpose 64bit) a été porté. Ce port comprend le support du JIT (C1 et C2) ainsi que de tous les GC existants (y compris ZGZ et Shenandoah).
Plus d'informations dans la JEP 422 : https://openjdk.java.net/jeps/422.

JEP 425 : Virtual Threads (Preview)

La JEP 425 introduit en preview feature les Virtual Threads (parfois aussi nommés green threads ou lightweight threads), ce sont des threads légers, avec un coût de création et de scheduling faible, qui permettent de faciliter l'écriture d'application concurrente.
C'est le projet Loom d'OpenJDK.
Les threads Java classiques sont implémentés via des threads OS, quel est le problème avec cette implémentation de thread ?

- Créer un thread en Java implique de créer un thread OS, et donc de réaliser un appel système, ce qui est coûteux.
- Un thread a une stack de taille fixe, par défaut de 1Mo sur Linux amd64, configurable via `-Xss` ou `-XX:ThreadStackSize`.
- Un thread sera schedulé via le scheduler de l'OS, chaque changement d'exécution d'un thread entraînera un context switch qui est une opération coûteuse.

Pour pallier ces problèmes, les applications concurrentes ont eu recours à plusieurs types de construction :

- Les pools de threads qui permettent de réutiliser un thread pour plusieurs requêtes (HTTP, JDBC, …). Le degré de concurrence de l'application est donc la taille du pool.
- La programmation réactive qui va ouvrir un nombre de threads très réduit (1 ou 2 par unité de CPU), et se baser sur un système d'event loop (boucle d'évènements) pour traiter un ensemble de requêtes concurrentes sur ces quelques threads.

Mais ces constructions impliquent de la complexité lors du développement des applications. L'idée des Virtual Threads est d'offrir des threads peu coûteux à créer, et donc de permettre de se passer de pool de threads ou d'event loop et d'utiliser un thread pour chaque requête.
C'est la promesse du projet Loom.
Le principe est de proposer des threads virtuels qui sont créés et schedulés par la JVM, et utilisent un pool de thread OS, ce que l'on appelle des carrier threads (thread porteur). La JVM va alors automatiquement monter et démonter les threads virtuels des carrier threads quand nécessaire (lors des I/O par exemple). Les stacks des threads seront stockées dans la heap de la JVM, leur taille n'étant plus fixe, un gain en mémoire est possible.
La manière la plus simple de créer des threads virtuels est via `Executors.newVirtualThreadPerTaskExecutor()`, on a alors un `ExecutorService` qui exécutera chaque tâche dans un nouveau thread virtuel.
Là où les threads virtuels sont le plus intéressants est pour les applications qui nécessitent une grande concurrence (plusieurs dizaines de milliers de thread) et/ou qui ne sont pas CPU bound (généralement celles exécutant des I/O). Attention, utiliser des threads virtuels pour des applications CPU bound, des applications faisant des calculs intenses par exemple, est contre-productif, car avoir plus de threads que

de CPU pour ce type d'application impacte négativement les performances.

Il existe des limitations dans l'implémentation actuelle des threads virtuels, dans certains cas un thread virtuel va épingler (pinning) le carrier thread qui ne pourra pas traiter d'autres threads virtuels :

- L'utilisation du mot clé synchronized
- L'utilisation de méthode native ou de l'API Foreign Function de la JEP 424

À cause de ces limitations, les threads virtuels ne sont pas forcément la solution à tous les problèmes de concurrence, même si des améliorations pourront être faites dans les prochaines versions de Java. Il faudra aussi que l'écosystème devienne compatible avec les threads virtuels (par exemple en évitant l'utilisation de bloc synchronisé).

Plus d'informations dans la JEP 425 : https://openjdk.java.net/jeps/425.

JEP 428 : Structured Concurrency (Incubator)

Le but de la JEP 428 est de simplifier la programmation multithread en introduisant une API pour la concurrence structurée. La concurrence structurée traite plusieurs tâches s'exécutant dans différents threads comme une seule unité de travail, rationalisant ainsi la gestion et l'annulation des erreurs, améliorant la fiabilité et l'observabilité. C'est un incubator module

Même si les Virtual Threads sont les stars de cette nouvelle version de Java, l'API Structured Concurrency est pour moi aussi intéressante, si ce n'est plus, en tout cas pour un développeur, car cela va impacter fortement la manière d'écrire du code multi-threadé / concurrent.

Le but de cette API est de pouvoir écrire un code multi-threadé plus lisible et avec moins de risque d'erreur dans son implémentation via une meilleure gestion des erreurs.

Avant la Structured Concurrency API, pour exécuter un ensemble de tâches concurrentes, puis joindre les résultats ; on écrivait un code comme celui-ci (exemple pris de la JEP).

```
ExecutorService ex = Executors.newFixedThreadPool(nbCores);
Response handle() throws ExecutionException, InterruptedException {
  Future<String>  user  = es.submit(() -> findUser());
  Future<Integer> order = es.submit(() -> fetchOrder());
  String theUser  = user.get();  // Join findUser
  int    theOrder = order.get(); // Join fetchOrder
  return new Response(theUser, theOrder);
}
```

Il y a plusieurs problèmes ici :

- Si une erreur arrive dans la méthode fetchOrder(), on va quand même attendre la fin de la tâche findUser().
- Si une erreur arrive dans la méthode findUser(), alors la méthode handle() va se terminer, mais le thread qui exécute fetchOrder() va continuer à s'exécuter, c'est un thread leak.
- Si la méthode handle() est interrompue, cette interruption n'est pas propagée aux sous-tâches et leurs threads vont continuer à s'exécuter, c'est un thread leak.

Avec la Structured Concurrency API, on peut utiliser la classe StructuredTaskScope qui permet de lancer un ensemble de tâches et de les gérer comme une unité de traitement unique. Le principe est de découper une tâche en sous-tâches (via fork()) qui vont se terminer au même endroit : scope.join(). Voici l'exemple pris de la JEP.

```
Response handle() throws ExecutionException, InterruptedException {
  try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
    Future<String>  user  = scope.fork(() -> findUser());
    Future<Integer> order = scope.fork(() -> fetchOrder());

    scope.join();       // Join both forks
    scope.throwIfFailed(); // ... and propagate errors

    // Here, both forks have succeeded, so compose their results
    return new Response(user.resultNow(), order.resultNow());
  }
}
```

Ici les deux sous-tâches findUser() et fetchOrder() sont démarrées dans leur propre thread (par défaut un thread virtuel), mais sont jointes et potentiellement annulées ensemble (comme une unité de traitement). Leurs exceptions et leurs résultats seront agrégés dans la tâche parente, et toute interruption du thread parent sera propagée aux threads enfants. Lors d'un thread dump, on verra bien les sous-tâches comme enfants du thread parent, facilitant le debugging de code écrit de cette manière. Plus d'informations dans la JEP 428 : https://openjdk.java.net/jeps/428.

Les fonctionnalités qui restent en preview

Les fonctionnalités suivantes restent en preview (ou en incubator module). Pour les détails sur celles-ci, vous pouvez vous référer à mes articles précédents.

Vector API : quatrième incubation de la fonctionnalité. Vector API est une nouvelle API qui permet d'exprimer des calculs de vecteur (calcul matriciel entre autres), qui seront exécutés via des instructions machines optimales en fonction de la plateforme d'exécution.

Plus d'informations dans la JEP 426 : https://openjdk.org/jeps/426.

Foreign Function & Memory API : après deux incubations pour ces deux fonctionnalités rassemblées dans une même JEP, celles-ci passent en preview. Foreign Memory API permet de gérer des segments mémoire (on heap ou off heap) tandis que Foreign Function l'interconnexion de la JVM avec du code natif (en C par exemple) de façon facile et performante. Ces deux API sont les bases du projet Panama.

Plus d'informations dans la JEP 424 : https://openjdk.org/jeps/424.

Pattern Matching for switch : troisième preview avec l'introduction de la clause when pour les guards à la place du &&. Le support du pattern matching dans les switch permet d'écrire un switch sur le type d'une variable, donc sur sa classe / son interface. Plus d'informations dans la JEP 427 : https://openjdk.org/jeps/427.

Conclusion

La tant attendue JEP 425 Virtual Threads est enfin arrivée et pourrait bien changer la donne dans la programmation concurrente en baissant le coût de création des threads en Java. En conjonction avec la JEP 428 qui permet d'écrire du code concurrent plus simplement et avec une meilleure gestion des erreurs, c'est une mini révolution qui se prépare. Bien sûr, l'adoption des virtual threads se fera lentement, car l'écosystème devra se mettre à jour pour les supporter, mais c'est une étape importante pour Java et la JVM.

Seifeddin MANSRI

Seifeddin est le directeur de l'ingénierie cloud chez Sfeir, responsable des activités Cloud Infrastructure et DevOps, il gère ainsi plusieurs équipes d'experts cloud. Il a de multiples certifications techniques cloud (GCP, AWS, Kubernetes). Il est aussi formateur officiel Google Cloud et Kubernetes. Il est reconnu, depuis cette année, comme étant Google Developer Expert (GDE) sur les technos Google Cloud Platform (GCP).

Anthos : écrire une fois, exécuter partout

Anthos a été introduit le 9 avril 2019 lors de la conférence annuelle Google Cloud Next par Sundar Pichai, le CEO de Google. Anthos permet d'exécuter les applications sur les infrastructures existantes on-premises ou sur le cloud public, de manière simple, flexible et sécurisée.

La promesse d'Anthos : incarner le principe « écrire une fois et exécuter partout », et cela dans le Cloud avec bien évidemment Google Cloud, mais également sur d'autres cloud providers comme Amazon Web Services ou Microsoft Azure, mais aussi on-premises où il est possible de déployer Anthos sur du VMware ou même directement sur des serveurs physiques (bare metal).

Anthos peut être considéré comme une offre de modernisation d'applications hybrides et multi-cloud, basée principalement sur Kubernetes. Toutefois, Anthos s'appuie également sur d'autres technologies open source moderne qui augmentent sa puissance telle que Istio ou Knative. Il est ainsi bâti sur les fondations de Google Kubernetes Engine (GKE), qui joue le rôle de centre de commande et de contrôle. Les utilisateurs peuvent ainsi gérer leur infrastructure distribuée sur le cloud ou dans un data center sur site.

Anthos vient du grec ancien ανθος, ánthos, et signifie fleur. La raison pour laquelle ils ont choisi ce nom est que les fleurs poussent sur place (on-prem), mais qu'elles ont besoin de la pluie des nuages (Cloud) pour s'épanouir.

Les composants clés

Anthos est constitué de plusieurs composants qui peuvent tous fonctionner ensemble pour aider les entreprises à créer et à gérer des applications hybrides et multi-cloud modernes dans tous les environnements (Dev, Staging et Prod). **Figure 2**

Figure 1
Introduction d'Anthos par Sundar Pichai

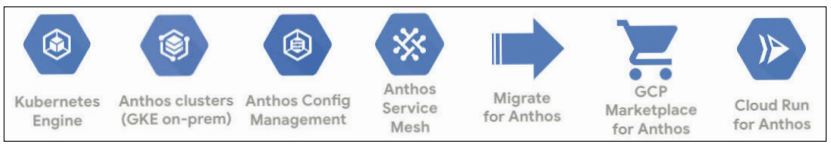


Figure 2

Kubernetes (K8S)

L'environnement d'exécution d'applications dans Anthos s'appuie sur Google Kubernetes Engine (GKE) et Anthos clusters (GKE On-Prem) pour gérer les installations Kubernetes dans les différents environnements où sont déployées les applications. Pour rappel, Kubernetes est un orchestrateur de conteneurs open source créé initialement chez Google en s'inspirant de Borg, l'orchestrateur interne de ses datacenters. K8S a été par la suite donné à la Cloud Native Computing Foundation (CNCF) qui s'en occupe depuis 2015. Techniquement parlant, Kubernetes est composé principalement de deux parties : le plan de contrôle (control plane) qui constitue le cerveau du cluster et les nœuds où seront généralement déployées les applications.

Google Kubernetes Engine (GKE)

Il s'agit de la version gérée par Google de Kubernetes qui permet de déployer en production des applications sécurisées et hautement scalables en s'appuyant sur l'infrastructure de Google Cloud. Avec GKE, Google Cloud héberge le plan de contrôle, et c'est uniquement le serveur d'API Kubernetes qui est accessible aux clients pour évidemment interagir avec le cluster. Quant aux nœuds, c'est sans surprise qu'ils sont dans le projet de l'utilisateur à l'aide d'instances Google Compute Engine (GCE), le service de provisionnement (provisioning) de machines virtuelles de Google Cloud.

Anthos clusters (GKE On-Prem)

Un composant clé d'Anthos est la version on-prem de GKE, qui permet aux clients de tirer parti de Kubernetes et de la technologie cloud dans leur propre centre de données. Ils bénéficient d'une expérience GKE avec des installations rapides, gérées et simples, ainsi que des mises à jour validées par Google. C'est tout simplement un concentré du savoir-faire de Google en matière d'orchestration mise à disposition des clients pour être utilisée à l'extérieur de Google Cloud. C'est aussi un moyen efficace pour garantir la compatibilité des clusters Anthos avec le reste de la suite technique.

Anthos Config Management (ACM)

Anthos Config Management est un gestionnaire de configuration multi-cluster qui permet d'avoir des règles cohérentes sur l'ensemble des clusters, qu'ils soient on-prem ou dans le Cloud. Il s'appuie sur un référentiel Central Git pour gérer les politiques de contrôle d'accès, les quotas de ressources et les namespaces. C'est déclaratif et continu : une fois que vous déclarez le nouvel état souhaité, l'outil vérifie en permanence les changements qui vont à l'encontre de l'état désiré.

ACM combine trois composants : Policy Controller, Config Sync et Config Controller. Ensemble, ils ont la responsabilité de protéger et de configurer en continu vos ressources Google Cloud et Kubernetes, comme illustré dans le schéma suivant : **(Figure 3).**

Policy Controller

Anthos Config Management Policy Controller offre une intégration entre Open Policy Agent (OPA) et Kubernetes via un contrôleur personnalisé. OPA est une solution open source qui permet de mettre en place des contrôles basés sur des règles pour les environnements cloud natifs. Dans le contexte Kubernetes, OPA transforme les politiques Rego en ressources Kubernetes, leur permettant d'être personnalisées et déployées à l'aide de flux de travail standard.

Les politiques permettent à l'application de mettre en place des règles entièrement programmables pour les clusters qui empêchent toute modification de la configuration de l'API Kubernetes, de violer les contrôles de sécurité, opérationnels ou de conformité.

Config Sync

Config Sync est un outil open source qui permet une gestion centralisée des configurations et des politiques de sécurité qui peuvent être déployées sur plusieurs clusters pouvant s'étendre sur des environnements hybrides et multicloud. Ce processus simplifie et automatise la configuration et la gestion des politiques de sécurité à grande échelle.

Config Sync est tout simplement une boucle de réconciliation entre les configurations déclarées (appelées configs) dans un repo Git et l'état du ou des clusters. Il est considéré comme un outil de Configuration-as-Code puisqu'il permet d'adopter une approche GitOps quand il s'agit d'administrer une flotte de clusters. **Figure 4**

Config Controller

Config Controller est un service permettant de provisionner et d'orchestrer les ressources Anthos et Google Cloud. Ce composant offre un point de terminaison d'API qui peut provisionner, activer et orchestrer les ressources Google Cloud. Il s'appuie principalement sur Config Connector qui est un add-on open source dans Kubernetes qui permet de faire de l'Infrastructure-as-Code pour déployer des ressources Google cloud en utilisant la syntaxe et le flux de travail Kubernetes.

Istio

Istio est une solution de maillage (mesh) de services open source qui réduit la complexité des déploiements et la gestion des interactions de service entre les conteneurs et les machines virtuelles. Il permet de connecter Google Cloud, d'autres fournisseurs cloud, des bases de données et d'autres composants dans un maillage de services unique, prenant en charge l'équilibrage de charge (load balancing), la surveillance d'un grand nombre de clusters et la gestion du trafic.

Au niveau de l'utilisateur, il offre des fonctionnalités très intéressantes telles que les circuit-breakers, timeouts, retries, traffic splitting ou encore les health checks actifs et passifs… Istio vous permet de créer facilement des clusters et peut prendre en charge les opérations sur des clusters déjà déployés, offrant une meilleure visibilité sur le comportement des services, les performances et l'état de santé des applications.

Anthos Service Mesh (ASM)

Bâti sur les API d'Istio, Anthos Service Mesh permet de créer facilement un réseau des services déployés (maillage de services) et offre une télémétrie prête à l'emploi pour la gestion des services. Il prend également en charge des fonctionnalités avancées telles que l'équilibrage de charge de service à service, l'authentification, la collecte des métriques

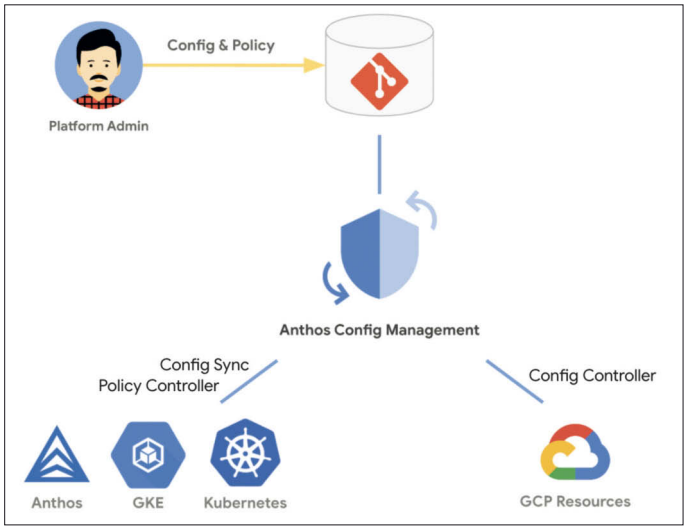


Figure 3

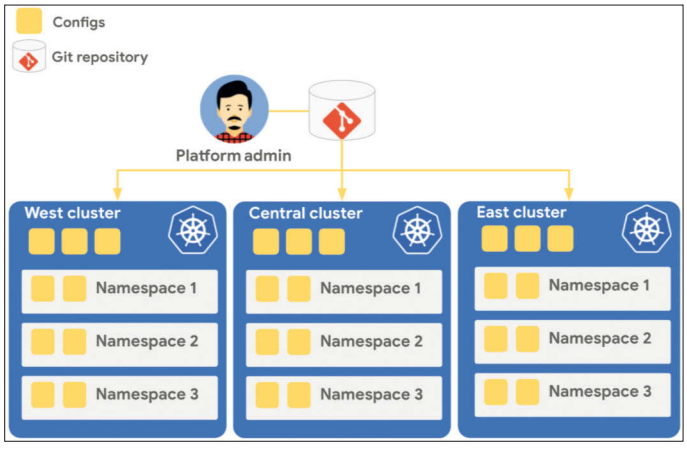


Figure 4

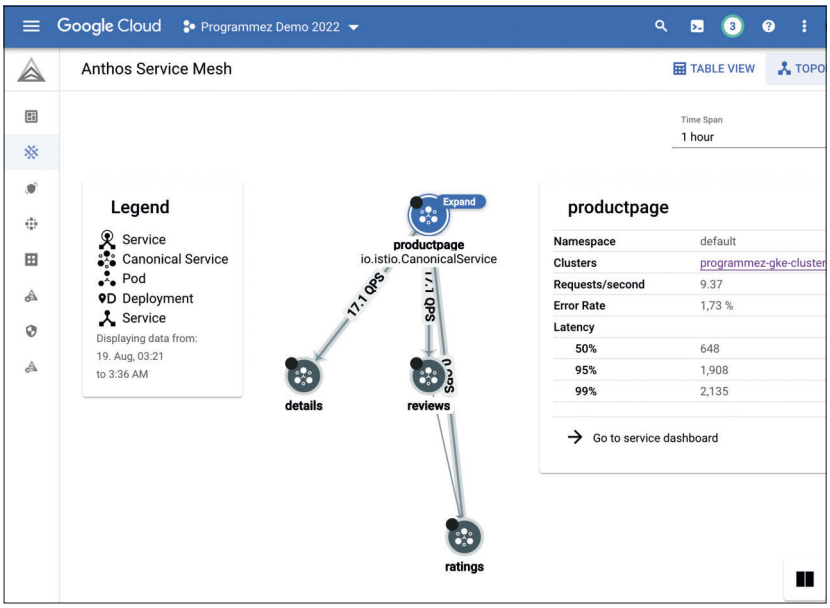


Figure 5

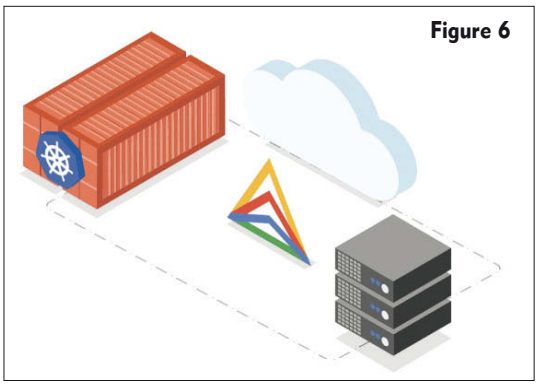


Figure 6

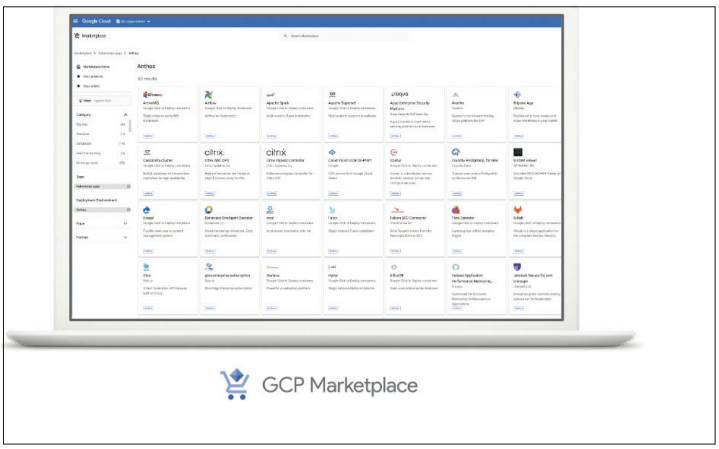


Figure 7

de niveau de service (requêtes par seconde, taux d'erreur, latence, etc.) et les alertes par rapport aux objectifs de niveau de service (SLO).

Anthos Service Mesh peut injecter de manière transparente un proxy side-car léger dans les conteneurs qui sera responsable d'appliquer des politiques uniformes pour toutes les communications entre les services sans nécessiter de modifications de code ou d'application.

Avec Anthos Service Mesh, les développeurs n'ont pas à coder en dur les configurations pour la gestion du trafic, les quotas et l'authentification dans l'application. Ils peuvent continuer à développer de nouvelles fonctionnalités métiers sans avoir à se soucier de la sécurité et des opérations.

Pour pouvoir tester ASM, nous avons créé un cluster GKE *programmez-gke-cluster*. L'option Anthos Service Mesh a été activée au moment de la création et la istio-ingressgateway a été installée (https://cloud.google.com/service-mesh/docs/gateways) qui est, en version courte, un service de type LoadBalancer et qui permet d'exposer les services à l'extérieur du maillage de services (mesh). Par la suite, la fameuse application de démo Bookinfo (https://istio.io/latest/docs/examples/bookinfo/) a été déployée. Enfin, un stress test a été réalisé à l'aide de l'outil siege pour envoyer du trafic vers l'application.

Le mesh peut être ainsi visualisé à travers la console Google Cloud avec quelques métriques sur le trafic reçu par l'application. **Figure 5**

Migrate for Anthos

Migrate pour Anthos donne la possibilité aux applications existantes, qui tournent sur des machines virtuelles Linux ou Windows, de bénéficier de toutes les possibilités offertes par la conteneurisation. Ce service permet, de moderniser facilement et rapidement ces applications en migrant les VM à partir de VMware, AWS, Azure ou même GCE vers des conteneurs gérés par Anthos. Cette démarche est souvent la meilleure option pour accélérer les migrations et l'intégration dans des infrastructures modernes. **Figure 6**

Google Cloud Marketplace

Google Cloud Marketplace pour Anthos vous permet de gérer de manière centralisée des logiciels prêts à être déployés en production en libre-service, et cela, en quelques clics. Il s'agit d'applications Kubernetes pré-intégrées compatibles avec Anthos et dont les mises à jour sont déjà gérées. Enfin, les utilisateurs paient une facture unique qui englobe les services Google et éventuellement les licences tierces des éditeurs de logiciels.**Figure 7**

Cloud Run for Anthos

Cloud Run est un produit d'exécution d'application serverless, entièrement gérée par Google Cloud et compatible avec Knative qui est une solution open source pour gérer les environnements serverless sur Kubernetes.

Cloud Run for Anthos permet une utilisation du serverless sur votre propre cluster, partout où Anthos peut s'exécuter.

Cela permet aux organisations de tirer parti de leur investissement existant dans Kubernetes, tout en adoptant les pratiques de développement Serverless. Ils bénéficieront ainsi de tous les avantages de la mise à l'échelle automatique, y compris jusqu'à zéro instance de conteneur, d'URL automatiques, de la répartition du trafic entre plusieurs versions, etc.

Cloud Run fournit des abstractions serverless conviviales pour les développeurs leur permettant d'en faire plus en moins de temps et par conséquent gagner en productivité. Et, comme Cloud Run est profondément intégré au reste

d'Anthos, vous pouvez tirer parti d'Anthos Service Mesh et d'Anthos Configuration Management en toute transparence. Vous bénéficiez également d'une visibilité opérationnelle, clé en main pour tous vos services.
Cloud Run pour Anthos vous permet même d'exploiter du matériel avancé, tel que des types de machines personnalisés, des GPU, des TPU et des capacités de mémoire élevées.
En bref, Cloud Run pour Anthos vous permet de combiner le meilleur du Serverless et de Kubernetes.

Options de déploiement d'Anthos

Il existe plusieurs options pour le déploiement d'Anthos qui se répartissent globalement en deux familles : le Cloud et on-premises.
Dans le cadre de cet article, on va plutôt se concentrer sur les architectures multi-cloud. À savoir les Anthos clusters sur AWS ou AZURE et les Attached Clusters avec EKS ou AKS qui sont respectivement les deux produits gérés autour de Kubernetes d'AWS et d'AZURE. **Figure 8**

Anthos clusters on AWS

Anthos clusters on AWS vous permet de gérer les clusters GKE exécutés sur l'infrastructure AWS via l'API Anthos Multi-Cloud. Associés à Connect, les clusters Anthos sur AWS vous permettent de gérer les clusters GKE sur Google Cloud et AWS à partir de la console Google Cloud.
Lorsque vous créez un cluster avec des clusters Anthos sur AWS, Google crée les ressources AWS dont vous avez besoin et affiche un cluster en votre nom. Vous pouvez ensuite déployer vos applications avec les outils de ligne de commande gcloud et kubectl.
La méthode la plus simple, mais aussi qui est recommandée pour l'installation d'un cluster Anthos est d'utiliser Terraform l'outil d'infrastructure-as-Code (IaC). Pour cela, Google Cloud met à disposition un repo GitHub (https://github.com/GoogleCloudPlatform/anthos-samples) qui contient les scripts nécessaires (sous le répertoire /anthos-multi-cloud/AWS/) pour le déploiement et la configuration des ressources AWS telles que VPC, subnets, internet gateway, iam roles, route tables et bien d'autres. Il faudra aussi installer et configurer, si ce n'est déjà fait, les outils de ligne de commande gcloud et aws. Il faudra aussi personnaliser le script en renseignant les variables dans le fichier *terraform.tfvars* telles que l'id du projet Google cloud, les types d'instances ou encore la localisation.

```
gcp_project_id = "programmez-demo-2022"
#add up to 10 GCP lds for cluster admin via connect gateway
admin_users = ["mansri.s@sfeir.com"]
name_prefix = "programmez-aws-cluster"
/* supported instance types
https://cloud.google.com/anthos/clusters/docs/multi-cloud/aws/reference
/supported-instance-types
*/
node_pool_instance_type    = "t3.medium"
control_plane_instance_type = "t3.medium"
cluster_version          = "1.22.8-gke.2100"
/* supported regions
https://cloud.google.com/anthos/clusters/docs/multi-cloud/aws/reference
/supported-regions
*/
gcp_location          = "europe-west1"
aws_region            = "eu-west-3"
subnet_availability_zones = ["eu-west-3a", "eu-west-3b", "eu-west-3c"]
```

Une fois les scripts terraform exécutés, il ne vous reste plus qu'à récupérer les informations de connexion pour la configuration du contexte pour kubectl qui est l'outil ligne de commande de Kubernetes qu'on utilisera par la suite pour interagir avec notre cluster Anthos sur AWS. Il est à noter qu'il faut rajouter l'option *"–location europe-west1"* à la commande proposée par le script. **Figure 9**
Pour vérifier que le contexte a été bien récupéré, il suffit d'exécuter la commande *kubectl get nodes* pour afficher les nœuds du cluster. **Figure 10**
Techniquement parlant, cinq instances EC2 ont été déployées : 3 control plane chacun dans une zone de disponibilité et un node pool avec 2 VM. **Figure 11**
On peut également observer le résultat à travers la console Google Cloud où on voit clairement la différence de type

Figure 8

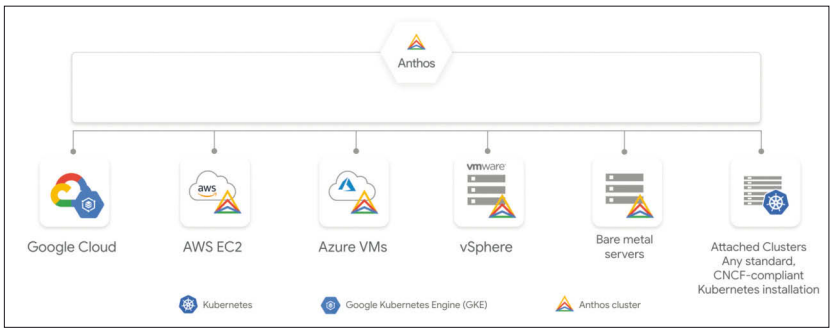


Figure 9

```
Apply complete! Resources: 61 added, 0 changed, 0 destroyed.

Outputs:

cluster_name = "programmez-aws-cluster-jd"
message = "To connect to your cluster issue the command: gcloud container aws clusters get-credentials
programmez-aws-cluster-jd"
```

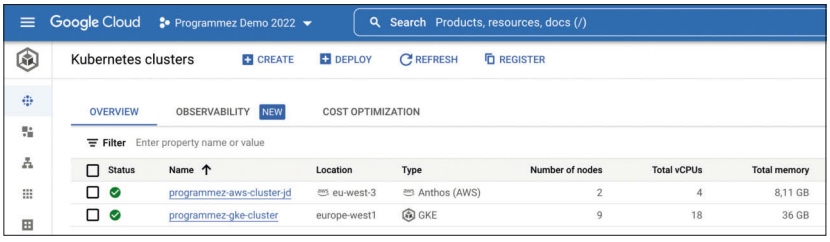
Figure 10

```
A new kubeconfig entry "gke_aws_programmez-demo-2022_europe-west1_programmez-aws-cluster-jd"
has been generated and set as the current context.
seifeddinemansri@Mac603 AWS % kubectl get nodes
NAME                                          STATUS   ROLES    AGE   VERSION
ip-10-0-1-145.eu-west-3.compute.internal      Ready    <none>   14m   v1.22.8-gke.2100
ip-10-0-1-70.eu-west-3.compute.internal       Ready    <none>   14m   v1.22.8-gke.2100
```

Figure 11

Name	ID d'instance	État de l'insta...	Type d'instance	Contrôle des statuts	Statut d'alar...	Zone de dispon...
programmez-aws-cluster-jd-cp	i-01ffe2c9578c1bb72	En cours d'exé	t3.medium	2/2 vérifications réussi	Aucune al...	eu-west-3b
programmez-aws-cluster-jd-cp	i-05ec6ed42eefcebfe	En cours d'exé	t3.medium	2/2 vérifications réussi	Aucune al...	eu-west-3c
programmez-aws-cluster-jd-cp	i-0ac48f5755bd9f548	En cours d'exé	t3.medium	2/2 vérifications réussi	Aucune al...	eu-west-3a
programmez-aws-cluster-jd-nodepool	i-0e92811569baa0f87	En cours d'exé	t3.medium	2/2 vérifications réussi	Aucune al...	eu-west-3a
programmez-aws-cluster-jd-nodepool	i-03212b1837625909f	En cours d'exé	t3.medium	2/2 vérifications réussi	Aucune al...	eu-west-3a

Figure 12



entre un cluster GKE créé précédemment et notre cluster Anthos dans AWS créé via Terraform. **Figure 12**

Anthos clusters on Azure

Les mêmes étapes que pour Anthos clusters on AWS peuvent être déroulées en exécutant les scripts nécessaires (sous le répertoire /anthos-multi-cloud/AZURE/) et en personnalisant le fichier *terraform.tfvars*

Figure 13

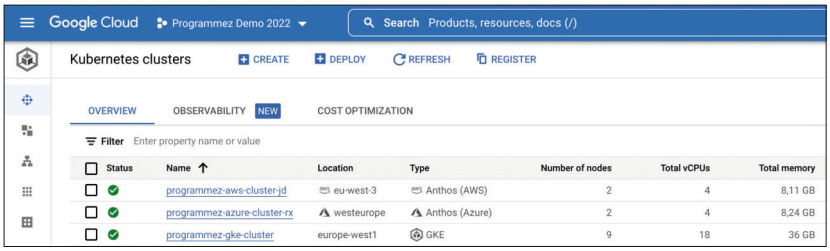


Figure 14

Figure 15

```
seifeddinemansri@Mac603 eks % export CLUSTER_NAME=$(terraform output -raw cluster_name) \
export MEMBERSHIP_NAME="programmez-eks-attached" \
export AWS_REGION=$(terraform output -raw region) \
export KUBECONFIG_CONTEXT=$(kubectl config current-context) \
export OIDC_URL=$(aws eks describe-cluster --name $CLUSTER_NAME --region $AWS_REGION --query "cluster.identity.oidc.issuer"
--output text)
```

Figure 16

```
seifeddinemansri@Mac603 eks % gcloud container hub memberships register $MEMBERSHIP_NAME \
--context=$KUBECONFIG_CONTEXT \
--kubeconfig="~/.kube/config" \
--enable-workload-identity \
--public-issuer-url=$OIDC_URL
Waiting for membership to be created...done.
Created a new membership [projects/programmez-demo-2022/locations/global/memberships/programmez-eks-attached] for the cluster
 [programmez-eks-attached]
Generating the Connect Agent manifest...
Deploying the Connect Agent on cluster [programmez-eks-attached] in namespace [gke-connect]...
Deployed the Connect Agent on cluster [programmez-eks-attached] in namespace [gke-connect].
Finished registering the cluster [programmez-eks-attached] with the Fleet.
```

Figures 17 et 18

```
seifeddinemansri@Mac603 eks % kubectl create serviceaccount -n kube-system admin-user
serviceaccount/admin-user created
```

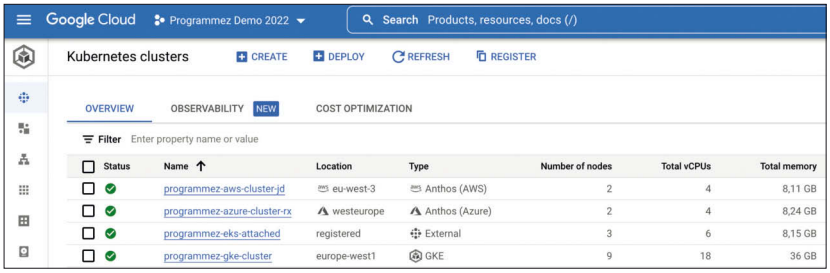
```
seifeddinemansri@Mac603 eks % kubectl create clusterrolebinding admin-user-binding \
--clusterrole cluster-admin --serviceaccount kube-system:admin-user
clusterrolebinding.rbac.authorization.k8s.io/admin-user-binding created
```

Figures 19 et 20

```
seifeddinemansri@Mac603 eks % SECRET_NAME=$(kubectl get serviceaccount -n kube-system admin-user
-o jsonpath='{$.secrets[0].name}')
```

```
seifeddinemansri@Mac603 eks % kubectl get secret -n kube-system ${SECRET_NAME} -o jsonpath='{$.data.token}'
 | base64 -d | sed $'s/$/\\\n/g'
eyJhbGciOiJSUzI1NiIsImtpZCI6IlUzX19PeGllSnZrSVJ4b09Bd2FXakd6ZDZHQm9abW96dnh5bWR1Wnc0ajQifQ.eyJpc3MiOiJrdWJl
cm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt
1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi1c2VyLXRva2VuLWdjODZuIiwia3ViZXJuZXRlcy5pby
9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L
3NlcnZpY2UtYWNjb3VudC51aWQiOiJmMTM5ZDBhYi1mNTNlLTQ2NDctODdlMy05ODZmMzJmNWM4YTQiLCJzdWIiOiJzeXN0ZW06c2Vydmlj
ZWFjY291bnQ6a3ViZS1zeXN0ZW06YWRtaW4tdXNlciJ9.qcXupZKgmdtDT9YrHRnNHECG5UvAxHvyXMYK53K5kZ0Cs30UcQ3sgfVEumObl-
Vv-uAW4w9MJNFNcXb1cUG8nP3PvwJRnP66DkYHFKPGgqzxcuNsid44HnHOaxtqz6ls0Ohq-PGUO-TEZpZFisfq80zgsHnPvh1lCrBe88OTT
o41CDwsTx46RUcoZu7Vu6dYo8s_f4gSS5En_0UHdzbJ2pAevJLSiBph-jNM4hQK-0FB0L9Z4b8HD_DViFF-GIS0mDMJ9tg1OKYVNKR_fc40
ZqdGlJ3klM4IZ2lTDqbSYJrmSKC6YLzOyBmEGcy9V_VSa9OJqF5O08doPaIFRR_Wlg
```

Figure 21




```
gcp_project_id = "programmez-demo-2022"
#add up to 10 GCP Ids for cluster admin via connect gateway
admin_users = ["mansri.s@sfeir.com"]
name_prefix = "programmez-azure-cluster"
/* supported instance types
https://cloud.google.com/anthos/clusters/docs/multi-cloud/azure/reference
/supported-vms
*/
control_plane_instance_type = "Standard_DS2_v2"
node_pool_instance_type    = "Standard_DS2_v2"
cluster_version          = "1.22.8-gke.2100"
/* supported regions
https://cloud.google.com/anthos/clusters/docs/multi-cloud/azure/reference
/supported-regions
*/
gcp_location = "europe-west1"
azure_region = "westeurope"
```

Et voilà, on a trois clusters Anthos créés dans les trois fournisseurs cloud majeurs ! **Figure 13**

Anthos attached clusters

Attacher des clusters vous permet de voir vos clusters Kubernetes existants dans la console Google Cloud côte à côte avec vos clusters Anthos et d'activer au sein de ceux-ci plusieurs fonctionnalités Anthos, comme la gestion centralisée des configurations avec Anthos Config Management ou la gestion d'architectures de type microservices avec Anthos Service Mesh.

Voyons maintenant comment rattacher un cluster EKS nouvellement créé à Anthos et ainsi utiliser GCP comme interface unique de gestion et de surveillance du cluster. Il faut imaginer que les étapes pour rattacher un cluster AKS dans Azure sont sensiblement les mêmes à quelques détails près.

D'abord, il faudra provisionner un cluster EKS qui est le service géré de Kubernetes sur AWS. Pour cela, vous pouvez vous appuyer sur le tutoriel officiel de Terraform pour la création du cluster (https://learn.hashicorp.com/tutorials/terraform/eks).

Figure 14

Une fois le cluster EKS créé et prêt à l'utilisation, il faudra l'enregistrer dans Google Cloud.

Commençons d'abord par exporter les variables d'environnement qui serviront pour l'enregistrement. **Figure 15**

Par la suite, nous utiliserons la CLI gcloud container hub membership pour effectuer l'enregistrement. Cela a pour effet, l'installation de l'agent Connect dans le cluster pour gérer la connexion à sa flotte. **Figure 16**

Enfin, nous avons besoin de nous connecter au cluster EKS attaché à partir de la console.

Cette opération commence par la création d'un compte de service Kubernetes avec le rôle d'administrateur du cluster (attention c'est bien différent d'un compte de service Google cloud). **Figures 17 et 18**

On utilisera ce compte de service pour générer le token qu'on utilisera pour se connecter au cluster EKS à partir de la console Google cloud. **Figures 19 et 20**

En plus du cluster GKE et des clusters Anthos dans AWS et AZURE, créés via Terraform, on peut maintenant voir le cluster EKS attaché. **Figure 21**

Bac à sable pour les développeurs

Envie de découvrir comment vous pouvez commencer à développer vos applications sur Anthos ? Sachez que Google met à disposition un bac à sable Anthos pour les développeurs, qui peut être utilisé gratuitement. Il suffit pour en disposer de posséder un compte Google.

Indépendamment de votre choix de plateforme d'exécution, le bac à sable Anthos vous permet d'effectuer certaines tâches quotidiennes de développement telles que :

- Le déploiement d'applications sur un cluster Kubernetes local avec minikube ;
- L'exécution des tests d'une application en local avec l'outil de création local Cloud Build (cloud-build-local) ;
- Le déploiement d'applications sur l'émulateur Cloud Run local.

https://cloud.google.com/blog/topics/anthos/introducing-the-anthos-developer-sandbox

Conclusion

Vous l'avez compris, Anthos fournit une plateforme unifiée pour la gestion des clusters Kubernetes sur différents Clouds et on-prem. Ces clusters peuvent être exécutés à l'intérieur de machines virtuelles ou sur du bare metal.

Anthos est un produit 100 % logiciel qui étend les services Google Cloud et les pratiques d'ingénierie à vos environnements afin que vous puissiez moderniser vos applications plus rapidement et établir une cohérence opérationnelle entre elles.

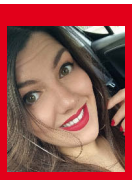
Il gère non seulement les clusters Anthos GKE on-prem et dans d'autres Cloud providers, mais également tous les clusters Kubernetes conformes à la norme CNCF, tels que AKS, EKS, OpenShift, etc.

Anthos est un excellent outil de gestion multicluster pour gérer, standardiser et sécuriser vos clusters dans plusieurs environnements cloud et fournisseurs de distributions Kubernetes.

À ce stade, Anthos n'a plus de secrets pour vous. Il ne vous reste plus qu'à le tester vous-même sur Google Cloud, on-prem ou sur d'autres fournisseurs de Cloud.

Resources

https://www.openpolicyagent.org/docs/latest/
https://istio.io/latest/docs/
https://istio.io/latest/docs/examples/bookinfo/
https://istio.io/latest/docs/tasks/traffic-management/ingress/ingress-control/
https://cloud.google.com/service-mesh/docs/install-anthos-service-mesh-console
https://cloud.google.com/service-mesh/docs/anthos-service-mesh-proxy-injection
https://cloud.google.com/service-mesh/docs/gateways
https://github.com/GoogleCloudPlatform/anthos-service-mesh-packages
https://cloud.google.com/anthos-config-management/docs/concepts/config-controller-overview
https://cloud.google.com/config-connector/docs/overview
https://cloud.google.com/anthos-config-management
https://cloud.google.com/anthos/clusters/docs/attached/how-to/attach-kubernetes-clusters
https://learn.hashicorp.com/tutorials/terraform/eks
https://github.com/GoogleCloudPlatform/anthos-samples/
https://knative.dev/docs/
https://linux.die.net/man/1/siege



Guendalina Caldarini

Guendalina est une ingénieure linguistique et doctorante dans le TAL, passionnée par le Deep Learning et les modèles linguistiques. Sa passion pour les langues (humaines et de programmation) l'a amenée à vivre et à travailler dans plusieurs pays dans le monde. Elle parle TAL chaque semaine sur son blog towardsnlp.com.

Les Transformers : deep dive pour mieux comprendre les gros modèles qui sont en train de révolutionner le Deep Learning

Si vous êtes passionnés de Machine Learning, vous avez sûrement entendu parler des Transformers. Il s'agit de modèles de Deep Learning très puissants, entraînés sur des millions de données, qui offrent des performances exceptionnelles dans plusieurs tâches. Mais d'où vient cette puissance ? Et pourquoi les Transformers sont-ils devenus si populaires ? Nous allons le découvrir ensemble.

Une Solution élégante pour un problème de longue date

Pour réaliser la plupart des tâches de Traitement Automatique du Langage (TAL ou NLP en anglais), nous utilisons aujourd'hui des modèles appelés Transformers. Ces derniers prennent une séquence de mots en entrée et génèrent une phrase cible, selon la tâche spécifiée.

Les Transformers sont considérés comme des modèles de post-deep learning, à cause de leur capacité de parallélisation des calculs et leur capacité d'homogénéisation. Grâce à l'homogénéisation, les Transformers peuvent accomplir plusieurs tâches sans besoin d'entraînement supplémentaire. Ils offrent un mode d'apprentissage auto-supervisé (self-supervised learning) sur des milliards de données brutes non structurées et réalisent des calculs sur des milliards de paramètres. Pour ces raisons, ils sont aussi appelés modèles fondateurs (foundational models).

Des Transformers ont été entraînés pour des tâches comme la traduction, la synthèse et la paraphrase, et ils peuvent même répondre à des questions ou décrire des images.

Figure 0

Pour bien comprendre pourquoi les Transformers sont si populaire et tellement performant, il est important de comprendre le mécanisme clé derrière les performances *nec plus ultra* des Transformers : l'Attention. Toutefois, avant de vous parler de ce mécanisme, il convient de faire un bond dans le passé qui nous permettra d'apprécier davantage l'arrivée des transformers. Avant l'arrivée des Transformers, pour la plupart des tâches de Traitement Automatique du Langage, les modèles Seq2Seq composés de *Recurrent Neural Networks* (RNNs) étaient la solution la plus utilisée. Dans toutes leurs formes (LSTMs et GRUs), ils présentent une structure assez simple : un encodeur (Encoder) et un décodeur (Decoder). L'encodeur s'occupe, comme son nom l'indique, d'encoder notre phrase d'entrée mot par mot. Une fois que tous les mots dans la phrase ont été encodés, une "contextualisation" de la phrase, appelée *context vector* **(Fig. 1)**, est donnée au décodeur, qui l'utilise pour générer une réponse appropriée. La sortie est encore une fois générée mot à mot. Or, face à une phrase particulièrement longue, on risque de perdre des informations importantes par rapport à l'interdépendance entre les mots. Le modèle n'arrive pas à se souvenir correctement des premiers mots encodés lorsqu'il résume la phrase et la séquence de sortie générée n'est pas de qualité satisfaisante.

Pour résoudre cette problématique, Bahdanau et al. (2015), ont développé une solution simple et élégante : le mécanisme de l'Attention. Le principe est facile à comprendre : en plus du

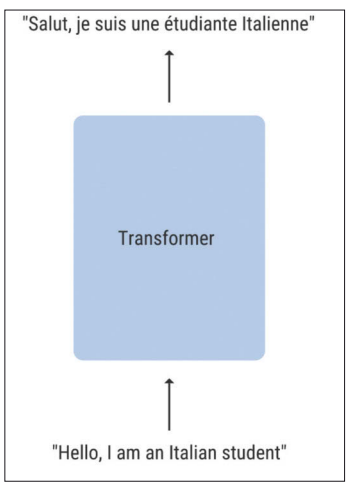


Figure 0. *Abstraction du fonctionnement d'un Transformer. Image créée par l'auteur.*

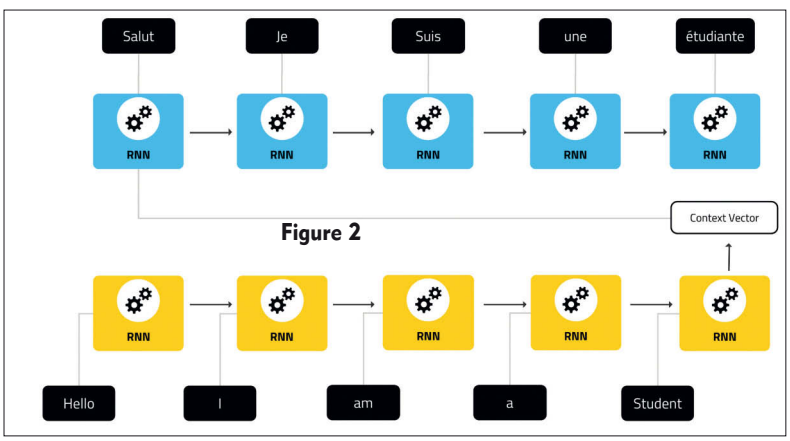


Figure 2

Figure 1. *Illustration d'un modèle Encoder (en jaune) - Decoder (en bleu). Image créée par l'auteur.*

contexte, on fait générer au modèle un "poids" pour chaque mot dans la phrase. Ce poids indique l'importance ou l'*attention* qu'on doit prêter au mot dans le contexte. L'ensemble des poids est donné au décodeur avec le *context vector*, ce qui lui permet de bien interpréter la phrase **(Fig. 2)**. L'attention pèse chaque mot de la séquence d'entrée en fonction de l'impact qu'il a sur la génération de la séquence cible.

L'Attention, la vraie puissance des Transformers

Quel lien entre l'Attention et les Transformers ? En effet, c'est précisément ce mécanisme qui rend les Transformers à la fois puissants et flexibles. Les Transformers sont des modèles basés uniquement sur ce mécanisme d'Attention : on encode chaque mot selon le poids qu'il a dans la phrase et en relation aux autres mots, sans passer par le *context vector*. La notion de "contexte" est incluse dans l'encodage fait par l'Attention, vu que chaque mot est encodé dans un principe d'interdépendance **(Fig. 3).**

Ce mécanisme est appelé auto-attention (Self-Attention), car la phrase initiale "prête attention" à elle-même. En traitant un mot, l'auto-attention permet au modèle de se concentrer sur d'autres mots de la séquence d'entrée qui sont étroitement liés à celui-ci.

Pour lui permettre de traiter plus de nuances sur l'intention (c'est-à-dire le sens et le ton de la phrase) et la sémantique de la phrase, un Transformer inclut des scores de multiples attentions pour chaque mot.

Pendant le traitement d'un mot, l'attention permet au modèle de se concentrer sur d'autres mots de l'entrée qui sont étroitement liés à ce mot.

Par exemple, dans la figure suivante, le lemme "chat" est étroitement lié aux mots "noir" et "caresse". En revanche, "noir" n'est pas lié à "fille". **Figure 4.**

L'architecture Transformer utilise l'auto-attention en reliant chaque mot de la séquence d'entrée à tous les autres mots.

Par exemple. Considérons deux phrases :

- Le chat a bu le lait parce qu'il avait faim.
- Le chat a bu le lait parce qu'il était sucré.

Dans la première phrase, le mot "il" fait référence au "chat", tandis que dans la seconde, il fait référence au "lait". Lorsque le modèle traite le mot "il", l'auto-attention lui donne plus d'informations sur sa signification afin qu'il puisse associer correctement "il" aux autres lemmes de la séquence.

Par ailleurs, pour lui permettre de traiter plus de nuances sur l'intention et la sémantique de la phrase, les Transformers incluent des scores de multiples attentions pour chaque mot. Par conséquent, lors du traitement du mot "il", le premier score met en évidence "chat", tandis que le deuxième score met en évidence "avait faim". Ainsi, lorsqu'il décode le mot "il", en le traduisant dans une autre langue, il va choisir le bon mot dans la langue d'arrivée en tenant compte de la relation entre "il", "chat" et "faim".

Dans l'Attention, la séquence intégrée passe par trois couches linéaires qui produisent trois matrices distinctes, appelées requête, clé et valeur. Ce sont ces trois matrices qui sont utilisées pour calculer le score d'attention. Il est important de garder à l'esprit que chaque "ligne" de ces matrices correspond à un mot de la séquence source, ce qui signifie

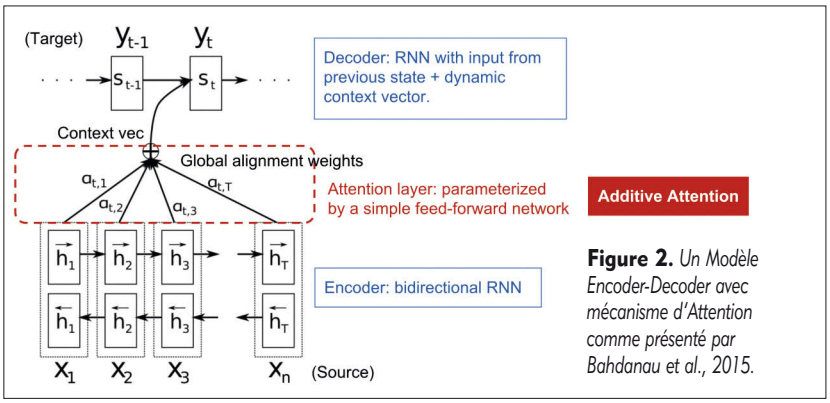


Figure 2. *Un Modèle Encoder-Decoder avec mécanisme d'Attention comme présenté par Bahdanau et al., 2015.*

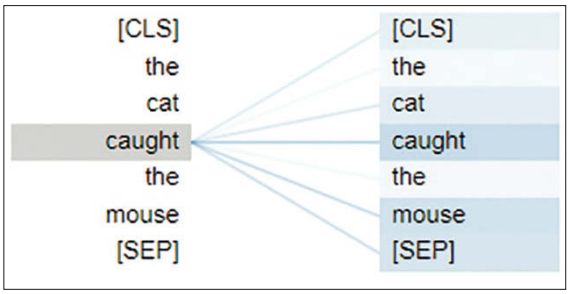


Figure 3. *Exemple des poids de chaque mot dans une phrase par rapport au mot en gris. Les nuances plus foncées représentent un poids plus élevé. Visualisation créée avec BertViz.*

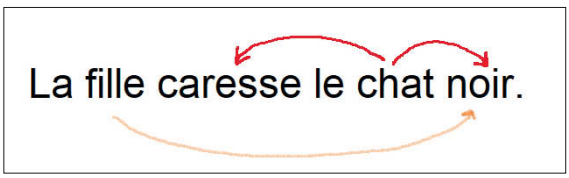


Figure 4. *Exemple d'Attention - Figure créée par l'auteur*

que chaque mot de la séquence est représenté par un vecteur. On ne va pas rentrer dans les détails du calcul de la matrice. Il suffit de penser à ce trois matrices et à leur utilisation comme une version plus soft d'un dictionnaire(1) :

dict = { 'a' : 1, 'b' : 2, 'c' : 3}

Dans le dictionnaire *dict* en exemple, 'c' est une Key, 3 est un Value.

Pour rechercher une valeur spécifique, on peut effectuer une recherche dans *dict* : dict['b'] (dict['b'] étant la Query). Dans le mécanisme de l'Attention, Key, Query and Value sont des vecteurs. Chaque clé (Key) est comparée à une requête (Query), et une valeur (Value) est retournée.

Vaswani et al. (2017) ont été les premiers à proposer ce nouveau modèle, et depuis, les entreprises et les laboratoires de recherche ont continué à les utiliser pour accomplir différentes tâches de Machine Learning, dans le TAL, mais pas seulement.

En effet, baser un modèle exclusivement sur l'Attention ne donne pas seulement de meilleurs résultats en termes de qualité de la réponse, mais permet aussi de paralléliser les calculs, ce que les anciens modèles ne permettaient pas à cause de leur nature séquentielle. Avec les Transformers, on ne doit plus attendre que le premier mot soit encodé pour encoder le deuxième, et on peut donc encoder chaque mot en parallèle, améliorant largement les performances en termes de calculs.

(1) https://www.w3schools.com/python/python_dictionaries.asp

On peut comparer les complexités des différents modèles dans le tableau suivant. **Table 1** -

Ici, d (ou d_model) est la dimension de représentation ou dimension d'intégration d'un mot (généralement dans la gamme 128-512), n est la longueur de la séquence (généralement dans la gamme 40-70), k est la taille du noyau de la convolution et *r* est la taille de la fenêtre d'attention pour l'auto-attention restreinte.

À partir de ce tableau, nous pouvons déduire ce qui suit :

- Il est clair que la complexité de calcul par couche de l'auto-attention est bien inférieure à celle des autres.
- Ces améliorations apportées par les Transformers se révèlent être des vrais atouts. Allons voir plus dans le détail la structure de ces modèles.

La Structure d'un Transformer

En **Figure 4**, vous trouverez la structure d'un Transformer dans son intégralité. À la base, il contient une pile de couches encodeur et décodeur. Pour éviter toute confusion, nous ferons référence à la couche individuelle en tant qu'encodeur ou décodeur et utiliserons pile encodeur ou pile décodeur pour un groupe de couches encodeurs.

La pile d'encodeurs et la pile de décodeurs ont chacune leurs couches d'incorporation correspondantes pour leurs entrées respectives. Enfin, il existe une couche de sortie pour générer la sortie finale.

Démontons donc un Transformer pour en étudier chaque composant : **Figure 5**

Table 1 - *Comparaison des modèles basés sur le RNN, le CNN et l'auto-attention sur la base de paramètres d'efficacité informatique via "Attention is all you need", Vaswani et al. (2017)*

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

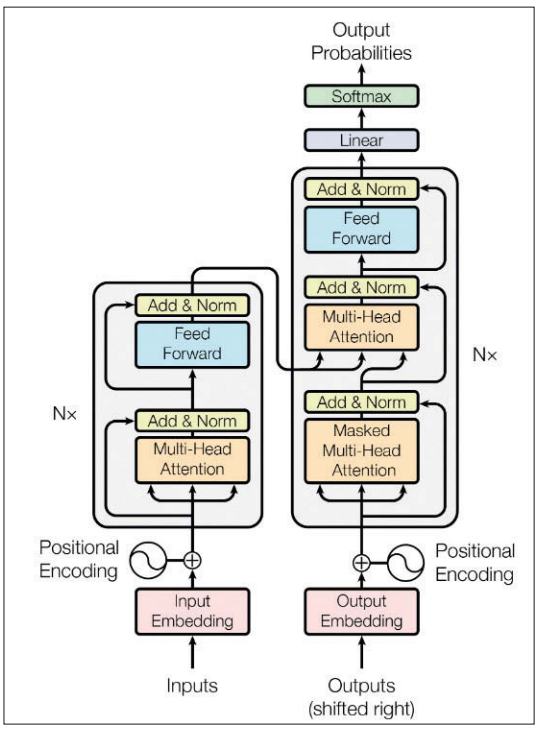


Figure 5. *Transformer via "Attention is all you need", Vaswani et al. (2017)*

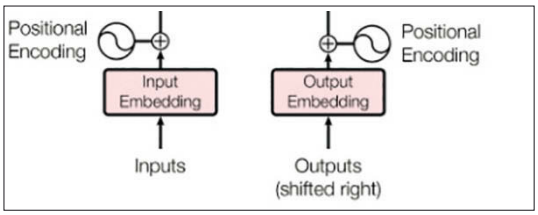


Figure 6. *Positional Encodings dans un Transformer via "Attention is all you need", Vaswani et al. (2017)*

Positional Encoding Figure 6.

On part de l'encodage positionnel (Positional Encodings). Nous savons que les calculs d'auto-attention (Self-Attention) n'ont aucune notion d'ordre des mots parmi les séquences. Ainsi que les RNN bien qu'ils soient lents, conservent l'ordre des mots par leur nature séquentielle. Ainsi, pour obtenir cette notion de positionnement des mots dans la séquence, des encodages positionnels sont ajoutés aux poids calculés par l'attention.

La dimension des encodages positionnels est la même que celle des embeddings(2) (d_model) pour faciliter la sommation des deux. Dans l'article, les encodages positionnels sont obtenus à l'aide de ces formules :

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Figure 7. *L'utilisation de sinusoïdes pour les encodages positionnels via "Attention is all you need", Vaswani et al. (2017)*

Ici, *i* est la dimension et pos est la position du mot. Nous utilisons le sinus pour les valeurs paires (2i) des dimensions et le cosinus pour les valeurs impaires (2i + 1). Il y a plusieurs choix pour les encodages positionnels (appris ou fixe). Dans notre cas, il s'agit de la méthode fixe, car l'article indique que les méthodes apprises et fixes donnent des résultats identiques.

L'idée générale derrière cela est qu'un décalage k fixe, PE peut être représenté comme une fonction linéaire de PE.

Masking

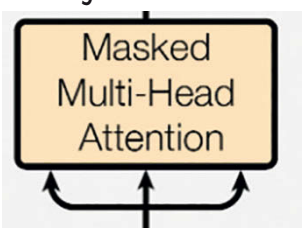


Figure 8. *Masking utilisé dans le Multi-Head Attention via "Attention is all you need", Vaswani et al. (2017)*

Il existe deux types de masques utilisés dans le mécanisme d'attention multi-têtes du Transformer.

- Masque de remplissage ou *Padding Mask* : le vecteur d'entrée des séquences est censé être de longueur fixe. Par conséquent, un paramètre max_length définit la longueur maximale d'une séquence que le transformer peut accepter. Toutes les séquences dont la longueur est supérieure à max_length sont tronquées, tandis que les séquences plus courtes sont complétées par des zéros. Toutefois, les zéros ne sont pas censés contribuer au calcul de l'attention ni à la génération de la séquence cible. C'est pour ça qu'on

(2) https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture#:~:text=An%20embedding%20is%20a%20relatively,like%20sparse%20vectors%20representing%20words.

introduit le masque de remplissage qui indique au modèle de ne pas prendre en considération ces zéros pendant l'encodage.

- *Look-ahead Mask* : lors de la génération de séquences cibles au niveau du décodeur, puisque le Transformer utilise l'auto-attention, il a tendance à inclure tous les mots de la phrase cible. Mais, en réalité, c'est de la triche ! C'est comme si on regardait les réponses du test avant de le passer. Seuls les mots précédant le mot courant peuvent contribuer à la génération du mot suivant, C'est pour cela que dans la phase de décodage, on masque les mots qui suivent.

Scaled Dot-Product Attention

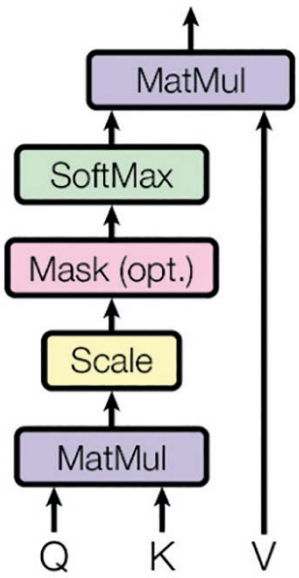


Figure 9. *Scaled Dot-Product Attention via "Attention is all you need", Vaswani et al. (2017)*

Il s'agit de l'étape principale du "calcul de l'attention" que nous avons abordée précédemment dans la section sur l'auto-attention. Elle comporte plusieurs phases :

- MatMul : il s'agit d'une opération de produit matriciel. La requête (Query) et la clé (Key) sont d'abord soumises à cette opération. Celle-ci peut être représentée mathématiquement comme suit :

  MatMul(Q, K) = Q.K

- qcale √la sortie de lgppération de produit scalaire hdotb productmpeut conduire f de ` randes valeurs´ . ui peuvent perturàer le so't, aa dans la partie suivanteexar conséb . uent´ nous les , ettons f lgéc- elle en les divisant par un 'acteur dgéc- elle (d□e
- Mask : Le masque de remplissage optionnel est déjà abordé dans la section masquage ci-dessus.
- Softmax : La fonction softmax ramène les valeurs à une distribution de probabilité, c'est-à-dire une valeur entre 0 et 1.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Équation 1. *Équation pour Scaled Dot-Product Attention via "Attention is all you need", Vaswani et al. (2017)*

L'attention du produit scalaire est une composante majeure de l'attention multi-têtes que nous allons voir dans la prochaine sous-section.

Multi-Head Attention

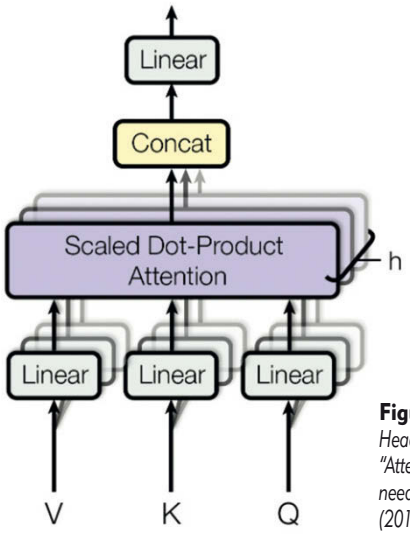


Figure 10. *Multi-Head Attention via "Attention is all you need", Vaswani et al. (2017)*

L'attention multi-têtes est essentiellement l'intégration de tous les micro-concepts discutés précédemment.
Dans la figure ci-contre, h est le nombre de têtes. En ce qui concerne les mathématiques, les entrées initiales de l'attention multi-têtes sont divisées en h parties. Chacune de ces parties a des requêtes, des clés et des valeurs, pour max_length de mots dans une séquence etpour des séquences de taille batch_size. Les dimensions de Q, K, V sont appelées *depth*. *Depth* est calculée comme suit :

depth = d_model // h

C'est la raison pour laquelle d_model doit être complètement divisible par h. Ainsi, lors de la division, les vecteurs de forme d_model sont divisés en h vecteurs de forme depth. Ces vecteurs sont passés en tant que Q, K, V au produit scalaire, et la sortie est concaténée en remodelant à nouveau les h vecteurs en un vecteur de forme d_model. Ce vecteur reconstruit est ensuite transmis à une couche de réseau neuronal à action directe (qui transmet l'information dans une seule direction).

Point-Wise Feed Forward Network and Residual Dropout

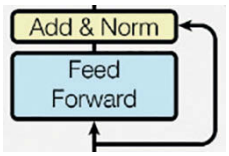


Figure 11. *Point-wise FFN and Residual Dropout via "Attention is all you need", Vaswani et al. (2017)*

Le bloc Point-Wise Feed Forward Network est essentiellement une transformation linéaire à deux couches qui est utilisée de manière identique dans toute l'architecture du modèle, généralement après les blocs d'attention.
Pour la régularisation, un Dropout est appliqué à la sortie de chaque sous-couche avant d'être ajouté aux entrées de la sous-couche et normalisé.

Le Flux des Données dans le Transformer

Le transformer ou transformeur fonctionne de façon légèrement différente entre l'apprentissage et l'inférence.

Apprentissage

Examinons d'abord le flux de données pendant l'apprentissage. Celles utilisées pour l'entraînement se composent de deux parties :

- La séquence source ou d'entrée (par exemple, "You are welcome" en anglais, pour un problème de traduction).
- La destination ou séquence cible (par exemple "De nada" en espagnol).

L'objectif du Transformer est d'apprendre à produire la séquence cible, en utilisant à la fois la séquence d'entrée et la séquence cible.

Le transformer traite les données comme suit :

1. La séquence d'entrée est convertie en Embeddings (avec codage de position) et transmise à l'Encodeur.
2. La pile d'encodeurs traite ces données et produit une représentation codée de la séquence d'entrée.
3. La séquence cible est précédée d'un jeton de début de phrase, convertie en imbrications (avec codage de position) et envoyée au décodeur.
4. La pile de décodeurs traite cette séquence avec la représentation codée de la pile de codeurs pour produire une représentation codée de la séquence cible.
5. La couche de sortie l'a convertie en probabilités de mots et en séquence de sortie finale.
6. La fonction de perte du transformer compare cette séquence de sortie avec la séquence cible provenant des données d'apprentissage. Cette perte est utilisée pour générer des gradients afin d'entraîner le transformer pendant la rétro-propagation.

Inférence

Pendant l'inférence, nous n'avons que la séquence d'entrée et nous n'avons pas la séquence cible à transmettre comme entrée au décodeur. Le but du transformer est de produire la séquence cible à partir de la seule séquence d'entrée.

Ainsi, comme dans un modèle Seq2Seq, nous générons la sortie dans une boucle et alimentons le décodeur avec la séquence de sortie de l'étape précédente dans l'étape suivante jusqu'à ce que nous rencontrions un jeton de fin de phrase.

La différence avec le modèle Seq2Seq est qu'à chaque étape, nous réinjectons la séquence de sortie entière générée jusqu'à présent, plutôt que le dernier mot.

Le flux de données pendant l'inférence est le suivant :

1. La séquence d'entrée est convertie en Embeddings (avec codage de position) et transmise à l'encodeur.
2. La pile d'encodeurs traite ces données et produit une représentation encodée de la séquence d'entrée.
3. Au lieu de la séquence cible, nous utilisons une séquence vide avec seulement un jeton de début de phrase. Celle-ci est convertie en Embeddings (avec codage de position) et envoyée au décodeur.
4. La pile de décodeurs la traite avec la représentation codée de la pile de codeurs pour produire une représentation codée de la séquence cible.
5. La couche de sortie l'a convertie en probabilités de mots et produit une séquence de sortie.
6. Nous prenons le dernier mot de la séquence de sortie comme le mot prédit. Ce mot est maintenant rempli dans la deuxième position de notre séquence d'entrée du décodeur, qui contient maintenant un jeton de début de phrase et le premier mot.
7. Retournez à l'étape 3. Comme précédemment, introduisez la nouvelle séquence du décodeur dans le modèle. Ensuite, prenez le deuxième mot de la sortie et ajoutez-le à la séquence du décodeur. Répétez cette opération jusqu'à ce que le modèle prédise un mot de fin de phrase. Notez que puisque la séquence de l'Encodeur ne change pas pour chaque itération, vous n'avez pas besoin de répéter les étapes une et deux.

Teacher Forcing

L'approche consistant à fournir la séquence cible au décodeur pendant l'apprentissage est connue sous le nom de Teacher Forcing. Pourquoi le faisons-nous et que signifie ce terme ?

Pendant l'apprentissage, nous aurions pu utiliser la même approche que celle utilisée durant l'inférence. En d'autres termes, faire tourner le transformer en boucle, prendre le dernier mot de la séquence de sortie, l'ajouter à l'entrée du décodeur et le transmettre au décodeur pour l'itération suivante. Enfin, lorsque le jeton de fin de phrase est prédit, la fonction de perte compare la séquence de sortie générée à la séquence cible afin d'entraîner le réseau.

Non seulement cette boucle rend l'apprentissage plus long, mais aussi plus difficile. Le modèle devrait prédire le deuxième mot sur la base d'un premier mot prédit, potentiellement erroné, et ainsi de suite.

Au lieu de cela, en fournissant la séquence cible au décodeur, nous lui donnons un indice, ainsi que le ferait un enseignant. Même s'il a prédit un premier mot erroné, il peut utiliser le mot correct pour prédire le mot suivant afin d'éviter que ces erreurs ne s'accumulent.

En outre, le transformer est capable de produire tous les mots en parallèle, sans boucle, ce qui accélère considérablement l'apprentissage.

Conclusion

Et voilà, on a reconstruit un Transformer. Vous savez maintenant comment l'Attention est utilisée pour donner un poids à chaque mot de la phrase d'entrée, et comment ces poids sont utilisés pour produire une sortie. Maintenant que vous connaissez son architecture et les fonctionnalités principales, vous pouvez explorer davantage ces modèles si puissants et performants. Il y a plusieurs librairies qui vous permettent de faire du Fine-Tuning sur des modèles déjà entraînés et des nombreux articles qui expliquent la structure des Transformers plus populaires.

Essayez vous-mêmes, et vous serez surpris de leurs performances et leur flexibilité !

Cafetière Arduino

On peut facilement trouver une cafetière de type Senseo pour 50 €, neuve et entre 20 et 30 €, d'occasion. Elles sont faciles à utiliser, les dosettes ne sont pas très chères et elles permettent de faire du café à la demande. Cette cafetière est extrêmement simple, elle n'a que 3 boutons. Un bouton central qui permet de l'allumer et un bouton de chaque côté pour choisir la taille du café (simple ou double).

Mais avec le modèle de base, cela demande 3 étapes :

1 Mettre la tasse, l'eau, 1 ou 2 dosettes et allumer la cafetière en appuyant sur le bouton central.

Attendre presque 1 min 30 que l'eau soit suffisamment chaude.

2 Sélectionner le café désiré (simple tasse ou double tasse).

Attendre encore environ 1 min que le café coule dans la tasse.

3 Prendre le café et le boire

Ce serait quand même plus pratique si l'on pouvait supprimer l'étape n°2 et sélectionner son café dès le début (comme avec la Senseo Viva). Et ensuite on pourrait en profiter pour ajouter de nouvelles fonctionalités...

Ce tuto utilise une Senseo modèle 7820. Si vous possédez une autre cafetière, il faudra peut-être modifier un peu le code ou les branchements, mais le principe reste le même

Préparation de la cafetière

Pour contrôler la cafetière, il faut simuler l'appui sur les boutons. La première étape est donc de démonter la cafetière, pour accéder à la carte de contrôle. Ensuite, il faudra repérer les contacts correspondant aux 3 boutons poussoirs et souder des fils qui seront reliés à des relais.

Il est évident qu'après cette manipulation, la garantie du constructeur ne pourra plus être invoquée en cas de panne de la cafetière. Je vous conseille donc d'utiliser une vieille cafetière.

Préparation de la cafetière

L'accès à la cafetière n'est pas extrêmement facile, car il faut déjà démonter l'arrière, qui tient avec une vis torx et qui est clipsée, avant d'accéder aux 2 vis torx qui maintiennent la plaque du dessous, sur laquelle est fixée la carte de contrôle. N'hésitez pas à consulter le guide du Repair Café, à cette adresse : https://smogey.org/wp-content/uploads/2017/09/Guide-de-reparation-Senseo-Version-4.1.1-fr-Beta-du-6-decembre-2015-2.pdf

Figure 1

Je vous conseille de faire preuve d'un minimum de délicatesse, si vous ne voulez pas rencontrer le même problème que moi : en insérant un gros tournevis à l'intérieur de la cafetière pour déclipser l'avant qui faisait un peu de résistance, j'ai pris appui sur le bouton central, qui n'est apparemment pas prévu pour supporter ce type de traitement… **Figure 2**

Heureusement, j'ai réussi sans problèmes, à le dessouder et le remplacer par un autre (carré). **Figure 3**

Une fois la carte de contrôle extraite, soudez les fils sur les contacts des 3 boutons (au dos). Il suffit de seulement 4 fils, car les 3 boutons sont reliés à la masse, mais si vous préférez, vous pouvez aussi utiliser 6 fils (2 x 3 boutons).

Quelques conseils : Utilisez un fer à souder de bonne qualité, ne dénudez pas les fils sur une trop grande longueur et étamez-les avant la soudure. **Figure 4**

Repérez bien les fils et remontez la cafetière. **Figure 5**

Les fils seront ensuite branchés à un module 4 relais (Alimentation de la cafetière + 3 boutons). **Figure 6**

Attention l'une des bornes est reliée à l'alimentation de la cafetière (230V).

Jean-Christophe Quetin

J'ai débuté sur MO5 avec le LOGO et sa petite tortue (un des seuls langages en français). J'ai retrouvé l'électronique et la programmation avec le Raspberry Pi et l'Arduino. J'ai eu l'occasion de travailler avec des élèves de collège sur l'Arduino et d'écrire en 2018 mon premier livre (dont la seconde édition est parue en 2021).

Arduino - Apprivoisez l'électronique et le codage pour donner vie à vos projets *(2e édition) au Editions ENI.*

micro:bit - Programmez la carte avec MakeCode et MicroPython

twitter.com/jcquetin

arduiblog.com

youtube.com/channel/UCQXCp5srIwc8eCv0ULUjrdA

Figure 1

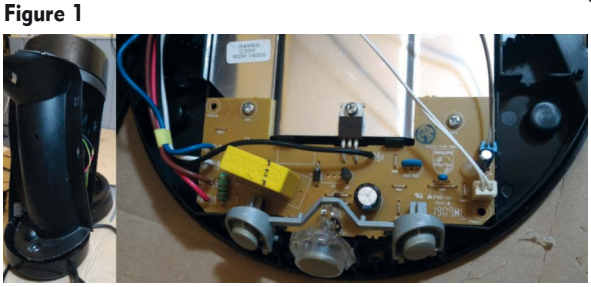


Figure 2

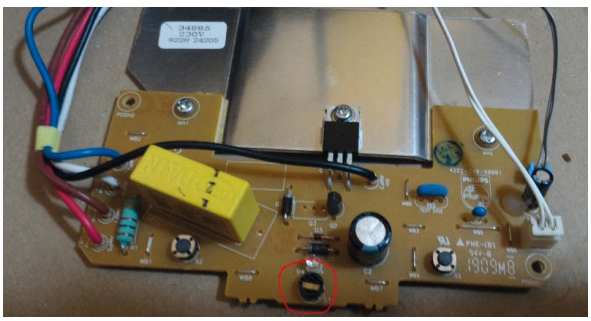


Figure 3

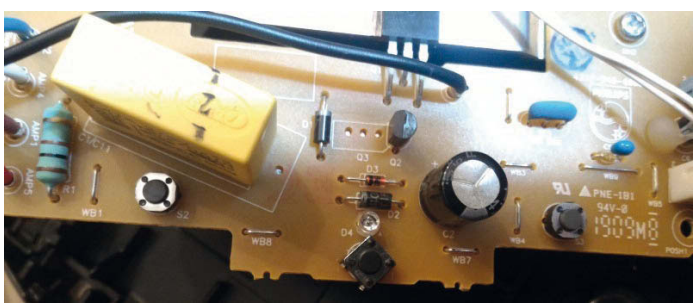


Figure 4

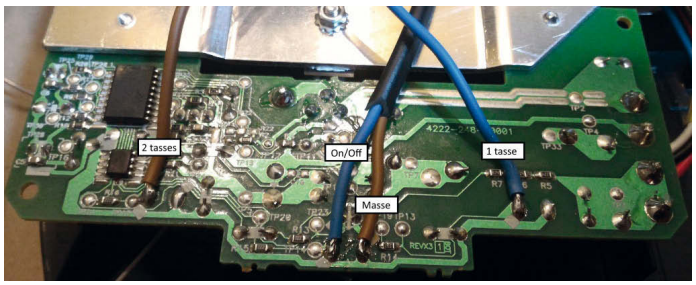


Figure 5

Figure 6

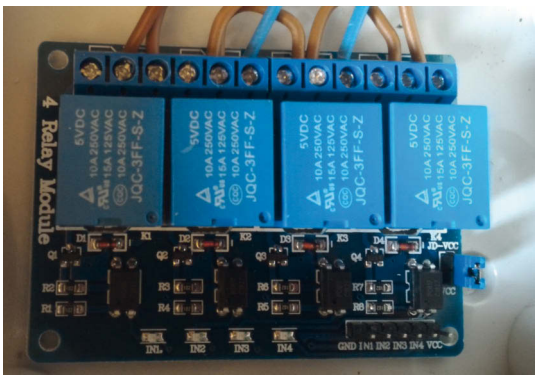


Figure 7

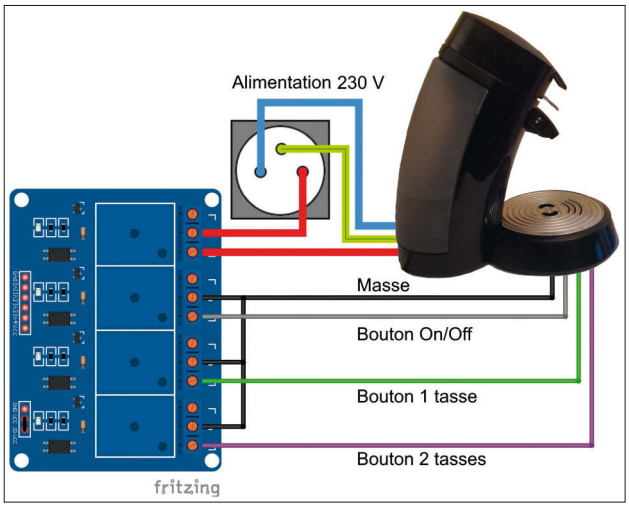


Figure 8

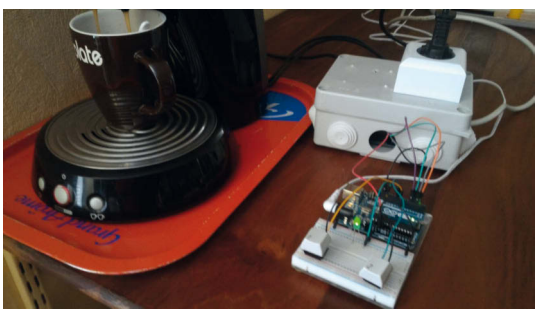
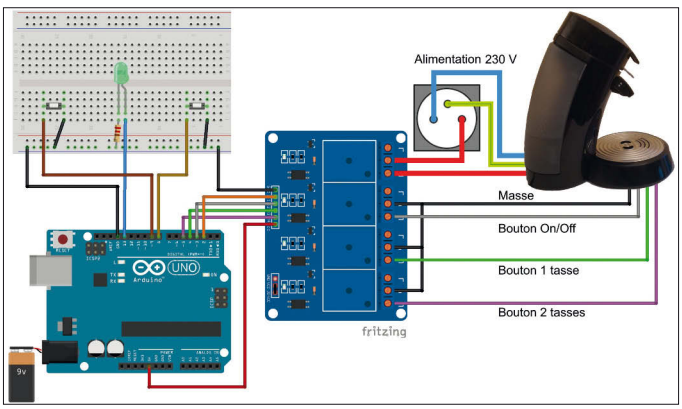


Figure 9



Travaillez toujours hors tension et enfermez le module relais dans un boîtier isolé.

Vérifiez aussi que la puissance de votre cafetière (1450W, dans cet exemple avec la senseo 7820) ne dépasse pas la puissance maximum admise par vos relais (10A X 230V = 2300W). Si c'est le cas, utilisez un relais plus puissant pour gérer l'alimentation. **Figure 7**

Vous pensez peut-être que l'utilisation de relais est un peu démesurée. En effet la carte de contrôle de la Senseo 7820 fonctionne en 5V (comme l'Arduino). Il serait donc possible d'utiliser de simples transistors ou mosfet (sauf pour l'alimentation).

Mais j'aimerais faire un tuto simple et universel, ou du moins qui puisse s'adapter facilement à d'autres modèles de cafetière. Les relais permettent de simuler des interrupteurs et ils acceptent jusqu'à 250V et 10A, autant dire qu'il y a de la marge (sauf si la puissance de votre cafetière dépasse 2300W). De plus, les relais sont beaucoup plus sûrs, car les circuits de la cafetière sont séparés de l'appareil qui la contrôlera, il y a donc peu de risque d'endommager votre matériel.

Un Arduino avec 2 boutons et une LED

C'est la 1ère étape. L'Arduino est certainement l'une des solutions les plus simples pour contrôler les relais. Dans cet exemple, il suffit de 2 boutons pour gérer la cafetière. Le bouton n°1 pour une simple tasse et le bouton n°2 pour une double tasse. La LED n'est pas obligatoire, mais elle permet d'indiquer que l'appui sur l'un des 2 boutons a bien été pris en compte, que le café est en cours de préparation et qu'il faut attendre avant de pouvoir en refaire un autre.

Matériel nécessaire :

- La cafetière modifiée (avec 1 module d'au moins 4 relais)
- 1 Arduino Uno et son alimentation,
- 1 breadboard avec des câbles Dupont,
- 2 boutons poussoirs,
- 1 LED et 1 résistance d'environ 220 Ω (facultatives, car vous pouvez aussi regarder la LED interne de l'Arduino qui est reliée à la même borne). **Figure 8**

Vous pouvez reproduire le schéma ci-dessous : **Figure 9**

Et téléverser le code correspondant dans l'Arduino (en adaptant éventuellement le temps de chauffe à votre cafetière) :

Code complet sur programmez.com & github

Pour préparer un café, les 3 boutons en façade de la cafetière sont désormais inutiles puisqu'ils sont gérés par l'Arduino. Lorsque la LED témoin est allumée en continu, il suffit de :

1 Mettre la tasse, l'eau, 1 ou 2 dosettes et d'appuyer sur un des 2 boutons de la breadboard (simple ou double tasse)

L'Arduino prend en charge l'alimentation et allume la cafetière, puis la LED clignote, l'eau chauffe, le café coule et l'eau contenue dans le réservoir se remet à chauffer.

2 Prendre le café et le boire

Lorsque la LED s'arrête de clignoter, vous pouvez préparer un autre café. Mais de toute façon, la cafetière s'éteindra au bout de 5 minutes d'inactivité.

Il faut quand même se lever pour appuyer sur le bouton de l'Arduino. Ce serait quand même plus pratique de déclencher la cafetière à distance. ***A suivre.***

Compiler du code C # avec Roslyn

Je vous propose de découvrir comment C#, et .Net, permet de compiler dynamiquement du code et de produire une assembly .Net qui contiendra votre code compilé. Bien sûr l'exécution ne demande pas d'avoir Visual Studio sur le poste d'exécution, mais uniquement un runtime .Net Framework.

Fred Berton

(frederic.berton@capgemini.com)

Je pratique le développement logiciel depuis plus de 35 ans, depuis 13 ans chez Capgemini dans une approche en phase avec la mouvance des artisans du logiciel "Software Craftsmanship". Ma motivation : relever le niveau du développement professionnel de logiciels par la pratique et en aidant les autres à acquérir le savoir-faire.

Roslyn, c'est le joli petit nom que Microsoft a donné au « Kit SDK .NET Compiler Platform » (forcément c'est moins sympa). Ce SDK open source fournit les outils permettant l'analyse, et la compilation de code exprimé avec les langages C# et VB.Net. Le principal objectif de cet outil est de permettre le développement d'outils d'analyse de code statique basés sur l'analyse syntaxique et sémantique qui donne les moyens de comprendre le code.

Vous l'avez compris c'est donc avec cet outil que l'on peut écrire des outils pour contrôler la qualité du code, ou faire de l'IntelliSense par exemple. Roslyn fournit l'ensemble des API nécessaire pour : l'analyse lexicale, l'analyse sémantique, la compilation dynamique, la génération de code, et la compilation statique. Dans cet article nous allons nous intéresser plus particulièrement à cette dernière fonctionnalité pour la compilation de notre code.

Les prérequis

Pour utiliser le compilateur Roslyn dans une application, c'est très simple, il suffit d'installer le package Nuget « Microsoft.CodeAnalysis. CSharp » (au moment de l'écriture de cet article en version 4.2.0), ou « Microsoft.CodeAnalysis. VisualBasic ».

Ce package Nuget va installer les assembly suivantes :

- Microsoft.CodeAnalysis.Analyzers
- Microsoft.CodeAnalysis.Common
- Microsoft.CodeAnalysis.CSharp **Figure 1**

Il existe également un package dit « all-in-one » qui installe l'ensemble des fonctionnalités y compris la prise en charge de Visual Basic. Le package s'appelle : Microsoft.Code Analysis

Vous pouvez utiliser le Roslyn dans un projet .NET Framework à partir de la version 4.5 ou .NET Core toutes versions.

Pour utiliser les API de Roslyn dans le code C#, il faut utiliser le using suivant :

```
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis;
```

Présentation des classes nécessaires

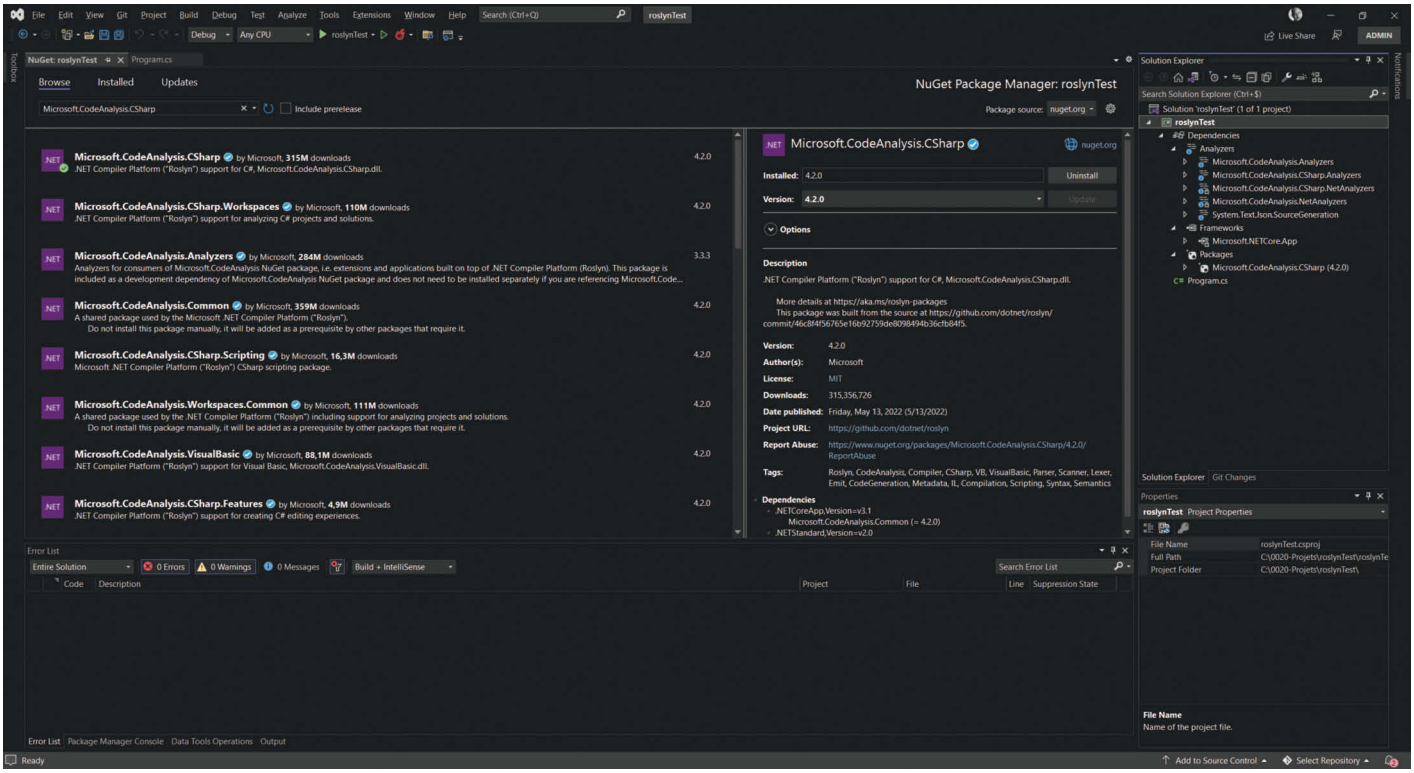
Voir tableau ci-contre

Notre exemple de code C# à compiler

Nous allons voir comment compiler du code C# dans une assembly de type librairie de classe dynamique (DLL). Puis nous réaliserons un changement dynamique de l'assembly que nous avons compilé. Finalement nous réaliserons une instance de la classe ClassACompiler.

Notre exemple de code C# est stocké dans un fichier : ClassACompiler.cs qui est un simple fichier texte. Il s'agit d'une

Figure 1



Classes utiles pour la compilation de code.

Classe	Description
CSharpSyntaxTree	Cette classe contient les méthodes nécessaires pour faire le parsing d'un code C#.
MetadataReference	Cette classe dispose des fonctions statiques qui permettent de créer les métadonnées nécessaires pour référencer une assembly pour la classe CSharpCompilation.
CSharpCompilation	Cette classe contient les méthodes nécessaires à la compilation d'un code contenu dans un SyntaxTree. Elle permet également de définir les références à d'autres assembly utilisées. Elle contient une méthode qui permet de générer une assembly avec le code compilé.
CSharpCompilation Options	Cette classe permet de définir des options utilisées par la classe CSharpCompilation. Par exemple le type d'assembly (exe ou dll) que l'on souhaite produire.

Classes utiles pour le chargement dynamique d'assembly.

Assembly	La classe Assembly contient les méthodes statiques nécessaires pour le chargement d'une assembly. Il est possible de charger une assembly soit avec son nom (Assembly.Load), soit avec son nom de fichier (Assembly.LoadFrom).
Activator	La classe Activator contient les méthodes statiques nécessaires pour créer une instance d'une classe à partir de son type.

simplification, le compilateur utilise une chaîne de caractère comme source.

```
using System;
using Interfaces;

namespace MyCompilation
{
  public class ClassACompiler : IInterfaceAdresse
  {
    public ClassACompiler()
    {
      Console.WriteLine($"Execute : MyCompilation.ClassACompiler()");
    }

    public void Init(string _Nom, string _Mail)
    {
      Console.WriteLine($"Execute : MyCompilation.Init({_Nom},{_Mail})");
      Nom = _Nom;
      Mail = _Mail;
    }

    public string Nom { get; set; }
    public string Mail { get; set; }
  }
}
```

Comme on peut le constater la classe ClassACompiler implémente l'interface IInterfaceAdresse ci-dessous

```
namespace Interfaces
{
  public interface IInterfaceAdresse
  {
    void Init(string _Nom, string _Mail);
    string Nom { get; set; }
    string Mail { get; set; }
  }
}
```

Cette interface qui sera connue de notre application nous permettra d'utiliser la classe ClassACompiler sans avoir de liaison statique vers l'assembly qui la contient. Heureusement, car justement cette assembly peut ne pas exister au moment de la compilation de notre application.

Compiler le code et produire une assembly

Pour compiler notre code, la première chose à faire est de construire un arbre de syntaxe avec la méthode ParseText de la classe CSharpSyntaxTree. Cet arbre permet d'analyser le code source sous la forme d'un arbre qui représente les concepts du langage. Il peut être utilisé pour la compilation de code ce qui est notre objectif ici, mais il peut aussi être utilisé pour analyser le code.

```
// Lecture dans un string du fichier avec le code de la classe
string sCodeSource = File.ReadAllText(@"..\..\ClassACompiler.cs");

// Parsing du code, creation du syntax tree
SyntaxTree tree = CSharpSyntaxTree.ParseText(sCodeSource);
```

Figure 2

L'avantage est que cette analyse est statique, utilise le texte de notre code source et non le code compilé. En revanche les fonctionnalités rendues sont les mêmes que celles offertes par la réflexion de code.

Avant de pouvoir compiler notre code à partir de l'analyse syntaxique, nous devons donner au compilateur la liste des assemblys qui sont référencées par celui-ci. Au minimum il faut donner la référence sur l'assembly System du .NET Framework qui contient la définition des types de bases. Pour définir une référence sur une assembly on utilise la fonction CreateFromFile de la classe MetadataReference. Cette méthode prend en paramètre le chemin du fichier de l'assembly pour lequel nous souhaitons obtenir une référence.

```
// Creation des references sur les assembly utilisées
string PathAssembly_System = typeof(object).Assembly.Location;
string PathAssembly_Interfaces = typeof(IInterfaceAdresse).Assembly.
Location;
var AssemblyRef_System = MetadataReference.CreateFromFile
(PathAssembly_System);
var AssemblyRef_Interfaces = MetadataReference.CreateFromFile
(PathAssembly_Interfaces);
```

La mise en place de la structure de compilation du code est réalisée par la fonction Create de la classe CSharpCompilation. Cette fonction prend en paramètres l'arbre de syntaxe à compiler, un tableau des assemblys qui sont référencées, et d'éventuelles options de compilation.

```
var CompilationOption = new CSharpCompilationOptions(OutputKind.
```

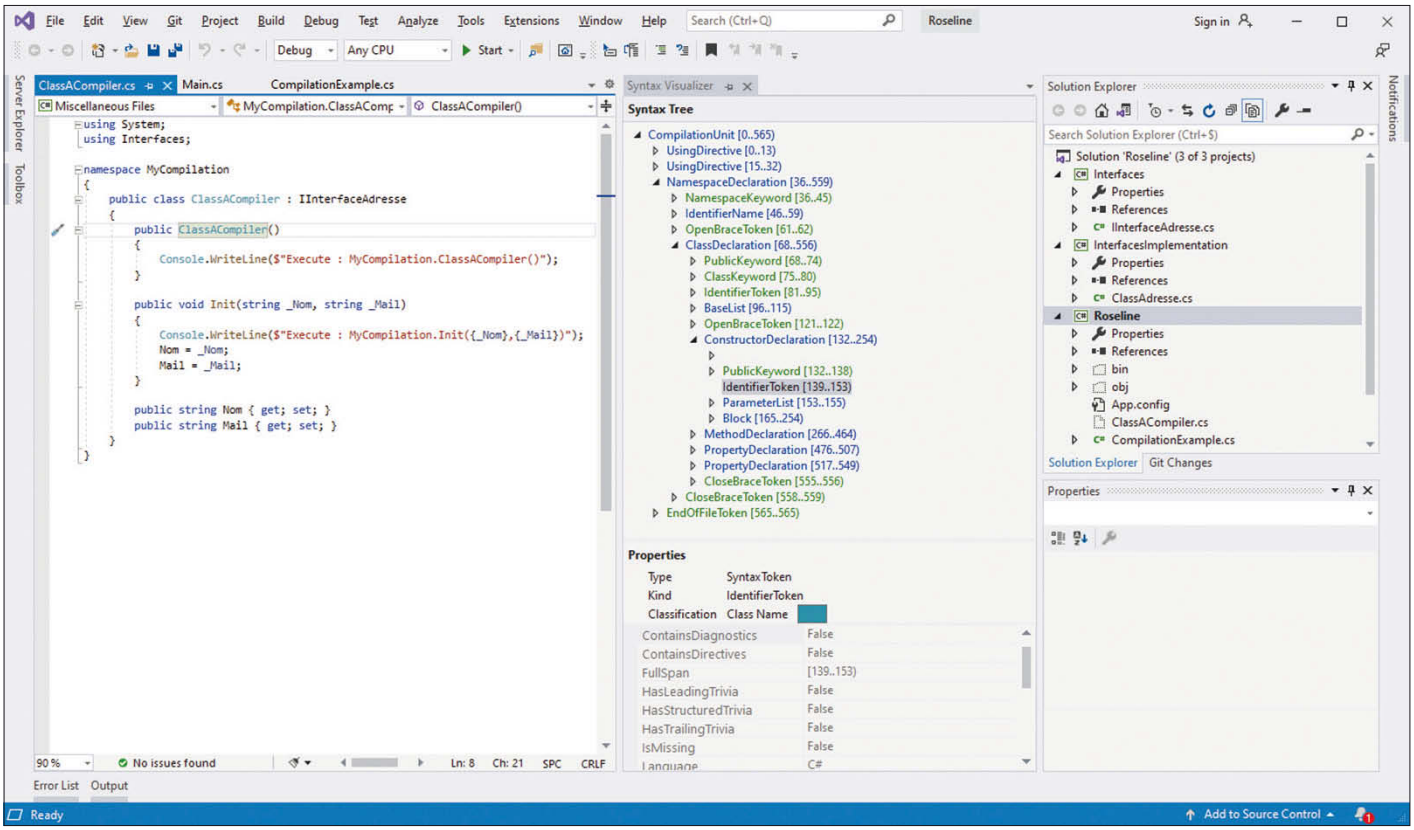


Figure 2 - *Arbre de syntaxe dans Visual Studio*

```
DynamicallyLinkedLibrary);
var compilation = CSharpCompilation.Create("MyCompilation",
                    syntaxTrees: new[] { tree },
                  references: new[] { AssemblyRef_System, AssemblyRef_Interfaces },
                    options: CompilationOption);
```

Les options de compilation définies dans la classe CSharpCompilationOptions permettent, entre autre, d'indiquer notre souhait de produire une assembly de type DLL lors de la compilation (OutputKind. DynamicallyLinkedLibrary). Beaucoup d'autres options sont disponibles, par exemple le niveau de warning, la cible CPU, etc.

La génération de l'assembly est produite par la fonction Emit de la classe CSharpCompilation, à laquelle on donne en paramètre le nom du fichier qui contiendra l'assembly.

```
EmitResult emitResult = compilation.Emit("output.dll");
```

D'autres paramètres sont possibles pour spécifier, par exemple : un fichier pdb pour les informations de debug, ou un fichier xml pour la documentation.

Cette fonction renvoie une instance de la classe EmitResult qui permet d'analyser le résultat de la compilation et de la production de l'assembly. Si l'opération a réussi la propriété Success est à True. En cas d'erreur, ou de warning, elles sont disponibles dans la propriété Diagnostics.

Le code suivant permet d'exploiter ce retour d'informations.

```
// S'il y a des erreurs de compilation le dire
if (emitResult.Success == false)
  Console.WriteLine("Compilation KO");
else
  Console.WriteLine("Compilation OK");

// Afficher les résultats de la compilation.
// il peut s'agir d'erreur ou de warning
Console.WriteLine("Résultat de la compilation");
foreach (Diagnostic diagnostic in emitResult.Diagnostics)
  Console.WriteLine(diagnostic.ToString());
```

Chargement dynamique de l'assembly

À ce stade, et s'il n'y a pas d'erreur de compilation, nous avons produit une assembly avec notre code.

L'utilisation de cette assembly peut être réalisée via un chargement dynamique, ou plus classiquement, par une référence depuis Visual Studio.

Dans le code suivant nous illustrons un chargement dynamique et la création d'une instance de la classe, enfin nous invoquons une méthode sur cette instance.

Le chargement de l'assembly est réalisé par la fonction Assembly.LoadFrom

```
var myAssembly = Assembly.LoadFrom("output.dll");
```

Une fois l'assembly chargée, on peut créer une instance de la classe dont on retrouve le nom via la fonction GetType. On voit également l'utilisation de notre interface pour manipuler l'instance de la classe sans connaitre au préalable son type. Ceci est possible, car nous avons défini cette interface dans une assembly qui est connu à la fois du programme appelant et également par le code et l'assembly que nous avons compilé.

```
Type typeMyClass = myAssembly.GetType("MyCompilation.ClassACompiler");
```

```
        IInterfaceAdresse myclass = (IInterfaceAdresse)Activator.CreateInstance
(typeMyClass);
```

Nous disposons maintenant d'une instance de la classe dont nous pouvons invoquée simplement des méthodes.

```
        myclass.Init("BERTON", "frederic.berton@capgemini.com");
        Console.WriteLine($"Nom {myclass.Nom} Mail {myclass.Mail}");
```

Voici le code complet :

```
using System;
using System.IO;
using System.Reflection;
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.Emit;

// Cet espace de nom, est défini dans une assembly qui contient
 l'interface IInterfaceAdresse
// Cette assembly doit être référencée statiquement par le programme
using Interfaces;

namespace Roseline
{
  public class CompilationExample
  {
    static public void Compile_And_Execute()
    {
      // Lecture dans un string du fichier avec le code de la classe
      string sCodeSource = File.ReadAllText(@"..\..\ClassACompiler.cs");

      // Parsing du code, creation du syntax tree
      SyntaxTree tree = CSharpSyntaxTree.ParseText(sCodeSource);

      // Creation des references sur les assembly utilisée
      string PathAssembly_System = typeof(object).Assembly.Location;
      string PathAssembly_Interfaces = typeof(IInterfaceAdresse).Assembly.
Location;
      var AssemblyRef_System = MetadataReference.CreateFromFile
(PathAssembly_System);
      var AssemblyRef_Interfaces = MetadataReference.CreateFromFile
(PathAssembly_Interfaces);

      // Compilation du code
      var CompilationOption = new CSharpCompilationOptions(Output
Kind.DynamicallyLinkedLibrary);
      var compilation = CSharpCompilation.Create("MyCompilation",
                                syntaxTrees: new[] { tree },
                              references: new[] { AssemblyRef_System,
                                        AssemblyRef_Interfaces },
                                options: CompilationOption);

      // Creation de l'assembly
      Console.WriteLine("COMPILATION");
      EmitResult emitResult = compilation.Emit("output.dll");

      // S'il y a des erreurs de compilation le dire
      if (emitResult.Success == false)
        Console.WriteLine("Compilation KO");
      else
        Console.WriteLine("Compilation OK");

      // Afficher les résultats de la compilation.
      // il peut s'agir d'erreur ou de warning
      Console.WriteLine("Résultat de la compilation");
      foreach (Diagnostic diagnostic in emitResult.Diagnostics)
        Console.WriteLine(diagnostic.ToString());

      Console.WriteLine("Appuyez sur une touche pour charger l'assembly");
      Console.ReadKey();

      // Chargement dynamique de l'assembly que l'on viens de compiler
      Console.WriteLine("Chargment de assembly : output.dll");
      var myAssembly = Assembly.LoadFrom("output.dll");

     // CReation d'une instance de la classe : MyCompilation.ClassACompiler
      Console.WriteLine("Creation d'une instance de la classe : MyCompilation.
ClassACompiler");
      Type typeMyClass = myAssembly.GetType("MyCompilation.ClassA
Compiler");
      IInterfaceAdresse myclass = (IInterfaceAdresse)Activator.Create
Instance(typeMyClass);

      // invocation des methodes avec l'interface contenu du programme appelant
      myclass.Init("BERTON", "frederic.berton@capgemini.com");
      Console.WriteLine($"Nom {myclass.Nom} Mail {myclass.Mail}");

      Console.WriteLine("Appuyez sur une touche pour quitter");
      Console.ReadKey();
    }
  }
}
```

Liens web sur Microsoft Roslyn

Nous donnons ici des liens sur des articles ou des documentations qui permettent une prise en main avancée du compilateur Microsoft Roslyn.

La documentation et les codes sources sont disponibles à l'adresse : https://github.com/dotnet/roslyn
Une partie de la documentation de Roslyn est disponible sur le site de Microsoft à l'adresse : https://docs.microsoft.com/en-us/dotnet/api/microsoft.codeanalysis.
Cette documentation couvre principalement les fonctionnalités d'analyse de code de Roslyn qui ne nous intéressent pas ici.
Pour trouver l'ensemble de la documentation nécessaire à l'utilisation de Roslyn (comme le compilateur par exemple) il faut se référer au site web : https://github.com/dotnet/roslyn/tree/master/docs qui contient la documentation complète et les codes sources de Roslyn.
Documentation sur l'analyseur syntaxique : https://docs.microsoft.com/fr-fr/dotnet/csharp/roslyn-sdk/get-started/syntax-analysis

Lien sur des articles traitants de Roslyn comme compilateur : https://www.tugberkugurlu.com/archive/compiling-c-sharp-code-into-memory-and-executing-it-with-roslyn

Le balisage sémantique : réconcilier l'humain et la machine en donnant du sens.

Traditionnellement une page, un contenu écrit qu'il soit présent dans un livre ou sur une feuille volante est destiné à être lu par un humain. On écrit pour être lu par l'humain.

Max Antoine BRUN

Fort de près d'une décennie dans le monde du SEO ; de nombreux succès notables pour des sites de toutes tailles, évoluant dans des secteurs divers tels que : l'immobilier ; la cosmétique ; l'assurance ; la banque ; la succession ; etc. ; j'ai décidé de me mettre à mon compte il y a maintenant quelques années et j'ai créé Search Artisan, agence SEO axée sur la conversion dédiée aux indépendants et aux Grands Comptes.

Avec l'avènement d'internet, un nouveau « consommateur » de contenu écrit est apparu, à savoir : le bot informatique. *Un robot d'indexation est un agent logiciel automatique ou semi-automatique qui interagit avec des serveurs informatiques.* Le travail du robot d'indexation consiste à « explorer » des pages web pour récupérer les informations qui s'y trouvent.

Une page web est donc, destinée à être « survolée » par l'humain ET par les bots informatiques (ou robots d'indexation) ; c'est sans doute la différence la plus notable avec la page physique qui elle, n'est lue que par l'humain.

Or, le robot n'étant pas encore capable de comprendre la « nature » d'une partie d'une page (s'agit-il d'un menu ? du contenu principal de la page ? du contenu annexe ?) encore moins leur signification dans un contexte (pragmatisme) ; les développeurs ont dû penser à des balises servant à donner des informations sur le contenu qu'elle encadre.

Ces balises dites sémantiques permettent ainsi de structurer les pages et de leur donner du sens pour les robots informatiques. Elles s'avèrent donc utiles pour le positionnement de ces pages. Les mots-clés compris dans certaines balises sont, en effet, mieux appréciés par Google.

Pour commencer, qu'est-ce qu'une balise sémantique ?

Au début du web, avant le développement des balises sémantiques, seules les balises et étaient utilisées pour donner des informations sur la structure d'une page aux robots d'indexation.

Ainsi, les balises sémantiques sont venues combler un vide sémantique quant à l'appréciation des bots lors de leur visite sur une page web. Il y a encore quelques années, il était impossible pour un robot d'indexation de comprendre la structure d'une page web. Les balises et alors utilisées ne donnant aucune information sur la nature des sections qu'elles entourent.

En effet, la balise sert à coder un bloc de page, un bloc pouvant comprendre un ou plusieurs paragraphes, une ou plusieurs images avec leurs légendes, etc.

La balise est utilisée pour spécifier une ligne ; une phrase par exemple. Le robot d'indexation des moteurs de recherche avec ces seules balises n'était donc pas capable d'isoler les différentes parties d'une page ou plus précisément d'en comprendre leur sens.

Il pouvait ainsi s'agir aussi bien d'un <menu>, d'une section <article> annexe ou encore d'une en-tête (<header>) sans que le robot ne les considère pas comme étant différents.

C'est de ce vide sémantique que sont venues combler les balises sémantiques. Elles permettent de structurer une page et donc d'envoyer une information qui est déjà présente pour l'humain ; mais qui ne l'était alors pas pour le robot. Elle ajoute ainsi de la cohérence en revoyant la même information aux deux types de visiteurs.

Quelles sont les différentes balises sémantiques et à quoi renvoient-elles ?

Les balises sémantiques peuvent donner des informations sur une section d'une page, mais également sur la partie d'un contenu. Elles permettent ainsi de structurer la page web :

- En-tête ;
- Menu de navigation ;
- Contenu principal ;
- Contenus indépendants ;
- Section ;
- Contenus annexes ;
- Pied de page.

Mais également le contenu présent sur la page :

- Titrage ;
- Liste à puce ;
- Mise en forme ;
- Citations ;
- Adresse ;
- Etc.

Nous allons dans cet article nous concentrer sur les balises sémantiques qui permettent de structurer une page web.

Les balises sémantiques qui permettent de structurer une page

Les balises sémantiques permettent de structurer une page web et ainsi de donner des informations cruciales aux robots ; notamment aux robots des moteurs de recherche. La pondération des mots (poids donné à un mot par les robots des moteurs de recherche) est plus ou moins importante selon l'emplacement auquel il se trouve.

Ainsi, un mot présent dans l'en-tête et dans la barre de navigation sera plus fortement pondéré qu'un mot présent dans un paragraphe de milieu de page.

Les balises sémantiques ont donc deux forts intérêts :

1. Donner une structure à la page ;
2. Accentuer la pondération du mot-clé (requête cible) pour les robots des moteurs de recherche.

Quelles sont ces balises qui permettent de structurer une page ?

La balise d'en-tête <header>

Il s'agit de la partie la plus haute de la page ; celle qui sera vu en premier par l'internaute. Communément, on y trouve le logo, le menu de navigation (avec l'ensemble des liens qui y sont présents). Peuvent, également, se trouver dans cette zone :

- Le panier ;
- Le bouton de connexion ;
- Le bouton d'inscription ;
- etc.

À noter que la présence de plusieurs balises <header> n'est pas du tout interdite, voire non pénalisante. Mais, en abuser ne serait pas une bonne idée, car elle créerait de la confusion dans la compréhension de la structure de votre page par les robots.

Le ou les menus de navigation <nav>

La balise <nav> permet de préciser aux robots qu'il s'agit d'un menu de navigation. Il peut s'agir du menu principal du site ; comme d'un menu présent dans le contenu de la page.

Le contenu principal avec <main>

La balise <main> est importante, car elle permet de spécifier à Google la partie de votre page qui contient le contenu principal de la page ; celle qui est spécifique à la page. C'est cette partie qui est la plus importante pour le référencement de vos pages. Celle-ci doit idéalement être utilisée à une seule reprise dans votre page. Vous comprendrez donc qu'il ne peut pas y avoir plusieurs parties principales, ce serait très perturbant pour le bot.

Segmentation du contenu principal <section>

Votre partie principale vous semble trop dense, ne vous inquiétez pas ! Vous pouvez utiliser la balise <section> pour la segmenter en plusieurs sous-parties. *Chaque balise <section> indique aux robots d'indexation une variété de contenu différente sur une même page*. En référencement, je conseille l'utilisation de la balise <section> pour segmenter la partie principale par sémantique.

Disons que vous rédigez un article sur l'importance du choix du mot-clé pour convertir le trafic provenant du levier « SEO » ; vous pourriez segmenter votre contenu principal (balise <main>) en 3 balises <section> :

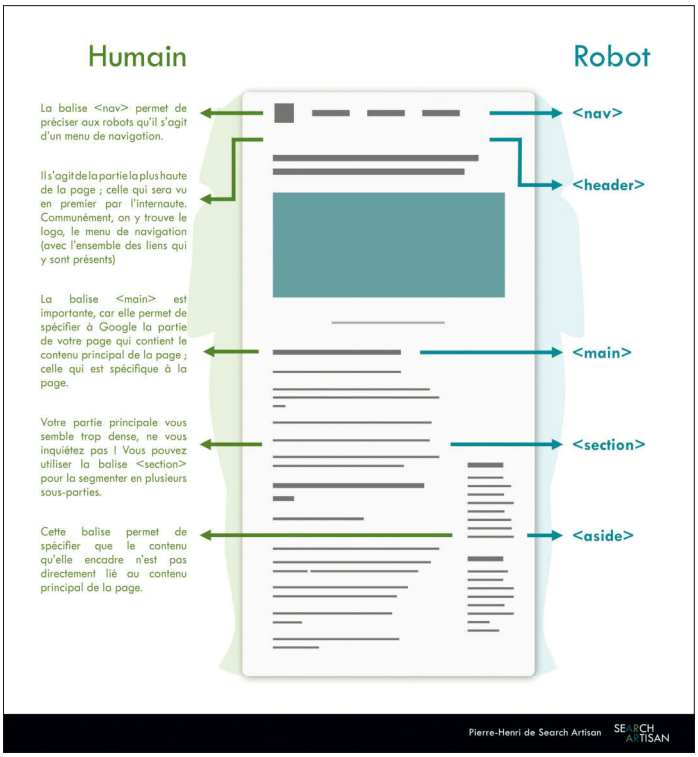
- <section> Leviers SEO
- <section> choix du mot-clé ;
- <section> conversion.

La première section s'articulerait autour de la présentation du SEO (qu'est-ce que le SEO ? Optimiser un site, comment faire ? etc.) La deuxième section s'organiserait autour des mots-clés (qu'est-ce qu'un mot-clé ? Quels sont les composants possibles d'un mot-clé ? Comment choisir son mot-clé ? etc.). Enfin, la troisième section parlerait de la conversion issue d' un site web (comment mesurer le trafic SEO ? Quelles sont les données à privilégier dans le tracking de la conversion du trafic SEO…).

Le contenu annexe avec <aside>

Cette balise permet de spécifier que le contenu qu'elle encadre n'est pas directement lié au contenu principal de la page. Il s'agit d'un contenu annexe. Il peut s'agir d'article similaire ou de bandeau publicitaire par exemple. Cette section prend souvent la forme d'une sidebar (barre latérale).

Les mots compris dans la balise <aside> sont moins valorisés par Google, car ils ne sont pas directement liés à l'article. Ils sont moins importants pour la compréhension de celui-ci par l'humain ET donc par les robots d'indexation.



Conclusion

Il y a encore quelques années seules les balises et étaient utilisées pour donner des informations sur la structure d'une page aux robots d'indexation.

Ces informations étant très partielles, elles ne suffisaient pas à permettre aux bots de comprendre précisément la structure d'une page web. Pour remédier à cette "carence" sémantique, les développeurs ont donc créé les balises sémantiques.

Les balises sémantiques ont une importance cruciale pour la compréhension d'une page web par les robots d'indexation. Sans celles-ci le robot est incapable de structurer celle-ci. En effet, il lui est impossible autrement de déterminer quel est l'en-tête, le menu, le contenu principal ou encore le contenu annexe d'une page.

La compréhension de la structure de la page joue un rôle en SEO, car elle permet de "scorer" les différentes sections d'une page. Ainsi, un mot présent dans le <header>, dans la <nav> ou encore dans le <main> aura un "score sémantique" plus élevé qu'un mot présent dans une section <aside>. En cela, le balisage sémantique est un pas solide et ferme vers une lecture, une compréhension similaire d'une page web par l'humain et par le robot.

Si l'article vous a plu, n'hésitez pas à visiter le site de mon agence SXO : Search Artisan (https://www.search-artisan.com/). Search artisan est une agence SXO, axée sur la conversion, dédiée aux entrepreneurs et aux Grands Comptes.

Ressources web

https://fr.wikipedia.org/wiki/Web_s%C3%A9mantique#:~:text=Le%20Web%20s%C3%A9mantique%2C%20ou%20toile,Resource%20Description%20Framework%20(RDF)
https://www.enssib.fr/le-dictionnaire/web-semantique
https://www.lafabriquedunet.fr/seo/articles/mettre-en-place-installer-balisage-semantique-schema-org/
https://www.fun-mooc.fr/fr/cours/web-semantique-et-web-de-donnees/
https://oseox.fr/referencement/balisage-semantique.html

Ressources papier

Référencement Google mode d'emploi – Olivier Andrieu, Paris, Éditions EYROLLES, "Collection", juin 2021, 262.

Le web sémantique – Fabien Gandon, Catherine Faron-Zucker, Olivier Corby, Paris, Dunod, "InfoPro", mai 2012, 224.

NextJS, le marketing au service des développeurs

Ne partez pas tout de suite ! Je sais, je vais vous parler de marketing alors que nous préférons tous laisser ceci de côté en tant que développeurs. Ne vous inquiétez pas, je vais quand même vous parler technique ! Le sujet du jour est NextJS !

Loïc GUILLEBEAU

J'ai 21 ans et pendant mon cursus d'ingénieur en informatique à EPITA, j'ai créé ASIG-Tech que je dirige depuis maintenant 4 ans.

J'ai commencé à développer au collège avec Java et le fameux jeu Minecraft. Un début plutôt créatif ! Au fil des années, j'ai appris d'autres langages.

Je lance un nouveau projet qui s'appelle Accelerio. Nous utilisons NextJS pour créer une plateforme de réservation en ligne pour les sports mécaniques.

Vous retrouvez mes projets ici :

https://asig-tech.com

https://accelerio.com

Les équipes de NextJS ont pensé leur framework afin de vous faciliter le développement et la mise en production. Cela va passer par l'architecture de vos fichiers, l'intégration de l'API, la manière de récupérer vos données et la gestion de votre balise < head > dans chaque page ! Et cela en vue de servir l'une des choses les plus importantes d'internet : le SEO (Search Engine Optimization) ! Nous appellerons notre projet : Mecanique Venture.

En quoi l'architecture de mes fichiers influe sur le SEO ?

En règle générale, quand tu es développeur, il est important d'avoir une arborescence de fichiers structurée et compréhensible par tous. Oui, il arrive que ton collègue en marketing décide de regarder à quoi ça ressemble la « Programmation ».

NextJS vous oblige à avoir une structure très précise afin d'être prêt à envoyer en production très rapidement. Quand vous allez initialiser votre projet Next avec `npx create-next-app@latest OU yarn create next-app`, vous aurez déjà l'architecture construite, il vous suffira de l'enrichir.

Vous retrouverez un dossier **styles/** qui regroupe tous vos fichiers CSS/SCSS. Le dossier **public/** qui permet de mettre à disposition de l'utilisateur des images, des pdfs, des vidéos ou d'autres types de contenu. Il a le même but que le dossier public propre à un projet Express/Node, à savoir servir des fichiers statiques. Enfin, le plus important demeure le dossier **pages/** avec un sous-dossier **api/** et un fichier index.js. Ce fichier contient l'exemple de départ et constitue la page centrale qui est affichée lorsqu'un utilisateur se rend à la racine (root) de votre application.

Ce dossier est essentiel, c'est lui qui va gérer toute l'arborescence de votre site web !

Dans notre projet Mecanique Venture. On aura donc plusieurs pages. Notre page d'accueil pour donner envie à nos utilisateurs d'acheter nos pièces, également une page de blog avec tous nos articles pour attirer du monde sur notre site par le biais de questions souvent posées. Cette page blog sera liée à une page template d'article qui sera générée automatiquement grâce à son slug. Il ne faut pas oublier, que nous sommes un site e-commerce, nous aurons un catalogue de tous nos produits et des catégories par produit. De la même manière que le blog à une page template d'article, ici nous aurons une page template d'un produit.

Il nous faut aussi un tunnel d'achat avec un panier, un formulaire de paiement et un formulaire de livraison. Puis pour rassurer nos clients, nous rajoutons une page sur notre histoire et une page pour nous contacter.

Vous pouvez visualiser cette architecture en consultant l'image ci-dessous. **Figure 1.** Cette architecture va aussi vous permettre de créer votre sitemap beaucoup plus simplement. Il existe des packages comme next-sitemap qui vont se baser sur votre build pour créer le fichier sitemap.xml. Ainsi cette phase prend en compte votre architecture du dossier page, d'où la nécessité de bien le structurer. Il y a aussi une partie importante liée à l'api, que nous verrons plus loin.

Notez que si votre application n'est guère destinée à un usage en interne et cible un large public, alors il sera très important de créer ce fichier pour que Google trouve votre site et vous référence correctement. Vous devrez aller sur Google Search Console pour leur envoyer votre nouveau sitemap via l'url de votre site !

Comment récupérer mes données pour le SEO ?

Maintenant que vous avez une architecture au top, il faut maintenant penser à récupérer les données pour les mettre en forme sur votre site. Pour cela, il faut savoir comment vous allez gérer vos informations. Il se peut que cela soit géré du côté serveur (Server-side rendering - SSR) ou du côté client (Client-side rendering - CSR). Avec SSR, l'API est "atteinte" avant que votre page termine de se charger, et cela, malgré l'effet de délai qu'on peut constater avant l'affichage. Tandis qu'avec CSR, votre fetch sera exécuté après que la page s'est chargée. Nous pouvons aussi gérer cela de manière statique, c'est ce que nous appelons : Static-site generation (SSG). Cette technique permet de récolter toutes les données au

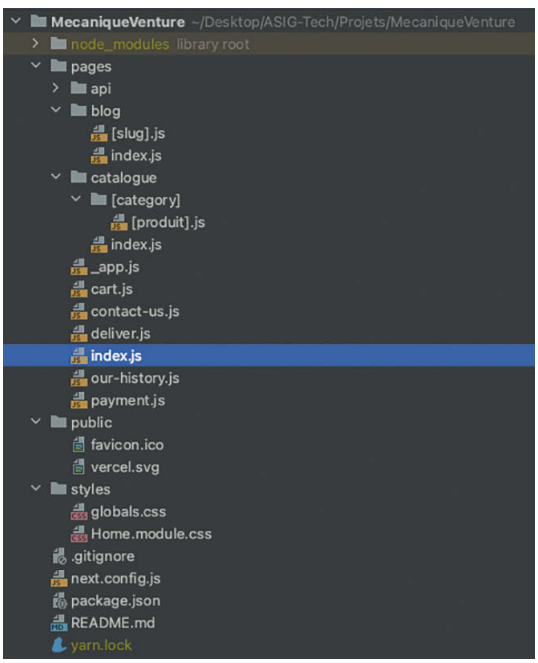


Figure 1

moment du build et d'afficher une donnée directement sur le site sans appeler le serveur. Ceci est la meilleure technique pour le SEO, car Google ne laisse pas le temps à votre site d'appeler votre serveur. De plus, le SSG fait au moment du build un pré-rendu qui génère un fichier HTML avec vos données pour chacune de vos pages. Puis, il reste une dernière technique que nous appelons : Incremental Static Regeneration (ISR). Certains diront qu'elle ressemble à SSG et d'autres que c'est plutôt une combinaison entre SSG et SSR ; celle-ci permet, une fois que le site est build et mis en production, de rafraîchir les données au bout d'un intervalle défini. Cela peut être utile lorsque vous créez une fiche produit que vous mettez à jour à l'aide d'un back-office externe. Aujourd'hui, NextJS gère ces quatre façons de récupérer les données. Pour cela, il faut appeler des fonctions spécifiques à ce framework. Pour la première manière, le server-side rendering se fait grâce à la fonction asynchrone **getServer SideProps()**. Cette fonction
s'applique, seulement, dans un fichier inclus dans le dossier **pages/**. Dans les paramètres de la fonction vous allez pouvoir récupérer tout le contexte de votre page. Par exemple, vous allez pouvoir accéder à l'url de la page, pour récupérer des données en fonction d'un slug. Voici un exemple de comment cela fonctionne :

```
export async function getServerSideProps({params}) {
  const result = await axios.get('/api/v1/product/slug/' +    params.slug)

  return {
    props: {
      product: result.data
    }
  }
}
```

La spécificité de CSR est qu'il peut s'exécuter au sein vos composants appelés. Cette technique est plutôt utile quand vos données sont amenées à changer assez souvent et que le SEO ne fait pas partie de vos préoccupations. De plus, NextJS vous offre un React Hook qui est le SWR, il permet de récupérer vos données, de les passer directement en cache, de les rafraîchir et les mettre à jour à intervalles réguliers.

```
const fetcher = url => axios.get(url).then(res => res.data)

function App ({slug}) {
  const { data, error } = useSWR(`/api/v1/product/slug/${slug}`, fetcher)

  if (error) return <div>{error}</div>
  if (!data) return <div>Chargement...</div>

  //...
}
```

Passons, maintenant, au cœur de notre sujet, à savoir la récupération des données pour le SEO. Comme nous l'évoquions précédemment, la meilleure technique reste la génération statique (SSG). Afin d'y arriver vous allez devoir appeler deux fonctions dans votre fichier : **getStaticPaths()** et **getStaticProps().** La première va permettre de récupérer une liste d'URL possibles (de chemins) en fonction des routes dynamiques associées à votre page. Ici, nous allons récupérer tous les slugs de nos produits. Cela permettra au site de reconnaître les URL qui doivent être appelées en statique. Elle va ensuite appeler votre API pour récupérer l'information. C'est ici qu'intervient la deuxième fonction, vous allez faire votre appel API dedans puis gérer vos données pour les envoyer à votre composant react, qui recevra **product** comme prop. Notons que tout cela s'exécute au moment du « build » ce qui permet de générer une page html pour chacune des URL trouvées dans votre première fonction.
De plus, pour trouver cette liste d'url, vous pouvez appeler votre base de données, votre CMS ou votre système de fichiers afin de la récupérer dynamiquement. Ainsi, cela vous permet de ne pas écrire à la main chaque url possible.
Code complet sur programmez.com & github

Pour finir, nous avons la technique de la régénération statique de vos pages (ISR). Elle vient parfaitement compléter le SSG. Elle nous permet de privilégier la génération statique de nos pages, sans avoir à reconstruire tout le site à chaque fois. C'est très pratique quand votre collègue veut changer une image sur la fiche produit. Cela permet de vous laisser tranquille sans avoir à mettre une nouvelle fois en production tout votre site. Ici, nous faisons appel aux mêmes fonctions que nous avons vu juste avant, mais avec deux paramètres différents. Tout d'abord, dans le **getStaticProps()** vous allez pouvoir retourner un paramètre **revalidate**. Il permettra de dire au site de se régénérer à des intervalles réguliers lorsqu'une requête est effectuée sur la page. Dans notre exemple, la valeur sera de 60 secondes. Ensuite, il reste la fonction **getStaticPaths()** à laquelle nous pouvons retourner le paramètre **fallback**. Ce dernier peut prendre trois valeurs : false, true ou blocking. Cela va permettre de ne pas générer toutes les pages dynamiques lors du build initial et de produire un effet sur la façon dont la fonction **getStaticProps()** va se comporter. Si le paramètre est à false la fonction sera appelée au moment du build et une page 404 sera retournée dès lors qu'il n' y aura aucune correspondance avec les chemins retournés par **getStaticPaths()**, ce type d'erreur permet de mieux contrôler les pages qui fonctionnent, on privilégiera ce mode lorsque l'on souhaite générer l'ensemble de nos routes dynamiques au moment du build. Si nous passons le fallback à true, la fonction **getStaticProps()** sera exécutée en arrière-plan pour tous les chemins qui n'auront pas été générés au moment du build, une page de secours (fallback) s'affichera le temps de la construction de la page finale, cela est très appréciable pour un site e-commerce dans lequel vous avez généré vos 100 meilleures pages au moment du build et envisagez de laisser les autres se générer lors de la première visite. Enfin, avec la valeur blocking, aucune page de secours (fallback) n'apparaîtra, il faudra attendre l'affichage de la page, ce qui sera marqué par une temporisation de quelques secondes à l'image d'un rendu côté serveur (SSR), ce qui n'est pas forcément recherché ici.

```
export async function getStaticPaths() {
  const {data: result} = await axios.get('/api/v1/product/slug')
  let products = []
```

```
  result.map(data => {
    products.push({params: {slug: data.slug}})
  })

  return {
    paths: products,
    fallback: true
  }
}

export async function getStaticProps({params}) {
  const result = await axios.get('/api/v1/product/slug/' + params.slug)

  return {
    props: {
      product: result.data
    },
    revalidate: 60
  }
}
```

Vous savez tout sur la manière de gérer vos données sur NextJS. Maintenant, c'est à vous de choisir ce que vous préférez selon le contexte de votre prochain projet !

Comment gérer ma balise < head > ?

La gestion de votre balise < head > n'est pas la partie la plus compliquée, mais reste non négligeable en SEO. C'est ici que Google va lire toutes vos métadonnées et que vous allez pouvoir mettre en place OpenGraph ! Dans le cadre de cet article, nous allons nous intéresser sur la manière d'enrichir votre balise < head > dynamiquement.

Dans cette partie-là, nous retrouvons les balises meta classiques que vous pouvez avoir sur un site écrit en HTML pur. Il y a la balise meta pour la description, la gestion des robots, les mots-clés de votre site, le robot google et bien d'autres. En plus de cela, il est important d'avoir une balise < title > et un lien canonique pour Google. En revanche, vous n'allez pas écrire à la main chaque page HTML de vos dix mille produits. Cela va vous amener à utiliser toutes les données que vous allez récupérer, comme nous l'avons vu précédemment pour la mettre dans votre balise < Head > de NextJS. Il faut savoir que dans ce framework nous pouvons gérer une balise < Head > par fichier a contrario de React où il n'y a qu'un fichier HTML. Même si vous utilisez React Helmet, cela ne permet pas une génération de votre page web en statique, donc le référencement sera de moins bonne qualité. Revenons sur notre balise < Head >, vous avez maintenant toutes vos données et vous allez pouvoir les ajouter à vos balises. Cela se gère de la même manière que si vous intégrez une variable dans votre code, rien de plus simple.

La gestion des balises pour OpenGraph est semblable, mais il faut avoir les mots-clés spécifiques à cette technologie. Nous allons retrouver les propriétés « og : type », « og : url », « og : title », « og : description » et « og : image ». Ces propriétés représentent l'une des configurations les plus simples, il en existe d'autres. OpenGraph vous permet d'afficher des informations lorsque vous partagez votre page sur différents réseaux sociaux.

Une astuce intéressante est de prévoir lors de votre architecture de la base de données des champs spécifiques à votre SEO. Cela vous permettra de générer votre balise < Head > de manière beaucoup plus simple plutôt que de devoir retravailler chaque variable de votre produit ou article.

Code complet sur programmez.com & github

Toutes ces informations permettront à Google de vous adorer !

Conclusion

NextJS facilite le développement pour un usage marketing en ayant une structure pensée pour la mise en production. Fini de se casser la tête avec React Helmet pour le SEO et la manière de récupérer ses données. Ici, nous voyons la fracture entre React et NextJS. React, lui sera plus utilisé dans le cadre d'une application interne à l'entreprise, alors que NextJS sera plus utilisé pour une application web sur le marché, avec un besoin de référencement !

Comment créer mon API sur NextJS ?

Un petit bonus, cela ne fait jamais de mal ! Ce qui est intéressant aussi avec NextJS, c'est sa gestion de l'API directement sur le framework. Il n'est pas obligatoire de passer par un backend en NodeJS avec Express ou autres. Encore une fois, nous retrouvons dans cette partie l'importance de notre architecture ! Elle est directement liée à la route de votre API. Dans notre exemple, nous avons une API pour récupérer toutes les marques de voitures. Ce qui donne une architecture de dossier comme celle-ci « /api/v1/make/index.js ». Ensuite vous retrouverez une fonction classique comme dans NodeJS, avec en paramètres la requête et la réponse à renvoyer. Dans mon exemple, j'utilise prisma pour récupérer mes données ! C'est un ORM très pratique. Pour finir, il vous suffira d'appeler votre url : « /api/v1/make ». Cela peut se faire avec fetch() ou axios comme vous préférez.

```
export default async function handle(req, res) {
  try {
    const result = await prisma.vehicles.findMany()
    res.status(200).json(result)
  } catch (e) {
    res.status(500).json({ error: e })
  }
}
```

Et voilà, vous savez gérer l'API sur NextJS, facile non ?

Ressources

https://github.com/LoicGuillebeau/MecaniqueVenture
https://nextjs.org/docs/basic-features/data-fetching/overview
https://nextjs.org/docs/routing/dynamic-routes
https://nextjs.org/docs/api-routes/introduction
https://nextjs.org/docs/basic-features/pages
https://www.prisma.io/docs/
https://ogp.me/
https://developers.google.com/search/docs/advanced/crawling/special-tags
https://developers.google.com/search/docs/advanced/sitemaps/build-sitemap
https://nextjs.org/docs/api-reference/data-fetching/get-server-side-props

Une gestion efficace des configurations Python avec Hydra

Thibaud Vienne
est senior data scientist et le fondateur de Clevergence, une fintech spécialisée dans le développement de modèles de risques financiers pour les professionnels de l'investissement. Il est également professeur vacataire en machine learning au sein de l'université Paris-Dauphine.

Hamza Aziz
est machine learning engineer. Actuellement, il implémente des outils MLops au sein de Clevergence.

Mi-recherche, mi-dev, la réalisation d'applications, notamment machine learning, fait émerger de nombreux challenges dont sa reproductibilité, son déploiement et son suivi. Dans le cas du machine learning, une réponse à ces difficultés est le "MLOps", un ensemble de pratiques qui visent à développer, déployer et maintenir des modèles de machine learning de façon fiable et efficace.

Avec l'approche MLOps, le cycle de vie d'un projet machine learning n'est plus linéaire, mais prend la forme d'un processus itératif qui comprend l'émergence de fonctionnalités, le développement de celles-ci (de la collecte de données à l'entraînement du modèle) puis son déploiement en production. Cette boucle de rétroaction permet d'obtenir rapidement un feedback pour continuer à faire évoluer l'application. **Figure 1**
Ces nombreux allers-retours entre développement et déploiement engendrent de nombreuses expérimentations et essais. Cela pousse ainsi bien souvent l'équipe data science à utiliser des fichiers de configuration pour paramétrer l'application. Par exemple, pour pouvoir tester une nouvelle structure de modèle ou encore ajouter / supprimer une variable explicative (c'est-à-dire une variable utilisée pour entraîner le modèle).
Cela étant, ces fichiers de configuration échouent bien souvent à concilier robustesse du code et flexibilité. Hydra, un module python développé par Facebook AI Research, permet de dépasser ces difficultés.

Argparse et les fichiers de configuration

Avec Python, il existe plusieurs façons de gérer ces fichiers de configuration. Les plus communément utilisées sont le passage d'arguments par la ligne de commande ou bien le recours à des fichiers de configuration.

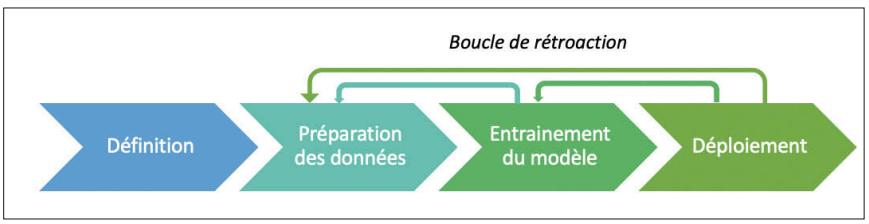
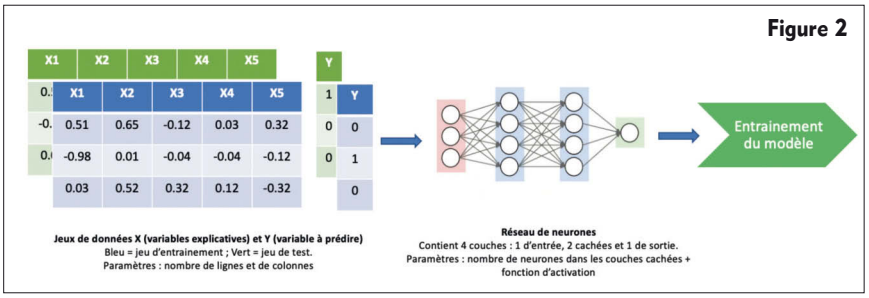


Figure 1- *Le déploiement de l'application permet le feebdack utilisateur et donc l'émergence de nouvelles actions correctives (besoin de plus de données, nouvelles variables explicatives, etc.).*

Figure 2



Avec la ligne de commande, les différents paramètres sont passés à l'interpréteur python lors de l'exécution du script au travers de "*flags*". Ceux-ci peuvent ensuite être récupérés et retraités convenablement avec certains modules tels que **argparse**.
Dans l'exemple ci-dessous (cf. Annexe, *répertoire GitHub*), nous allons implémenter une application simple de machine learning comprenant la création de données aléatoires (N lignes et M colonnes) puis la création et l'entraînement d'un réseau de neurones avec Keras. Cette application contient de nombreux paramètres tels que le format des données d'entrée ou l'architecture du réseau de neurones. **Figure 2**
La récupération des paramètres avec Argparse se fait de la façon suivante :

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("—n_rows_train", type=int, help="number of rows in training set")
parser.add_argument("—n_rows_test", type=int, help="number of rows in test set")
parser.add_argument("—n_columns", type=int, help="number of input features (columns)")
parser.add_argument("—n_neurons_1", type=int, help="number of neurons in the first layer.")
parser.add_argument("—activation_1", type=str, help="activation function in first layer.")
parser.add_argument("—n_neurons_2", type=int, help="number of neurons in the second layer.")
parser.add_argument("—activation_2", type=str, help="activation in the second layer.")
parser.add_argument("—learning_rate", type=float, help="learning rate")
conf = parser.parse_args()
```

Passage et récupération d'arguments avec Argparse.

Avec ces paramètres, le code devient alors générique. L'utilisateur n'a alors qu'à les modifier pour obtenir le comportement souhaité de l'application. Avec Argparse, le passage des arguments se fait à l'appel du script.

```
> python main_with_argparse.py —n_rows_train 1000 —n_rows_test 200 —n_columns 5 —n_neurons_1 10 —n_neurons_2 5 —activation_1 relu —activation_2 tanh —learning_rate 0.05
```

Passage des paramètres lors de l'appel du script

De plus, Argparse permet aussi de pouvoir assigner des valeurs par défaut ou d'introduire des arguments optionnels. Malgré ces fonctionnalités, argparse rencontre plusieurs limitations :

- **Passage d'arguments laborieux,** dès que l'application dépasse quelques paramètres. Il faut à la fois modifier la ligne de commande et le parsing des arguments dans le script. Ce qui peut vite devenir fastidieux ;
- **Absence de hiérarchie,** qui empêche de regrouper les paramètres en groupe. Ce qui est pourtant utile si ceux-ci agissent sur un même composant (par exemple l'architecture du modèle).

Peut-on faire autrement qu'avec la ligne de commande ? Oui, en utilisant des fichiers de configuration. Cela consiste à stocker nos paramètres dans un fichier puis à les lire dans l'application. Ci-dessous, un exemple réalisé avec un simple dictionnaire Python créé dans le fichier conf.py, puis importé et utilisé dans le main :

```
src
├── config
│   └── conf.py
└── main_with_conf.py
```

```
conf = {
 # data
 "n_rows_train": 1000,
 "n_rows_test": 200,
 "n_columns": 5,

 # model
 "n_neurons_1": 10,
 "activation_1": "relu",
 "n_neurons_2": 5,
 "activation_2": "tanh",
 "learning_rate": 0.05
}
```

fichier conf.py : les paramètres sont encapsulés dans un dictionnaire.

```
def main(conf):

 # create fake data
 x_train = np.random.randn(conf["n_rows_train"], conf["n_columns"])
 y_train = np.random.choice([0, 1], size=(conf["n_rows_train"],), p=
[1./2, 1./2])
 x_test = np.random.randn(conf["n_rows_train"], conf["n_columns"])
 y_test = np.random.choice([0, 1], size=(conf["n_rows_test"]), p=
[1./2, 1./2])

 ...

if __name__ == "__main__":

 # test 2 : configuration file
 from src.config.conf import conf
 main(conf)
```

fichier main_with_conf.py : import de la configuration puis appel des paramètres à partir des clés du dictionnaire.

Cette approche apporte beaucoup plus d'efficacité que la méthode "argparse". Ainsi, il n'y a pas besoin de parser les paramètres comme précédemment. On peut ainsi ajouter ou modifier un paramètre facilement. De plus, il devient possible de passer n'importe quel type d'objet en paramètre tel que des listes, dictionnaires, voire même des fonctions ou des classes, etc.

Bien que plus flexible qu'Argparse, chaque changement de paramètre nécessite une modification du fichier de configuration. Quand le fichier est versionné (par exemple avec Git) et couplé avec une approche de déploiement continu (CI/CD), chaque modification devra entraîner un commit. Ce qui pourra se révéler peu pratique lorsque l'on souhaite réaliser de nombreuses expérimentations.

Hydra, une solution plus adaptée

Hydra permet de résoudre l'ensemble des difficultés évoquées précédemment. Il s'agit d'un module python open source développé par Facebook AI Research (FAIR). Avec Hydra, les fichiers de configuration sont écrits au format yaml, puis sont chargés dans l'application. Il est également possible d'utiliser la ligne de commande (comme pour Argparse) afin d'outrepasser le fichier de configuration.

L'installation de hydra est simple pour tout système d'exploitation et se fait avec le gestionnaire pip :

```
> pip install hydra-core
```

Le format yaml utilisé par Hydra offre l'avantage de pouvoir hiérarchiser les paramètres, comme dans l'exemple ci-dessous, avec les sections "data" et "model". Le chargement de la configuration se fait dans le *main* avec l'ajout d'un simple décorateur hydra qui spécifie le chemin du fichier de configuration (avec le "*config_path*" et le "*config_name*" respectivement le chemin du répertoire et le point d'entrée). Le fichier de configuration est ensuite chargé dans un dictionnaire qui devient alors accessible, ici avec la variable **conf**.

```
src
├── config
│   └── conf.py
└── main_with_conf.py
```

```
data:
 n_rows_train: 1000
 n_rows_test: 200
 n_columns: 5

model:
 n_neurons_1: 10
 activation_1: "relu"
 n_neurons_2: 5
 activation_2: "tanh"
 learning_rate: 0.05
```

conf.yaml : le format yaml permet de facilement hiérarchiser les paramètres.

```
import hydra

@hydra.main(config_path="config", config_name="conf")
def main(conf):

 # create fake data
 print(conf)
 x_train = np.random.randn(conf.data.n_rows_train, conf.data.n_
columns)
 y_train = np.random.choice([0, 1], size=(conf.data.n_rows_train,)
```

```
, p=[1./2, 1./2])
 x_test = np.random.randn(conf.data.n_rows_train, conf.data.n_columns)
 y_test = np.random.choice([0, 1], size=(conf.data.n_rows_test,), p=
[1./2, 1./2])

                                    ...

if __name__ == "__main__":

 # test 4 : hydra
 main()

                                  output

{'data': {'n_rows_train': 1000, 'n_rows_test': 200, 'n_columns': 5},
'model': {'n_neurons_1': 10, 'activation_1': 'relu', 'n_neurons_2': 5,
'activation_2': 'tanh', 'learning_rate': 0.05}}
```

main_with_hydra.py : utilisation du décorateur Hydra pour appeller la configuration.

Avec Hydra, il devient également possible d'alterner facilement entre plusieurs expérimentations. Nous pouvons, par exemple, construire des sous-configurations pour représenter deux modèles différents "*small*" et "*large*". La structure des répertoires de notre application ressemble alors à ce qui suit :

Dans le fichier principal **conf.yaml**, il est alors possible de basculer facilement d'un fichier de configuration à l'autre en assignant au paramètre **model**, la valeur "*model_small*" ou "*model_large*".

```
src
├── config
│   └── conf.yaml
│   ├── models
│   │   └── model_small.yaml
│   │   └── model_large.yaml
└── main_with_hydra.py
```

```
n_neurons_1: 10
activation_1: "relu"
n_neurons_2: 5
activation_2: "tanh"
learning_rate: 0.05
```

```
n_neurons_1: 500
activation_1: "relu"
n_neurons_2: 100
activation_2: "tanh"
learning_rate: 0.1
```

À gauche, l'architecture du projet
Au milieu - model_small.yaml : un petit réseau de neurones.
À droite - model_large.yaml : un réseau de neurones plus large.

```
defaults:
- model: model_large

data:
n_rows_train: 1000
n_rows_test:  200
n_columns: 5
```

conf.yaml : la section "model" est maintenant composée à partir du sous-fichier de configuration model/model_large.yaml

```
import hydra

@hydra.main(config_path="config", config_name="conf")
def main(conf):
```

```
 # create fake data
 print(conf)
 x_train = np.random.randn(conf.data.n_rows_train, conf.data.n_
columns)
 y_train = np.random.choice([0, 1], size=(conf.data.n_rows_train,),
p=[1./2, 1./2])
 x_test = np.random.randn(conf.data.n_rows_train, conf.data.n_
columns)
 y_test = np.random.choice([0, 1], size=(conf.data.n_rows_test,),
p=[1./2, 1./2])

                                    ...

if __name__ == "__main__":

 # test 4 : hydra
 main()

                                  output

{'model': {'n_neurons_1': 500, 'activation_1': 'relu', 'n_neurons_2':
100, 'activation_2': 'tanh', 'learning_rate': 0.1}, 'data': {'n_rows_train':
1000, 'n_rows_test': 200, 'n_columns': 5}}
```

main_with_hydra.py : le modèle "large" a bien été pris en compte.

Hydra permet ainsi d'écrire, hiérarchiser et basculer facilement entre plusieurs configurations. Ce qui rend les expérimentations flexibles et reproductibles. La ligne de commande peut également être utilisée pour outrepasser le fichier yaml. Enfin, Hydra possède de nombreuses autres fonctionnalités parmi lesquelles :

- **La gestion des formats**, des différents paramètres passés en configuration ;
- **Le multi-run**, pour exécuter l'application avec plusieurs paramètres en même temps. Par exemple avec la ligne de commande : > *python main_with_hydra.py —multirun n_rows_train=1000,2000 ;*
- Et plein d'autres encore, nous vous invitons à consulter la documentation officielle pour plus de détails.

Conclusion

Hydra permet d'adresser efficacement les limitations de Argparse et des fichiers de configurations. Ce qui en fait un outil de choix pour les applications data science nécessitant de nombreuses expérimentations.

Annexes

Repo GitHub : https://github.com/tvienne/demo-hydra
Wikipedia : https://fr.wikipedia.org/wiki/MLOps
Documentation Hydra : https://hydra.cc/docs/intro

Comprendre l'UI Design sous Figma

Figma est LE logiciel pour faire du Design d'interface ces dernières années. Son succès lui vient notamment d'une interface web ou logicielle qui se rapproche de plus en plus du développement web en utilisant des notions CSS et un jargon connu : space-between, containers, padding, margin, position absolute… (Presque) tout y est !

Margaux Membré

Passionnée de graphisme et de visuels depuis une bonne dizaine d'années, j'ai commencé à m'intéresser au Design avec Photoshop. J'ai été Développeuse Front-End pendant quelques mois en 2019/2020, et je me suis reconvertie dans l'UX/UI Design fin 2020.

Je travaille actuellement à La Redoute dans le cadre de mon alternance, autant sur l'interface web que sur l'interface mobile, et sur le Design System.

Vous pourriez avoir besoin de comprendre sa mécanique, notamment si vous envisagez d'approfondir vos connaissances dans l'UI Design. Nous verrons les bases de l'UI Design et la prise en main de Figma. Il se peut que dans votre quotidien, vous puissiez avoir besoin de comprendre sa mécanique. C'est parti !

Revenons aux fondamentaux : l'UI Design

UI signifie User Interface. Mais ce n'est pas simplement le fait de faire des carrés et des ronds de couleur, loin de là !

Le rôle de l'UI Designer est de créer de la hiérarchie visuelle, grâce à des rapports de proportion et l'utilisation de l'espace blanc (white space), afin d'aider l'utilisateur à remplir l'objectif qu'il s'est fixé en arrivant sur le site. Il est donc le guide de l'utilisateur dans son parcours sur le site.

L'UI Design ne s'apprend jamais seul dans les écoles (en tout cas, à ma connaissance). Et ça se comprend ! Un bon UI Designer fait attention aux attentes, besoins et motivations de l'utilisateur. Il le comprend et essaie de trouver des solutions à ses frustrations pour qu'il ait une expérience agréable durant tout son parcours sur l'interface ou le service. L'UI Designer a donc des bases en UX Design.

L'UX Designer utilise des méthodes comme les entretiens, les questionnaires, les cartes d'empathie, etc. Il est souvent amené à tester l'ergonomie de son travail via les tests utilisateurs, et il est parfois adepte du *Design Thinking(1)*.

Benchmark

L'UI Designer commence souvent son travail par un benchmark des concurrents de son entreprise, c'est-à-dire qu'il va regarder les fonctionnalités (*features)* créées par d'autres acteurs… Qu'ils soient en lien direct avec le secteur de l'entreprise ou non !

Prenons un exemple : je travaille chez **La Redoute**, où nous vendons du Prêt-à-porter, de la Maison et beaucoup d'autres choses.

Je peux être amenée à "benchmarker" (barbarisme du métier) un site comme **Adidas** et **Zalando** qui sont tous les deux des e-commerces de l'habillement/sport. Je peux aussi benchmarker des sites comme **Leroy Merlin** ou **Castorama** **qui ne vendent pas du tout le même type de produits. Ça reste du e-commerce, mais je pourrais même sortir de ce domaine pour aller benchmarker **Airbnb** par exemple !

Tous les sites qui ont des fonctionnalités intéressantes qui pourraient être utiles sur votre interface valent la peine d'être creusés. Néanmoins, comme en code, ce qui marche quelque part ne fonctionnera peut-être pas chez vous…

Ce dont je vous parle ici, c'est du benchmark fonctionnel, c'est-à-dire que je vais me focaliser sur les fonctionnalités. Il existe aussi un benchmark inspirationnel qui consiste à regarder davantage les codes graphiques et à s'en inspirer. *(1)* Nous n'entrerons pas davantage dans les détails, je vous laisse le soin de creuser si le sujet vous intéresse. Une ressource : https://www.usabilis.com/quest-ce-que-le-design-thinking/#:~:text=Le design thinking est une,des services ou produits innovants.

L'outil de l'UI Designer : Figma

Je ne suis pas neutre dans ce domaine puisque j'adore Figma, mais il existe aussi Sketch ainsi qu'Adobe eXperience Design (Adobe XD) qui permettent de concevoir des interfaces digitales.

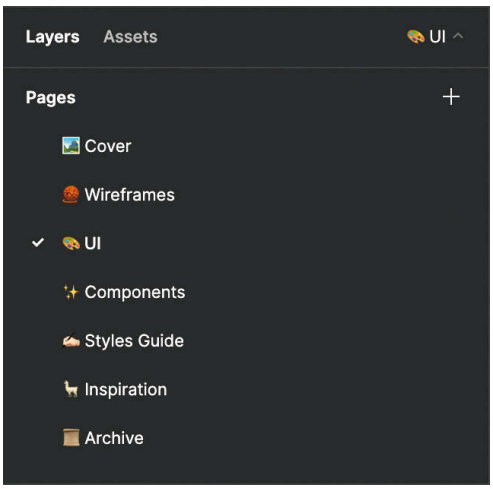
Figma est un outil gratuit (dans une moindre mesure) qui existe dans une version *light* et, depuis peu, dans une version *dark*.

C'est lorsque vous déciderez de créer des *Teams* que la question de payer un abonnement se posera. Les *teams* sont des endroits comme des répertoires Github : seuls ceux que vous invitez peuvent contribuer aux fichiers (éditer), et vous pouvez ainsi séparer différentes *teams* par *Features Teams* ou *Business Units* par exemple.

Il ne reste plus qu'à créer un nouveau fichier ou à entrer dans un fichier déjà existant !

La structure des fichiers

Chaque entreprise a son fonctionnement, mais ce qui reste commun est le fait d'utiliser plusieurs pages.



On définit toujours une image de couverture (*cover* ou *thumbnail*) qui permet d'identifier le projet. En général, on y fait figurer :

- Le nom du projet ou de la feature. Par exemple : panier (*basket*).
- La personne qui s'en occupe ou qui en est responsable.
- Le statut du fichier : en cours, complété (côté design), prêt pour le développement, terminé, etc.

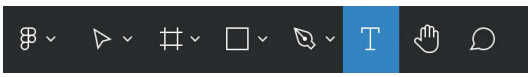
Commencer à créer

Comme les développeurs, les designers fonctionnent avec des composants (*components*). Les composants garantissent une meilleure cohérence graphique, puisqu'ils évitent de créer de nouveaux éléments à chaque nouveau besoin.

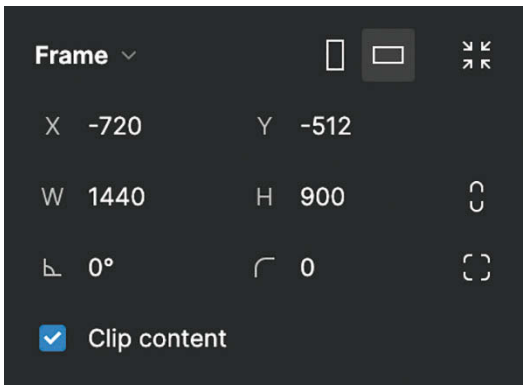
Les designers travaillent souvent avec l'*Atomic Design*, concept créé par Brad Frost, qui consiste à séparer les éléments selon qu'ils sont des atomes, des molécules, des organismes, des *templates* ou des pages.

Partons sur un exemple pratique : les boutons !

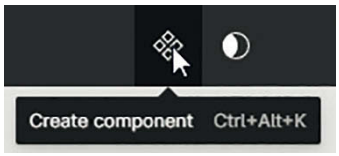
Commencez à écrire votre texte grâce à l'outil Texte (*Text*).

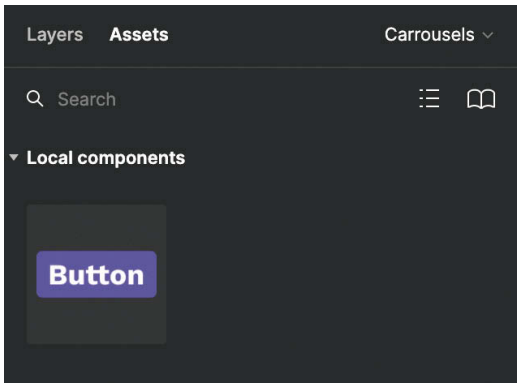


Avec le raccourci *Shift* + *A*, transformez votre texte en auto-layout, notion sur laquelle nous reviendrons plus bas. Remplissez le "Fill" avec une couleur et ajoutez un border-radius si vous le souhaitez.

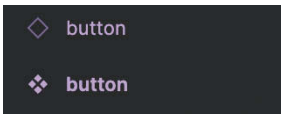


Dans la barre en haut, cliquez sur le losange composé de 4 carrés*(6)* en sélectionnant votre élément. Vous avez maintenant un composant*(7)* ! Plus qu'à le renommer "button" et le tour est joué. Dans le panneau *Assets(8)*, vous pouvez maintenant retrouver votre button et le réutiliser où vous voulez.





Quand vous réutiliserez votre *button*, il aura un symbole différent dans le panneau *Layers*. Ce sera à présent un losange vide*(9)*.



La logique des composants est très semblable à celle des développeurs : l'idée est de créer le composant le plus générique possible, et donc un composant réutilisable. Ici, nous avons vu un élément très basique, mais la prise en main est similaire pour les éléments plus complexes. Je vous conseille de creuser dans les vidéos, qui seront sans doute un support plus adapté qu'un article :

https://www.youtube.com/watch?v=e68PKFYWfoQ
https://www.youtube.com/watch?v=e68PKFYWfoQ&feature=emb_imp_woyt
https://www.youtube.com/watch?v=Pip1uFrB8Il
https://www.youtube.com/watch?v=Pip1uFrB8Il
(Pour la mise à jour)

Auto-layout et contraintes

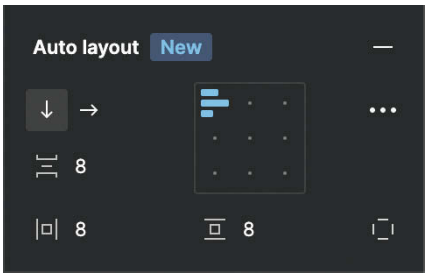
Ces fonctionnalités sont LA grande spécificité de Figma par rapport à Adobe XD ou Sketch, disponible nativement dans l'outil. C'est aussi ce qui se rapproche le plus de la propriété CSS Flexbox. Lien vers la propriété CSS Flexbox : https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Auto-layout

Il est possible de mettre à peu près tout en auto-layout. Reprenons notre exemple de bouton. Prenez l'outil Texte (*Text*) en appuyant sur T et écrivez "Button label". Ensuite, faites le raccourci clavier *Shift* + *A*. Vous pouvez voir un nouveau symbole dans le panneau de gauche :

(Auto-layout horizontal : les éléments se mettront les uns à côté des autres. En CSS, cela équivaut à un flex-direction: row;)

(Auto-layout vertical : les éléments se mettront les uns sous les autres. En CSS, cela équivaut à un flex-direction: column;) Et du côté du panneau de droite, de nouvelles propriétés sont disponibles.



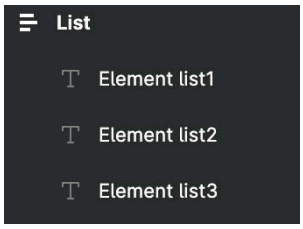
Il est possible de :

- Changer le sens de l'auto-layout : vertical ou horizontal.
- Ajouter, modifier ou supprimer la valeur de margin (espace entre les éléments).
- Ajouter, modifier ou supprimer la valeur de padding (espace interne aux éléments) sur la gauche et la droite.
- Ajouter, modifier ou supprimer la valeur de padding en haut et en bas.
- Gérer l'alignement des éléments présents dans l'auto-layout.

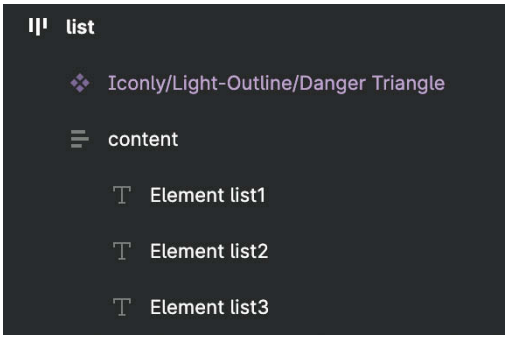
Dans l'exemple de notre bouton, nous n'avons qu'un texte ; si nous ajoutons une icône, l'alignement peut être intéressant puisque nous pourrons centrer les deux éléments.

Essayons de construire une liste en auto-layout.

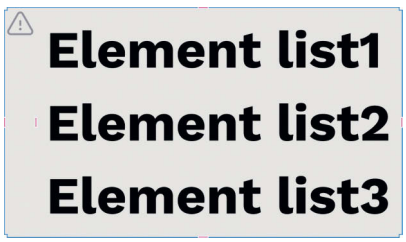
Pour ça, créez plusieurs blocs de texte. Notez qu'il est important que ce soit bien des blocs de texte indépendants et non des retours à la ligne. Par défaut, Figma positionnera l'auto-layout en horizontal. Passez-le en vertical. Dans le panneau de gauche, vous devriez avoir ça :



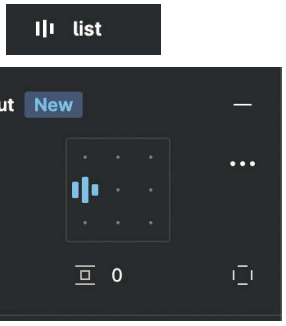
Ajoutons une icône, créons un nouvel auto-layout autour de l'ensemble, et passons l'auto-layout en alignement horizontal. De base, l'alignement devrait être en haut à gauche (*top bottom*).



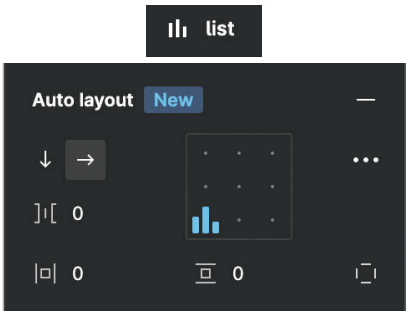
En haut à gauche



Centré à gauche



En bas à gauche

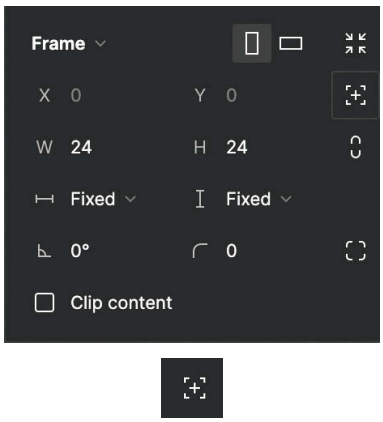


L'icône de l'auto-layout dans le panneau de gauche sera amenée à changer visuellement pour représenter l'alignement choisi.

Je vous invite à regarder des tutoriels vidéo tels que ceux de **Jessica Trocmé** pour mieux visualiser et comprendre la manière dont nous pouvons utiliser l'auto-layout. Et surtout, essayez par vous-même ! C'est en essayant qu'on apprend.

Pour finir sur l'auto-layout, la grande nouveauté intégrée à

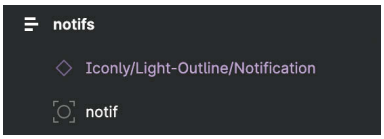
Figma en 2022, c'est la position *absolute*. Si vous cliquez sur votre icône, vous devriez avoir un symbole particulier qui apparaît en haut du panneau de droite.

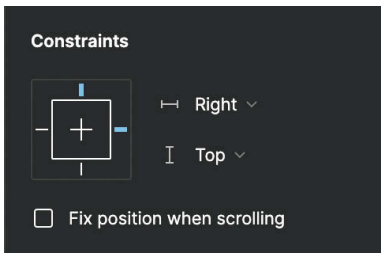


Dès que vous cliquerez sur le symbole, l'icône sortira du flux et votre icône aura un nouveau symbole dans le panneau de gauche.

Il est maintenant possible de placer votre icône où vous le souhaitez. C'est une fonctionnalité très pratique, notamment dans le cas de la pastille de notifications à placer en haut à gauche de la cloche.

Il ne reste plus qu'à définir les contraintes de l'élément en position *absolute* pour s'assurer qu'il sera toujours bien placé. Dans le cas de ma notification, je la veux en haut à droite.

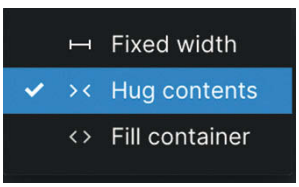




(Cliquez bien sur l'élément en absolute pour définir ses contraintes, pas sur le parent.)

Contraintes

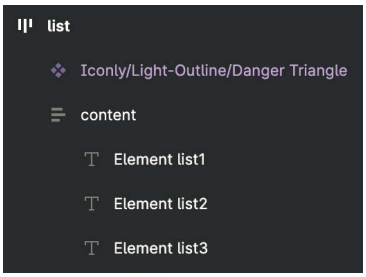
Les contraintes (*constraints*) sont situées dans le panneau de droite de Figma. Elles disparaissent quand l'élément est en auto-layout, pour proposer d'autres options situées en haut du panneau de droite sous la largeur (*width*) et la hauteur (*height*) :



De base, elles devraient être en *Hug contents*, ce qui signifie qu'elles suivent le comportement de l'enfant. Dans le cas du texte, s'il est en *Hug contents*, il pourra être amené à dépasser la width de la liste et à sortir du conteneur (*container*).

On laisse rarement une width en *Hug contents*, par contre il est très fréquent pour avoir un comportement responsive de laisser la height en *Hug contents*.

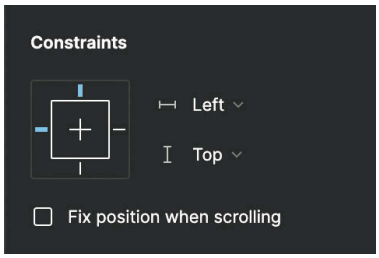
Pour la *width*, on va en général préférer l'option *Fill container* qui n'apparaît que si le parent est lui-même en auto-layout. Ici, mon content est en auto-layout et ma liste également.



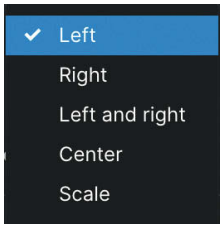
Ce qu'il faudrait faire, c'est passer chaque bloc de texte en *Fill container* au niveau de la *width* pour s'assurer que le texte ne sortira pas de son container. Vous verrez alors un message : "*Width set to Fixed for content*"

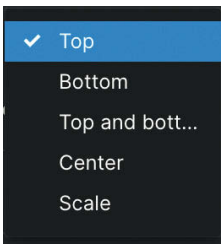
En effet, si je demande à mes éléments de remplir leur *container*, celui-ci doit prendre une *width* fixe afin de créer un comportement responsive.

Et puis il y a les contraintes qui ne sont pas liées à l'auto-layout. À l'origine, elles sont positionnées en haut à gauche (*top left*).



Elles peuvent avoir ces propriétés :





De mon côté, je touche assez peu aux contraintes puisque mes éléments sont souvent en auto-layout. Je mets simplement mes icônes en *Scale* pour qu'elles s'agrandissent si je change la hauteur ou la largeur de mon élément.

Il est possible de simuler un comportement d'auto-layout avec les propriétés *Left and right* et *Top and bottom*. Là encore, je vous conseille d'essayer par vous-même ou de

consulter des vidéos sur YouTube pour mieux comprendre et vous faire la main sur l'outil.
https://www.youtube.com/watch?v=floQKLsWAy4
https://www.youtube.com/watch?v=floQKLsWAy4&feature=emb_imp_woyt

Les armes de l'UI Designer : l'UI Kit et le Styles Guide !

L'UI Kit

L'UI Kit est l'ensemble des composants utilisés dans une interface. En d'autres termes, l'UI Kit, c'est la bibliothèque de composants de votre projet !
Il rassemble tous les composants au même endroit, facilitant ainsi la visualisation de l'ensemble des composants de l'interface et leur utilisation. Il permet aussi de garder la cohérence graphique et l'harmonie générale en utilisant des composants dans le but d'éviter de créer sans cesse de nouveaux éléments.
L'UI Kit contient également toutes les variations d'un composant, ce qui permet aux designers de pouvoir changer facilement certaines propriétés et d'avoir accès à toutes les déclinaisons d'un composant.
Puisque l'UI Kit est composé des composants *Masters*, il permet de garder l'interface à jour et de changer très facilement des éléments de style comme la typographie ou encore la couleur.
L'UI Kit permet d'industrialiser le processus de design, puisqu'il s'agit ensuite de faire du Lego : on assemble chaque composant pour créer une page. Pour concevoir un UI Kit efficace, je vous conseille de vous intéresser à l'*Atomic Design* et de lire des articles sur le sujet.
Pour autant, un UI Kit ne sert absolument à rien sans Styles Guide !

Styles Guide

Le Styles Guide, c'est la documentation de votre UI Kit.
Chaque composant doit avoir un rôle et doit être explicité pour permettre aux designers de votre équipe de l'utiliser facilement sans se poser de questions.
Le Styles Guide et l'UI Kit constituent un premier pas vers un Design System, et la meilleure documentation que je pourrais vous conseiller pour servir d'exemple vient tout droit des Design Systems.
En général, on y explique :

- Ce à quoi sert le composant ;
- Quand les utiliser ou quand ne pas les utiliser ;
- Les variants et dans quels cas les utiliser ;
- Les tailles (pour les boutons par exemple) ;
- Comment les utiliser (les *do & don't*).

Ce n'est jamais très drôle d'écrire une documentation, mais c'est pourtant primordial pour l'entreprise. La documentation permet à tous les designers d'avoir accès aux mêmes informations et de pouvoir produire des pages de la même qualité en garantissant l'harmonie et la cohérence graphique.
Ce sont des lignes de conduite à adopter, des règles à suivre. Il est donc important de passer du temps à les écrire pour chaque composant, et même pour les styles (couleurs, typographie, grille…).
On prévoit toutes les variations d'un composant pour avoir une librairie la plus exhaustive possible. Il est toujours possible d'ajouter de nouveaux éléments, mais il faut en général un ou plusieurs "garant(s)" de l'UI Kit/Design System qui valident les demandes et qui garantissent la bonne cohérence des éléments de l'interface (des composants de la marque).

Introduction au Design System

Le Design System est la Bible du designer et de l'entreprise en matière de design, de *ton of voice* et de code.
Encore jeune, il remplace peu à peu la charte graphique PDF pour s'exporter sur le web où il prend la forme d'un site répertoriant les lignes directrices (*guidelines*), les composants, la façon d'écrire (*copywritting*) et également des spécifications techniques.
Il s'agit d'un produit qui évolue et qui contient différentes versions pour permettre aux designers de proposer de nouveaux éléments, tout en permettant au reste de l'entreprise de continuer d'utiliser la version en cours. De ce fait, il a souvent un nom propre.
Il devient de plus en plus important pour les entreprises d'investir dans le Design System et d'y dédier des ressources propres afin de permettre une meilleure évolution, autant en terme de design que d'expérience utilisateur.
Je vous invite à consulter plusieurs Design Systems :

- Carbon, IBM : https://carbondesignsystem.com/
- Spectrum, Adobe : https://spectrum.adobe.com/
- Mozaic, Adeo : https://mozaic.adeo.cloud/
- Vitamin, Decathlon :

https://www.decathlon.design/726f8c765/p/71b8e3-decathlon-design-system

- Oxygen, Doctolib :

https://oxygen.doctolib.design/60b411768/p/77fd2d-doctolib-design-system

Vous pouvez aussi consulter la liste des designs systems français sur Design Systems France :
https://www.designsystems.fr/liste-des-designs-systems-francais

Le Design, une philosophie

Tout au long de cet article, nous sommes restés dans une vision très orientée Design visuel (*UI Design*). Pour autant, il ne faut pas oublier l'Expérience Utilisateur (*UX Design*) qui passe par beaucoup d'interactions avec les utilisateurs et de la recherche pour comprendre leurs besoins, leurs attentes, leurs motivations et leurs frustrations.
Il s'agit de créer des produits pour répondre aux besoins des utilisateurs et solutionner leurs problèmes. Dans le cas où le profit serait la seule motivation, le produit risquerait de ne pas fonctionner.
Concernant l'UI Design, il s'agit d'une étape complexe qui est primordiale puisqu'elle peut avoir un impact émotionnel pour l'utilisateur et le pousser à apprécier le site et la marque. Attention néanmoins à ne pas tomber dans de mauvaises pratiques (*Dark Patterns*) qui auraient pour but de tromper l'utilisateur afin de le faire consommer !
Enfin, il faut de la rigueur et être à l'aise avec son logiciel pour faciliter et accélérer le travail. Il faut également beaucoup de réflexion en amont et bien construire sa base, c'est-à-dire l'UI Kit, le Styles Guide et le Design System. Le Design, c'est beaucoup plus que faire des cercles et des rectangles de couleur !

Louis de Choulot
Étudiant en 2e année d'étude de programmation informatique à ALGOSUP, Vierzon France.
louisdechoulot@gmail.com

Les VPN comme système de contournement des restrictions et sécurisation de vos connexions.

Le VPN, ou Virtual Private Network, est un système de chiffrement et de protection des données né en 1996 avec comme but de donner un accès à distance sécurisé aux fichiers d'une entreprise par ses employés, il ne concernait pas l'internet global, mais plutôt les intranets.

Aujourd'hui, des services VPN existent et proposent différents types de protocoles. Devant cette gamme nous pouvons nous demander lesquels utiliser ?

Les principes de base des protocoles VPN

Si nous parlons aujourd'hui du VPN comme étant un moyen sécurisé d'envoyer des données en les cryptant, rappelons-nous qu'à l'époque le VPN définissait le **PPTP** soit *Point to Point Tunneling Protocol*.
Ce système servait de "tunnel" protégé entourant la donnée pour empêcher son interception, elle n'était pas pour autant cryptée. Ainsi, la faille avait vite été découverte, le tunnel était facile à décrypter pour quiconque connaissant un peu le domaine.
Bien que ce protocole ait connu un succès long terne dû à son implémentation par un de ses créateurs : Windows, sur le système Windows XP. Celui-ci n'est aujourd'hui plus proposé par les services VPN.

En 1998 le protocole **IKEV** pour *internet Key Exchange* fait son apparition, ce dernier se sert de clefs de cryptage pour crypter la donnée elle-même. Microsoft et Cisco, ses créateurs, venaient de créer la base du protocole VPN moderne.
IKEV est devenu obsolète avec l'arrivée du nouveau **IKEV2** en 2005, on parle plus souvent de celui-ci comme une amélioration et c'est pourquoi IKEV porte aussi le nom d'IKEV1.
Les faiblesses majeures d'IKEV1 étaient :

- Sa grande consommation de ressources ;
- L'authentification symétrique, on lui préférera l'asymétrique qui utilise 2 clefs de cryptage. Une par appareil, au lieu d'une seule commune à chaque appareil connecté ;
- Aucun accès à distance possible ;
- IKEV utilise toujours le port UDP 500.

Ces faiblesses sont toutes réglées dans la version de 2005 IKEV2.

Enfin, les bases autour des protocoles VPN actuels, ne sauraient être complètes sans parler d' **IPSec** ou *internet Protocol Security*. IPSec représente un ensemble de protocoles utilisés dans la sécurisation des données qui transitent sur internet, il se trouve directement sur le réseau et ne nécessite aucune configuration extérieure.
On entend souvent parler d'IPSec avec **L2TP** *Layer 2 Tunneling Protocol*, une évolution du PPTP, développé par Microsoft et Cisco. L2TP ne contenant pas de cryptage des données et n'ayant pour but que la création du tunnel entre les appareils connectés, fait d'IPSec le parfait "partenaire".

Remote access Vpn et Site-to-Site Vpn

Le Vpn possède deux possibilités d'accès, que ce soit en intranet [1] ou en extranet [2].
Le Vpn *site to site* est celui que vous utilisez quand vous vous rendez sur internet, si vous possédez un service Vpn de bas niveau, c'est un cryptage de la donnée entre votre appareil et celui qu'il doit atteindre.
Le remote access Vpn ou *Vpn avec accès à distance* n'est pas plus compliqué, vous gardez le principe du *site to site* Vpn et vous ajoutez un appareil, le vôtre. Puis vous allez vous connecter à ce réseau site to site de façon sécurisée, mais à distance

Il n'y a pas de désavantage à se connecter en remote, le seul mauvais point qu'on peut lui trouver est dans la surveillance des connexions au *site to site*, une brèche de sécurité peut souvent apparaître à cause d'une négligence telle que ne pas consulter les logs/historiques de connexions pour observer si les appareils ayant accédés au site sont bien enregistrés. L'erreur viendra plutôt de l'Humain, notamment s'il accorde des droits d'accès aux mauvais appareils, c'est une maladresse assez courante, elle s'est par exemple produite à la NASA. En effet, un raspberry pi avait été branché dans l'enceinte de l'établissement et avait eu accès au réseau interne. Les hackers n'avaient alors plus qu'à se connecter au raspberry à distance, puis à collecter autant de données que possible avant que la brèche ne soit décelée, leurs identités n'ont toujours pas été découvertes.

Les protocoles VPN actuels

Les protocoles VPN n'ont de cesse de s'améliorer, le plus souvent grâce à l'aide des géants de la tech comme les GAFAM [3], nous l'avons vu précédemment avec Microsoft qui s'est impliqué dans la création de plusieurs des protocoles VPN basiques. De nos jours, ces derniers sont le plus souvent utilisés par le biais d'un service VPN, il n'est donc pas rare que l'utilisateur ne sache pas que la plupart proposent les

mêmes services et protocoles avec pour seule différence la qualité de ceux-ci. Avec un service VPN payant, il y aura sûrement une connexion sécurisée à un serveur qui s'occupera d'effectuer pour vous les demandes de connexion aux pages web et renverra ce qu'il obtient à votre appareil, il s'agit souvent de connexions en remote access.

Parmi les plus populaires et utilisés, on retrouve **OpenVPN (2002)**, sécurisé et rapide, il est issu de technologies open source/ libre d'accès.
S'il n'est pas le plus rapide, il possède l'un des meilleurs alliages stabilité, sécurité et compatibilité. Il est compatible sur tous les appareils avec comme seul bémol qu'il nécessite un tiers pour l'installer. Son support mobile laisse place à des améliorations.

En second, il s'agit d'un VPN beaucoup plus jeune, à savoir **SoftEther (2014)**. Celui-ci se base sur d'autres protocoles VPN qui ne lui sont pas "propres". Ce nouveau venu possède un haut niveau de sécurité, mais surtout une grande rapidité. Il n'a aucun problème de compatibilité, quel que soit l'OS [4], car dès le départ il a été pensé multi-plate-forme. Il se pourrait qu'il dépasse OpenVPN dans les années à venir, tant en popularité qu'en efficacité.
Le troisième est encore plus nouveau et surtout promet d'être plus efficace que OpenVPN sur bien des aspects : **WireGuard**.
15% plus rapide, une sécurité au top, avec une promesse de multi-plate-forme, il ne date que de 2019 pour sa version la plus récente, mais fait déjà partie des références du VPN. Si on peut lui reprocher une chose, c'est qu'il échappe beaucoup plus difficilement aux filtres anti-VPN que OpenVPN et SoftEther ce qui est une donnée importante dans le choix de son protocole, car les filtres anti-VPN se popularisent et beaucoup de grandes entreprises les mettent en place, en vue de vous empêcher d'avoir un abonnement à un prix moins cher sur l'un de leurs services ou de participer à une bêta ouverte uniquement dans des zones géographiques spécifiques. Un autre challenger anti-VPN est l'état, en vous connectant en dehors du pays vous échapper à la censure appliquée sur les réseaux.
Néanmoins, WireGuard est toujours en développement et ses failles pourraient être comblées lors de futures mises à jour.
Enfin, **SSTP**, développé par Microsoft, ne fait aucune promesse de multi-plate-forme, mais fonctionne sous certaines distributions Linux : Apple et mobile passez votre chemin !
SSTP est toujours en développement et grâce aux robustes équipes de Microsoft, il est extrêmement efficace sur Windows. Son protocole de cryptage est le même que celui utilisé par OpenVPN soit du 256 bits, celui-ci est aussi appelé AES pour *Advanced Encryption Standard*. AES est réputé pour n'avoir aucune backdoor, le seul moyen de le cracker est d'utiliser des techniques plus ou moins proches du brut force, il faudrait plusieurs millions d'années avec le meilleur ordinateur du monde pour le cracker, car le nombre de possibilités pour AES est de 2 à la puissance 256 soit 1.157920892373162e+77 soit bien plus qu'un milliard de milliards de milliards.(15,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,039,457,584,007,913,129,639,936 de possibilités)
SSTP est un peu le PPTP/LT2P de notre époque.

Ressources

[1] Intranet : connexion entre plusieurs appareils accessibles dans un lieu donné en local. Il s'agit d'un réseau interne.
[2] Extranet : Fait souvent référence à l'internet global et toutes les connexions accessibles à distance. Il s'agit d'un réseau externe.
[3] Google, Apple, Facebook, Amazon et Microsoft.
[4] Système d'exploitation.

Sources

https://www.vpnmonde.com/levolution-des-vpn-de-1995-a-nos-jours/
https://www.le-vpn.com/fr/histoire-des-vpn/
https://www.fibre-pro.fr/vpn-ipsec/
https://www.cisco.com/c/en/us/support/docs/security-vpn/ipsec-negotiation-ike-protocols/217432-understand-ipsec-ikev1-protocol.html
https://nordvpn.com/fr/vpn-client/
https://nordvpn.com/fr/what-is-a-vpn/#3972593d-0
https://surfshark.com/fr/
https://surfshark.com/fr/learn/vpn-cest-quoi
https://openvpn.net/
https://www.softether.org/
https://thebestvpn.com/pptp-l2tp-openvpn-sstp-ikev2-protocols/
https://www.security.org/vpn/best/
https://www.journaldunet.fr/web-tech/guide-de-l-entreprise-digitale/1506369-openvpn-tout-savoir-sur-le-vpn-open-source-pour-les-pros/
https://le-routeur-wifi.com/site-bloque-vpn/
https://cybernews.com/what-is-vpn/wireguard-protocol/
https://www.vpnxd.com/softether-vs-openvpn-which-one-better/
https://lecrabeinfo.net/vpn-les-protocoles-de-chiffrement.html#openvpn
https://fr.vpnmentor.com/blog/les-differents-types-de-vpn-et-quand-les-utiliser/#section-5
https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#Performance
https://www.thesslstore.com/blog/what-is-256-bit-encryption/

le bon outil pour le bon projet... ou pas.

Directives de compilation

PROGRAMMEZ!

Programmez! n°254 - Septembre - Octobre 2022

Seigneur Sith niveau directeur des rédactions :
François Tonic - ftonic@programmez.com

Rédactrice en chef du numéro : **Jinane Mehdad**

Code review : **Dorra Bartaguiz & Jinane Mehdad**

Contacter la rédaction : redaction@programmez.com

Ont collaboré à ce numéro : **Checkmax, CommitStrip**

Les contributeurs techniques

Jinane Mehdad
Romy Alula
Laurent Ellerbach
Christophe Villeneuve
Pierre Bouillon
Sylvain Cuenca
Jean-Christophe Riat
Guillaume Saupin
Pierre-Gilles Leymarie
Loïc Mathieu
Seifeddin Mansri
Guendalina Caldarini
Jean-Christophe Quétin
Fred Berton
Max Antoine Brun
Loïc Guillebeau
Thibaud Vienne et Hamza Aziz
Margaux Membré
Louis de Choulot
CommitStrip

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
1225K78366

ISSN
1627-0908

Abonnement
Abonnement (tarifs France) : 55 € pour 1 an, 90 € pour 2 ans. Etudiants : 45 €. Europe et Suisse : 65 € - Algérie, Maroc, Tunisie : 64 € - Canada : 80 € - Tom - Dom : voir www.programmez.com.
Autres pays : consultez les tarifs sur www.programmez.com.

Pour toute question sur l'abonnement :
abonnements@programmez.com

Abonnement PDF
monde entier : 45 € pour 1 an.
Accès aux archives : 20 €.

Nefer-IT
57 rue de Gisors, 95300 Pontoise France
redaction@programmez.com
Tél. : 09 86 73 61 08

Ce numéro comporte un encart jeté ComponentSource pour les abonnés.

INFORMER
pour transformer l'entreprise

La dématérialisation, le Cloud,
les communications unifiées,
les nécessités de la cybersécurité
transforment le travail et toute l'entreprise,
les services publics.
Le magazine, le site, ses newsletters
vous informent sur cette actualité mouvante
et vous aident à décoder les tendances

Abonnez-vous

www.solutions-numeriques.com/abonnement/

Toutes les ressources et newsletters en ligne

Pour répondre à vos besoins d'information

- La cybersécurité vous concerne ? Cliquez sur l'onglet. Vous trouverez les infos, l'annuaire, le lexique, etc
- L'emploi, les salaires, les formations, les offres vous intéressent ? Le site sur l'Emploi dans le numérique est à votre disposition
- Votre site Télétravail. Infos et bonnes pratiques pour le travail à distance et le travail hybride
- Solutions Channel. L'info des partenaires, intégrateurs, distributeurs

www.solutions-numeriques.com