PROGRAMMEZ!

Le magazine des développeurs

09/10 2020 N°242 23e année

PHP
Slim 4 : un microframework à découvrir

C'est quoi
GRAPHQL ?

Agile & DevOps :
amis ou ennemis ?

Utiliser la plateforme
VMWARE TANZU

Algorithmes
Génétiques
Enjeux & utilisations

Dans ce numéro, trouve ton bonheur de codeur :

Scala vs JDK 14
Deno 1.0
Réplication
Jenkins
Quantique

© François Tonic

M 04319 - 242 - F: 6,50 € - RD

Le seul magazine écrit par et pour les développeurs

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

EDITO

Zénitude & équilibre :
le développeur est un empilement de galets

C'est quoi être développeur en 2020 ? Un peu comme en 2019, 2018, année en cours -n. Sauf que 2020 ressemble plus à 2012. 2012, le film, pas l'année. Ou 2012, comme la fin du monde pour certains. Bon ok, la fin du monde n'est pas vraiment arrivée en 2012. Mais franchement, 2020, y'a du bon level avec les grèves, les manifs, le Covid et la canicule. On va se calmer un peu pour 2021...

Donc, c'est quoi être développeur en 2020 ? À en croire, les annonces d'emplois, les médias et certaines entreprises, le développeur serait une espèce rare, pas facile à trouver et un peu timide. Mais surtout, quand on regarde les caractéristiques de l'animal, il est bizarre : il doit savoir tout faire, vive le développeur full-stack ; il aime communiquer (courrier, mail, réunions, téléphone, visioconférence, très sociable avec les clients et la direction) ; il est acrobate niveau 300 (il n'est pas agiliste pour rien) ; il parle une multitude de langues et de dialectes (même le Cobol et le VB6) ; il aime râler contre un vieux Pentium avec 1 Go de RAM et 500 Go de disque dur (ben ouais, on va pas lui changer son poste de dév tous les ans : faut pas déconner, selon des sources anonymes d'entreprises) mais visiblement râler ferait partie de son mode de vie, avec le troll. Mais surtout, il est prêt à accepter un salaire niveau -300 car c'est la crise, ou il/elle est jeune, sans expérience, à peine sorti du nid ou de sa grotte ou en reconversion.

Ah oui ! J'oubliais un élément important... Il doit avoir au moins 10 ans d'expérience sur des technologies qui n'ont pas 5 ans ! (non, non, on ne rigole pas). « Cherche développeur avec 15 ans d'expérience sur Flutter même si techno n'existe que depuis 2017 ! »

Franchement, ça donne envie d'être développeur en 2020 !

Bref, le développeur est comme un empilement de galets : zen et équilibre. Il faut au moins ça quand on entend : c'est rapide, pas plus de 2h de codage, tiens voilà la chaîne intégrée DevOps qui est intégrée à la main, ce serait top si tu apprenais ce nouveau framework pour demain matin, ouais le projet va se faire en 2 mois au lieu de 6 mois, c'est un top défi non ? Eh, tu peux dépanner mon PC, mon téléphone et ma trottinette électrique ?

Re-bref, le développeur est un empilement de galets : zen et équilibre avec supplément pizza et café (je veux bien être zen et tout ce que l'on veut, mais il me faut la base).

Re-re-bref : bon code de rentrée avec Programmez!

Votre maître zen ceinture étoile noire,

François Tonic - ftonic@programmez.com

SOMMAIRE

Prochain numéro : PROGRAMMEZ! 243
Disponible le 30 octobre 2020
LOW-CODE / NO-CODE : la mort du développeur ?

Le monde merveilleux des ransomwares : un été chargé

Les cyberattaques ne s'arrêtent pas avec l'été, à tel point que c'est difficile de vous résumer les nombreuses affaires de ces dernières semaines. On essaie quand même : Garmin s'est fait avoir, et a visiblement payé la rançon pour récupérer ses données. CarlsonWagonLit Travel aussi : on est à peu près sûrs qu'ils se sont délestés de 4,5 millions de dollars pour s'en sortir, bien qu'ils ne l'admettront probablement jamais. On peut aussi citer LG, Xerox et Canon, tous trois piratés par le groupe Maze. En France, MMA s'est fait pirater, mais aussi le département d'Eure et Loire, le groupe de BTP Rabot Dutilleul, Orange, la mairie de Mitry-Mory… On ne s'ennuie donc pas trop, mais on espère que vous avez patché avant de partir en vacances.

Schrems II : le jugement dernier

Dans son arrêt baptisé « Schrems II » faisant suite au procès opposant le militant autrichien Maximilian Schrems à Facebook, la justice européenne a déclaré le Privacy Shield invalide. Ce texte était ni plus ni moins que le dispositif juridique encadrant et autorisant les échanges de données entre l'Europe et les États unis. Le Privacy Shield était le texte venu pallier le vide laissé par l'invalidation en 2015 du Safe Harbor, le précédent texte également mis à mal par la CJUE. Celle-ci s'exprimait déjà à l'époque sur l'affaire opposant Schrems à Facebook, débutée en 2009.

Ledger piraté ! Mais ça va aller

Si vous possédez des cryptomonnaies et que vous êtes quelqu'un de prudent, vous avez peut-être fait l'acquisition d'un porte-monnaie hardware pour garder en sécurité tous vos cryptosous. Et si vous avez du goût, vous avez peut-être opté pour Ledger, société française en pointe sur ce secteur. À la fin du mois de juillet, Ledger a néanmoins annoncé sur son site avoir été victime d'un piratage ayant permis à un tiers d'accéder à sa base de données clients : soit les adresses mail d'un peu plus d'un million de clients et des données personnelles qui traînaient çà et là. Une bonne nouvelle néanmoins : les porte-monnaie n'ont eux rien à craindre, et les cryptosous sont bien gardés. Ouf !

SoLocal : le chômage partiel va coûter cher… aux salariés

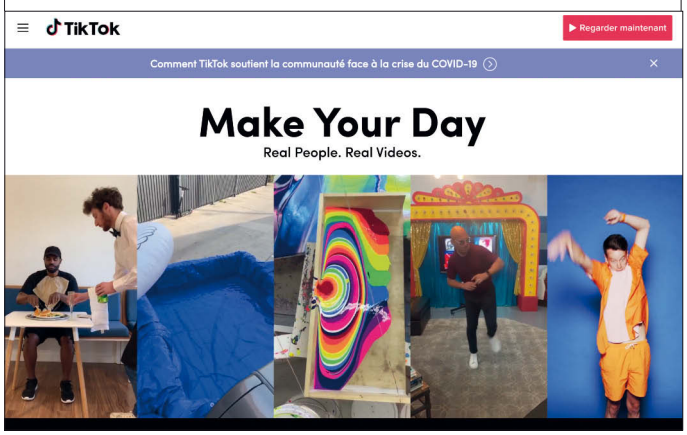
Les employés de SoLocal, ex-Pages jaunes, vont faire grise mine : un bug découvert au moment de la mise en place du chômage partiel dans le système de traitement de la paie a révélé plusieurs erreurs ayant conduit à surévaluer certaines primes. Résultat des courses : le groupe demande maintenant à environ 750 employés de rembourser le trop-perçu, certains remontant au mois de mars 2019. Sans surprise, les syndicats sont moyennement emballés.

Google, Fitbit et la Commission européenne

Google a annoncé en 2019 le rachat de Fitbit, poids lourd américain du marché des wearables et autres bracelets connectés. Mais certaines ONG de défense des consommateurs s'inquiétaient de cette fusion, estimant que ce rachat offrait à Google une position un peu trop dominante sur le marché de la publicité en ligne. On doit bien convenir qu'en mettant la main sur Fitbit, Google s'offre surtout un trésor de données pour alimenter ses offres de publicités ciblées. Et force est de constater que la Commission européenne s'en inquiète aussi : celle-ci a annoncé à la fin du mois de juillet l'ouverture d'une « enquête approfondie » sur cette fusion. Google jure de son côté que la seule chose qui l'intéresse dans ce deal, ce sont les objets connectés et qu'ils n'ont pas un seul instant pensé aux données. Promis, juré.

TikTok, by Microsoft

Ce nouvel épisode de la guerre économique entre la chine et les États unis se focalise sur TikTok, cette application chinoise de vidéos courtes qui pousse les adolescents du monde entier à danser frénétiquement devant leur smartphone. Accusée de siphonner les données des utilisateurs américains, celle-ci est menacée d'interdiction sur le sol américain par Donald Trump, début août. TikTok s'en défend vertement, mais les États unis ont déjà posé un ultimatum. Seule issue envisagée : celle d'une fusion avec une entreprise américaine. Et c'est Microsoft qui semble aujourd'hui le candidat tout désigné pour reprendre la main sur l'application. On se demande quand même ce que vient faire Microsoft dans cette galère, mais l'occasion fait le larron.



MEETUPS PROGRAMMEZ!

8 septembre : PHP8 avec Christophe Villeneuve
13 octobre : recherche avancée pour votre application legacy, avec David Pilato (elastic.co)
10 novembre : sujet à venir
22 décembre : C++20, avec Christophe Pichaud

A PARTIR DE 18H30

Informations & inscription : programmez.com

DEVCON#9 : SPÉCIALE .NET 5 ET TECHNOLOGIES MICROSOFT

1 journée entière, +10 sessions & ateliers

19 novembre,
À PARTIR DE 9H30 À EPITECH / PARIS

Informations & inscription : programmez.com

SEPTEMBRE

1	2	3	4	5	6	7
8	9	10	11	12	13	14
Meetup Programmez!/ Paris Détail sur programmez.com			JUG Summer Camp / La Rochelle			BIG DATA PARIS
15	16	17	18	19	20	21
BIG DATA PARIS						
22	23	24	25	26	27	28
Solutions RH/Paris	Cloud Expo & IoT World / Paris Solutions RH/Paris	Cloud Expo & IoT World / Paris Solutions RH/Paris				
29	30					

OCTOBRE

1	2	3	4	5	6	7
8	9	10	11	12	13	14
					Meetup Programmez!	
15	16	17	18	19	20	21
DevOps Days / Paris Volcamp / Clermond Ferrand	Volcamp / Clermond Ferrand					
22	23	24	25	26	27	28
					European Cloud Conference / Nice	European Cloud Conference / Nice
29	30	31				
European Cloud Conference / Nice Agile Tour Nantes/Nantes	BDX I/O / Bordeaux Agile Tour Nantes					

Sous réserve de Covid-19

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

PROGRAMMEZ!

Le magazine des développeurs

A PARTIR DU 1ER SEPTEMBRE 2020

LES CONTACTS POUR LES ABONNEMENTS ET RÉABONNEMENTS CHANGENT.

Une unique adresse :

Programmez
service abonnement
57 rue de Gisors
95300 Pontoise

Une seule adresse mail :

abonnements@programmez.com

Sans oublier notre boutique en ligne : www.programmez.com

Roadmap des langages

Déjà disponible

TypeScript 4.0

La v4 du langage n'est pas une version majeure, plutôt une mise à jour douce. Parmi les nouveautés : les types tuple, des améliorations sur l'inférence de propriété de classe, apparition de jsxFragmentFactory, performances améliorées en mode build (avec – noEmitOnError), nouveautés de l'éditeur TypeScript dans Visual Studio. Attention : des casses de compatibilité sont annoncées notamment sur lib.d.ts. Node Factory est déprécié. Consultez bien la documentation de version.

Septembre - octobre

Java 15

Septembre

Java 15 supportera les blocs de texte. Cette fonctionnalité est attendue depuis les premières préversions sorties avec Java 13. Les blocs de texte sont des chaînes à plusieurs lignes. C'est tout simple, mais cela permet d'insérer beaucoup plus facilement, dans le code Java, des extraits de code en HTML, XML, SQL, JSON, etc. Java 15 fera aussi du ménage en retirant Nashorn, le moteur JS d'Oracle. GraalVM rend obsolète le moteur.

Kotlin 1.4

Septembre

La principale nouveauté est un nouvel algo d'inférence de type (en préversion dans la 1.3). Le nouvel algorithme déduira des types dans de nombreux cas où l'ancienne inférence vous obligeait à les spécifier explicitement. Par exemple, dans l'exemple suivant, le type du paramètre lambda it est-il correctement déduit comme étant String ?

```
val rulesMap : Map<String, (String ?) -> Boolean> = mapOf(
  "weak" to { it != null },
  "medium" to { !it.isNullOrBlank() },
  "strong" to { it != null && "^[a-zA-Z0-9]+$".toRegex().matches(it) }
)
```

Le nouvel allocateur d'objets fonctionnera jusqu'à deux fois plus rapidement sur certains benchmarks, selon JetBrains. Cette version inclura aussi un nouveau compilateur IR (intermediate representation) côté backend Kotlin / JS. L'éditeur prévient qu'il y aura un problème de compatibilité binaire.

Python 3.9.0

Octobre

Les premières versions alpha de la 3.9 ont été distribuées fin janvier. Désormais, il faudra s'attendre à une mise à jour annuelle du langage. Bref : moins de nouveautés, mais plus d'évolutions et d'améliorations. Parmi les nouveautés attendues : des améliorations sur le garbage collector et la résurrection d'objets (pouvant provoquer des blocages), évolution de la stabilité ABI pour éviter les incompatibilités sur les différents systèmes.

Bootstrap 4.5

Cette version corrigera des bugs des versions précédentes et promet une petite optimisation sur le poids des fichiers. Les exemples et la documentation sont maintenant disponibles en fichier ZIP. Cette version continue à s'appuyer sur jQuery qui sera retiré avec la v5.

Fin 2020

Swift 5.3

?

La grosse nouveauté du langage sera l'arrivée officielle de Swift sur Windows. Côté Linux, le langage arrive sur de nouvelles distributions Linux : Ubuntu 20.04, CentOS 8 et Amazon Linux 2 ! Avec l'arrivée des Mac ARM, le langage sera aussi disponible nativement sur ARM.
Au-delà, la version 6 apportera pas mal de nouveautés et d'améliorations :

- Possibilité de porter le langage sur de nouvelles plateformes ;
- Des améliorations sur le gestionnaire de paquets ;
- Temps de compilation réduits ;
- Meilleure autocomplétion du code.

Aucun agenda n'a pour le moment été communiqué.

.Net 5

Novembre

.Net 5 doit apporter une unification des différentes éditions de .Net et servir de fondation sur toutes les plateformes supportées. Parmi les nombreuses nouveautés et évolutions annoncées, notons :

- Expérience du SDK .NET unifié :
  - BCL (bibliothèque de classes de base) unique dans toutes les applications .NET 5. Aujourd'hui, les applications Xamarin utilisent le Mono BCL, mais viendront à utiliser le .NET Core BCL, améliorant ainsi la compatibilité entre les modèles d'applications.
  - Le développement mobile (Xamarin) est intégré dans .NET 5. Cela signifie que le SDK .NET prend en charge les mobiles. Par exemple, vous pouvez utiliser «dotnet new XamarinForms» pour créer une application mobile.
- Applications natives prenant en charge plusieurs plateformes : projet Single Device qui prend en charge une application pouvant fonctionner sur plusieurs appareils. Par exemple Windows Desktop, Microsoft Duo (Android) et iOS à l'aide des contrôles natifs seront pris en charge.
- Applications Web prenant en charge plusieurs plateformes : projet Single Blazor qui prend en charge une application pouvant fonctionner dans les navigateurs, sur les appareils mobiles et en tant qu'application de bureau native (Windows 10x par exemple).
- Cloud Native Applications : hautes performances, fichier unique (.exe) <50 Mo de microservices et prise en charge de la création de plusieurs projets (API, frontaux Web, conteneurs) à la fois localement et dans le cloud.
- Améliorations continues, telles que : algorithmes plus rapides dans la BCL, meilleure prise en charge des conteneurs lors de l'exécution, prise en charge de HTTP3.

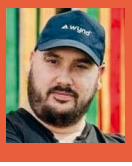
Avec .Net 5, Microsoft inaugure une nouvelle cadence d'évolution annuelle de .Net pour ne plus attendre plusieurs années pour introduire une évolution majeure.

PHP 8.0

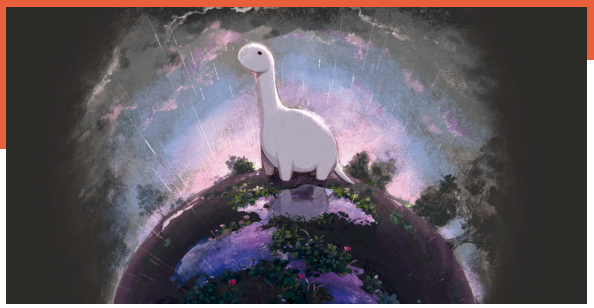
Décembre

La prochaine grande version de PHP doit arriver fin 2020. Cette version annonce beaucoup de choses, dont :

- casse de compatibilité avec les versions 7.x sur certaines évolutions ;
- Les types Union vont arriver avec PHP 8, qui permettront de définir plusieurs types pour les arguments reçus par une fonction, ainsi que pour la valeur qu'elle retourne. En plus du type self, le type static va devenir un type de valeur retournée valide ;
- Le compilateur JIT de PHP 8 promet d'importantes améliorations de performances.



Lilian Saget-Lethias
Software Architect
ParisDeno Creator
@lsagetlethias



https://deno.paris

Deno 1.0

LE NODE.JS KILLER (!?)

Dans un numéro précédent(1), nous avions fait une introduction à Deno qui était à l'époque en version 0.21.0. Après quelques mois d'attente, la version 1.0.0 est enfin sortie et s'accompagne de son lot de nouveautés. Node a-t-il du souci à se faire ? Etat des lieux par rapport à Node et petit tour d'horizon sur les changements apportés, les ajouts, et l'avenir de la technologie.

(1) "Deno, première approche", Programmez! n°237

EVOLUTIONS

Rusty V8

Le module libdeno était à la l'origine le lien qui unissait V8 à Deno. Malheureusement, avec le temps, la maintenance était de plus en plus compliquée du fait de la non homogénéité de la bibliothèque. En effet, libdeno était en majeure partie développée en C++, avec un interfaçage en Rust. Il fallait donc trouver une solution pour optimiser l'interface (et ses performances) et unifier les développements avec le Core de Deno.
Rusty V8, née de ce besoin, est une crate Rust indépendante qui permet de faciliter le lien avec V8 en se passant d'une couche d'abstraction supplémentaire. Cette couche est celle qui, entre autres, empêchait l'utilisation correcte de l'Inspector API pour le debug.

- Équivalent Node : C/C++ Node binding.

Deno format

Avant la 1.0, l'outil intégré de formatage de code était un fork du très connu Prettier. Le problème était qu'en tant que fork, l'outil était donc développé en TypeScript et exécuté par Deno (ce qu'on peut appeler du *dogfooding*) ; à la première exécution, l'outil était lent par rapport à l'exécution d'un Prettier "natif" par exemple ; de surcroît, plus la taille d'une codebase augmentait, plus le temps de l'exécution augmentait en conséquence.
Le choix a donc été fait de changer le coeur de l'outil pour un module natif écrit en Rust, dprint[2], plus performant, et maintenu par un tiers avec la participation au besoin de la team Deno.

- Équivalent Node : Prettier, ESLint, (parmi d'autres)

Compiler API

Jusqu'à maintenant, la CLI Deno donnait la possibilité de compiler, de mettre en cache, ou d'empaqueter, le tout avec les commandes deno compile, deno cache (avant deno fetch), et deno bundle.
Afin de donner plus de liberté, ces commandes sont maintenant accessibles de manière programmable et centralisées au sein de la Compiler API. Cette uniformisation est, à l'heure où ces lignes sont écrites, toujours en cours d'implémentation. D'autres commandes arriveront dès qu'un consensus sera établi(3).

- Aucune équivalence Node.

Deno upgrade

Deno, et sa CLI, sont assez simples à installer ; que ce soit manuellement ou via Homebrew pour macOS par exemple. En revanche, la mise à jour peut très vite devenir fastidieuse quand elle est faite manuellement. Enfin, via les gestionnaires de dépendances (comme brew), les maintenir peut devenir très vite encombrant, d'autant plus si les gestionnaires se multiplient.
Pour pallier ce potentiel problème, et rester dans un esprit purement dogmatique, la commande CLI deno upgrade a été ajoutée pour facilement mettre à jour Deno. Avec cette commande on peut donc soit mettre à jour Deno à la dernière version, soit installer une version au choix.

- Équivalent Node : nvm ou n.

RECONSIDÉRATIONS

Compatibilité Nodejs

Depuis ses débuts, Deno a essayé de ne pas copier Node au niveau de l'implémentation. Aussi, l'isomorphisme n'était pas envisagé ou envisageable car initialement, Deno est un système différent de Node. Mais même si l'implémentation est différente les similarités subsistent.
Malheureusement, lesdites similarités ont conduit la communauté à réimplémenter l'API de Node pour faciliter le port de bibliothèques, voir l'import direct sans conversion.
Ce qu'il ne faut pas oublier, c'est que cette implémentation est strictement destinée à l'extérieur, c'est à dire qu'elle n'est qu'un module de compatibilité qui ne doit, en aucun cas, être utilisé en interne dans la bibliothèque Standard.

Sortie avant les performances désirées de HTTP

Pour la sortie de Deno, l'accent a été principalement mis sur l'expérience utilisateur et la stabilisation de l'API. De ce fait, même si le *monitoring*(4) et la non régression des performances du serveur HTTP ont été maintenus, lesdites performances pour la 1.0 ne sont à ce jour pas au niveau souhaité initialement. Le problème vient principalement du fait que le serveur HTTP de Deno est écrit en TypeScript et implémente une couche TCP bas niveau ; contrairement à Node où le serveur HTTP lui-même est écrit en C au bas niveau. Le but n'est donc pas de réécrire le serveur HTTP en C ou Rust, mais d'améliorer les performances de TCP qui est directement utilisé par HTTP et dans d'autres cas (WebSocket par exemple).

Etude de la réécriture du TypeScript Compiler en Rust (?)

Comme dit plusieurs fois , la couche TypeScript (et notamment la *transpilation* à la volée) peut poser des problèmes de performance dans certains cas. Pour rappel et pour simplifier, TypeScript est un langage destiné à la compilation, et non à l'exécution. Donc, au moment du premier lancement d'un programme, Deno souffre d'un temps d'exécution plus long causé par une compilation et une mise en cache initiale. Plus précisément, le *type checking* est la partie qui prend le plus de temps. L'équipe planche donc sur

(2) https://github.com/dprint/dprint

(3) https://github.com/denoland/deno/issues/4752

(4) https://deno.land/benchmarks

l'éventualité de soit réécrire cette partie critique en natif dans Deno, soit l'optimiser d'une manière ou d'une autre(5). Le but n'étant pas de "refaire" TypeScript bien évidemment.

Flag "--unstable"

Depuis la sortie de la 1.0, un certain nombre de fonctionnalités sont restées considérées comme instables, et ne sont donc pas utilisables en production.

Même s'ils n'ont pas besoin du flag --unstable pour être utilisés, les standards (disponibles sous https://deno.land/std/) sont eux aussi considérés instables. A l'heure où ces lignes sont écrites, les standards sont en version 0.56.0, décorrélés de la version de la CLI de Deno afin de refléter leur non stabilité.

Une fonctionnalité est définie comme instable lorsqu'elle n'est pas passée par toutes les revues de sécurité considérées essentielles pour son utilisation en production. Ladite fonctionnalité est alors potentiellement soumise à des changements rompant la rétrocompatibilité (*breaking changes*).

AJOUTS

Plugins

Toujours au statut expérimental, un système de *plugins* est en cours d'ajout dans Deno. Ces plugins pourraient prendre plusieurs formes. Les premières cibles sont les "ops", une couche intermédiaire développée en Rust et branchée sur Rusty V8. Mais d'autres variantes sont envisagées, comme par exemple des sources Rust directement (ex : `import { add } from "./add.rs";`) comme le fait Parcel(6), ou certains types de bibliothèques (.dll, .so, …)(7).

WebAssembly (WASM)(8)

Dans la même veine que les plugins, il est aussi possible de charger du WebAssembly directement dans Deno. Un code web assembly peut être chargé soit via un fichier binaire, soit via un tableau directement dans le code. Afin de rester *web compatible*, l'API pour charger le module est quant à elle la même que pour les navigateurs.

Fichiers "lockfile" et "import_maps

Il est compréhensible pour les développeurs venant de Node ou du monde web d'être déconcertés par l'absence d'un gestionnaire de dépendances dans Deno. Il est tout de même facile de simuler une bonne gestion des dépendances en combinant plusieurs fonctionnalités. Parmi elles, on retrouve un fichier lockfile et un fichier import_maps.

Le lockfile permet de garantir l'intégrité des dépendances pour chaque module utilisé par un point d'entrée donné. Ce fichier est activable avec les flags --lock=<fichier> (pour la vérification) et --lock-write (pour l'écriture et la mise à jour dudit fichier).

L'import_maps (qui est instable !) est la même fonctionnalité trouvable dans le monde web par exemple(9). Ce fichier est activable avec le flag --importmap=<fichier>. Il permet, pour rappel, de lier des urls (ou bases d'url) à des raccourcis de noms de modules afin d'en simplifier leur utilisation dans le code. Il peut aussi être utilisé afin de centraliser par exemple les versions des dépendances d'un programme. Pour rappel, il est possible dans une url d'import de préciser une référence git, par exemple : https://deno.land/std@0.56.0/

Debug et inspection

Le protocole d'inspection V8 est implémenté dans Deno permettant à un programme d'être débuggé via n'importe quel client comme les Chrome Devtools par exemple. A la manière de Node, le debug est activable avec les flags --inspect ou --inspect-brk.

Pour rappel le premier fera s'arrêter le programme *que* s'il trouve un point d'arrêt ou le mot clef debugger, tandis pour que le second, cela se fera dès le début de l'exécution.

Cette fonctionnalité ne fonctionne très bien pour l'instant qu'avec les Chrome DevTools, mais le support par les IDE est encore limité. L'extension officielle VSCode par exemple n'est aujourd'hui pas encore prête directement(10).

Doc et website

Qui dit sortie officielle dit *branding*. Pour l'occasion le site deno.land a fait peau neuve et propose une refonte du manuel d'utilisation et de la documentation. Le tout, mis à jour automatiquement à chaque nouvelle version de Deno. On y retrouve donc le manuel qui regroupe entre autres les instructions de premières utilisations ; la documentation auto-générée de l'API du Runtime (fonctions identiques au navigateur, le namespace Deno) ; la documentation de l'API Standard reprise du dépôt officiel ; et enfin le moteur de recherche officiel des bibliothèques tierces (qui sont exposées derrière l'url de module https://deno.land/x/).

ROADMAP

Deno est-il donc prêt pour une utilisation en production ? Avec tout ce qui a été énuméré précédemment : **oui**. L'écosystème est maintenant suffisamment stable pour capitaliser dessus et continuera de se stabiliser et de gagner en performances avec le temps.

D'ailleurs, la performance est le maître mot des prochains objectifs de refonte qui seront établis avec notamment une intégration plus poussée de l'arbre de dépendances afin d'améliorer grandement les temps de (re)compilation ; mais aussi une refonte de la mécanique du transport HTTP, comme dit précédemment, qui est le dernier point sur lequel Deno peut être amélioré. Enfin, il est possible de suivre la liste des fonctionnalités suggérées, étudiées, et/ou considérées avec le *milestone* nommé "future" sur Github.

Parmi cette liste, on retrouve par exemple :

- L'exposition publique des API pour les compilateurs externes ; l'idée ici serait de donner la possibilité de brancher un autre langage haut niveau similaire à TypeScript à l'API de Deno. Cela permettrait potentiellement l'utilisation de Dart, Flow, ou CoffeeScript et ce "*out of the box*" comme c'est le cas pour TypeScript actuellement ;
- Le support de WebGL nativement dans le core ;
- L'analyse de code et la génération d'un rapport de couverture de code lors de l'exécution des tests, et ce *built-in* dans Deno ;
- L'utilisation de l'API fetch pour des fichiers locaux ;
- Une granularité des droits d'exécution plus fine.

OUTRO

Peut-on alors réellement dire que Deno est "*le Node.js killer*" ? Pas vraiment, et il ne le sera sans doute jamais. Au même titre que C# n'est pas un C++ killer, Scala un Java killer, ou TypeScript un JavaScript killer par exemple.

Deno n'a pas pour but de remplacer Node. C'est un écosystème à part entière, basé sur des principes architecturaux différents, qui vient d'effectuer un premier pas vers la maturité. Le but de la team Deno vis-à-vis des développeurs Node est d'offrir une transition douce afin de migrer de Node vers Deno. Maintenant, certains projets seront plus appropriés en Node qu'en Deno et vice-versa. De la même manière qu'il n'y pas de "gagnant" entre la programmation fonctionnelle et la programmation objet, entre Pepsi et Coca. Pour finir, les communautés mondiales autour de Deno sont en train d'émerger et de grandir, et notamment la communauté française et parisienne. N'hésitez pas à nous rendre visite https://deno.paris et nous nous ferons un plaisir de répondre à vos questions ! Restez à l'écoute !

(5) https://github.com/denoland/deno/issues/5432

(6) https://parceljs.org/rust.html

(7) https://github.com/denoland/deno/pull/3372

(8) https://webassembly.org

(9) https://github.com/WICG/import-maps

(10) https://github.com/denoland/vscode_deno/issues/12

Epitech : des fondamentaux aux projets concrets

Epitech fait partie des écoles d'informatique les plus réputées en France. Ses atouts ? La diversité de ses cursus : le Programme Grande École en 5 ans, ou encore des formations courtes en 2 ou 3 ans pour se reconvertir. Plus que jamais dans un monde de plus en plus numérique, l'école renforce ses axes technologiques et ouvre l'esprit des futurs développeurs aux prochaines révolutions technologiques, sans compromis avec les fondamentaux.

Depuis plus de vingt ans, Epitech forme ses étudiants à réfléchir à leur métier de manière intelligente : à quoi sert la technologie ? Quelles sont les valeurs qu'un développeur peut apporter ? Pourquoi est-ce qu'on ne code pas de la même manière pour un éditeur, une ONG ou un utilisateur final ? Le défi principal de l'école Epitech est d'intéresser les étudiants aux enjeux et aux contextes de leurs projets informatiques, et pas uniquement à la technique pour la technique.

Au-delà de la technique, la deep tech

Cette sensibilisation est un élément clé dans la pédagogie de l'école. Pour autant, pas question de remettre en cause les fondamentaux de l'administration système ou de la programmation (notamment avec les bases du C++).
Axelle Ziegler, Directrice des Etudes d'Epitech, précise : « *Nous abordons différemment les spécialisations techniques dans nos formations. Notre but n'est pas d'apprendre le dernier framework JavaScript à la mode ou tel ou tel nouveau langage. Nous sommes sur les fondamentaux et sur les "nouveaux" domaines technologiques : Machine et Deep learning, cybersécurité, etc. Quand on parle de Machine Learning, le but n'est pas d'apprendre TensorFlow, car les étudiants savent le faire ; mais de comprendre comment fonctionne le machine learning, comment fonctionne un réseau neuronal* ».

Le cursus type

Epitech est une école qui forme des experts en informatique en cinq ans. Le cursus classique est le suivant :
Année 1 : fondamentaux et autonomie (notamment administration et développement système) ;
Année 2 : conception et travail d'équipe (notamment programmation système avancée, shell, objet, fonctionnelle…) ;
Année 3 : diversification et innovation (notamment Java, .Net, C++ avancé, réseau..) ;
Année 4 : ouverture à l'international et expériences multiculturelles (choix d'Universités parmi 100 partenariats) ;
Année 5 : leadership (notamment, développement d'expériences interactives, IoT, et Blockchain).

COMMENT REJOINDRE EPITECH ?

« La condition principale pour être admis à Epitech est d'aimer vraiment l'informatique. »

Les épreuves suivent la même typologie, quels que soient l'année d'intégration et le programme retenu :

- Entretien oral de motivation (1h) ;
- Tests d'anglais (30 min) ;
- Tests de logique (15 min).

Tous les dossiers sont reçus à partir du mois d'octobre.
Ils doivent être remplis en ligne sur le portail de candidature le plus tôt possible dans l'année, et jusqu'à la rentrée, en dehors de toute procédure Parcoursup.
Chaque candidat est évalué individuellement sur sa motivation et son adaptabilité à la pédagogie particulière de l'école et à l'esprit d'entreprise.

AXELLE ZIEGLER

Directrice des Etudes d'Epitech

Quels sont les défis qu'Epitech doit relever dans un monde technologique de plus en plus diversifié et fragmenté ?

AZ : Notre choix, résolument, est celui de former des généralistes capables de découvrir et de s'approprier de nouvelles technologies. Sur des carrières de plus en plus longues et fragmentées, former des spécialistes d'une techno, d'un framework ou d'un langage n'a aucun sens. Nous attendons de nos étudiants qu'ils se construisent une culture générale, qu'ils approfondissent un certain nombre de technologies à travers leurs projets, mais aussi qu'ils soient capables d'en maîtriser de nouvelles. Si certains se spécialisent bien sûr ensuite dans des domaines technologiques (Sécurité, IA, Embarqué, etc.), cette culture générale est fondamentale.

Cet objectif, nous l'atteignons en proposant à nos étudiants à la fois des contenus permettant de développer cette maîtrise des fondamentaux de la programmation, de l'administration système, de la qualité de code, de la gestion de projet, tout en laissant suffisamment de souplesse pour insérer des projets plus spécialisés, encadrés par des intervenants extérieurs pointus dans leur domaine. Notre défi est de former des généralistes capables de se spécialiser, ou des spécialistes avec une culture générale suffisamment importante. Ce défi se relève à travers les apports méthodologiques de l'école, et le développement très tôt dans le cursus de la capacité à se former de façon rigoureuse dans un nouveau domaine.

Epitech peut-elle aider à réduire la fracture numérique, au moins, faire comprendre aux étudiants qu'elle existe ?

AZ : C'est un sujet qui est au cœur de nos préoccupations. Nos étudiants sont incités à faire du volontariat pour initier au code des publics les plus larges possible, à travers des ateliers à l'école ou dans des structures extérieures. Cette initiative est aujourd'hui déclinée en Epitech Diversity, dont l'objectif est de monter des partenariats avec des grands réseaux d'associations pour proposer l'aide de nos étudiants sur de l'accompagnement aux démarches numériques, de la prise en charge des outils à la découverte du code.

Ce travail sur la fracture numérique se fait aussi en gardant des écoles ouvertes sur le monde. Nos étudiants sont en effet sensibilisés dès le début de la 3ème année à l'importance de réfléchir à l'impact sociétal des projets qu'ils portent.

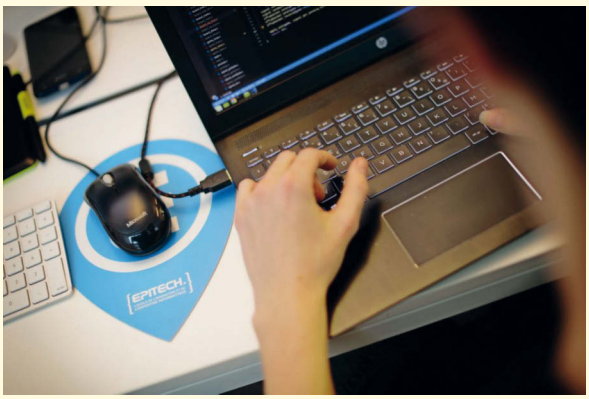
Toutes les écoles d'informatique œuvrent par exemple pour favoriser la diversité et attirer les lycéennes et étudiantes. Qu'en est-il à l'Epitech ?

AZ : Epitech est le berceau d'E-mma, une des associations étudiantes les plus dynamiques dans le numérique. C'est une association visant à promouvoir la mixité dans le domaine du numérique, qui est presque exclusivement masculin. Nous voulons encourager les femmes à se lancer sur des terrains dont elles ont été éloignées par leur éducation et leur culture. Nous estimons que nul domaine ne peut être productif s'il n'est pas composé d'hommes et de femmes. Nous voulons donc également encourager les hommes à nous rejoindre afin de travailler ensemble, main dans la main. E-mma voit l'avenir en double, c'est-à-dire où la mixité devient banalité, et où la productivité a pour essence l'égalité.

ATELIER BY EPITECH : LE LIEU DES PROJETS VISIBLES

Un nouveau lieu de rencontre entre étudiants, chercheurs et entreprises est ouvert à Epitech Paris, Atelier by Epitech. « On y trouvera des experts techniques de l'école, des chercheurs de laboratoires de recherche et des entreprises. L'idée est de pouvoir aborder des technologies nouvelles, créer des prototypes, faire découvrir à nos étudiants le monde de la recherche technologique même si notre but n'est pas de faire de la recherche fondamentale, mais plutôt de réaliser des projets prospectifs » explique Axelle Ziegler.

Atelier by Epitech sera un lieu d'échanges, de rencontres et de projets. Ce lieu permettra à la fois d'apporter une culture technologique et des spécialisations nouvelles. Contrairement aux Hubs Innovation qui existent dans l'ensemble des campus Epitech, Atelier by Epitech sera un lieu unique, à vocation nationale dont l'objectif est d'explorer et de tester : il sera résolument orienté PoC et prospective, intégrant l'approche deep tech qui fait la spécificité d'Epitech.



Cette structure sera intégrée au cursus Epitech. Un étudiant pourra remplacer un projet / un module par un projet réalisé dans Atelier by Epitech.

Un comité d'experts sera mis en place pour proposer les orientations technologiques qui seront mises en pratique dans Atelier by Epitech. Les entreprises peuvent d'ores et déjà contacter l'école pour des appels à projets à atelier@epitech.eu.

Epitech en quelques chiffres

- **14** campus en France ;
- + de **3000** entreprises partenaires ;
- **9500** étudiants dans le monde ;
- **100%** d'embauche à la sortie du Programme Grande École ;
- **7** campus à l'international.

Franck Franchin
Fondateur du Swiss Quantum Hub et de la société de conseil en technologie Voltanode. Il conseille de grands groupes suisses et français dans les innovations technologiques et en particulier en Informatique Quantique. Il a travaillé dans l'innovation et la cybersécurité chez Thalès, Vivendi et Orange. Il a aussi créé et revendu des startups dans les domaines de la cryptographie et des contenus numériques. Il est ingénieur diplômé de CentraleSupelec et titulaire d'un MBA de l'ESCP.

Informatique quantique et finance

La finance moderne a toujours été étroitement liée à l'évolution de l'informatique, des télécommunications et des mathématiques financières. L'Intelligence Artificielle et le Machine Learning sont d'ailleurs en train de profondément modifier les métiers de la banque et des assurances, en parallèle à leur numérisation accélérée ces dernières années. L'Informatique Quantique, nouvellement arrivée dans le paysage informatique, marquera-t-elle un changement de paradigme pour le monde des institutions bancaires et financières ?

Depuis quelques années, des établissements financiers comme Caixa Bank, Barclays, Morgan Stanley, Goldman Sachs ou BBVA investissent fortement dans des équipes spécialisées en algorithmiques quantiques et nouent des partenariats avec des universités et des startups. Quel peut-être l'impact de l'Informatique Quantique sur les domaines de la Finance, hormis les cryptographies post-quantiques ou la sécurisation des échanges grâce à la QKD (Quantum Key Distribution), qui ne sont pas l'objet de cet article ? Révolution ou évolution ?

Historique

Tout a commencé dans les années 70 avec l'arrivée des produits dérivés basés sur la célèbre formule de Black-Scholes-Merton, suivie par une multitude de méthodes et d'algorithmes mathématiques pour calculer les prix de ces produits et les risques liés.

Les produits dérivés sont des outils financiers, souvent considérés comme des outils de spéculation sur les évolutions des cours des devises, des actions, des indices ou des matières premières. Ils ne sont pas directement associés à des biens tangibles, mais à des sortes de contrats intangibles entre deux ou plusieurs parties. En général, leur valeur dépend d'un ou plusieurs actifs sous-jacents. L'intérêt spéculatif est lié à l'effet de levier : on ne détient pas l'actif sous-jacent, mais on spécule sur l'évolution à la hausse ou à la baisse de sa valeur. Les *options* sont les produits dérivés les plus connus du grand public.

Une option est un contrat qui donne à l'acheteur (le propriétaire ou le titulaire de l'option) le droit, mais non l'obligation, d'acheter ou de vendre un actif ou un instrument sous-jacent à un prix d'exercice spécifié avant ou à une date spécifiée, selon la forme de l'option. Le prix d'exercice peut être fixé par référence au prix au comptant (prix du marché) du titre ou de la marchandise sous-jacente le jour de la souscription d'une option, ou il peut être fixé à escompte ou à prime. Le vendeur a l'obligation correspondante de réaliser la transaction - vendre ou acheter - si l'acheteur (le propriétaire) *exerce* l'option. Une option qui confère au propriétaire le droit d'acheter à un prix spécifique est appelée un *call*; une option qui confère au propriétaire le droit de vendre à un prix spécifique est appelée un *put*.

Dans les années 80, il fallait encore disposer d'ordinateurs extrêmement puissants pour calculer les prix des produits dérivés. Les grandes entreprises possédaient donc un avantage concurrentiel majeur. Avant la crise des subprimes de 2008, le volume des échanges de produits dérivés était proche de 1 milliard de dollars par jour. Cette crise financière majeure et la faillite retentissante de Lehman Brothers anéantissent la (fausse) conviction d'une liquidité infinie. La chute spectaculaire de nombreuses banques montra alors que la couverture d'une opération de produits dérivés ne pouvait être parfaite que tant que les contreparties restaient solvables.

Le monde de la finance comprit (apprit ?) alors douloureusement qu'il existait un risque dans les produits dérivés et que les marchés libres ne s'autorégulaient pas. La réalité se dévoila bien éloignée de ce que les traders avaient cru apprendre dans les livres d'école.

Après la crise, le monde financier se tourna vers une meilleure compréhension des risques et vers la mise en place de barrières (de survie ?) pour limiter les manipulations de marché rendues possibles par le *trading automatisé*.

Le trading automatisé est une méthode d'exécution des ordres utilisant des instructions de trading préprogrammées et automatisées tenant compte de variables telles que l'heure, le prix et le volume pour envoyer de petites tranches d'ordres sur le marché au fil du temps. Ces méthodes ont été développées pour que les opérateurs, les fameux *traders*, n'aient pas besoin de surveiller constamment une action ou une obligation et d'envoyer ces ordres manuellement à plusieurs reprises. Le trading automatisé est largement utilisé par les banques d'investissement, les fonds de pension, les fonds communs de placement et les fonds spéculatifs (*hedge funds*), car ils doivent exécuter de gros passages d'ordre sur des marchés qui ne peuvent pas toujours supporter tous ces volumes à la fois. Ce terme est également connu pour désigner le trading algorithmique ou le trading en boîte noire. Ces stratégies englobent des stratégies de trading fortement tributaires de formules mathématiques complexes et de programmes informatiques qui tournent sur des calculateurs à haute performance (HPC).

Le trading automatisé a aussi évolué vers le *trading à haute fréquence* (High Frequency Trading - HFT) qui est un type de trading algorithmique automatisé caractérisé par des opérations très rapides qui s'appuient sur des données financières captées (échantillonnées) et analysées à haute fréquence. Le HFT peut être considéré comme une des formes principales du trading algorithmique. Il est basé sur l'utilisa-

tion d'outils technologiques sophistiqués et d'algorithmes informatiques pour négocier rapidement des titres. Le HFT utilise des stratégies de trading souvent propres à chaque établissement financier et permet d'évoluer sur des positions en quelques secondes ou fractions de seconde.

Est-ce que la leçon a été suffisamment et durement apprise par les loups de la finance internationale ? Est-ce que le trading automatisé est suffisamment contrôlé et encadré ? Point n'est ici le propos d'apprécier si les lois et régulations internationales remplissent correctement leurs missions de contrôle, mais force est de constater que le trading automatisé n'a fait que se développer depuis 2008.

Et le quantique dans tout ça ?

Commençons par rappeler pourquoi l'Informatique Quantique peut-elle être une technologie de rupture dans le calcul mathématique.

Quand l'informatique classique représente, stocke et traite les données sous forme binaire (un bit vaut 0 ou 1), l'Informatique Quantique manipule, elle, des *qubits* qui peuvent être dans n'importe quelle combinaison statistique de 0 et de 1 simultanément. C'est ce qu'on appelle l'état de *superposition*. Si on décide de travailler sur 10 qubits liés par l'*intrication* (une autre propriété quantique), on obtient une superposition de 2^{10} états, soit 1024. On a donc une croissance exponentielle du nombre possible d'états, en fonction du nombre de qubits.

L'idée d'utiliser la physique en ingénierie financière n'est pas nouvelle. Il n'est pas rare de trouver de bons vieux physiciens dans les équipes d'*analyse quantitative* des établissements financiers les plus renommés. De nombreux problèmes financiers peuvent s'exprimer directement sous des formes propres à la physique quantique. La fameuse formule financière de Black-Scholes-Merton peut d'ailleurs se lire dans l'esprit de l'équation de Schrödinger, chère aux physiciens.

L'analyse quantitative est un domaine des mathématiques appliquées qui traite de la modélisation mathématique des marchés financiers. En règle générale, les mathématiques financières utilisent des modèles mathématiques qui n'ont pas forcément de liens avec les théories financières ou économiques. Si un analyste financier va plutôt étudier les raisons structurelles pour lesquelles une entreprise peut avoir un certain cours de bourse, un mathématicien financier peut simplement prendre le cours de l'action comme une donnée et tenter d'utiliser le calcul stochastique pour obtenir la valorisation (*pricing*) des options sur cette action.

De nombreux problèmes en finances peuvent se réduire à des problèmes de type optimisation. Mais il existe aussi des problématiques complexes d'analyse de risques telles que la VaR (Value at Risk) et la CVaR (Conditional Value at Risk). La VaR est une mesure statistique des pertes potentielles d'un portefeuille et donc du risque associé. La CVaR est liée à la perte attendue pour un portefeuille pour des pertes plus grandes que la VaR. La prise en compte de la CVaR est plus conservative en termes de risque que la VaR.

Les données financières, de marchés et liées à l'économie sont excessivement complexes et leur analyse, même très poussée, ne mène pas nécessairement à des prévisions plus précises étant donné cette complexité et toutes les corrélations possibles et intrinsèques.

" En d'autres termes, il y a peu de chances que l'utilisation de l'Informatique Quantique réduise l'incertitude et les aléas des marchés."

Il suffit de se rappeler que le crash de 2008 est dû en grande partie à l'extrapolation automatisée des performances passées d'actifs de type prêts alors que la réalité des subprimes en 2007 était bien plus toxique.

De nombreux problèmes auxquels la communauté financière est confrontée n'ont pas de solution analytique connue et sont au contraire d'une grande complexité algorithmique et informatique. Pire, ils convergent lentement vers une solution sur les ordinateurs classiques. En particulier, l'évolution rapide des marchés génère une complexité supplémentaire pour la valorisation des options.

La principale évolution réside dans le temps de calcul nécessaire, car l'ordinateur quantique offre (parfois) une accélération exponentielle ou (souvent) quadratique pour certains algorithmes utilisés en Finance comme les simulations de Monte-Carlo (évaluation du risque, de l'incertitude et valorisation des produits dérivés).

Simulations de type Monte-Carlo

La méthode de Monte-Carlo est une technique qui permet d'estimer de manière stochastique (sur la base de *chaînes de Markov*) les propriétés d'un système, par échantillonnage statistique des états du système. C'est une méthodologie très utilisée en physique, en chimie et en ingénierie.

En Finance, l'approche stochastique est généralement employée pour simuler l'effet des incertitudes qui affectent un actif financier comme une action, une option ou un portefeuille de titres. Les méthodes de type Monte-Carlo sont ainsi applicables à l'évaluation de portefeuille, aux prévisions de gains et de rentabilité, à l'évaluation des risques, à l'optimisation des capitaux investis, aux stratégies de couverture (*hedging*) et au calcul des prix des produits dérivés (et donc des options).

Les algorithmes de type Monte-Carlo ont besoin d'un générateur de nombres aléatoires de la meilleure qualité possible afin d'éviter toute corrélation qui pourrait engendrer des résultats indésirables. C'est d'ailleurs l'une des toutes premières applications du Quantique en Finance grâce aux générateurs quantiques de nombres aléatoires (Quantum Random Number Generators - QRNG).

Considérons qu'on veuille échantillonner une distribution statistique (probabiliste) dont la variance est s^2 et la moyenne est m. La *loi des grands nombres* établit (inégalité de Chebyshev) qu'il suffit de prendre *k* échantillons (*k* étant fonction de s^2 et du taux d'erreur souhaité au carré) pour estimer m. Toutefois, si la distribution a une variance forte ou si on a besoin d'une erreur très faible, le nombre d'échantillons nécessaires peut devenir énorme.

Des chercheurs ont prouvé qu'il était pos-

sible grâce à des algorithmes quantiques de réduire le nombre d'échantillons *k* de manière quadratique, fonction de s et du taux d'erreur. Plusieurs méthodes et algorithmes quantiques ont vu le jour : Variational Monte-Carlo (VMQ) ou de Diffusion Monte-Carlo (DMC), Gaussian Quantum, Path Integral Ground State, Reputation, etc.

Dans le cas de Monte-Carlo, les algorithmes quantiques vont donc permettre de travailler avec beaucoup moins de données (échantillons). Il est ainsi possible de gagner typiquement 3 décades sur le nombre d'échantillons nécessaires, ce qui permet de passer d'un batch de nuit à du quasi-temps réel pour des prises de décision basées sur du Monte-Carlo. Outre cette meilleure réactivité face à une volatilité toujours accrue des marchés, il est clair que la possibilité de lancer plus souvent et plus vite des simulations ou des optimisations permet une meilleure allocation des capitaux disponibles. Toutefois, les simulations de type Monte-Carlo s'avèrent plus efficaces dans le cas de stratégies obligataires ou de taux qu'avec des stratégies liées à des actions ou à des options.

Les problèmes NP-complets

Certains problèmes en Finance sont considérés comme *NP-complets*, c'est-à-dire qu'on ne peut les résoudre que grâce à des méthodes heuristiques ou à des algorithmes d'approximation.

Pour cette classe de problèmes, il est possible de *vérifier* une solution en un temps polynomial, mais il n'existe pas de moyen connu pour *trouver* la ou les solutions, hormis en force brute (*brute force*). Cette dernière méthode s'avère en général impraticable dès que la taille du problème augmente. Un grand classique est le problème du voyageur de commerce.

" Hormis quelques rares exceptions algorithmiques, et malgré ce qu'on peut parfois lire sur le Net, l'ordinateur quantique n'est pas capable de résoudre des problèmes NP-complets en un temps polynomial."

Toutefois, des algorithmes tels que le Variational Quantum Eigensolver (VQE) et le Quantum Approximate Optimization Algorithm (QAOA) sont conçus pour résoudre les problèmes d'optimisation binaire quadratique sans contrainte (Quadratic Unconstrained Binary Optimization - QUBO). Il existe aussi de nombreux autres problèmes pertinents comme ceux d'optimisation binaire mixte (Mixed Binary Optimization - MBO), avec des variables discrètes et/ou continues, ou avec des contraintes qui ne peuvent pas être modélisées dans le cadre d'un problème QUBO, par ex. avec des contraintes d'inégalité. Il a été démontré que les ordinateurs quantiques surpassent les ordinateurs classiques pour ce type de calculs.

Quelques exemples d'applications financières

Nous allons maintenant donner quelques exemples de problématiques financières qui peuvent (pourront) trouver des optimisations grâce à l'Informatique Quantique.

Nous précisons « *pourront* » parce qu'en général il est nécessaire de disposer de bien plus de qubits logiques que n'en offrent aujourd'hui les ordinateurs quantiques de type NISQ.

Optimisation de portefeuille d'actifs

Tout gestionnaire de portefeuille d'actifs (Asset Manager) se pose les questions suivantes : quels actifs dois-je choisir pour constituer un portefeuille optimal ? Comment faire évoluer la composition de mon portefeuille par rapport aux évolutions des marchés ? Comment détecter de nouvelles opportunités ? Comment estimer le risque et le rendement de mon portefeuille ou d'un de ses actifs ?

Une approche consiste à rechercher quel est le portefeuille d'actifs qui minimise le risque pour un rendement donné. Ou pour un risque donné, le portefeuille qui maximise le rendement.

L'allocation optimale d'un capital donné à un ensemble d'actifs est un des grands problèmes classiques des mathématiques financières, connus sous le terme de problème de programmation quadratique (et non linéaire). Le but est de trouver la meilleure combinaison rendement/risque pour un portefeuille donné et avec des contraintes spécifiques. On parle aussi d'optimisation de la moyenne-variance.

Plusieurs algorithmes théoriques existent, mais leurs applications dans le monde réel se heurtent à des considérations pratiques importantes. Dans la plupart des algorithmes proposés, il est nécessaire d'inverser une matrice de covariance liée à des données aussi incertaines que celles de marchés très actifs. Une autre solution basée sur la notion de parité des risques (Inverse-Variance Parity - IVP) permet de s'affranchir de la matrice de covariances entre actifs et de travailler uniquement sur la variance de chaque actif. Il reste toutefois important d'apprécier les corrélations entre les actifs.

Un des algorithmes possibles consiste à utiliser un arbre de classification hiérarchique (Hierarchical Risk Parity – HRP) basé sur la matrice de covariance des rendements des actifs. La démarche s'articule autour de la notion de structure hiérarchisée des actifs : une action se comporte différemment qu'une obligation, une action d'un opérateur télécom que celle d'une petite société de produits de luxe, etc.

C'est le rôle de la matrice de corrélation d'exprimer les différences entre les actifs. Il est alors possible de regrouper dynamiquement des actifs sous forme de clusters, la structure hiérarchisée de l'arbre. Lorsque des actifs sont proches, on les regroupe dans le même cluster. On peut aussi regrouper deux clusters similaires pour former un nouveau cluster. Il suffit ensuite d'associer des poids à chaque cluster à partir de la variance de chaque cluster contenu et de la matrice de covariance.

Une version quantique de l'algorithme HRP existe (QHRP) et s'avère être un problème NP-complexe qui peut s'exprimer sous la forme d'un problème de type QUBO.

Cette méthode a été expérimentée avec succès sur un petit portefeuille typique d'une cinquantaine d'actifs de commodités (pétrole, blé, or, actions, obligations) à corrélation faible et sur les 30 titres (variables) qui composent l'indice Dow Jones Industrial Average (DJIA) à corrélation forte.

Ces tests de l'algorithme QHRP ont prouvé qu'il était plus efficace dans l'allocation d'actifs que les méthodes classiques (variance minimale ou MV, IVP, HRP). Par

exemple, la volatilité annualisée via la méthode MV s'est avérée trois fois plus élevée de celle via la méthode QHRP.

Arbitrage

L'arbitrage est un terme générique utilisé en Finance pour représenter un choix entre différents moyens de générer un profit sur le même actif dont le prix est différent sur des marchés différents.

Par exemple, il est possible que l'opération de change suivante génère un profit, même faible, net des coûts de transactions : conversion de dollars (USD) en Francs Suisses (CHF), puis conversion des CHF en Euros (EUR), puis conversion des EUR en USD.

La modélisation mathématique pour trouver au moins une opportunité d'arbitrage est bien connue et s'appuie sur la théorie des graphes. Les algorithmes de Bellman-Ford et de Floyd-Warshall permettent de résoudre ces problèmes selon une complexité polynomiale.

Par contre, trouver la meilleure opportunité d'arbitrage, celle qui génère le profit le plus important, s'avère bien plus difficile, car c'est un problème de type NP-complet à résolution exponentielle et non polynomiale. Le Quantum-Annealing, prôné par la société D-Wave, se prête bien à ce type de problème grâce aux algorithmes de type VQE et QUBO.

Swaps

Un swap est un contrat pour échanger des flux financiers à des dates spécifiques selon un cadre particulier. Par exemple, le processus de compensation d'un swap entre la contrepartie A et la contrepartie B correspond à la transformation du contrat initial en deux contrats distincts : l'un entre la contrepartie A et le compensateur, l'autre entre le compensateur et la contrepartie B. Ainsi, le compensateur protège chacune des contreparties de tout risque de défaillance de l'autre. Le compensateur détient le collatéral des deux contreparties, évalué selon la valeur brute des flux financiers sous-jacents.

Il est tout à fait possible que le compensateur doive garantir des flux financiers pour des durées très longues (jusqu'à trente ans). Le compensateur peut « netter » une partie ou la totalité des flux financiers afin de réduire son exposition aux risques. Les contreparties pourront ainsi reprendre l'usage du capital qui était immobilisé dans des comptes de marge. La forme la plus simple du *netting* consiste à compenser des flux identiques, mais il est aussi possible de compenser entre eux plusieurs flux (des chaînes de flux) même s'il reste un petit montant résiduel.

Il peut s'avérer aussi pertinent de combiner la compensation de plusieurs flux dont les termes économiques sont différents, par exemple certains avec des taux d'intérêt fixes, d'autres avec des taux d'intérêts variables ou indexés, voire les deux types combinés pour le même contrat.

Trouver ces combinaisons de netting permet à l'évidence une utilisation optimale des capitaux disponibles.

Le Quantum-Annealing s'applique bien aussi à ce type de problème grâce au QUBO.

Marché des commodités

Le marché des commodités est aussi un champ d'application très intéressant pour l'Informatique Quantique.

Le commerce des commodités est souvent basé sur des segments temporels de 3, 6 ou 12 mois, dont la complexité et la durée ne facilitent pas les cycles d'arbitrage et la détection du meilleur arbitrage.

En particulier, le marché du pétrole brut et du gaz est très dynamique et implique des stratégies complexes d'achat/vente et de couverture des cours.

Généralement, les entreprises impliquées utilisent des outils de couverture comme les futures, les options, les swaps, etc. Quand le marché est orienté à la hausse (*bullish*), elles font appel à des options de type *call* pour se protéger de la hausse des prix. Si, au contraire, le marché est à la baisse (*bearish*), elles emploient des options de type *put*.

Le monde de l'analyse quantitative dispose depuis longtemps d'algorithmes pour le calcul ou l'estimation de la volatilité implicite (IV) d'un contrat d'options sur la base de simulations de type Monte-Carlo et du fameux modèle de Black-Scholes-Merton. L'analyse prédictive et les modèles de prix dérivés sont utilisés quotidiennement, mais se heurtent à la taille importante des modèles de données ou aux limites de temps de calcul imposées par la dynamique des marchés.

Ces algorithmes ont été implémentés avec un certain succès en Informatique Quantique ce qui laisse entrevoir pour un futur proche des gains en efficacité et en réactivité sur ces marchés.

Credit scoring

L'attribution erronée de prêts à des entreprises ou à des particuliers est une des causes majeures des pertes voire des défaillances des établissements financiers. Les experts s'entendent pour considérer que 2 % des prêts accordés, toutes catégories confondues, sont défaillants. Pourtant, aux USA, environ 40 % des prêts aux particuliers sont refusés.

Le *credit scoring* est donc un facteur clé de succès pour les banques et les organismes de crédit.

Une nouvelle fois, il apparaît que certains problèmes du credit scoring sont de type QUBO et se résolvent mieux avec l'Informatique Quantique qu'avec les méthodes classiques de régressions, d'arbres de décision, de réseaux de neurones ou de classificateurs.

Quel futur ?

Plusieurs grandes banques internationales ont déjà des équipes qui travaillent sur ces concepts d'algorithmiques quantiques en collaboration avec des sociétés expertes comme IBM, Ernst&Young, D-Wave ou des startups spécialisées.

Leur but est de comprendre le potentiel futur de l'Informatique Quantique, de travailler sur ces algorithmes spécifiques, d'identifier les différentes technologies et de créer des liens avec les labos et les startups.

Le meilleur moyen pour apprendre (la fameuse *learning curve*) consiste à monter des « use cases » et des « PoCs ». L'expérimentation est la clé. Programmer un ordinateur quantique ne ressemble à rien de ce qu'on connaît déjà.

Les premiers à adopter ces nouvelles technologies disposeront d'avantages clés dès lors que l'Informatique Quantique passera à l'étape suivante, après les ordinateurs de type NISQ, c'est à dire avec des systèmes quantiques à tolérance aux erreurs et avec des milliers de qubits.

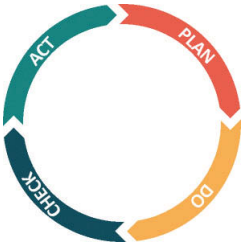
Emeric MARTINEAU
Consultant DevOps @ Zenika Nantes
Je travaille dans l'informatique depuis 2001. Après plusieurs métiers dans l'IT, je me suis spécialisé depuis 2014 sur les sujets autour de la fabrication, livraison, exploitation des logiciels.

De l'agilité au DevOps : retour sur leurs origines

Les gens s'interrogent souvent sur la différence entre agilité et DevOps. Est-ce différent ? Est-ce complémentaire ? Prenons le temps de refaire l'historique de l'agilité, des méthodes de gestion de projet, du DevOps pour mieux comprendre quelles sont les problématiques adressées et quelles solutions sont apportées. Un sujet comme celui-là, pour être traité dans sa globalité, nécessiterait plusieurs tomes. J'encourage le lecteur à prendre les sujets abordés, à se renseigner par lui-même, et se faire sa propre opinion.

L'agilité

Les premiers travaux que l'on retrouve dans le concept d'agilité viennent de Walter A. Shewhart, physicien et statisticien américain, qui travailla toute sa carrière au sein de *Bell Telephone Laboratories*. Il mit au point *la théorie des variations* exposée dans un premier livre en 1931. Selon lui, la recherche de la qualité est proche de la recherche expérimentale(1).



Cette idée est à l'origine du cycle PDCA (Plan, Do, Check, Act)(2) présenté par W. Edwards Deming, statisticien, professeur d'université, auteur et consultant américain aux industriels japonais dans les années 50, que l'on traduit souvent par :

- **Plan** : préparer, planifier ;
- **Do** : développer, réaliser, mettre en œuvre ;
- **Check** : contrôler, vérifier ;
- **Act** : agir, ajuster, réagir.

Le but est de s'ajuster en permanence pour s'améliorer de façon continue.

En 1970, Alvin Toffler, écrivain, sociologue et futurologue américain, présente un concept novateur dans son livre *Le Choc du futur* : l'adhocratie.

L'adhocratie est l'opposé de la bureaucratie. C'est une organisation qui mobilise, dans un environnement instable et complexe, des compétences multiples et transverses.

Les équipes ont une autonomie importante, les ajustements sont mutuels, le management est souple. On doit la popularisation du concept à Robert Waterman Jr, un consultant et auteur américain, spécialiste du management dans les années 90.

Sept ans après *Le Choc du futur*, John Kenneth Galbraith présente le concept de *Design Organisationnel* qui cherche à mettre en cohérence(3) :

- la stratégie de l'organisation ;
- l'organisation du travail ;
- la coordination des unités et du personnel.

Le but est d'aligner l'entreprise aux caractéristiques économiques, technologiques et sociologiques de leur environnement.

La mondialisation des marchés, l'accélération des innovations techniques et le changement de l'économie nécessitent une flexibilité stratégique. Pour y répondre, la décentralisation des prises de décision, la réduction des niveaux hiérarchiques, un management par le leadership et l'auto-organisation des équipes vont permettre l'utilisation stratégique de la vitesse et des premières expériences.

C'est ce que présentent sous le nom d'*organisation nouvelle* Leblond et Paquet en 1988(4).

En 1991, Roger N. Nagel, qui est considéré comme le père du mot Agile(5), Steven L. Goldman, Kenneth Preiss(6) et Rick Dove(7) théorisent l'entreprise agile comme :

- la vente de « solutions totales » mélangeant biens, services et informations ;
- un haut niveau de différenciation ;
- un prix de série ;
- un degré d'exigence occidental.

On retrouve aussi ces éléments dans le livre *Théorie de l'entreprise agile* d'Olivier Badot en 1998.

Toyota, une source d'inspiration de l'agilité

Toyota est souvent cité comme référence dans le monde de l'agilité. En effet, deux principes nés chez Toyota se rencontrent régulièrement dans l'informatique.

Kanban

Kanban a été inventé dans les années 50. Le but est d'améliorer le flux de production, en se basant sur un flux tiré, c'est la demande du client (sa consommation) qui va donner le rythme de la production. Afin de ne pas engorger un poste en attente, le nombre de tâches en cours est limité. Cela va permettre de réduire les stocks ou encore d'optimiser les capacités humaines et techniques.

(1) http://www.fr-deming.org/theovar.pdf

(2) *Par la suite W. Edwards Deming renomma PDCA en Plan, Do, Study, Act.*

(3) https://www.strategie-aims.com/events/conferences/23-xxiieme-conference-de-l-aims/communications/3041-quels-nouveaux-fondements-theoriques-pour-un-nouveau-design-organisationnel/download

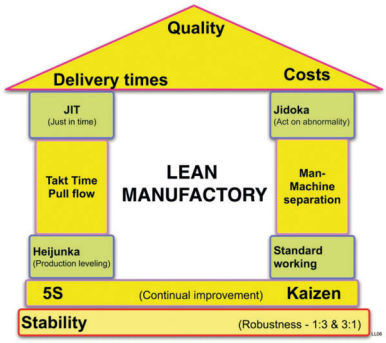
(4) *Stratégie vulpine et structure modulaire dans une ère de tohu-bohu André LeBlond, Gilles Paquet, Université d'Ottawa. Faculté d'administration*

(5) https://www.myagilepartner.com/blog/index.php/2019/03/07/the-origin-of-agility-existed-before-the-agile-manifesto/
https://blog.myagilepartner.fr/index.php/2017/10/05/l-origine-de-l-agilite-bien-avant-le-manifeste-agile/

(6) *Agile Competitors and Virtual Organizations: Strategies for Enriching the Customer, novembre 1994 Steven L. Goldman, Roger N. Nagel, Kenneth Preiss*

(7) *Response Ability: The Language, Structure, and Culture of the Agile Enterprise - Rick Dove* https://web.stevens.edu/facultyprofile/?id=714

Lean

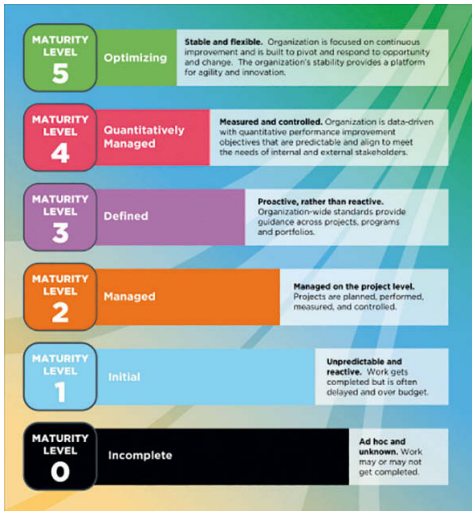


© Crédit Laurens van Lieshout

Le Lean trouve son origine dans le système de production de Toyota. Son but est la recherche de la performance (productivité, qualité, délais, coûts), le tout en utilisant l'amélioration continue et la suppression du gaspillage. Le principe du Lean est né dans un pays en ruine, à la fin de la Deuxième Guerre mondiale. Toyota devait faire face à un grand nombre de pénuries. Le marché de l'époque exclut les grandes séries et un grand nombre de concurrents. Dans la partie "Agilité", on se rend compte que l'entreprise agile s'est principalement axée sur l'optimisation de l'organisation, du management, des décisions, de la stratégie.
Toutefois, Toyota a pris aussi en compte les aspects techniques comme clefs du succès.

Capability Maturity Model Integration

Pour les besoins du département de la Défense des États-Unis (DoD), le SEI (Software Engineering Institute) présente en 1991 le Capability Maturity Model (CMM). Il s'agit d'un modèle de référence qui ne concerne que les bonnes pratiques du génie logiciel. Ceci afin de mesurer la capacité des projets à s'achever correctement en termes de délais, de fonctionnalités et de budget.
En 2001 sont ajoutées les bonnes pratiques d'autres modèles, sauf la gestion des ressources humaines. En 2006, un nouveau modèle plus simple et intégrant les composants de type matériel est publié. La dernière version date de 2011.
Dans ce référentiel, le CMMI-DEV est un modèle de processus pour réaliser des produits. Il est utilisé dans le développement et la maintenance de logiciel. Sa portée s'étend de l'apparition d'un besoin jusqu'à la livraison du produit correspondant.



Crédit cmmi institute

CMMI est un modèle d'amélioration continue qui définit cinq niveaux de maturité. Les niveaux permettent de savoir où en sont les projets ou les entreprises.
L'amélioration continue rejoint les principes vus précédemment de l'agilité. Toutefois, CMMI va aussi à l'encontre de l'agilité, en réduisant par exemple l'autonomie des équipes.
CMMI se focalise aussi beaucoup sur la standardisation, les normes, les processus et la documentation, au risque de détériorer la qualité (du produit, service, environnement de travail), qui est paradoxalement le but premier de CMMI(8).
Pour augmenter de niveau, les entreprises sont tentées de mettre en place toute une série de cellules de vérification ou de comités de validation… ce qui peut aller à l'encontre de l'agilité.
Il existe de nombreux autres problèmes(9) créés principalement par la mise en œuvre de CMMI et non par CMMI lui-même.
Pour l'infrastructure et les opérations dans le domaine du logiciel, un autre référentiel est très répandu. Il s'agit d'ITIL.
Dans une même entreprise, les deux référentiels peuvent cohabiter, générant parfois quelques incompréhensions.
Comme indiqué au début, CMMI est avant tout, au départ du moins, un recueil de bonnes pratiques. L'article *Synergies entre CMMI et les Méthodes Agiles*(10) semble montrer qu'un certain nombre de personnes ont vécu CMMI plus comme une contrainte, une rigidité de l'entreprise. CMMI a eu le mérite d'essayer de regrouper les différentes pratiques et expériences dans un but clair : apporter de la valeur ajoutée au client.

Information Technology Infrastructure Library

ITIL est un ensemble d'ouvrages regroupant les bonnes pratiques du management du système d'informations apparu en 1980 et rédigé à l'origine par l'Office public britannique du Commerce. Depuis la version 3, d'autres acteurs y contribuent, notamment des sociétés de services.

ITIL comprend cinq grands points :
- Stratégie des services ;
- Conception des services ;
- Transition des services ;
- Exploitation des services ;
- Amélioration continue des services.

ITIL, comme CMMI, semble être compatible avec l'agilité. Toutefois, avec le nombre de processus (5 domains, 26 processus), de documentations qu'il semble mettre en oeuvre, cela peut poser question… Est-ce que la mise en œuvre d'ITIL a tendance à rigidifier l'entreprise et à la focaliser plus sur les processus que sur la qualité ou le service rendu ?
ITIL apporte un nombre important d'améliorations(11) lorsqu'une entreprise l'adopte. ITIL permet une vue globale d'un produit informatique, du développement à l'exécution.
C'est aussi un apport conséquent dans le domaine de

(8) https://rollout.io/blog/cmmi-dead-yet/
(9) http://www.volle.com/travaux/cmmi.htm
(10) https://blog.octo.com/synergies-entre-cmmi-et-les-methodes-agiles/
(11) https://www.itiltraining.com/blog/2017/01/17/top-10-itil-benefits/

l'infrastructure informatique avec par exemple la mise en place de CMDB(12). Toutefois, côté développement informatique (développeur, MOA, MOE), ITIL semble avoir été mis de côté. ITIL a aussi généré de la frustration côté infrastructure(13) (14).
Un défaut est peut-être, comme CMMI, d'avoir finalement organisé et justifié le cloisonnement de l'entreprise et généré beaucoup de frustration. En effet, on remarque souvent, une plus grande difficulté de coopération entre deux personnes de services différents qu'entre deux personnes d'un même service(15).
Mais ITIL est dans certaines entreprises un succès. Ce succès est peut-être moins diffusé. Le document *Bilan de 4 ans d'ITIL à l'Université de Strasbourg*(16) en est un exemple. Il analyse en détail ce qui est bien dans ITIL, ce qui l'est moins et surtout les raisons du succès ou de l'échec d'ITIL. La phrase suivante qui se trouve dans la conclusion résume bien la dynamique autour de l'adoption d'une démarche :

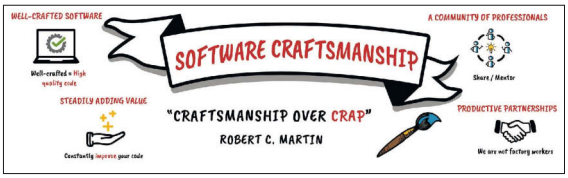
> *« Fait paradoxal, si beaucoup rejettent ITIL, vécu comme une «religion» ou une idée completement décorrélée du «terrain», bon nombre sont familiers et adherent aux concepts et processus véhiculés par ITIL. Il ne faut donc pas se focaliser sur l'adoption d'ITIL en tant que tel mais plutôt sur les processus mis en œuvre. »*

La version 4 d'ITIL, publiée en février 2019, s'est allégée et prend en compte le Lean, l'agilité et le DevOps.

Développement agile

Durant les années 1990 plusieurs méthodes de développement logiciel font leur apparition (RAD, UP, DSDM, XP). Elles ont pour point commun de mettre de côté le micro-management, des planifications de plusieurs semaines à plusieurs mois, et d'autres aspects très contractuels. À cette époque, la méthode la plus courante était la méthode dite *Cycle en V* où il fallait attendre le cahier des charges, les spécifications générales, puis détaillées pour commencer à développer. Entre le début du projet et l'écriture de la première ligne de code, il pouvait s'écouler plusieurs mois, voire plusieurs années.
La conséquence était de générer beaucoup de frustration pour les clients car leurs besoins avaient souvent évolué (ou le cadre réglementaire) entre le début de rédaction des spécifications et le début du développement. Cette frustration était bien souvent partagée par les développeurs car il était presque impossible de changer quelque chose dans les spécifications ou le planning.

Software craftsmanship



Crédit agilepartner

En 1992, Jack W. Reeves publie un essai *What Is Software Design?*(17) où il suggère que le développement se rapproche plus du travail d'artisanat que du travail d'ingénierie.
En effet, si vous reprenez les précédentes parties, il y a une forte notion de production de série.
Sept ans plus tard, le livre *The Pragmatic Programmer: From Journeyman to Master*(18) est publié. Les auteurs y expliquent que les développeurs passent par des étapes, un peu comme les traditions médiévales ou même le compagnonnage.
En 2001, le livre *Software Craftsmanship: The New Imperative*(19), émet l'idée que le développement logiciel a besoin d'une approche différente que celle de l'ingénierie traditionnelle.
Un manifeste(20) sera publié en 2009.

Crédit agilepartner

Le software craftsmanship a permis de populariser les notions suivantes.

Le refactoring de code et la qualité de code

Le code est quelque chose de vivant. Les développeurs sur un projet changent et leurs expériences sont différentes.
Des évolutions de périmètres et de fonctionnalités vont arriver.
Pour permettre cette dynamique avec le moins de difficultés possible, le refactoring va être essentiel pour rendre un code plus lisible, permettant ainsi à un nouveau développeur de comprendre rapidement le code, ou de préparer l'arrivée de nouvelles fonctionnalités.
Des outils comme SonarQube, les linters, Checkmarx permettent d'aider le développeur dans ces tâches.

(12) https://fr.wikipedia.org/wiki/Configuration_management_database
(13) https://community.spiceworks.com/topic/624406-itil-for-sysadmin
(14) https://community.infosecinstitute.com/discussion/79505/help-me-with-itil-and-sysadmin
(15) https://www.sage.com/fr-ca/blog/9-conseils-pour-encourager-la-collaboration-entre-les-services/
(16) https://2013.jres.org/archives/6/paper6_article.pdf

(17) http://www.developerdotstar.com/mag/articles/reeves_design.html
(18) Andy Hunt & David Thomas - 1999.
(19) Pete McBreen - Août 2001
(20) http://manifesto.softwarecraftsmanship.org/

L'intégration continue

Le principe de l'intégration continue est que chaque fois qu'une modification est faite sur du code et mise du gestionnaire de source, le logiciel va être construit et un certain nombre de vérifications va être opéré.

Cela permet de détecter au plus tôt d'éventuelles erreurs et de permettre qu'un logiciel puisse être livré en permanence.

Ainsi, s'il faut livrer une correction urgente (faille de sécurité, problème qui menace l'intégrité des données…), l'intégration continue va permettre d'éviter de nombreuses inquiétudes.

Il y a de très nombreux outils pour cela, le plus connu étant certainement Jenkins. Mais d'autres solutions comme Concourse-CI méritent d'être testées.

Test Driven Development

Le TDD(21), c'est l'idée de commencer à écrire les tests en premier, avant le code, idéalement par une personne différente de celle qui va écrire le code.

On va écrire les tests des cas d'usage principaux, et ensuite écrire seulement le code nécessaire pour faire passer les tests.

Et encore plus...

La liste des bonnes pratiques est encore longue et passionnante. Car apprendre des choses c'est bien, les partager c'est fondamental. Le partage fait aussi partie du software craftsmanship. Pour compléter le sujet, il vous faudra approfondir les sujets suivants :

- Design Simple
- Conception objet
- Technique de testing
- Clean Code
- Refactoring
- Principes SOLID

Le manifeste agile

Crédit Loic Leofold

2001 est souvent pris comme référence de la naissance de l'agilité. Même si comme on l'a vu, l'agilité est née avant.

La publication cette année-là et l'adoption ensuite par des entreprises à succès comme Google, Amazon, Facebook, du manifeste agile va populariser le terme et les pratiques.

Les 4 valeurs et les 12 principes du manifeste sont très souvent cités. Pour bien les comprendre, il est très important de retenir la phrase d'introduction :

> *« Nous découvrons comment mieux développer des logiciels par la pratique et en aidant les autres à le faire. »*

Cette phrase permet de donner le sens de lecture du manifeste.

Dans le manifeste on trouve des principes qui doivent nous orienter.

> *« Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée. »*

Cela nous rappelle les idées évoquées dans l'agilité : prise de décision rapide. En informatique, livrer régulièrement du code passe obligatoirement par de l'automatisation la plus poussée possible.

> *« Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts. »*

Comme dit précédemment, pour livrer fréquemment il faut de l'automatisation. Et livrer fréquemment un logiciel opérationnel nécessite une chaîne fiable et automatisée du développement à la production, en passant notamment par la phase de recette.

Pour faire ça, tous les acteurs vont être mobilisés : métier, développeur, testeur, exploitant, administrateur système...

> *« Un logiciel opérationnel est la principale mesure d'avancement. »*

Pour faire un logiciel opérationnel, un code bien écrit, respectant les bonnes pratiques (software craftsmanship), un haut niveau de monitoring, une grande collaboration est nécessaire.

Il faut aussi définir une stratégie de test(22) et pour cela un spécialiste du test(23) est fort appréciable.

Crédit Growing Agile

« Fournissez-leur l'environnement et le soutien dont ils ont besoin et faites-leur confiance pour atteindre les objectifs fixés. »

On retrouve ici le concept de nouvelle organisation de Leblond et Paquet : un environnement, où les gens peuvent faire des erreurs et apprendre de celles-ci, avec un bon niveau d'autonomie.

> *« Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant. »*
>
> *« Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet. »*

(21) https://martinfowler.com/bliki/TestDrivenDevelopment.html

(22) https://www.all4test.fr/dossiers-thematiques/rediger-une-strategie-de-test/

(23) https://fr.wikipedia.org/wiki/ISTQB

Le principe d'embarquer tout le monde tout le temps est ici évoqué, même si on a l'impression que c'est limité à l'équipe de développement. Une notion également importante est un rythme constant et durable.

Que nous apprend ce rapide tour du manifeste agile ?

Le manifeste agile pose un premier cadre d'application de l'agilité dans l'informatique. Une forte orientation pour le développement est donnée, sans pour autant oublier les autres parties qui interviennent dans la vie du logiciel. Le manifeste énonce des principes. Beaucoup de personnes ont retenu l'axe de l'agilité tourné vers le client, en oubliant l'axe orienté vers les développeurs, même si dans les principes il est mentionné l'excellence technique. De nombreux développeurs ont la sensation que la technique est reléguée au fond du tiroir, et c'est un élément régulièrement cité pour expliquer qu'une application devient non maintenable, une cause de lassitude, de désengagement.

De plus, le développement logiciel a changé ses méthodes et ses fréquences de livraison mais la partie exploitation, infrastructure, a suivi un autre rythme.

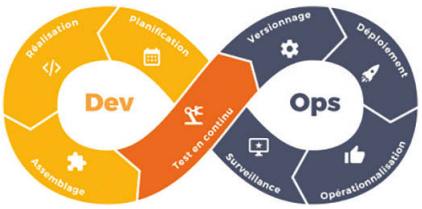
C'est un peu ce qui se passe avec la voiture électrique, où il existe plusieurs types de prises(24) mais les stations de recharge peinent à toutes les proposer.

Tout ceci a créé des tensions et un autre besoin s'est exprimé, c'est ce que nous allons voir ensuite.

L'article sur *Vert Citron* intitulé *Software Craftsmanship et Agilité*(25) résume bien le manque des méthodes agiles :

> « *Cependant, si les méthodes agiles donnent un cadre qui permet des apports essentiels, elles ne garantissent pas la qualité du produit. Tout au plus le développement par les tests permet de conserver une cohérence, mais tant que ça passe, rien n'empêche de coder n'importe comment…* »

DevOps



Crédit snapp

Le DevOps est né de la frustration aussi bien côté développeur qu'administrateur système.

En 2008, une conférence *Agile Infrastructure* doit se dérouler mais… il n'y a qu'un spectateur, Patrick Debois. L'orateur annule sa présentation. Toutefois les deux personnes discutent dans les couloirs de la conférence et forment *Agile Systems Administration Group*. En 2009, John Allspaw et Paul Hammond donnent une conférence : *10+ Deploys a Day: Dev and Ops Cooperation at Flickr(26)*, conférence devenue célèbre par la suite, où les auteurs expliquent les bénéfices de la collaboration entre les Dev et les Ops.

(24) https://www.automobile-propre.com/dossiers/recharge-voitures-electriques/

(25) https://www.vertcitron.fr/craftsmanship-et-agilite/

(26) http://fr.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr

Patrick Debois, ne pouvant assister à la conférence, sur une idée de Paul Nasrat, organise la même année la même conférence en Belgique. Toutefois, le nom *Agile Systems Administration Group* étant trop long, il prend les 3 premières lettres de développement et de opérationnels (admin' sys') : la conférence *DevOpsDays* est née.

Le constat

Imaginons-nous être un développeur :

- Pourquoi faut-il attendre plusieurs jours, plusieurs semaines ou plusieurs mois pour mettre en production le développement terminé puisque tous les tests sont validés, et que le métier a validé fonctionnellement ?
- Pour que l'application arrive en production, il faut suivre tout un processus de validation qui prend beaucoup de temps. Quelles en sont les raisons ?
- Il est parfois nécessaire de tester le logiciel dans un environnement iso-production, mais il est difficile de l'obtenir. Pourquoi ?

Imaginons-nous être un exploitant :

- Comment se fait-il que le livrable que je reçois ne fonctionne pas sur la plateforme de production alors qu'il fonctionne ailleurs ?
- Comment je fais pour surveiller cette application ? Comment je vois les ressources qu'elle consomme ?
- Pourquoi me livre-t-on tous ces fichiers ? Et c'est quoi ça ?
- J'ai bien un fichier de configuration, mais quelles valeurs je dois mettre ici ?

À toutes ces questions, il y a des réponses. Mais, le développeur, comme l'exploitant doivent-ils se poser ces questions ?

Chacun semble ignorer le besoin de l'autre, chacun étant fixé sur ses objectifs. Mais les objectifs sont-ils compatibles ?

Aux développeurs, il est demandé d'intégrer des fonctionnalités le plus rapidement possible et de livrer dès que c'est prêt.

Aux exploitants, il est assigné la mission de la garantie de service. Un service fonctionnant 24h/24, 7j/7 et sans perte de données.

Mais quand un changement intervient, il y a un risque que le service soit interrompu.

Comment résoudre ce dilemme ? La solution semble être d'avoir un objectif commun.

Qu'est-ce que le DevOps ?

Le DevOps est un référentiel de bonnes pratiques en constante évolution basées sur les pratiques agiles et Lean, porté et alimenté par une communauté.

L'objectif est de livrer des logiciels plus rapidement, de façon plus efficace, tout en garantissant la meilleure qualité et ce en adéquation avec les besoins de l'entreprise et de ses clients.

Ainsi, le mouvement DevOps donne un cadre aux différents acteurs qui interviennent dans le développement logiciel pour apporter plus de valeur aux utilisateurs, clients, fournisseurs et partenaires.

DevOps est un courant de pensée, un état d'esprit tout comme l'agilité. DevOps ne se limite pas, et ne doit pas se limiter, à l'adoption d'un ensemble d'outils et de processus.

Sa mise en place nécessite de la volonté et souvent un changement organisationnel profond, car une organisation DevOps se remet en question constamment.

Pourquoi adopter la démarche DevOps ?

Le changement, coûte de l'argent, du temps. Alors, ce changement doit être rentabilisé à son maximum. Pour éviter de faire des changements inutiles, il est important que ces changements soient motivés par la valeur créée en retour.

Il faut veiller à rendre les processus les plus naturels possibles en supprimant ceux qui sont inutiles. Plus les cycles seront courts, plus les personnes y seront habituées et plus elles seront efficaces.

Pour gagner des parts de marché, être plus compétitifs (en termes de prix comme de fonctionnalité), il faut réduire le temps entre le début du développement et l'arrivée sur le marché du produit (Time to Market (TTM)). Pour cela, il va falloir essayer souvent, et accepter de faire des erreurs, en transformant ces dernières en expériences enrichissantes.

La bonne CAMS

La démarche DevOps définit 4 valeurs fondamentales, regroupées sous l'acronyme CAMS(27) (ou SMAC si vous avez besoin de tendresse) :

- Culture : valoriser l'humain en tant qu'élément principal ;
- Automatisation : l'automatisation pour fiabiliser les produits et simplifier les processus ;
- Mesure : mesurer pour avoir des éléments factuels et non des impressions ;
- Share : partager la confiance et les compétences.

Les pratiques qui émergent avec le DevOps

La mise en place du DevOps doit normalement faire émerger un ensemble de pratiques telles que le développement et les tests sur des environnements le plus proche possible de la production.

Déployer des applications à travers des processus automatisés, fiables et répétables, doit devenir un non-évènement.

Les tests doivent être automatisés et faire partie intégrante de la conception du logiciel (plus de « *les tests on les fera plus tard quand on aura le temps* »).

Toute l'équipe (métiers, développeurs, exploitants) doit avoir l'ensemble des métriques pour leur permettre de prendre les décisions sur des éléments tangibles.

Chacun doit pouvoir faire son retour sur un élément et que ce retour soit considéré.

Tout ce qui est automatisable facilite la vie

Dans une organisation, un temps non négligeable est consacré à faire remplir des formulaires pour demander du matériel, des ressources… Il faut ensuite attendre qu'une personne valide selon des critères connus.

Pour peu qu'il y ait une erreur dans le formulaire ou qu'il manque une information, et il y a des allers-retours du formulaire.

Tout ceci pourrait en grande partie être simplifié et plus rapide grâce à l'automatisation. Pour ce faire, il est impératif que les éléments soient interconnectés et qu'ils exposent des API publiques. Prenons un exemple pour illustrer le propos(28) : la création de serveur (virtual machine).

Dans certaines entreprises, pour faire cela, il faut :

- un code projet, ;
- que la demande émane du chef de projet ou du directeur de projet ;
- le type de VM (qu'il faut aller regarder dans un catalogue, un fichier word sur le réseau) ;
- il faut envoyer la demande via une boîte e-mail générique à l'équipe infrastructure ;
- côté infrastructure un chef de projet va être nommé ;
- une réunion est organisée entre les deux chefs de projets car il faut motiver la demande ;
- une date de livraison est décidée ;
- il faudra attendre que la VM soit prête.

Enfin, vous avez votre VM… Mais aucun compte pour y accéder… C'est une autre équipe qu'il faut contacter… Seulement après que la VM ait été livrée.

Vous recommencez donc le même cycle et enchaînez avec l'équipe qui s'occupe des habilitations (pour faire le lien entre le compte et la VM) puis l'équipe qui s'occupe des mots de passe des comptes (car oui, le mot de passe ne vous est pas livré avec le compte).

Afin d'optimiser ce processus, il est nécessaire d'exposer des API et gérer son infrastructure as code(29).

En synthèse

Depuis longtemps, les organisations cherchent à améliorer leur efficacité. Le sujet est très compliqué et complexe et chaque méthode a essayé d'améliorer la précédente ou de l'adapter à un contexte différent.

Concernant CMMI et ITIL, on peut se demander si ces méthodes sont aptes à répondre à l'indétermination :

> « *Il faut accepter l'idée que la vie naturelle et sociale ne peut être totalement maîtrisée par des règles établies à l'avance. Seule la compétence augmentée et la coopération hautement fiable sont de bonnes réponses à l'indétermination.* »(30)

Une autre interrogation se pose sur le fait que des entreprises qui ont un intérêt financier participent aux rédactions de normes, de processus, des méthodes et qui proposent aussi des certifications :

> « *Un facteur important de l'inflation normative est qu'elle est source notable de revenus pour une quantité d'organisations qui n'ont, de ce fait, aucun intérêt à sa disparition. Le contrôle des règles se traduit par des certifications, audits, rapports de conformité qui font vivre nombre d'organisations.* »(31)

Et que dans les entreprises qui appliquent ces normes, certains risquent de perdre leur travail. Ils ont intérêt à générer beaucoup de processus au détriment même de leur entreprise.

S'il est difficile de faire la différence entre ITIL et DevOps(32). Les lignes principales montrent, qu'au départ du moins, il y a une

(27) https://medium.com/@brunodelb/the-cams-model-to-better-understand-the-devops-movement-ffe6713c3fd7

(28) *l'exemple vous semblera exagéré et pourtant je l'ai vécu.*

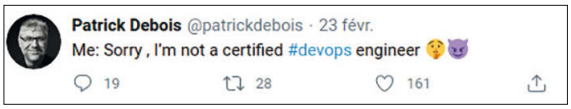
(29) https://www.lebigdata.fr/infrastructure-as-code-definition

(30) *Christian Morel, Les Décisions absurdes III. L'enfer des règles. Les pièges relationnels. p. 113*

(31) *Ibid p. 89*

(32) https://itsocial.fr/enjeux/production/itsm-production/change-itil-v4/

différence. DevOps a été créé par une communauté de personnes n'ayant aucun intérêt financier dans la création d'un mouvement.
Pourtant ITIL contient aussi des éléments absents de DevOps qui sont importants comme par exemple diffuser les éléments d'un langage commun(33).
Mais où va le mouvement DevOps ? Réussira-t-il là où les autres semblent avoir buté ?
Que ce soit le DevOps ou les autres (ITIL, CMMI), la raison de la réussite ou de l'échec tient plus à ce qui en est fait. Et pour cela, le tweet de Patrick Debois résume à lui seul la situation :



Et je citerais ce passage :
« [...] Taiichi Ohno (NDLR: créateur du Kanban) [...] disait souvent qu'il fallait passer son doigt sur les instructions, règles et procédures [...] S'il se couvrait de poussière, c'est qu'elles n'avaient pas été actualisées. [...] la réglementation en interne devrait être mise à jour en permanence, selon un processus quasi expérimental : conception de la règle, application, évaluation des impacts de cette règle, révision de la règle, voire abrogation et nouvelle règle. »(34)
Depuis quelques temps déjà, on trouve des formations certifiantes DevOps, certaines en 3 jours. On peut s'interroger.
Pour le développement logiciel, on est plus proche de l'artisanat que de la conception de grande série.

Prenons la construction d'une extension de maison. Il y a des éléments standard :
- brique ;
- béton ;
- prises électrique…

Mais d'autres éléments sont propres à cette extension :
- taille des poutres porteuses ;
- taille des fenêtres ;
- forme de la pièce...

L'artisanat est donc compatible avec les normes, mais l'artisanat c'est l'adaptation du besoin du client avec les normes.
Le compagnonnage a donc toute sa place pour faire face aux situations et il faut s'appuyer sur les compétences augmentées(35).
Je retiens cette phrase d'une collègue :

> *« Les logiciels, les outils et les processus sont à l'image de nos organisations. »*

Le DevOps n'est pas une fin en soi. C'est une démarche, plus proche de la réflexion personnelle que d'un dogme qui répond à tout.
La lecture du retour d'expérience *Bilan de 4 ans d'ITIL à l'Université de Strasbourg*(36) cité dans la partie ITIL, explique toutes ces problématiques. Je vous encourage vraiment à le lire.

Remerciement pour leur relecture et conseil : Johan Bonneau, Pierre-Yves Aillet, Eric Briand

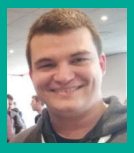
(33) c'est un point abordé dans le livre Christian Morel, Les Décisions absurdes III. L'enfer des règles. Les pièges relationnels.

(34) Ibid p. 117

(35) Ibid

(36) https://2013.jres.org/archives/6/paper6_article.pdf

Alexandre CAUSSIGNAC
Solution Engineer Manager chez VMware
Ma mission est de conseiller et d'accompagner les entreprises dans leur transformation digitale. Au-delà des technologies, j'adore échanger avec les gens sur leurs besoins finaux, car je pense que les seules barrières à ce jour dans notre secteur d'activité sont notre imagination et non notre savoir-faire technique.



Alexandre ROMAN
Solution Engineer chez VMware
Alexandre surfe sur la vague Java depuis presque 20 ans. Utilisateur de longue date de Spring Framework et Kotlin addict, il partage sa passion du développement cloud-native dans des conférences publiques comme Devoxx ou SpringOne et des webinars en ligne. Avec VMware, il aide les clients à entamer leur transformation numérique à grande échelle, en s'appuyant sur les technologies Kubernetes au cœur des solutions VMware Tanzu.

Kubernetes as a Service avec VMware Tanzu Kubernetes Grid

PARTIE 1 : TANZU KUBERNETES GRID

Dans cette première partie, nous allons nous focaliser sur la solution Tanzu Kubernetes Grid (TKG) de la gamme Tanzu (nouvelle offre de VMware orientée Cloud Native Apps). TKG se base sur le projet Cluster API, véritable pierre angulaire pour gérer l'intégralité du cycle de vie des clusters K8s. Il devient alors possible de provisionner des clusters K8s upstream sur n'importe quel type d'infrastructure en vue notamment d'une stratégie On-Premise/Multi-Cloud et d'une portabilité/réversibilité extrême. 1

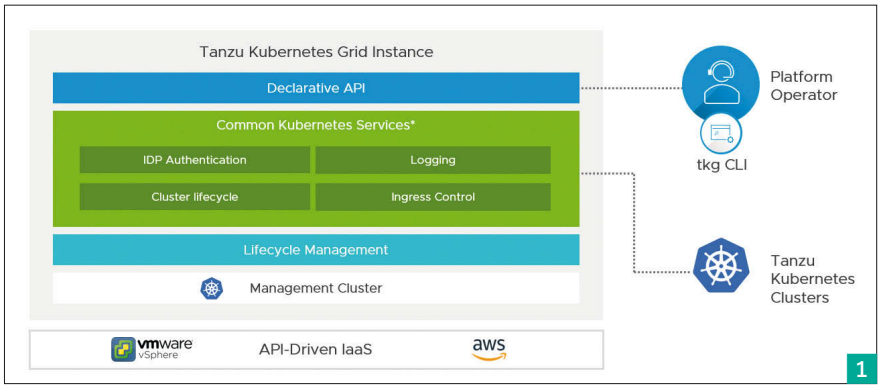
L'objectif de l'article pour le lecteur est de créer (Day 1) et de maintenir (Day 2) des clusters K8s dit Production ready de manière industrialisée, via des composants Open Source et avec le support d'un éditeur.

Les binaires de TKG sont fournis par VMware qui en assure le support et la maintenabilité. TKG est modulaire dans le sens où vous pouvez y ajouter les composants que vous souhaitez, qu'ils soient développés par VMware ou par un tiers (en cas de spécificité liée à votre environnement).
Voici une liste non exhaustive des modules intégrables à TKG. 2
En synthèse, TKG est une offre supportée de Kubernetes *as a Service*, modulaire, *Multi-Cloud* et basée sur des composants Open Source.

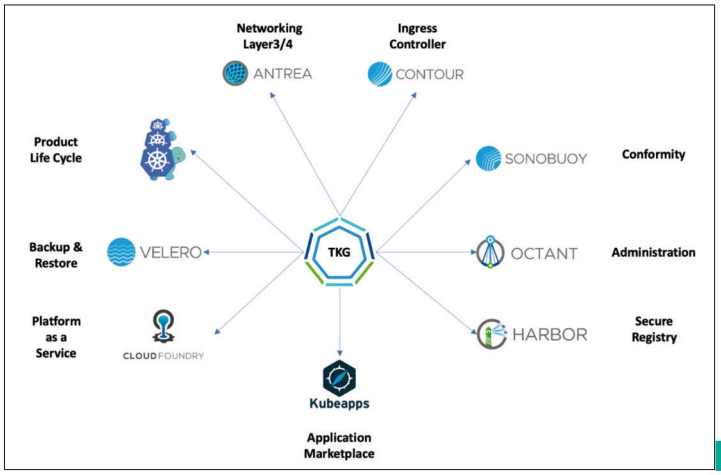
Note : la gamme VMware Tanzu permet d'avoir une offre modulaire via TKG ainsi qu'une approche plateforme via l'offre vSphere with Kubernetes qui intègre l'ensemble des couches calcul, réseau et stockage pour le monde des machines virtuelles et des containers. Ce qui donne aux utilisateurs la liberté de choisir ce qui leur convient le mieux en fonction notamment de leur patrimoine numérique, de leur maturité et du nombre de containers à gérer.

Concepts de TKG

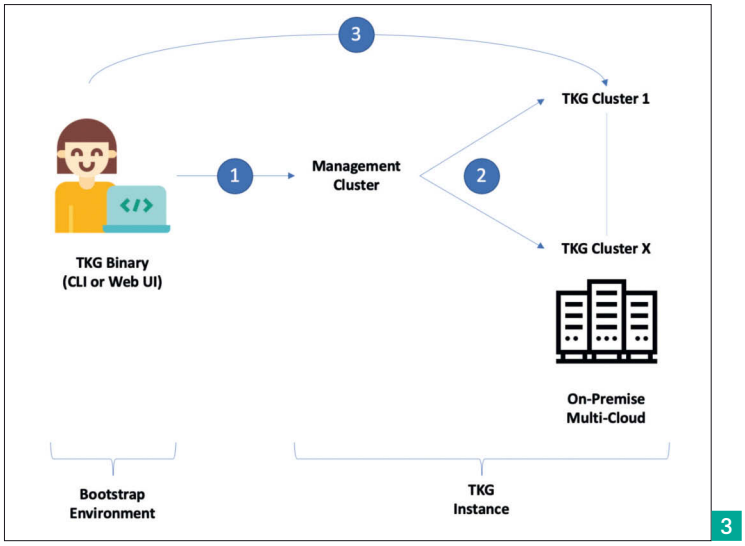
À partir d'un même point d'administration (Linux, macOS ou Windows), il suffit de récupérer le binaire TKG et d'initialiser le cluster de management. Ce dernier intègre les composants de Cluster API propres à la plateforme cible en vue de provisionner vos futurs clusters K8s. C'est la seule partie spécifique à faire sur les *providers* que l'on souhaite utiliser. Ensuite, la création d'un cluster K8s passe par une unique ligne de commande en utilisant les mêmes concepts d'architecture déclarative de K8s pour déployer (*Day 1*) et gérer (*Day 2*) les clusters eux-mêmes. Enfin, une fois le(s) cluster(s) provi-



1



2



3

sionné(s), il vous suffit d'accéder (ou donner accès à un tiers) à l'API via la commande kubectl par exemple. 3
Les clusters K8s sont pleinement opérationnels. Ces derniers intègrent nativement la *Container Storage Interface* (CSI) pour pouvoir provisionner des volumes persistants aux applications le nécessitant (base de données, bus de message, …). Ces volumes seront automatiquement fournis par l'infrastructure sur laquelle réside TKG. Les clusters disposent également de la *Container Network Interface* (CNI) basée à ce jour sur Calico (plus de choix à venir) pour la communication *pod-to-pod*. Il est possible d'utiliser des plans (*Blueprint*) lors du déploiement de ces clusters K8s. Cela permet de définir le gabarit et le nombre de nœuds sur la partie *control plane* et *compute*. Deux plans existent par défaut (dev et prod), et vous êtes libre d'ajouter les vôtres pour, par exemple, personnaliser les éléments déployés au sein des clusters.

Note : les providers de Cluster API supportés à ce jour par VMware dans TKG v1.0.0 sont AWS et vSphere.

K8s est un outil capable de s'intégrer avec un large écosystème via le biais des interfaces (CRI, CNI et CSI). Ces dernières permettent d'être libre dans le choix du runtime permettant de faire tourner des containers, mais aussi la manière de les connecter sur le réseau ou leur consommation en stockage. Nous allons donc voir comment configurer tout cela dans les prochaines sections.

Étape 0 - Préparation de l'environnement de *bootstrap*

Dans notre cas, l'environnement de *bootstrap* est une machine virtuelle Debian GNU/Linux 10 (buster) dont voici les prérequis :

• Configurer les accès sudo de l'utilisateur.

```
# cat /etc/sudoers.d/acaussignac
acaussignac   ALL=(ALL:ALL) NOPASSWD:ALL
```

• Installer le daemon Docker.

```
$ sudo apt install -y apt-transport-https ca-certificates curl gnupg2 software-properties-common
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian
 $(lsb_release -cs) stable"
$ sudo apt update
$ sudo apt-cache policy docker-ce
$ sudo apt install -y docker-ce
$ sudo usermod -aG docker acaussignac
```

• Installer le binaire kubectl.

```
$ curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage
.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
$ chmod +x ./kubectl
$ sudo mv ./kubectl /usr/local/bin/
$ sudo chown root:root /usr/local/bin/kubectl
```

• Installer les binaires jq et govc.

```
$ sudo apt install -y jq
$ curl -LO https://github.com/vmware/govmomi/releases/download/v0.22.1/govc_ linux_amd64.gz
$ gunzip govc_linux_amd64.gz
$ chmod +x ./govc_linux_amd64
$ sudo mv ./govc_linux_amd64 /usr/local/bin/govc
$ sudo chown root:root /usr/local/bin/govc
```

• Installer les binaires go et yq.

```
$ curl -O https://dl.google.com/go/go1.12.7.linux-amd64.tar.gz
$ tar zxvf go1.12.7.linux-amd64.tar.gz
$ sudo mv go /usr/local/
$ sudo chown -R root:root /usr/local/go
$ mkdir $HOME/go
$ tail -2 ~/.profile
export GOPATH=$HOME/go
export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin
$ source ~/.profile
$ GO111MODULE=on go get github.com/mikefarah/yq/v3
$ rm -f go1.12.7.linux-amd64.tar.gz
```

• Installer le binaire TKG.

Récupérer manuellement le binaire TKG pour Linux à l'adresse suivante : https://www.vmware.com/go/get-tkg.

Note : *la version 1.0.0 était la dernière release en date à l'écriture de cet article. A ce jour, la version 1.1.2 est disponible mais les commandes listées ci-dessous restent les mêmes.* 4

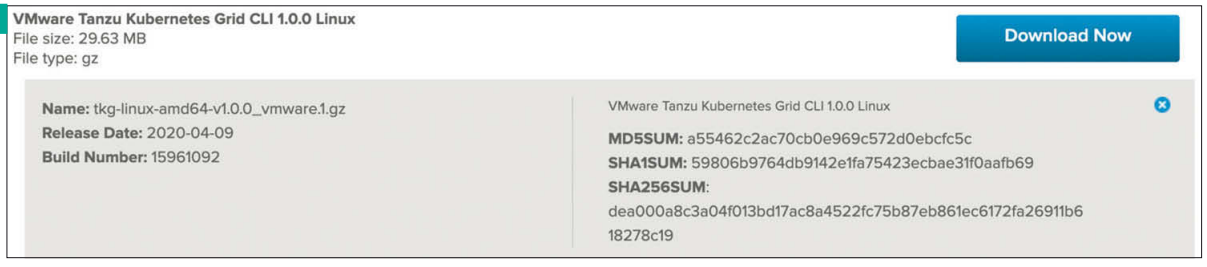
Ou automatiquement via vmw-cli en adaptant vos *credentials* (*username* et *password*).

```
$ mkdir vmw-cli
$ cd vmw-cli/
$ docker run -t --rm -e VMWUSER='username' -e VMWPASS='password' -v
${PWD}:/files apnex/vmw-cli index TKG-100
$ docker run -t --rm -e VMWUSER='username' -e VMWPASS='password' -v
${PWD}:/files apnex/vmw-cli get tkg-linux-amd64-v1.0.0_vmware.1.gz
$ gunzip tkg-linux-amd64-v1.0.0_vmware.1.gz
$ sudo chmod +x ./tkg-linux-amd64-v1.0.0_vmware.1
$ sudo mv tkg-linux-amd64-v1.0.0_vmware.1 /usr/local/bin/tkg && sudo chown root:root
/usr/local/bin/tkg
$ tkg version
Client:
    Version: v1.0.0
    Git commit: 60f6fd5f40101d6b78e95a33334498ecca86176e
```

• Installer le binaire clusterawsadm.

```
$ docker run -t --rm -e VMWUSER='username' -e VMWPASS='password' -v
${PWD}:/files apnex/vmw-cli get clusterawsadm-linux-amd64-v0.5.2_vmware.1.gz
```

4



```
$ gunzip clusterawsadm-linux-amd64-v0.5.2_vmware.1.gz
$ sudo chmod +x ./clusterawsadm-linux-amd64-v0.5.2_vmware.1
$ sudo mv clusterawsadm-linux-amd64-v0.5.2_vmware.1 /usr/local/bin/clusterawsadm &&
 sudo chown root:root /usr/local/bin/clusterawsadm
```

Note : le binaire clusterawsadm gère l'identité et l'accès aux ressources AWS provisionnées via Cluster API.

• Installer le binaire awscli.

```
$ cd
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
$ unzip awscliv2.zip
$ sudo ./aws/install
$ aws --version
aws-cli/2.0.13 Python/3.7.3 Linux/4.19.0-9-amd64 botocore/2.0.0dev17
$ rm -f awscliv2.zip
$ rm -rf aws/
```

L'environnement de *bootstrap* est désormais prêt et vous avez la possibilité de déployer le cluster de management sur une infrastructure AWS et/ou vSphere. Une fois le cluster de management déployé sur la plateforme(s) cible(s), il suffira d'utiliser le binaire TKG pour provisionner vos clusters K8s et ce sur n'importe quelle infrastructure.

Étape 1a - Déployer un cluster de management sur une infrastructure AWS

Create access key pour l'utilisateur AWS concerné dans la partie *Identity and Access Management* (IAM). 5

```
$ export AWS_ACCESS_KEY_ID=Access_key_ID
$ export AWS_SECRET_ACCESS_KEY=Secret_access_key
$ export AWS_REGION=us-east-2
```

Note : *si vous utilisez le **Multi-Factor Access**, il vous sera nécessaire de préciser votre **AWS session token** via la commande export AWS_SESSION_TOKEN=aws_session_token*

• Créer une *stack* CloudFormation (valable sur l'ensemble des régions) qui sera utilisée pour déployer le cluster de management et les clusters K8s.

```
$ clusterawsadm alpha bootstrap create-stack
```
6

• Générer une paire de clés SSH qui permettra d'administrer les clusters K8s.

```
$ aws ec2 create-key-pair --key-name tkg --output json | jq .KeyMaterial -r > tkg.pem
```

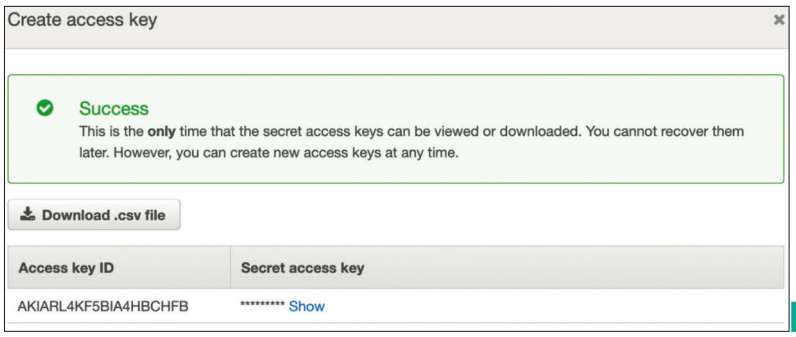
Note : *le format des clés utilisées par AWS étant différent de celui pour vSphere, il est nécessaire de générer une paire de clés par **provider**.*

• Fixer les variables d'authentification nécessaires à Cluster API.

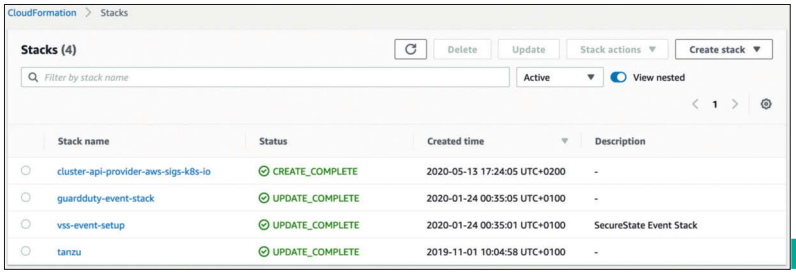
```
$ export AWS_CREDENTIALS=$(aws iam create-access-key --user-name bootstrapper.cluster-
api-provider-aws.sigs.k8s.io --output json)
$ export AWS_ACCESS_KEY_ID=$(echo $AWS_CREDENTIALS | jq .AccessKey.AccessKeyId -r)
$ export AWS_SECRET_ACCESS_KEY=$(echo $AWS_CREDENTIALS | jq .AccessKey.SecretAccess
Key -r)
$ export AWS_B64ENCODED_CREDENTIALS=$(clusterawsadm alpha bootstrap encode-aws-credentials)
```

• Définir l'identifiant de l'*Amazon Machine Image* (AMI) à utiliser en fonction de la région sélectionnée.

```
$ export AWS_AMI=ami-0e9216661312a1a35
```

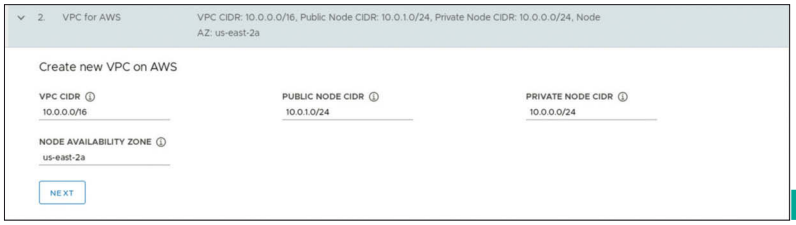


5

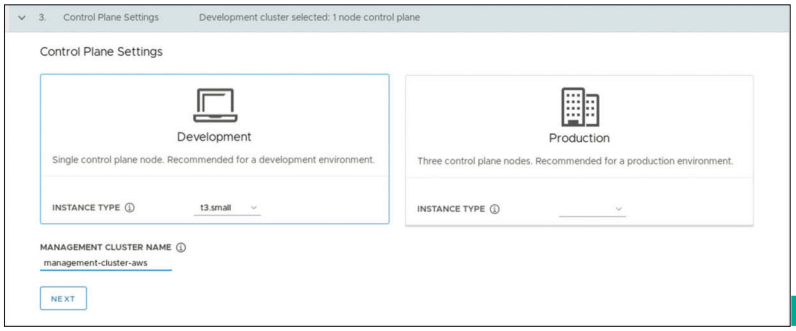


6

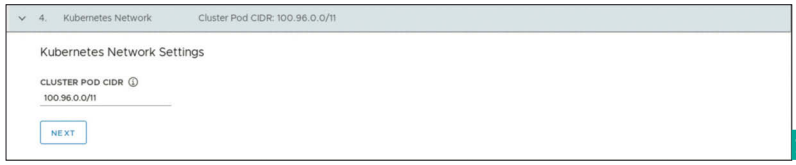
7



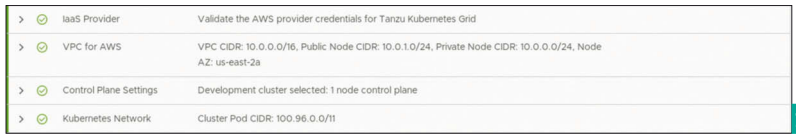
8



9



10



11

12

Note : *vous pouvez trouver le détail de ces identifiants par région sur la documentation officielle.*

• Lancer l'installation du cluster de management via l'interface Web.

```
$ tkg init --ui
```

7 8 9 10 11 12

L'installation dure environ une dizaine de minutes.

Note : *le fichier de configuration généré se trouve par défaut dans le répertoire $HOME/.tkg et se nomme config.yaml. Il est aussi possible de générer ce fichier en se basant sur l'existant comme modèle et de lancer l'installation du cluster de management via la commande tkg init --infrastructure=aws --name=management-cluster-aws --plan=dev. Pour plus d'informations, vous pouvez vous référer à la documentation officielle.*

• Vérifier l'installation du cluster de management.

```
$ tkg get management-cluster
+-------------------------+-----------------------------------------------------+
| MANAGEMENT CLUSTER NAME | CONTEXT NAME                                        |
+-------------------------+-----------------------------------------------------+
| management-cluster-aws * | management-cluster-aws-admin@management-cluster-aws |
+-------------------------+-----------------------------------------------------+
$ kubectl config get-contexts
CURRENT  NAME                                          CLUSTER              AUTHINFO                NAMESPACE
*        management-cluster-aws-admin@management-cluster-aws  management-cluster-aws  management-cluster-aws-admin
$ kubectl get nodes
NAME                                         STATUS  ROLES   AGE    VERSION
ip-10-0-0-16.us-east-2.compute.internal      Ready   <none>  9m10s  v1.17.3+vmware.2
ip-10-0-0-221.us-east-2.compute.internal     Ready   master  10m    v1.17.3+vmware.2
```

13

Étape 1b - Déployer un cluster de management sur une infrastructure vSphere (6.7 U3 minimum)

Note : *avant toute chose, un serveur DHCP est nécessaire sur votre infrastructure en vue de fournir des adresses IP aux différentes machines du cluster K8s. Pour information, il n'est pas nécessaire d'installer un cluster de management sur vSphere 7 si la fonctionnalité **vSphere with Kubernetes** a été activée (le **wizard** vous préviendra si c'est le cas).*

• Générer une paire de clés SSH qui permettra d'administrer les clusters K8s.

```
$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa_tkg
$ eval `ssh-agent`
Agent pid 17303
$ ssh-add ~/.ssh/id_rsa_tkg
```

• Configurer l'accès à la VCSA via govc.

```
$ cat ~/.govc
export GOVC_INSECURE=true
export GOVC_URL="https://vcsa.cpod-tkg.az-rbx.cloud-garage.net"
export GOVC_USERNAME="administrator@cpod-tkg.az-rbx.cloud-garage.net"
export GOVC_PASSWORD="password"
$ source .govc
$ govc about
Name:       VMware vCenter Server
Vendor:     VMware, Inc.
Version:    7.0.0
Build:      15952599
OS type:    linux-x64
API type:   VirtualCenter
API version: 7.0.0.0
Product ID: vpx
UUID:       5561022a-0846-4783-b225-04c5640ad59b
```

• Récupérer et importer les Templates au format OVA sur l'infrastructure vSphere.

```
$ cd vmw-cli/
$ docker run -t --rm -e VMWUSER='username' -e VMWPASS='password' -v ${PWD}:/files apnex/vmw-cli get photon-3-v1.17.3_vmware.2.ova
$ docker run -t --rm -e VMWUSER='username' -e VMWPASS='password' -v ${PWD}:/files apnex/vmw-cli get photon-3-capv-haproxy-v0.6.3_vmware.1.ova
$ govc pool.create */Resources/TKG
$ govc folder.create /cPod-TKG/vm/TKG
```

Note : *cPod-TKG est le nom du Datacenter sur cette infrastructure.*

```
$ govc import.spec photon-3-v1.17.3_vmware.2.ova | jq '.name="photon-3-kube-v1.17.3"' | jq '.NetworkMapping[0].Network="VM Network"' > photon-3-kube-v1.17.3.json
$ govc import.ova -pool=*/Resources/TKG -options=photon-3-kube-v1.17.3.json -ds=nfsDatastore photon-3-v1.17.3_vmware.2.ova
$ govc vm.upgrade -version=15 -vm photon-3-kube-v1.17.3
$ govc snapshot.create -vm photon-3-kube-v1.17.3 root
$ govc vm.markastemplate photon-3-kube-v1.17.3
$ govc object.mv /cPod-TKG/vm/photon-3-kube-v1.17.3 /cPod-TKG/vm/TKG
$ govc import.spec photon-3-capv-haproxy-v0.6.3_vmware.1.ova | jq '.name="capv-haproxy-v0.6.3"' | jq '.NetworkMapping[0].Network="VM Network"' > capv-haproxy-v0.6.3.json
$ govc import.ova -pool=*/Resources/TKG -options=capv-haproxy-v0.6.3.json -ds=nfsDatastore photon-3-capv-haproxy-v0.6.3_vmware.1.ova
$ govc vm.upgrade -version=15 -vm capv-haproxy-v0.6.3
$ govc snapshot.create -vm capv-haproxy-v0.6.3 root
$ govc vm.markastemplate capv-haproxy-v0.6.3
$ govc object.mv /cPod-TKG/vm/capv-haproxy-v0.6.3 /cPod-TKG/vm/TKG
$ cd
```

14

• Lancer l'installation du cluster de management depuis l'interface Web.

```
$ tkg init --ui
```

15

Note : *il s'agit de la clé SSH générée précédemment.*

16 17 18 19 20 21

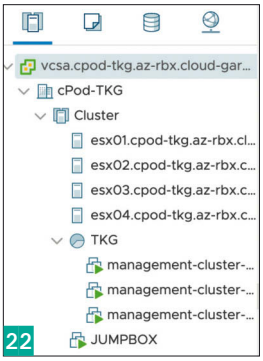
L'installation dure environ une dizaine de minutes.

Note : *le fichier de configuration généré se trouve par défaut dans le répertoire $HOME/.tkg et se nomme config.yaml. Il est aussi possible de générer ce fichier en se basant sur l'existant comme modèle et de lancer l'installation du cluster de management via la commande tkg init --infrastructure=vsphere --name=management-cluster-vsphere --plan=dev. Pour plus d'informations, vous pouvez vous référer à la documentation officielle.*

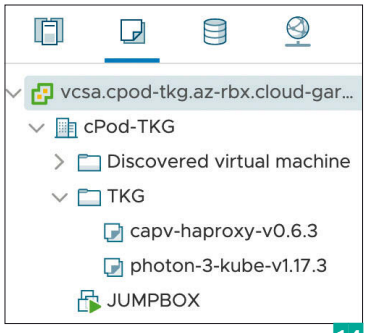
• Vérifier l'installation du cluster de management.

13

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
management...	i-05f34ef98622aeb7c	t3.small	us-east-2a	running	2/2 checks ...	OK
management...	i-060573fa0a06fe798	t3.small	us-east-2a	running	2/2 checks ...	OK
management...	i-080286ced85be6852	t2.micro	us-east-2a	running	2/2 checks ...	OK



22



14

```
$ tkg get management-cluster
+-----------------------------+-----------------------------------------------------+
| MANAGEMENT CLUSTER NAME     | CONTEXT NAME                                        |
+-----------------------------+-----------------------------------------------------+
| management-cluster-aws      | management-cluster-aws-admin@management-cluster-aws |
| management-cluster-vsphere * | management-cluster-vsphere-admin@management-cluster-vsphere |
+-----------------------------+-----------------------------------------------------+
$ kubectl config get-contexts
CURRENT  NAME                                                  CLUSTER                AUTHINFO                NAMESPACE
         management-cluster-aws-admin@management-cluster-aws  management-cluster-aws  management-cluster-aws-admin
*        management-cluster-vsphere-admin@management-cluster-vsphere  management-cluster-vsphere  management-cluster-vsphere-admin
$ kubectl get nodes
NAME                                                STATUS  ROLES   AGE  VERSION
management-cluster-vsphere-control-plane-9wghz      Ready   master  20m  v1.17.3+vmware.2
management-cluster-vsphere-md-0-6495b4c6d7-jrv46    Ready   <none>  19m  v1.17.3+vmware.2
```
22

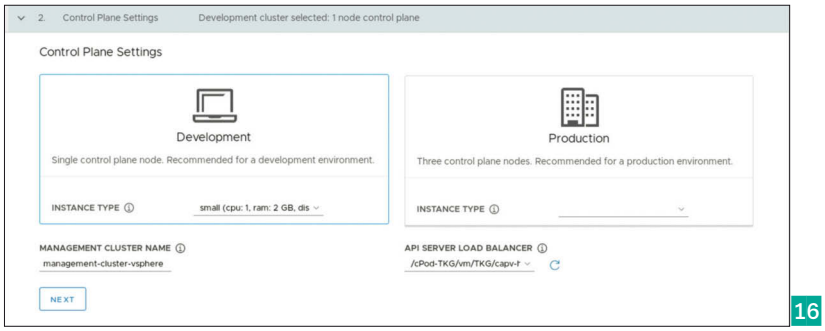
Étape 2 - Déployer des clusters K8s via le binaire TKG sur une infrastructure AWS et/ou vSphere

Une fois que le cluster de management a été créé sur une infrastructure AWS et/ou vSphere, nous allons pouvoir déployer des clusters K8s à la volée et directement prêts à l'emploi. Étant donné que TKG permet de provisionner d'un point central des clusters K8s sur tous types de plateformes compatibles avec Cluster API, il est donc nécessaire de choisir le cluster de management et le contexte K8s afférent. Dans l'exemple qui suit, j'ai pris une infrastructure vSphere mais les commandes sont exactement les mêmes sur une plateforme AWS. Les quelques spécificités liées à l'environnement seront mentionnées.

• Changer de cluster de management et de contexte K8s.

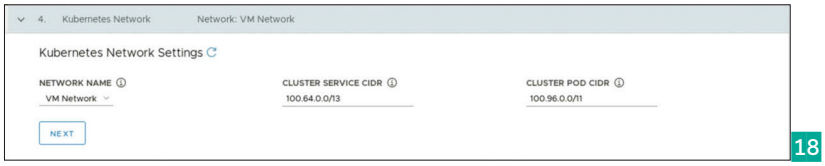
```
$ tkg set management-cluster management-cluster-vsphere
The current management cluster context is switched to management-cluster-vsphere
$ kubectl config use-context management-cluster-vsphere-admin@management-cluster-vsphere
Switched to context "management-cluster-vsphere-admin@management-cluster-vsphere".
$ tkg get management-cluster
+-----------------------------+-----------------------------------------------------+
| MANAGEMENT CLUSTER NAME     | CONTEXT NAME                                        |
+-----------------------------+-----------------------------------------------------+
| management-cluster-aws      | management-cluster-aws-admin@management-cluster-aws |
| management-cluster-vsphere * | management-cluster-vsphere-admin@management-cluster-vsphere |
+-----------------------------+-----------------------------------------------------+
$ kubectl config get-contexts
CURRENT  NAME                                                  CLUSTER                AUTHINFO                NAMESPACE
```

15



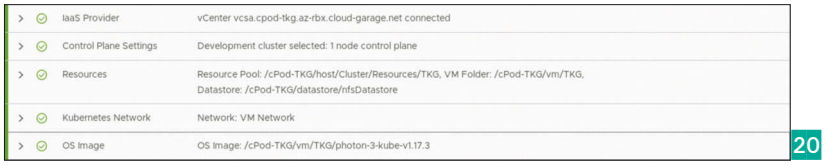
16

17



18

19



20

21

```
         management-cluster-aws-admin@management-cluster-aws  management-cluster-aws  management-cluster-aws-admin
*        management-cluster-vsphere-admin@management-cluster-vsphere  management-cluster-vsphere  management-cluster-vsphere-admin
$ kubectl get nodes
NAME                                                STATUS  ROLES   AGE   VERSION
management-cluster-vsphere-control-plane-9wghz      Ready   master  126m  v1.17.3+vmware.2
management-cluster-vsphere-md-0-6495b4c6d7-jrv46    Ready   <none>  124m  v1.17.3+vmware.2
```

• Créer un namespace par cluster K8s sur le cluster de management.

```
$ kubectl create ns vsphere-cluster-prod-001
```

Note : *la création d'un namespace dédié n'est pas obligatoire mais permet de classer plus facilement les clusters K8s. Attention à choisir des noms de clusters différents si vous utilisez plusieurs plateformes en parallèle.*

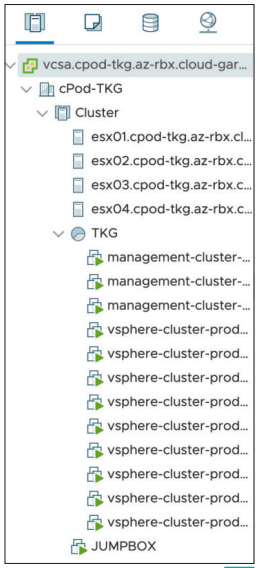
• Déployer un cluster K8s Multi-Master.

```
$ tkg create cluster vsphere-cluster-prod-001 -p prod -n vsphere-cluster-prod-001 -w 5
Logs of the command execution can also be found at: /tmp/tkg-20200514T100921573993618.log
Creating workload cluster 'vsphere-cluster-prod-001'...

Context set for workload cluster vsphere-cluster-prod-001 as vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001

Waiting for cluster nodes to be available...

Workload cluster 'vsphere-cluster-prod-001' created
```

Note : *le plan prod permet d'avoir un cluster K8s* ***Multi-Master*** *prêt à l'emploi (avec déploiement automatique d'un* ***Load Balancer***) *et l'option -w permet de spécifier le nombre de nœuds de* ***compute***. *La création d'un cluster* ***Multi-Master*** *dure une dizaine de minutes.*

```
$ kubectl config get-contexts
CURRENT  NAME                                                  CLUSTER                AUTHINFO                NAMESPACE
         management-cluster-aws-admin@management-cluster-aws   management-cluster-aws  management-cluster-aws-admin
         management-cluster-vsphere-admin@management-cluster-vsphere  management-cluster-vsphere  management-cluster-vsphere-admin
*        vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001  vsphere-cluster-prod-001  vsphere-cluster-prod-001-admin
$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE    VERSION
vsphere-cluster-prod-001-control-plane-ffg66  Ready    master   9m37s  v1.17.3+vmware.2
vsphere-cluster-prod-001-control-plane-hwhxj  Ready    master   3m7s   v1.17.3+vmware.2
vsphere-cluster-prod-001-control-plane-kf7c6  Ready    master   5m36s  v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-brbfg  Ready  <none>   5m43s  v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-m4tkx  Ready  <none>   5m44s  v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-s8ndg  Ready  <none>   5m42s  v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-z7wtb  Ready  <none>   6m16s  v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-z9htv  Ready  <none>   6m56s  v1.17.3+vmware.2
```

23



23

• Vérifier la création du cluster K8s dans la base ETCD du cluster de management.

```
$ kubectl config use-context management-cluster-vsphere-admin@management-cluster-vsphere
Switched to context "management-cluster-vsphere-admin@management-cluster-vsphere".
$ tkg get clusters -n vsphere-cluster-prod-001
+--------------------------+-------------+
| NAME                     | STATUS      |
+--------------------------+-------------+
| vsphere-cluster-prod-001 | Provisioned |
+--------------------------+-------------+
```

• Étendre à chaud le nombre de nœuds du *control plane* et de *compute* du cluster K8s.

```
$ tkg scale cluster vsphere-cluster-prod-001 -c5 -w7 -n vsphere-cluster-prod-001
Successfully updated control plane replica count for cluster vsphere-cluster-prod-001
Successfully updated worker node machine deployment replica count for cluster vsphere-cluster-prod-001
workload cluster vsphere-cluster-prod-001 is being scaled
$ kubectl config use-context vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001
Switched to context "vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001".
$ kubectl get nodes
NAME                                          STATUS  ROLES    AGE    VERSION
vsphere-cluster-prod-001-control-plane-ffg66  Ready   master   17m    v1.17.3+vmware.2
vsphere-cluster-prod-001-control-plane-hwhxj  Ready   master   10m    v1.17.3+vmware.2
vsphere-cluster-prod-001-control-plane-kf7c6  Ready   master   13m    v1.17.3+vmware.2
vsphere-cluster-prod-001-control-plane-pmnth  Ready   master   97s    v1.17.3+vmware.2
vsphere-cluster-prod-001-control-plane-v86zb  Ready   master   3m23s  v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-brbfg  Ready  <none>  13m    v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-cr686  Ready  <none>  3m26s  v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-lhtb8  Ready  <none>  3m45s  v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-m4tkx  Ready  <none>  13m    v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-s8ndg  Ready  <none>  13m    v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-z7wtb  Ready  <none>  13m    v1.17.3+vmware.2
vsphere-cluster-prod-001-md-0-6597cbbc9f-z9htv  Ready  <none>  14m    v1.17.3+vmware.2
```

Note : *il est également possible de visualiser sur la sortie standard la configuration au format YAML du cluster K8s avec l'option suivante tkg create cluster vsphere-cluster-prod-001 -p prod -n vsphere-cluster-prod-001 -w 5 --dry-run ou de générer cette même configuration en vue de la personnaliser avant de l'appliquer via la commande tkg config create. A noter dans ce cas qu'une étape supplémentaire (post installation) sera nécessaire pour configurer la partie CNI (alors que celle-ci est réalisée automatiquement via la commande tkg create cluster).*

• Déployer un cluster K8s à partir d'un fichier YAML.

```
$ kubectl config use-context management-cluster-vsphere-admin@management-cluster-vsphere
Switched to context "management-cluster-vsphere-admin@management-cluster-vsphere".
$ kubectl create ns vsphere-cluster-dev-001
$ tkg config cluster vsphere-cluster-dev-001 --infrastructure vsphere --kubernetes-version v1.17.3+vmware.1 --controlplane-machine-count 1 --worker-machine-count 3 --namespace vsphere-cluster-dev-001 --plan dev > vsphere-cluster-dev-001.yaml
```

Note : *spécifier l'option --infrastructure aws en fonction de la plateforme cible en adaptant également le nom du cluster.*

```
$ kubectl apply -f vsphere-cluster-dev-001.yaml
$ cat vsphere-cluster-dev-001.yaml | yq r -d7 - stringData.calicoYaml > postcreation_steps.yaml
$ tkg get clusters -n vsphere-cluster-dev-001
+--------------------------+-------------+
| NAME                     | STATUS      |
+--------------------------+-------------+
| vsphere-cluster-dev-001  | Provisioned |
+--------------------------+-------------+
$ tkg get credentials vsphere-cluster-dev-001 -n vsphere-cluster-dev-001
Credentials of workload cluster vsphere-cluster-dev-001 have been saved
You can now access the cluster by switching the context to vsphere-cluster-dev-001-admin@vsphere-cluster-dev-001 under /home/acaussignac/.kube/config
$ kubectl config get-contexts
CURRENT  NAME                                                  CLUSTER                AUTHINFO                NAMESPACE
         management-cluster-aws-admin@management-cluster-aws   management-cluster-aws  management-cluster-aws-admin
*        management-cluster-vsphere-admin@management-cluster-vsphere  management-cluster-vsphere  management-cluster-vsphere-admin
```

```
    vsphere-cluster-dev-001-admin@vsphere-cluster-dev-001      vsphere-cluster-dev-001
  vsphere-cluster-dev-001-admin
    vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001    vsphere-cluster-prod-001
  vsphere-cluster-prod-001-admin
$ kubectl config use-context vsphere-cluster-dev-001-admin@vsphere-cluster-dev-001
Switched to context "vsphere-cluster-dev-001-admin@vsphere-cluster-dev-001".
$ kubectl apply -f postcreation_steps.yaml
$ kubectl get nodes
NAME                                          STATUS  ROLES   AGE    VERSION
vsphere-cluster-dev-001-control-plane-mc99t   Ready   master  7m50s  v1.17.3+vmware.2
vsphere-cluster-dev-001-md-0-7854446464-6t58b Ready   <none>  5m39s  v1.17.3+vmware.2
vsphere-cluster-dev-001-md-0-7854446464-sfnk7 Ready   <none>  5m22s  v1.17.3+vmware.2
vsphere-cluster-dev-001-md-0-7854446464-vstqb Ready   <none>  5m21s  v1.17.3+vmware.2
```

Enfin, voici comment supprimer ce même cluster.

```
$ kubectl config use-context management-cluster-vsphere-admin@management-cluster-vsphere
Switched to context "management-cluster-vsphere-admin@management-cluster-vsphere".
$ tkg get cluster -n vsphere-cluster-dev-001
+-------------------------+-------------+
| NAME                    | STATUS      |
+-------------------------+-------------+
| vsphere-cluster-dev-001 | Provisioned |
+-------------------------+-------------+
$ tkg delete cluster vsphere-cluster-dev-001 -n vsphere-cluster-dev-001
Deleting workload cluster 'vsphere-cluster-dev-001'. Are you sure?: y
workload cluster vsphere-cluster-dev-001 is being deleted
$ tkg get cluster -n vsphere-cluster-dev-001
+------+--------+
| NAME | STATUS |
+------+--------+
+------+--------+
$ kubectl delete ns vsphere-cluster-dev-001
namespace "vsphere-cluster-dev-001" deleted
$ kubectl config delete-cluster vsphere-cluster-dev-001
deleted cluster vsphere-cluster-dev-001 from /home/acaussignac/.kube/config
$ kubectl config delete-context vsphere-cluster-dev-001-admin@vsphere-cluster-dev-001
deleted context vsphere-cluster-dev-001-admin@vsphere-cluster-dev-001 from /home/acaussignac/
.kube/config
```

Configurer la partie réseau *Container Network Interface* (à réaliser uniquement sur vSphere)

• Installer MetalLB pour apporter au cluster K8s la fonctionnalité de *Load Balancer as a Service* (déjà disponible sur AWS) qui nous permettra d'exposer nos services à l'extérieur de nos clusters K8s.

Note : *pour information, au même titre qu'un service managé chez un* ***hyperscaler,*** *tous ces composants réseau et stockage sont automatiquement fournis par la plateforme VMware Cloud Foundation quand vous utilisez notamment la solution* ***vSphere with Kubernetes***.

```
$ kubectl config use-context vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001
Switched to context "vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001".
$ kubectl apply -f https://raw.githubusercontent.com/google/metallb/v0.8.3/manifests/metallb.yaml
$ cat metallb-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
 namespace: metallb-system
 name: config
data:
 config: |
  address-pools:
  - name: default
   protocol: layer2
   addresses:
   - 172.23.5.100-172.23.5.150
```

Note : *le range 172.23.5.100-172.23.5.150 correspond au sous-réseau que j'utilise sur mon infrastructure vSphere. Il est donc à adapter en fonction de votre environnement.*

```
$ kubectl apply -f metallb-config.yaml
configmap/config created
```

Configurer la partie stockage *(Container Storage Interface)* sur une infrastructure AWS (driver gp2)

• Créer une *Storage Class* (toujours penser à changer de cluster de management et de contexte K8s selon le *provider* utilisé).

```
$ tkg set management-cluster management-cluster-aws
The current management cluster context is switched to management-cluster-aws
$ kubectl config use-context aws-cluster-prod-001-admin@aws-cluster-prod-001
Switched to context "aws-cluster-prod-001-admin@aws-cluster-prod-001".
$ cat k8s-sc-aws.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: k8s-sc-aws
 annotations:
  storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/aws-ebs
parameters:
 type: gp2
 fsType: ext4
$ kubectl create -f k8s-sc-aws.yaml
storageclass.storage.k8s.io/k8s-sc-aws created
$ kubectl get sc
NAME    PROVISIONER  RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE
k8s-sc-aws (default) kubernetes.io/aws-ebs Delete     Immediate       false            5s
```

Configurer la partie stockage *(Container Storage Interface)* sur une infrastructure vSphere (driver csi.vsphere)

Note : *l'installation du vSphere Cloud Controller Manager et du vSphere Cloud Storage Interface Driver est déjà intégrée dans Cluster API for vSphere (CAPV). Il reste juste à créer une* ***Storage Policy*** *sur vSphere mappée à une* ***Storage Class*** *sur K8s.*

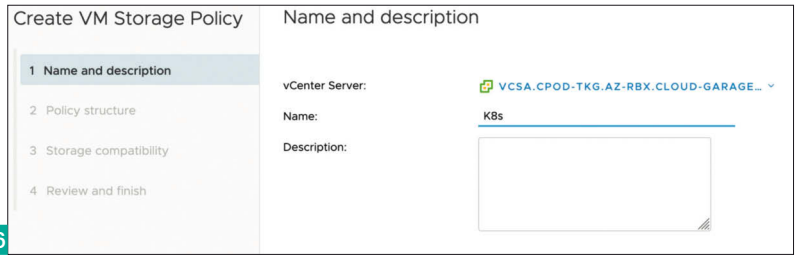
• Créer un *tag* nommé K8s et l'assigner au datastore adéquat.
24 25

• Créer une *Storage Policy* basée sur le *tag* créé précédemment.
26 27 28 29 30

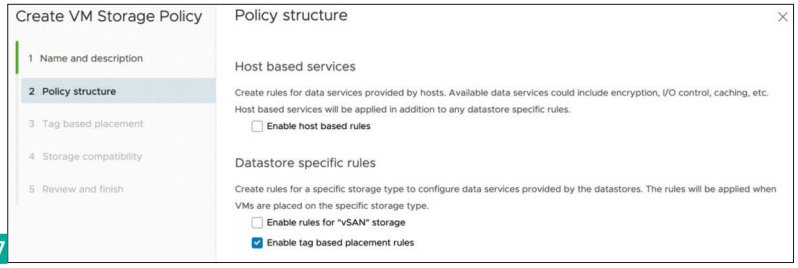
• Créer une *Storage Class* basée sur la *Storage Policy* (toujours penser à changer de cluster de management et de contexte K8s selon le *provider* utilisé).

```
$ tkg set management-cluster management-cluster-vsphere
The current management cluster context is switched to management-cluster-vsphere
$ kubectl config use-context vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001
Switched to context "vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001".
$ cat k8s-sc-vsphere.yaml
apiVersion: storage.k8s.io/v1
```

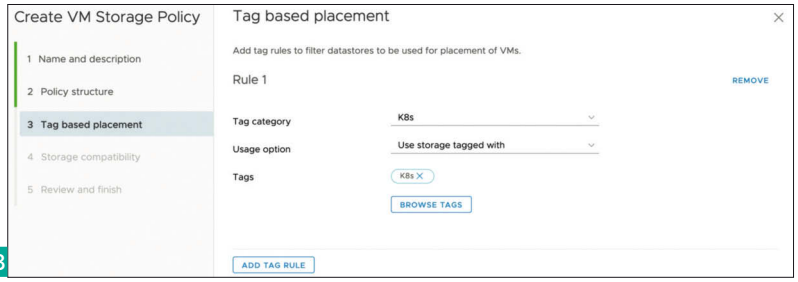
25



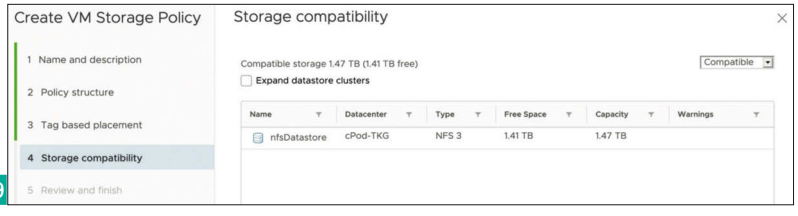
26



27



28



29

30

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: k8s-sc-vsphere
 annotations:
  storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi.vsphere.vmware.com
parameters:
 storagepolicyname: "K8s"
 fstype: ext4
$ kubectl create -f k8s-sc-vsphere.yaml
$ kubectl get sc
NAME  PROVISIONER  RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE
k8s-sc-vsphere (default) csi.vsphere.vmware.com Delete  Immediate  false  5s
```

Étape 3 - Installer l'application Rocket-chat (avec persistance de données) afin de tester notre configuration sur une infrastructure AWS et/ou vSphere

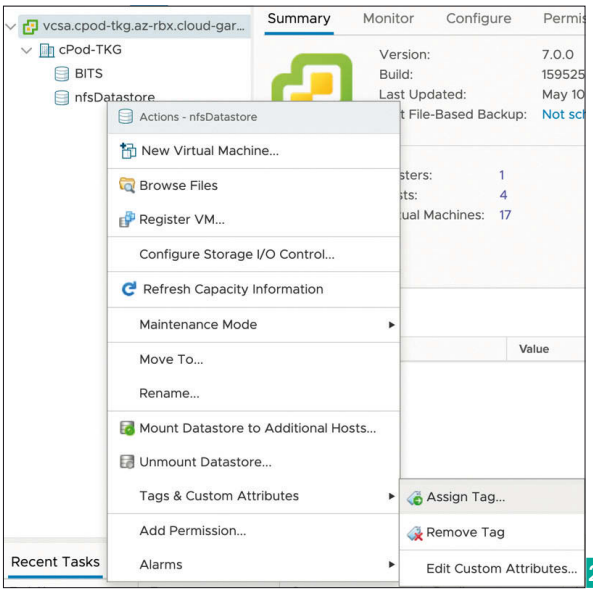
• Installer helm (*packager* d'applications basé sur des *marketplace*).

```
$ wget https://get.helm.sh/helm-v3.1.2-linux-amd64.tar.gz
$ tar zxvf helm-v3.1.2-linux-amd64.tar.gz
$ sudo mv linux-amd64/helm /usr/local/bin/ && sudo chown root:root /usr/local/bin/helm
$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
"stable" has been added to your repositories
$ helm repo add vmware-tanzu https://vmware-tanzu.github.io/helm-charts
"vmware-tanzu" has been added to your repositories
```

Note : *l'ajout des trois dépôts (repositories) permettra l'installation des outils/applications ci-dessous.*

• Installer l'application Rocketchat.

```
$ cat rocketchat.yaml
persistence:
 enabled: true
service:
```



24

```
 type: LoadBalancer
mongodb:
 mongodbPassword: password
 mongodbRootPassword: password
$ helm install rocketchat stable/rocketchat --namespace rocketchat -f rocketchat.yaml
$ kubectl get pods -n rocketchat
NAME                                READY   STATUS    RESTARTS   AGE
rocketchat-mongodb-primary-0        1/1     Running   0          m38s
rocketchat-rocketchat-8d4bfb57d-7dbnk   1/1     Running   2          2m38s
$ kubectl get svc -n rocketchat
NAME                         TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE
rocketchat-mongodb           ClusterIP      100.68.169.175  <none>         27017/TCP      2m53s
rocketchat-mongodb-headless  ClusterIP      None            <none>         27017/TCP      2m53s
rocketchat-rocketchat        LoadBalancer   100.66.242.80   172.23.5.100   80:30847/TCP   2m53s
$ kubectl get pvc -n rocketchat
NAME                 STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS  AGE
datadir-rocketchat-mongodb-primary-0  Bound  pvc-31e0c380-b96f-41f7-b79f-2c2e4a4d
2955  8Gi     RWO         k8s-sc-vsphere  14m
rocketchat-rocketchat           Bound  pvc-7e1a8418-efa3-47ee-ae56-99becf620ce5  8Gi
    RWO         k8s-sc-vsphere  14m
$ kubectl annotate pod -n rocketchat --selector=release=rocketchat,app=mongodb backup
.velero.io/backup-volumes=datadir --overwrite
```

Note : *la dernière commande permet de labelliser les Persistent Volumes (PV) afin que ces derniers soient inclus dans la sauvegarde Velero.* 31 32

Note : *résultat sur une infrastructure AWS.*

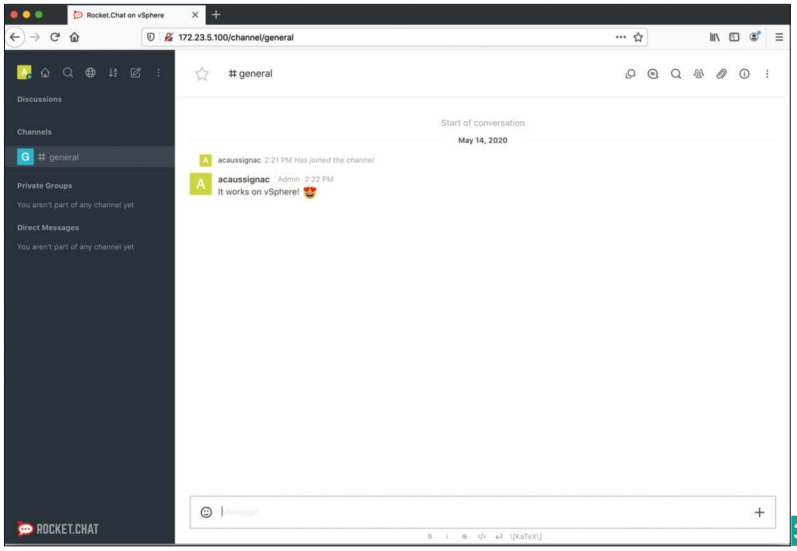
```
$ kubectl get pods -n rocketchat
NAME                                READY  STATUS   RESTARTS  AGE
rocketchat-mongodb-primary-0        1/1    Running  0         7m21s
rocketchat-rocketchat-8d4bfb57d-c777m  1/1    Running  0         7m21s
$ kubectl get svc -n rocketchat
NAME                         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
rocketchat-mongodb           ClusterIP      10.100.183.55   <none>        27017/TCP   2m19s
rocketchat-mongodb-headless  ClusterIP      None            <none>        27017/TCP   2m19s
rocketchat-rocketchat        LoadBalancer   10.108.203.88   a5be5e54df485419baf78d55b44dd
45a-128567579.us-east-2.elb.amazonaws.com   80:32712/TCP   2m19s
$ kubectl get pvc -n rocketchat
NAMESPACE  NAME                 STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS  AGE
rocketchat  datadir-rocketchat-mongodb-primary-0  Bound  pvc-6b992709-5c80-4c10-8169
-59110f61ec14  8Gi     RWO         k8s-sc-aws    8s
rocketchat  rocketchat-rocketchat           Bound  pvc-45f4c5d9-04fc-446c-8f1c-fa72d6b3
4b4c  8Gi     RWO         k8s-sc-aws    8s
```

33 34

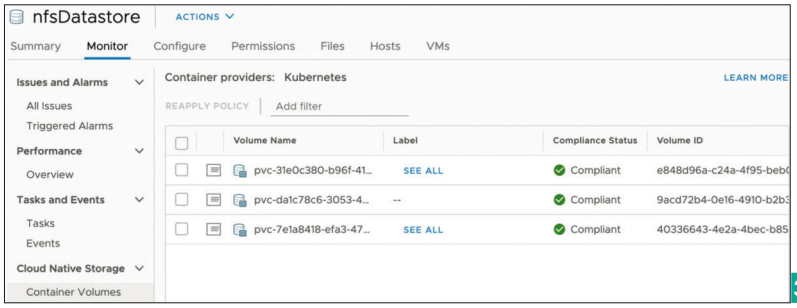
Étape 4a - Installer l'outil Velero afin de sauvegarder et restaurer l'application Rocketchat sur une infrastructure AWS

• Configurer un stockage de type objet (bucket S3) qui permettra de stocker les backups créés par Velero.

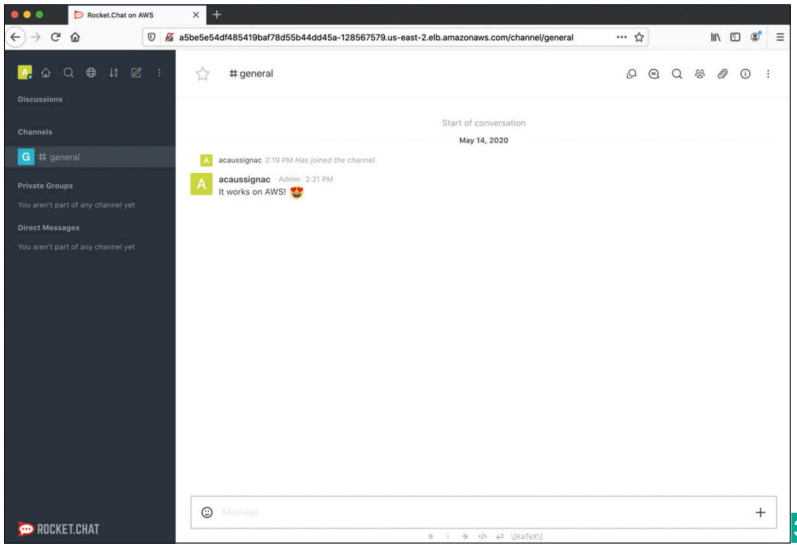
```
$ export AWS_ACCESS_KEY_ID=Access_key_ID
$ export AWS_SECRET_ACCESS_KEY=Secret_access_key
$ BUCKET=backup-velero
$ REGION=us-east-2
$ aws s3api create-bucket --bucket $BUCKET --region $REGION --create-bucket-configuration
 LocationConstraint=$REGION
$ aws iam create-user --user-name velero
```



31



32



33

34

```
$ cat > velero-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVolumes",
        "ec2:DescribeSnapshots",
```

```
        "ec2:CreateTags",
        "ec2:CreateVolume",
        "ec2:CreateSnapshot",
        "ec2:DeleteSnapshot"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:DeleteObject",
        "s3:PutObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
        "arn:aws:s3:::${BUCKET}/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::${BUCKET}"
      ]
    }
  ]
}
EOF

$ aws iam put-user-policy --user-name velero --policy-name velero --policy-document file://
velero-policy.json
$ aws iam create-access-key --user-name velero
{
  "AccessKey": {
    "UserName": "velero",
    "AccessKeyId": "Access_key_ID",
    "Status": "Active",
    "SecretAccessKey": "Secret_access_key",
    "CreateDate": "2020-05-14T14:22:32+00:00"
  }
}
```

Note : *récupérer* ***Access_key_ID et Secret_access_key*** *pour mettre à jour le fichier credentials-velero-aws ci-dessous.*

• Installer Velero server

```
$ wget https://github.com/vmware-tanzu/velero/releases/download/v1.3.2/velero-v1.3.2-
linux-amd64.tar.gz
$ tar zxvf velero-v1.3.2-linux-amd64.tar.gz
$ sudo mv velero-v1.3.2-linux-amd64/velero /usr/local/bin/velero && sudo chown root:root
/usr/local/bin/velero
$ cat credentials-velero-aws
[default]
aws_access_key_id=Access_key_ID
aws_secret_access_key=Secret_access_key
$ velero install --use-restic --provider aws --plugins velero/velero-plugin-for-aws:v1.0.1 --
bucket $BUCKET --backup-location-config region=$REGION --snapshot-location-config
region=$REGION --secret-file ./credentials-velero-aws
```

• Installer Velero CLI sur JUMPBOX.

```
$ go get github.com/vmware-tanzu/velero
$ velero backup create before-disaster --include-namespaces rocketchat
Backup request "before-disaster" submitted successfully.
Run `velero backup describe before-disaster` or `velero backup logs before-disaster` for more details.
$ velero backup describe before-disaster
...
Restic Backups (specify --details for more information):
  Completed:  1
```

35

• Supprimer entièrement l'application et la restaurer avec Velero.

```
$ kubectl delete ns rocketchat
namespace "rocketchat" deleted
```

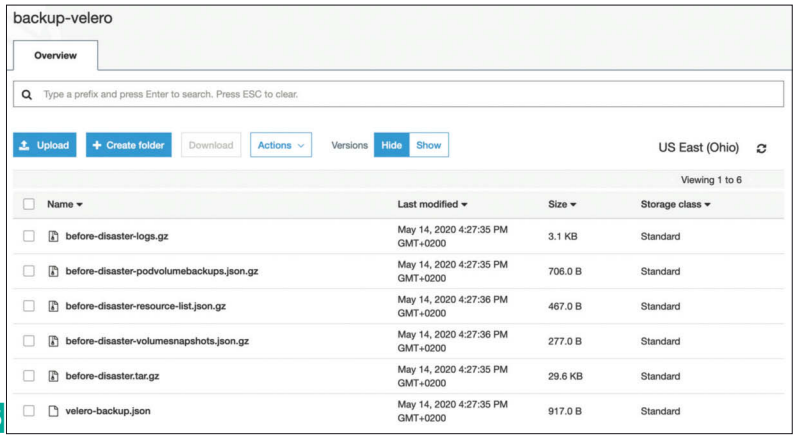
```
$ velero restore create --from-backup before-disaster --include-namespaces rocketchat
Restore request "before-disaster-20200514163124" submitted successfully.
Run `velero restore describe before-disaster-20200514163124` or `velero restore logs before-disaster
-20200514163124` for more details.
$ velero restore describe before-disaster-20200514163124
...
Restic Restores (specify --details for more information):
  Completed:  1
$ kubectl get svc -n rocketchat
NAME                        TYPE          CLUSTER-IP      EXTERNAL-IP  PORT(S)    AGE
rocketchat-mongodb          ClusterIP     10.106.115.246  <none>       27017/TCP3m4s
rocketchat-mongodb-headless ClusterIP     None            <none>       27017/TCP3m4s
rocketchat-rocketchat       LoadBalancer                  10.109.60.192  a97ba8ff0d398
4b5582e74786cf62de5-1943441192.us-east-2.elb.amazonaws.com  80:30266/TCP  3m4s
```

Note : *attention l'URL du service a changé.*

Étape 4b - Installer l'outil Velero afin de sauvegarder et restaurer l'application Rocketchat sur une infrastructure vSphere

• Configurer un stockage de type objet nommé Minio qui permettra de stocker les backups créés par Velero.

```
$ tkg set management-cluster management-cluster-vsphere
The current management cluster context is switched to management-cluster-vsphere
$ kubectl config use-context vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001
```



35

```
Switched to context "vsphere-cluster-prod-001-admin@vsphere-cluster-prod-001".
$ cat minio.yaml
image:
 tag: latest
accessKey: minio
secretKey: minio123
service:
 type: LoadBalancer
defaultBucket:
 enabled: true
 name: velero
persistence:
 size: 5G
$ kubectl create ns velero
namespace/velero created
$ helm install minio stable/minio --namespace velero -f minio.yaml
$ kubectl get svc --namespace velero -l app=minio
NAME   TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
minio  LoadBalancer  100.67.123.34  172.23.5.101  9000:32740/TCP  3m15s
```

36

• Installer Velero server

```
$  wget https://github.com/vmware-tanzu/velero/releases/download/v1.3.2/velero-v1.3.2-linux-amd64.tar.gz
$ tar zxvf velero-v1.3.2-linux-amd64.tar.gz
$ sudo mv velero-v1.3.2-linux-amd64/velero /usr/local/bin/velero && sudo chown root:root /usr/local/bin/velero
$ cat credentials-velero-vsphere
[default]
aws_access_key_id = minio
aws_secret_access_key = minio123
$ velero install --use-restic --provider aws --plugins velero/velero-plugin-for-aws:v1.0.0 --bucket velero --secret-file ./credentials-velero-vsphere --use-volume-snapshots=false --backup-location-config region=minio,s3ForcePathStyle="true",s3Url=http://minio.velero.svc:9000,publicUrl=http://172.23.5.101:9000
```

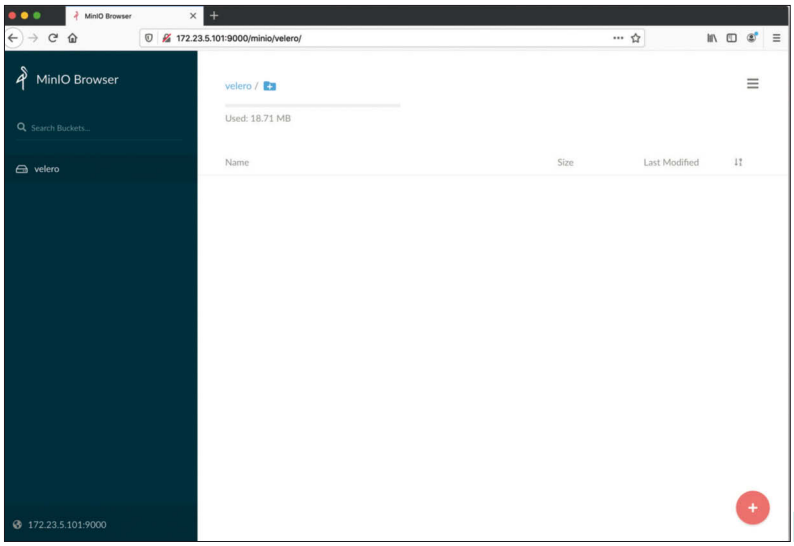
• Installer Velero CLI sur JUMPBOX.

```
$ go get github.com/vmware-tanzu/velero
$ velero backup create before-disaster --include-namespaces rocketchat
Backup request "before-disaster" submitted successfully.
Run `velero backup describe before-disaster` or `velero backup logs before-disaster` for more details.
$ velero backup describe before-disaster
...
Restic Backups (specify --details for more information):
 Completed:  1
```

• Supprimer entièrement l'application et la restaurer avec Velero.

```
$ kubectl delete ns rocketchat
namespace "rocketchat" deleted
```

```
$ velero restore create --from-backup before-disaster --include-namespaces rocketchat
Restore request "before-disaster-20200514145534" submitted successfully.
Run `velero restore describe before-disaster-20200514145534` or `velero restore logs before-disaster-20200514145534` for more details.
$ velero restore describe before-disaster-20200514145534
...
Restic Restores (specify --details for more information):
 Completed:  1
```



36

Étape 5 – Mettre à jour une installation TKG

TKG suit une cadence de sortie rapide (environ 6 semaines), ce qui permet d'offrir à ses utilisateurs la garantie de disposer d'une distribution Kubernetes en ligne avec les mises à jour proposées par la communauté (tout en bénéficiant du support VMware). Rappelons en effet qu'une nouvelle version majeure de Kubernetes sort tous les 3 mois : sans disposer de solides outils pour mettre à jour vos clusters Kubernetes, vous serez rapidement en retard par rapport au rythme de sortie des versions, ce qui très vite introduit de la dette technique sur votre infrastructure. En s'appuyant sur Cluster API et kubeadm, TKG vous permet de déployer ces mises à jour rapidement et sans interruption de service. Le déploiement d'une nouvelle version de Kubernetes sur un cluster est complètement automatisé : une seule commande à lancer, et laissez TKG s'occuper du reste !

```
$ tkg upgrade cluster vsphere-cluster-dev-001
```

L'exécution de cette commande prend environ 15 minutes sur un cluster composé de 3 nœuds. Retrouvez plus d'informations sur le déploiement des mises à jour TKG dans la documentation officielle en ligne.

Conclusion

Comme vous avez pu le constater, vous disposez à présent de clusters K8s pleinement fonctionnels, déployés et administrés depuis un même point centra et pleinement supportés par un éditeur et ce, qu'ils soient situés dans vos centres informatiques (On-Premise) ou sur des Clouds publics. C'est un point clé aujourd'hui de disposer d'une solution orienté portabilité. TKG a été pensé pour simplifier toutes les opérations *Day 1* et *Day 2* en vue de fournir des clusters K8s certifiés CNCF et instanciables sur tous types d'infrastructures via l'apport des fournisseurs (Alibaba, AWS, Azure, GCP, IBM Cloud, vSphere, …). C'est une technologie Open Source qui intègre également un ensemble de produits nécessaires à l'exploitation d'un cluster (Administration, Conformité, Sauvegarde, Restauration…). VMware en assure son support 24x7 et permet également le développement spécifique lié au contexte de ses utilisateurs. Nous vous donnons donc rendez-vous dans le prochain numéro pour voir comment consommer ces clusters K8s dans nos chaînes de CI/CD afin d'y installer nos applications.

Retrouvez la partie 2 dans Programmez! 243.

Thierry LERICHE
Architecte et tech lead
@ThierryLeriche

Daniel OPITZ
Software Developer
@dopitz

À la découverte de Slim 4

Slim est un microframework open source proposant un système de routing et de middleware très simple à mettre en œuvre. Moins connu que les mastodontes, il mérite un sérieux coup d'œil. Dans ce tuto, je vous propose d'écrire, de manière naïve et simplifiée, à l'aide de Slim 4, un petit site web reprenant les besoins et contraintes habituels.

Il existe, grosso modo, trois façons de créer un site Web en PHP :

- Tout écrire en « PHP pur », pour les purs et durs ;
- Employer un gros framework, comme Laravel (Programmez n° 235) ou Symphony, nécessitant une belle montée en compétence ;
- S'intégrer dans un CMS comme Wordpress ou Drupal, ce qui ne correspond pas à tous les cas d'utilisation.

L'utilisation d'un *framework* est toujours intéressante car elle s'accompagne habituellement d'un ensemble de normes, de bonnes pratiques et d'une communauté active. Cela simplifie notamment le passage d'un projet à l'autre ainsi que l'intégration de nouveaux membres dans l'équipe. Toutefois, au fur et à mesure des versions, les frameworks ajoutent de nouvelles composantes et responsabilités, les rendant toujours plus lourds et complexes.

Comme son nom l'indique, Slim a pour vocation de rester léger. On parlera de *microframework*. En se concentrant sur les aspects *routing* et *middleware*, Slim reste accessible, même aux débutants. Les aspects non gérés restent, si nécessaires, à la charge des développeurs. Mais c'est à la fois une liberté et une contrainte.

Note : les codes présentés dans cet article sont disponibles sur le site programmez.com et GitHub.

Routage

Disons-le carrément : on peut développer un site sans utiliser de système de routage. Ce n'est absolument pas indispensable, mais c'est pratique !

Un routeur va habituellement apporter, entre autres :

- Une centralisation de la gestion des routes ;
- La possibilité de regrouper des opérations sur plusieurs routes, comme l'obligation pour un utilisateur d'être authentifié, ou l'envoi automatique de logs vers le BI ;
- Des URL plus jolies.

Prenons un exemple pour ce dernier point. Disons qu'on souhaite consulter la fiche, en français, d'un livre sélectionné par son code ISBN. Sans système de routage, l'adresse utilisée ressemblerait à celle-ci :

www.monsite.com/book.php?lang=fr&isbn=12345

Cela fonctionne très bien mais on aimerait avoir une adresse plus jolie, sans extension et sans paramètre, comme celle-ci : www.monsite.com/fr/book/12345

On pourrait même vouloir traduire l' « action » dans la langue de l'utilisateur et même s'abstraire de l'indiquer, en ne changeant que la configuration (ie. sans toucher au code de la fonctionnalité, ni dupliquer le fichier PHP d'entrée). On aurait alors une adresse comme celle-ci : www.monsite.com/livre/12345

Cela nous intéresse pour les pages d'un site mais aussi pour des web services, pour lesquels c'est plus ou moins déjà la norme, et on aurait alors une adresse préfixée par « api » ainsi qu'un numéro de version, comme celle-ci :

www.monsite.com/api/v1/books/12345

On peut également imaginer qu'une action soit référencée par plusieurs adresses pour des raisons de SEO ou de promotion, par exemple :

www.monsite.com/promo/noel-2020/livre/12345

Il est bien entendu possible de créer son propre système de routage, à base de configuration Nginx/Apache et de règles PHP, mais c'est vite laborieux. Des bibliothèques comme Slim font déjà le travail. Elles sont légères et fiables. Et elles disposent de communautés pour répondre aux questions.

Note : Slim n'est pas seulement un routeur ; mais c'est certainement ce que vous découvrirez en premier.

Serveur

On peut servir le site Web depuis le serveur intégré à PHP, ce qui est certainement le plus simple en développement. On peut aussi le servir depuis une VM, en production comme en local, avec Vagrant par exemple.

La documentation de Slim propose des configurations pour les serveurs Apache, Nginx, IIS, Lighttpd, etc. Voici un exemple pour Nginx, sur lequel on voit que c'est très simple.

```
server {
 listen [::]:80;
 listen 80;
 server_name my-slim-website.com;

 index index.php;
 error_log /somepath/error.log;
 access_log /somepath/access.log;

 root /somepath/www/public;

 location / {
  try_files $uri /index.php$is_args$args;
 }

 location ~ \.php {
  try_files $uri =404;
  fastcgi_split_path_info ^(.+\.php)(/.+)$;
  include fastcgi_params;
  fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
  fastcgi_param SCRIPT_NAME $fastcgi_script_name;
  fastcgi_index index.php;
  fastcgi_pass unix:/var/run/php/php7.3-fpm.sock;
 }
}
```

L'idée est de n'avoir qu'*un seul et unique fichier PHP*, dans le dossier *public*, vers lequel on envoie toutes les requêtes. Et c'est Slim qui gère le routage.

Dans un premier temps, juste pour tester la configuration du serveur, j'écris un simple Hello World dans *index.php*.

Ajout de Slim 4

Slim est proposé en *Open Source* sous *licence MIT* et le code source est disponible sur GitHub.

Note : Il y a de nombreuses différences entre les versions 3 et 4 de Slim. La plupart des exemples utilisant Slim 3 ne fonctionnent plus avec Slim 4, même si une large majorité (mais pas tous) des concepts restent d'actualité.

J'utilise Composer pour gérer les dépendances. C'est un classique en PHP. Pour commencer, j'initialise un nouveau projet avec *composer init*. Bien entendu il faut ajouter Slim 4. On en profite pour ajouter immédiatement des dépendances pour les *PSR-7* (HTTP message) et *PSR-11* (Container).

```
composer require slim/slim:"4.*"
composer require slim/psr7
composer require php-di/php-di
```

Note : en fonction des besoins spécifiques, on peut préférer d'autres implémentations de PSR-7, comme *Nyhom* ou *Guzzle*.

Première route

Voyons donc comment transformer notre Hello World en utilisant Slim et une première route correspondant à la racine du site.

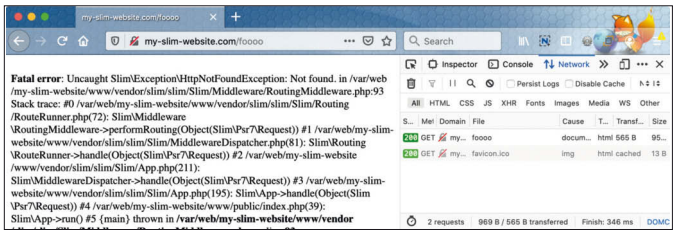
```
// 1
use Psr\Http\Message\ResponseInterface as Response;
use Psr\Http\Message\ServerRequestInterface as Request;

// 2
// use Slim\Psr7\Response;
// use Slim\Psr7\Request;

use Slim\Factory\AppFactory;

// 3
require __DIR__ . '/../vendor/autoload.php';

// 4
$app = AppFactory::create();

// 5
$app->get('/', function (Request $request, Response $response): Response {
 $response->getBody()->write('<h1>Hello World !</h1>');
 return $response;
});

// 6
$app->run();
```

Les actions de Slim manipulent les objets *Request* et *Response*. On peut utiliser ceux du PSR (1) ou directement ceux de Slim (2). L'avantage d'utiliser les classes du PSR est que cela reste plus standard. Il sera donc possible de passer sur une autre implémentation plus tard. Cela dit, il est finalement assez rare de changer de framework en cours de route, surtout s'il est structurant, et encore plus sur des petits projets. L'avantage d'utiliser les classes de Slim est qu'elles proposent des fonctionnalités additionnelles. Toutefois, une règle pourrait être d'utiliser les objets génériques tant qu'on n'a pas besoin de spécifique.
Bien entendu, puisqu'on utilise Composer, il faut charger l'*autoload* (3) et les dépendances installées dans le dossier *vendor*.

On entre ensuite dans le vif du sujet. Les blocs (4) et (6) permettent respectivement d'initialiser l'application et de la lancer. Slim 4 utilise un *factory* pour créer l'objet *Slim\App* alors que Slim 3 demandait d'appeler directement le constructeur.
Enfin, le bloc (5) définit les différentes routes. Le routeur de Slim est une surcouche de *Fast-route*. Il reconnaît les verbes *GET*, *POST*, *PUT*, *DELETE*, *OPTIONS* et *PATCH*. Il suffit d'appeler la fonction correspondante pour préciser celui qui nous intéresse. Il est également possible de tout accepter avec *any* ou d'en sélectionner plusieurs avec *map*.
La fonction *get/post/delete*/etc. prend une fonction en second argument. Celle-ci ne sera appelée que si la route correspond. Elle prend la requête et la réponse en entrée et renvoie la réponse complétée.

Affichage des erreurs

Pour l'instant il n'y qu'une seule route configurée, sur la racine « / » du site. Tous les appels d'autres URLs se termineront donc en erreur.



Bon, ce n'est pas très lisible. On va améliorer ça. C'est l'occasion d'ajouter un premier middleware basique.

```
$displayErrorDetails = true;
$logErrors = false;
$logErrorDetails = false;
$app->addErrorMiddleware($displayErrorDetails, $logErrors, logErrorDetails);
```

L'affichage devient tout de suite plus clair. Au passage, on remarque que le code http d'erreur est un *404* alors qu'on recevait un code *200* avant l'ajout du middleware d'erreur. Bien entendu, il faudra régler les paramètres plus finement avant de passer en prod.

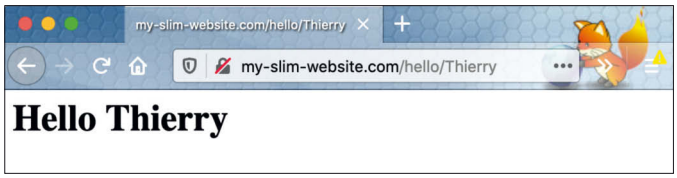
Route avec attributs

Un des grands intérêts d'utiliser un routeur est d'avoir des adresses avec des attributs. Pour définir une route avec des attributs, il suffit de les indiquer entre accolades :

```
$app->get('/', function (Request $request, Response $response): Response {
 $response->getBody()->write('<h1>Hello World !</h1>');
 return $response
});

$app->get('/hello/{name}', function (Request $request, Response $response, array $args): Response {
 $name = $args['name'];
 // $name = $request->getAttribute('name');
 $response->getBody()->write('<h1>');
 $response->getBody()->write('Hello ');
```

```
    $response->getBody()->write($name);
    $response->getBody()->write('</h1>');
    return $response;
});
```

Les valeurs sont alors fournies dans le tableau *args*, passé en troisième argument de la fonction, mais il est également possible de les récupérer directement depuis la requête.



Au passage, il est recommandé de nommer les routes. Ce sera utile plus tard, pour faire des redirections ou générer le href des liens par exemple.

```
return function (App $app) {
    $app->get('/', function (Request $request, Response $response): Response {
        ...
    })->setName('root');

    $app->get('/hello/{name}', function (Request $request, Response $response, array $args): Response {
        ...
    })->setName('hello')
};
```

Organisation

Avant d'aller plus loin, il vaut mieux réorganiser le code, faute de quoi il va rapidement devenir difficile à lire. Je crée les dossiers *config* et *src* à la racine, et donc au même niveau que le dossier *public*. Seul ce dernier est accessible de l'extérieur, donc il n'y a aucun souci à se faire. Le dossier *config* va contenir les fichiers nécessaires au lancement de l'application et le dossier *src* en contiendra les sources. Je crée ensuite *routes.php* dans *config* et j'y déplace le code des routes :

```
return function (App $app) {
    $app->get('/', function (Request $request, Response $response): Response {
        $response->getBody()->write('<h1>Hello World !</h1>');
        return $response;
    });

    $app->get('/hello/{name}', function (Request $request, Response $response, array $args): Response {
        $name = $args['name'];
        // $name = $request->getAttribute('name');
        $response->getBody()->write('<h1>');
        $response->getBody()->write('Hello ');
        $response->getBody()->write($name);
        $response->getBody()->write('<h1>');
        return $response;
    });
};
```

Le *require* renvoie une fonction que je peux appeler depuis *index.php*, qui devient donc bien plus simple :

```
$routes = require __DIR__ . '/../config/routes.php';
$routes($app);
```

Les puristes préfèreront même faire l'appel d'un seul coup pour éviter de déverser *routes* dans le scope global.

```
(require __DIR__ . '/../config/routes.php')($app);
```

Dans la suite, quand c'est possible, j'utiliserai systématiquement la même organisation.
Pendant qu'on y est, j'active dès à présent le PSR-4 (Autoloader) dans *composer.json*, en précisant donc le dossier *src* pour les sources, car on va très rapidement écrire quelques classes.

```
"autoload": {
    "psr-4": {
        "App\\": "src/"
    }
}
```

On n'oubliera pas de lancer un coup de *composer dump-autoload* pour que les nouveaux éléments soient bien pris en compte.
Je vous proposerai une seconde réorganisation à la fin.

Containeur

Le containeur permet notamment l'injection de dépendances. On peut le construire directement ou déléguer cette tâche à un *builder* auquel on passe la configuration. Il y a plusieurs manières de le faire mais restons simple. Je crée donc *settings.php* dans *config*, pour y placer mes éléments de conf (pour l'instant, il n'y a que l'affichage des erreurs) et j'y reviendrai au fur et à mesure.

```
return function (ContainerBuilder $containerBuilder) {
    $settings = [
        'displayErrorDetails' => true, // set to false in prod
        'logErrors' => false,
        'logErrorDetails' => false,
    ];

    $containerBuilder->addDefinitions([
        'settings' => $settings,
    ]);
};
```

Il est bien entendu possible de compléter cette configuration en fonction de l'environnement et de fichiers complémentaires.
Il n'y a plus qu'à l'utiliser dans *index.php* pour créer le containeur.

```
// 4
// Instantiate PHP-DI ContainerBuilder
$containerBuilder = new ContainerBuilder();

// Set up settings
(require __DIR__ . '/../config/settings.php')($containerBuilder);

// Build PHP-DI Container instance
$container = $containerBuilder->build();
AppFactory::setContainer($container);

// Instantiate the app
$app = AppFactory::create();
```

Je crée *middlewares.php* dans *config*, pour y placer (sans surprise) tout ce qui va concerner les middlewares. J'en profite pour activer le middleware de routing et configurer le middleware d'affichage des erreurs avec les éléments de setting récupérés depuis le container.

```
return function (App $app) {
 $container = $app->getContainer();

 // Settings
 $settings = $container->get('settings');

 // Add Routing Middleware
 $app->addRoutingMiddleware();

 // Add Error Middleware
 $displayErrorDetails = $settings['displayErrorDetails'];
 $logErrors = $settings['logErrors'];
 $logErrorDetails = $settings['logErrorDetails'];
 $app->addErrorMiddleware($displayErrorDetails, $logErrors, $logErrorDetails);
};
```

Logger

On en profite pour ajouter un logeur, qu'on pourra injecter dans nos composants. De nombreux choix sont disponibles en PHP. Je vous propose d'ajouter une dépendance à Monolog, qui implémente le PSR-3 (Logger) :

```
composer require monolog/monolog
```

La configuration des logs va naturellement aller dans *settings.php*.

```
return function (ContainerBuilder $containerBuilder) {
 $settings = [
  ...

  // Logger
  'logger' => [
   'name' => 'my-slim-website',
   'path' => '/somepath/app.log',
   'level' => Logger::DEBUG,
  ],
 ];
```

La suite est du déjà-vu. Je crée *loggers.php* dans *config* pour configurer le logeur à l'aide des valeurs indiquées dans les settings. Je définis directement un processeur d'UID, bien pratique en run (cf. plus loin).

```
return function($app) {
 $container = $app->getContainer();
 $container->set(LoggerInterface::class, function($container) {
  $settings = $container->get('settings');
  $loggerSettings = $settings['logger'];
  $name = $loggerSettings['name'];
  $path = $loggerSettings['path'];
  $level = $loggerSettings['level'];

  $logger = new Logger($name);
  $processor = new UidProcessor();
  $logger->pushProcessor($processor);
  $handler = new StreamHandler($path, $level);
  $logger->pushHandler($handler);
  return $logger;
 });
};
```

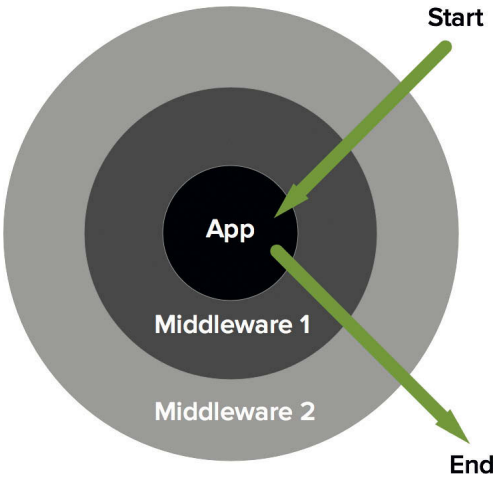
Je fixe le nom que je souhaite pour le logger dans le container. Le plus simple est de récupérer le nom de la classe principale ou de l'interface, ici *LoggerInterface ::class*.

On peut ajouter la dépendance dans le container directement ou en lui passant une fonction de construction, comme dans l'exemple. Passer une fonction a l'avantage de ne créer les objets que lorsqu'ils sont utilisés/injectés, en mode *lazy*.

Le logeur peut donc être injecté et utilisé dans l'app, par exemple dans les actions, les middlewares ou tout autre composant qu'on va créer. Pour l'instant, on le laisse tranquille mais on va l'injecter et l'utiliser dès la prochaine section.

Middlewares

Les middlewares de Slim 4 sont très simples à utiliser. Ils implémentent le PSR-15 (HTTP Server Request Handlers) et fonctionnent par couches concentriques (comme un oignon) qui s'additionnent les unes aux autres avec l'app. Il faut traverser les couches dans un sens puis dans l'autre, mais il faut arriver jusqu'à la couche centrale (ie. l'app) avant que la route ne soit activée.



Je crée un premier middleware invocable, inspiré de la documentation, dans *src/Middlewares* car ce n'est pas un élément de conf. N'oubliez pas de mettre à jour votre autoload.

```
class BeforeMiddleware {

 public function __invoke(Request $request, RequestHandler $handler): Response {
  $response = $handler->handle($request);
  $existingContent = (string) $response->getBody();

  $response = new Response();
  $response->getBody()->write('<div>BEFORE</div>' . $existingContent);

  return $response;
 }
```

La fonction *handhler* permet de lancer la suite du traitement.

Note : la signature de la function *__invoke* a changé depuis Slim 3, puisqu'elle prenait alors les arguments *$request*, *$response* et *$next*. Ce dernier était un callback pour continuer le traitement.

Sans surprise, je l'ajoute dans *middlewares.php*, à la suite par exemple.

```
return function (App $app) {
 $container = $app->getContainer();
 ...

 // My middlewares
 $app->add(BeforeMiddleware::class);
};
```

Dans l'exemple je me contente donc d'ajouter « BEFORE » en haut de la page.

Les plus attentifs auront remarqué que cet exemple, issu de la documentation, ne respecte pas le PSR-15. Il a également le gros défaut de redéfinir la réponse et donc de perdre les éléments éventuellement présents dans l'ancienne réponse. Il vaut mieux implémenter l'interface *MiddlewareInterface* et définir la fonction *process*.

```
class BeforeMiddleware implements Middleware {
 public function process(Request $request, RequestHandler $handler): Response {
  ...
```

J'injecte maintenant le logger dans le middleware, comme promis, histoire d'avoir un exemple, et d'écrire un premier log. Le plus simple est d'indiquer les dépendances au niveau du constructeur mais il y a d'autres possibilités. On aurait aussi pu annoter le code.

```
cclass BeforeMiddleware implements Middleware {

 private $logger;

 public function __construct(Logger $logger) {
  $this->logger = $logger;
 }

 public function process(Request $request, RequestHandler $handler): Response {
  $this->logger->info("Hi from the Before Middleware.");
  ...
```

Je vérifie que mon log apparaisse bien dans *app.log*. On voit que l'*uid* est automatiquement ajouté à la fin du log. Il est possible de le récupérer pour l'afficher dans vos pages d'erreur par exemple.

```
[2020-04-17T06:50:04.910541+00:00] my-slim-website.INFO: Hi from the Before Middleware.
[] {"uid":"d356fbb"}
```

Le middleware est ajouté pour l'ensemble de l'app, et donc l'ensemble des routes. Dans certains cas, comme pour protéger des routes nécessitant d'être authentifiées, on préférera ajouter le middleware au niveau du routeur, soit pour une route donnée, soit pour un groupe de routes.

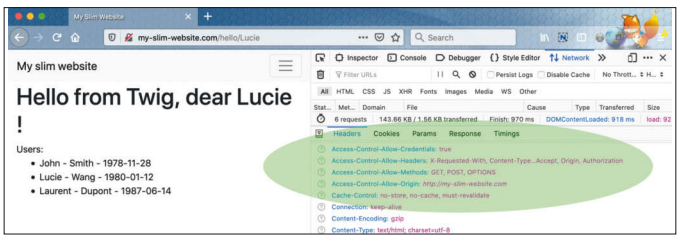
```
return function (App $app) {
 $app->get('/', function (Request $request, Response $response): Response {
  ...
 })->setName('root');

 $app->get('/hello/{name}', function (Request $request, Response $response, array $args): Response {
  ...
 })->setName('hello')
 ->add(AfterMiddleware::class);
};
```

Pour l'exemple, j'ai aussi écrit *AfterMiddleware* dont je vous laisse en deviner le contenu, très simple. Je l'ajoute uniquement sur la route « hello ». Il ne s'exécutera donc que sur cette route.

On peut créer des middlewares pour toutes sortes d'opérations plus intéressantes que d'écrire BEFORE et AFTER sur votre page. Par exemple, je vous invite à ajouter immédiatement un middleware qui s'occupe du CORS (Cross-Origin Resource Sharing). Il suffit d'ajouter les bons headers à votre réponse. Voici ce que cela donnerait dans les grandes lignes.

```
class CorsMiddleware implements Middleware {

 public function process(Request $request, RequestHandler $handler): Response {
  $response = $handler->handle($request);
  return $response
    ->withHeader('Access-Control-Allow-Origin', 'http://my-slim-website.com')
    ->withHeader('Access-Control-Allow-Headers', 'X-Requested-With, Content-Type, Accept, Origin,
Authorization')
    ->withHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS')
    ->withHeader('Access-Control-Allow-Credentials', 'true');
```

Ici je n'ai mis que GET et POST car je ne prévois pas d'utiliser autre chose.



Et vous l'aurez deviné, l'ordre compte. Les middlewares s'enchaînent dans l'ordre inverse de leur ajout. Le dernier ajouté sera donc le premier invoqué.

Actions et contrôleurs

C'est bien gentil d'avoir des fonctions dans le routeur mais je n'imagine absolument pas defaire un site complet de cette manière. Ça deviendrait ingérable et très vite chaotique. Je préfère donc écrire des classes d'action en remplacement. Elles seront automatiquement chargées sans avoir à les renseigner dans le container.

On peut écrire des contrôleurs ou des actions invocables. Les deux ont leurs

avantages et leurs inconvénients. Les actions n'ont que la fonction *__invoke*. Les contrôleurs peuvent définir plusieurs fonctions, comme getFoo ou postFoo par exemple.

```
class InvokableAction {
 public function __invoke(Request $request, Response $response, $args): Response {
  $id = $args['id'];

  $response->getBody()->write('<h1>');
  $response->getBody()->write('Invoke !');
  $response->getBody()->write($id);
  $response->getBody()->write('</h1>');

  return $response;
 }
```

Avec les contrôleurs, on peut préciser la méthode à appeler, par exemple en la faisant correspondre au verbe utilisé, ici *getFoo* et *postFoo*.

```
class CustomController {
 public function getFoo(Request $request, Response $response, $args): Response {
  $id = $args['id'];

  $response->getBody()->write('<h1>');
  $response->getBody()->write('Custom !');
  $response->getBody()->write($id);
  $response->getBody()->write('</h1>');
  return $response;
 }

 public function postFoo(Request $request, Response $response, $args): Response {
  $postParams = $request->getParsedBody();
  $firstname = $postParams['firstname'];

  $response->getBody()->write('<h1>');
  $response->getBody()->write('Custom !');
  $response->getBody()->write($firstname);
  $response->getBody()->write('</h1>');

  return $response;
 }
```

On peut convenir que *getFoo* permet d'afficher un formulaire et que *postFoo* permet de le soumettre. Avec des actions invocables, j'aurais plutôt écrit deux actions spécialisées, une pour l'affichage et une pour la soumission.
Le tableau *args* ne contient que les valeurs issues des variables des routes. Les valeurs indiquées dans les formulaires, et reçues en *POST*, peuvent être récupérées dans le body grâce à la fonction *getParsedBody*. Les valeurs passées dans la query (ie. l'adresse) peuvent être récupérées, quant à elles, grâce à la fonction *getQueryParams*. Je vous laisse deviner le nom de la fonction à utiliser pour les valeurs passées dans le header.
Dans le cas d'un web service, recevant du *JSON* ou du *XML*, qui ne sont pas pris en compte par le PSR-7, on activera le middleware dédié dans *middleware.php* pour ne pas avoir à faire le décodage soi-même.

```
// Parse json, form data and xml
$app->addBodyParsingMiddleware();
```

À titre d'illustration, je défini un groupe auquel j'ajoute *AfterMiddleware* que j'avais écrit plus tôt. J'indique deux routes différentes pour l'action maison : une pour *GET* et une pour *POST*. Les adresses correspondantes n'ont pas besoin d'être absolument les mêmes.

```
return function (App $app) {
 ...

 $app->group('/act', function(Group $group) {
  $group->get('/inv/{id}', InvokableAction::class);
  $group->get('/custom/{id}', CustomController::class . ':getFoo');
  $group->post('/custom', CustomController::class . ':postFoo');
 })->add(AfterMiddleware::class);
```

Pour la forme, voici le résultat dans Firefox.
Il n'est pas obligatoire d'indiquer un complément d'URL pour un group. Celui-ci peut être laissé vide. Et il est possible d'avoir plusieurs niveaux d'imbrication, avec ou sans complément. Cela permet, par exemple, de regrouper plusieurs routes d'administration, en y associant un middleware qui vérifie les droits de l'utilisateur connecté, sans préciser le pattern « admin » dans l'adresse.
Est-ce mieux d'écrire des actions ou des contrôleurs ? les contrôleurs sont plus pratiques/lisibles pour regrouper plusieurs verbes de manière logique. Les actions (en plus d'implémenter le *PSR-15*) sont plus efficaces, notamment parce qu'elles nécessitent moins d'injections. En effet, on ne va pas forcément injecter autant de choses pour un *GET* que pour un *POST*. Par exemple, on n'aura probablement pas besoin d'injecter *PDO* (cf. plus bas) pour afficher un formulaire. Et de la même manière, on n'aura pas besoin de *Twig* (cf. plus bas) pour valider un formulaire avant de forwarder. Et bien entendu, il est toujours possible de regrouper des actions par sous-dossier.

Twig

Pour le moment, les actions écrivent leurs réponses à coups de *write*. Ce n'est ni pratique ni agréable. On va donc installer un système de template dans l'application. Je vous propose Twig qui, en plus d'être à la mode, est simple, rapide, flexible et sécurisé.
Vous devez commencer à connaître la procédure. Il faut d'abord ajouter une dépendance vers Twig. Ici, je vais même choisir Twig-view qui sera encore plus facile à intégrer.

```
composer require slim/twig-view
```

J'ajoute un peu de conf dans *settings.php* pour spécifier où seront les templates et si j'active le cache. C'est important d'activer le cache de Twig car la compilation des templates est un peu longue. En local, toutefois, je le laisse désactivé car je vais modifier les fichiers tout le temps.

```
return function (ContainerBuilder $containerBuilder) {
 $settings = [
  ...

  'twig' => [
   'path' => dirname(__DIR__) . '/views',
   'cache' => false // '/somepath/tmp/cache',
  ],
 ];
```

Je crée *templates.php* dans *app* pour ajouter Twig au containeur.

```
return function($app) {
```

```
$container = $app->getContainer();

$container->set('view', function($container) {

 $settings = $container->get('settings');
 $twigSettings = $settings['twig'];
 $root = dirname(__DIR__);
 return  Twig::create($twigSettings['path'], [
  'cache' => $twigSettings['cache']
 ]);
});

$container->set(Twig::class, function($container) {
 return $container->get('view');
});
};
```

Et je n'oublie pas d'ajouter le middleware de Twig.

```
$app->add(TwigMiddleware::createFromContainer($app));
```

Cela me permet de l'injecter dans mes actions et mes contrôleurs, par exemple pour la route *hello* que j'ai transformée en contrôleur entre temps.

```
class HelloController {
 private $twig;

 public function __construct(Twig $twig) {
  $this->twig = $twig;
 }
```

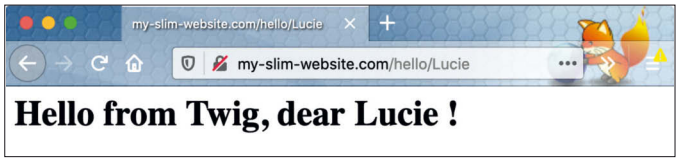
Je n'ai plus qu'à demander au contrôleur de déléguer le rendu à Twig en lui passant le nom du template et les éventuels paramètres issus de mon traitement (ici il n'y a que *name*).

```
class HelloController {
 ...

 public function hello(Request $request, Response $response, $args): Response {
  $name = $request->getAttribute('name');

  $params = [
   'name' => $name,
  ];

  return $this->twig->render($response, 'hello.twig', $params);
 }
```

Forcément, je dois aussi créer le fichier *hello.twig* dans *views*, conformément à la config.

```
<h1>
 Hello from Twig, dear {{ name }} !
</h1>
```

La syntaxe pour inclure le contenu d'une variable est de l'écrire entre double accolades, comme vous l'aviez probablement deviné en lisant l'exemple…



Histoire que ça ressemble à quelque chose, je complète le template pour qu'il s'intègre dans un vrai site, et je crée un petit menu.
Le fichier *hello.twig* étend *template.twig*

```
{% extends 'template.twig' %}

{% block content %}
 <h1>
  Hello from Twig, dear {{ name }} !
 </h1>
{% endblock %}
```

Le fichier *template.twig* inclue une structure HTML classique. J'en profite pour ajouter les dernières versions de *Bootstrap* et *jQuery*.

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="...">
  <title>My Slim Website</title>
  <link rel="stylesheet" href="https://.../bootstrap.min.css" ...>
 </head>

 <body>
  {% include "menu.twig" %}

  <div class="container-fluid">
   {% block content %}{% endblock %}
  </div>

  <script src="https://.../jquery.min.js" ...></script>
  <script src="https://.../popper.min.js" ...></script>
  <script src="https://.../bootstrap.min.js" ...></script>
 </body>
</html>
```

Et pour le menu, pour l'instant, je me contente de mettre des liens vers la racine et le Hello World.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
 <a class="navbar-brand" href="#">My slim website</a>
 <button class="navbar-toggler" type="button" ...>
  <span class="navbar-toggler-icon"></span>
 </button>
 <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
  <div class="navbar-nav">
   <a class="nav-item nav-link" href="/">Home <span class="sr-only"></span></a>
   <a class="nav-item nav-link" href="/hello/Lucie">Hello Lucie</a>
   <a class="nav-item nav-link" href="/hello/Thierry">Hello Thierry</a>
  </div>
 </div>
</nav>
```

L'objectif de cet article est de vous présenter Slim 4. Vous me pardonnerez si j'écris des templates simple, et moches.

On voit que les adresses indiquées dans *menu.twig* sont fixes. On pourrait les construire nous-même mais ce serait laborieux. Heureusement, le helper *url_for* permet de se lier directement au routeur, en exploitant le nom des routes, et en passant les paramètres si nécessaires.

```
<a class="..." href="{{ url_for(root) }}">Home <span class="sr-only"></span></a>
<a class="..." href="{{ url_for('hello', {'name': 'Lucie'}) }}">Hello Lucie</a>
<a class="..." href="{{ url_for('hello', {'name': 'Thierry'}) }}">Hello Thierry</a>
```

Responder

Parce que je veux séparer un peu plus les actions et Twig, et aussi parce que je prévois de m'en servir plus tard (erreurs, redirections), je me propose de créer un responder.

```
class Responder {
  private $twig;
  private $responseFactory;

  public function __construct(Twig $twig, ResponseFactoryInterface $responseFactory) {
    $this->twig = $twig;
    $this->responseFactory = $responseFactory;
  }

 public function render(Response $response, string $template, array $params = [], int $httpStatusCode
 = 200): Response {
   $params2 = array_merge($params, [
    'httpStatusCode' => $httpStatusCode,
   ]);
   // PHP 7.4: $params2 = [ ...$params, 'foo' => 'lorem' ]

   return $this->twig->render(
     $response->withHeader('Content-Type', 'text/html; charset=utf-8')->withStatus($httpStatusCode),
      $template,
      $params2);
 }
```

J'aimerais attirer votre attention sur le fait que la syntaxe de *array_merge* peut être simplifiée depuis PHP 7.4.
Si vous êtes attentif, vous aurez sans doute remarqué qu'il manque un peu de glue pour *ResponseFactoryInterface*. Je l'ajoute tout simplement dans *templates.php*, de manière lazy.

```
return function($app) {
  $container = $app->getContainer();
  ...

  $container->set(ResponseFactoryInterface::class, function($container) use ($app) {
    return $app->getResponseFactory();
  });
};
```

Handler d'erreur

Maintenant qu'on sait afficher de belles (no comment) pages, ce serait bien d'afficher de belles erreurs, par exemple quand on tombe sur une *404*. Et pour cela, il faut écrire un handler.

```
class MyErrorHandler {

  private $responder;
  private $logger;

  public function __construct(Responder $responder, Logger $logger) {
    $this->responder = $responder;
    $this->logger = $logger;
  }

  public function __invoke(Request $request, Throwable $exception, bool $displayErrorDetails, bool $logErrors):
 Response {

    $path = $request->getUri()->getPath();

    // Log error
    if ($logErrors) {
      $this->logger->error(sprintf(
        'Error: [%s] %s, Method: %s, Path: %s',
        $exception->getCode(),
        $exception->getMessage(),
        $request->getMethod(),
        $path
      ));
    }

    // Detect status code
    $httpStatusCode = $this->getHttpStatusCode($exception);

    // Error message
    $genericErrorMessage = $this->getErrorMessage($exception, $httpStatusCode, $displayErrorDetails);

    $response = $this->responder->createResponse();
    return $this->responder->render(
      $response,
      '/http_error.twig',
      ['httpStatusCode' => $httpStatusCode, 'genericErrorMessage' => $genericErrorMessage],
      $httpStatusCode);
  }
```

Je l'enregistre dans *middlewares.php*, où j'ai déjà mis le middleware de gestion d'erreurs.

```
$errorMiddleware = $app->addErrorMiddleware($displayErrorDetails, $logErrors, $logErrorDetails);
$errorMiddleware->setDefaultErrorHandler($container->get(MyErrorHandler::class));
```

Pour que ça fonctionne, je complète le responder, qui commence à prendre tout son sens.

```
class Responder {
  ...

  public function createResponse(): Response {
```

Boutique Programmez!

Les anciens numéros de PROGRAMMEZ!

Le magazine des développeurs

226

231

236

238

239

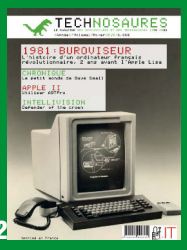
HS 01 été 2020

Tarif unitaire 6,5 € (frais postaux inclus)

Technosaures

Le magazine à remonter le temps !

N°1



N°2

N°3

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

Histoire de la micro-informatique 1973-2007

12,99 €
(frais postaux inclus)

☐ 226	: __ ex	☐ 238	: __ ex
☐ 231	: __ ex	☐ 239	: __ ex
☐ 236	: __ ex	☐ HS 01	: __ ex

soit __ exemplaires x 6,50 € = ____ €

Technosaures ☐ N°1 ☐ N°2 ☐ N°3
soit __ exemplaires x 7,66 € = ____ €
☐ Histoire de la Micro-informatique 12,99 €

.....____ € soit au **TOTAL** = ____ €

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

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

Prénom : ____ Nom : ____

Adresse : ____

Code postal : ____ Ville : ____

E-mail : ____ @ ____

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

Offres pouvant s'arrêter à tout moment, sans préavis PROG 242

```
  return $this->responseFactory->createResponse()->withHeader('Content-Type', 'text/html; charset=utf-8');
}
```

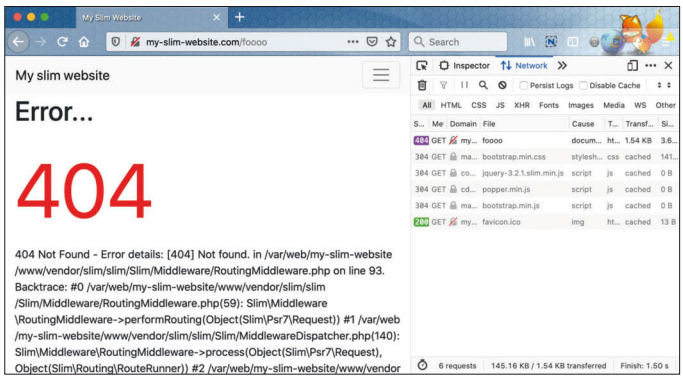
Je change la conf dans *settings.php*, en mettant *logErrors* et *logErrorDetails* à true, histoire d'avoir quelques logs et je vous épargne la trace complète.

```
[2020-04-17T16:56:28.214448+00:00] my-slim-website.ERROR: Error: [404] ... Method: GET, Path: /foooo [] {"uid":"ba01aa2"}
```

Enfin, j'écris le template *http_error.twig*, référencé dans le handler.

```
{% extends 'template.twig' %}

{% block content %}
 <style>
  .http-error .status-code {
   font-size: 7em;
   color: red;
  }
 </style>

 <div class="http-error">
  <h1>Error...</h1>
  <div class="status-code">{{ httpStatusCode }}</div>
  <div class="message">{{ genericErrorMessage }}</div>
 </div>
{% endblock %}
```

Les erreurs s'affichent maintenant avec le look du site. Il faudra encore affiner mais vous avez compris l'idée.



Database

Quelle application n'utilise pas de base de données ? J'ai besoin d'initialiser PDO. Je pense que vous commencez à connaître la chanson. J'ajoute les informations de connexion à ma base dans *settings.php*.

```
return function (ContainerBuilder $containerBuilder) {
 $settings = [
  ...

  'db' => [
   'user' => 'mylogin',
   'password' => 'mypwd',
   'host' => 'localhost',
   'name' => 'my_slim_website_db',
  ],
 ];
```

Je crée *databases.php* dans *config*, pour ajouter PDO dans le conteneur. La création d'une connexion vers MySQL est relativement coûteuse. C'est donc d'autant plus intéressant de le faire de manière lazy, et plus spécifiquement pour les cas où elle ne sera pas utilisée.

```
return function($app) {
 $container = $app->getContainer();

 $container->set(PDO::class, function($container): PDO {
  $settings = $container->get('settings');
  $databaseSettings = $settings['db'];
  $dbUser = $databaseSettings['user'];
  $dbPass = $databaseSettings['password'];
  $dbHost = $databaseSettings['host'];
  $dbName = $databaseSettings['name'];

  try {
   $pdo = new PDO("mysql:host=$dbHost;dbname=$dbName", $dbUser, $dbPass);
   $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   return $pdo;
  } catch (PDOException $e) {
   print($e);
   die();
  }
 });
};
```

Je l'injecte simplement dans *HelloAction*.

```
class HelloAction {
 private $responder;
 private $pdo;

 public function __construct(Responder $responder, PDO $pdo) {
  $this->responder = $responder;
  $this->pdo = $pdo;
 }
```

Je l'utilise directement pour exécuter une requête dans MySQL et j'envoie le résultat à Twig via le responder.

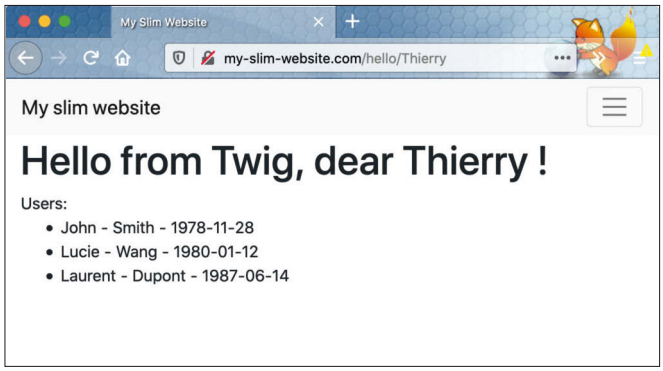
```
public function hello(Request $request, Response $response, $args): Response {

 $query = 'SELECT id, firstname, lastname, birthday FROM user';
 $stmt = $this->pdo->prepare($query);
 $stmt->execute();
 $users = $stmt->fetchAll(PDO::FETCH_OBJ);

 $name = $request->getAttribute('name');

 $params = [
  'name' => $name,
  'users' => $users
 ];
```

```
 return $this->responder->render($response, 'hello.twig', $params);
}
```

Il suffit de parcourir la liste dans le template pour l'afficher. J'aime beaucoup le fait que Twig gère automatiquement l'éventuelle nullité des variables, ce qui m'évite de devoir le faire et, surtout, ce qui évite de polluer le template avec du code technique.

```
<div>
 Users:
 <ul>
  {% for u in users %}
    <li>{{ u.firstname }} - {{ u.lastname }} - {{ u.birthday }}</li>
  {% endfor %}
 </ul>
</div>
```

Le résultat dans Firefox ne devrait pas vous surprendre.



Idéalement, dans une vraie app, je n'aurais pas injecté PDO directement dans le contrôleur ou dans l'action. J'aurais plutôt créé un helper Database, capable de prendre en charge mes requêtes. J'aurais sûrement écrit plusieurs DAO spécialisés par domaine, dans lesquels j'aurais injecté Database. Et finalement j'aurais injecté les DAOs nécessaires dans mes actions. J'aurais aussi pu mettre en œuvre un ORM.

Formulaire de login

Je m'intéresse maintenant au login. J'aime créer des objets pour gérer les valeurs des formulaires. Cela n'engage que moi. Pour le login, cela ressemblerait à ça.

```
class LoginForm {
 public $login = '';
 public $password = '';
```

Il suffit d'initialiser le form dans le contrôleur, et de le passer au template.

```
class LoginController {
 private $responder;
 private $pdo;

 public function __construct(Responder $responder, PDO $pdo) {
  $this->responder = $responder;
  $this->pdo = $pdo;
 }

 public function getLogin(Request $request, Response $response, $args): Response {
  $loginForm = new LoginForm();

  $params = [
   'form' => $loginForm,
  ];

  return $this->responder->render($response, 'login.twig', $params);
 }
```

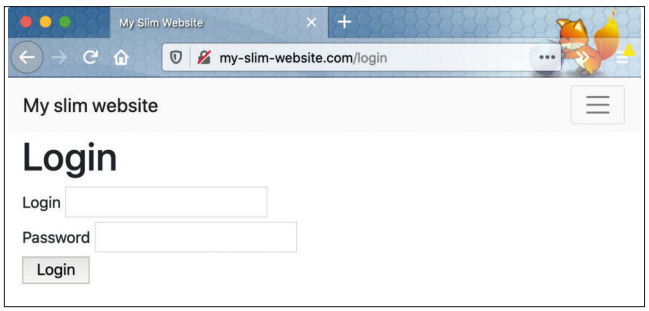
Pour le template, j'écris un formulaire très simple. Pour accéder aux propriétés d'un objet avec Twig, il suffit de remplacer la flèche par un point.

```
{% block content %}
 <h1>
  Login
 </h1>

 <form method="post" action="{{ url_for('login-submit') }}">
  <div>
   <label>Login</label>
   <input type="text" name="login" value="{{ form.login }}" />
  </div>
  <div>
   <label>Password</label>
   <input type="password" name="password" value="{{ form.password }}" />
  </div>
  <div>
   <input type="submit" value="Login" />
  </div>
 </form>
{% endblock %}
```

Et bien entendu, j'ajoute la route correspondante.

```
$app->get('/login', LoginController::class . ':getLogin')->setName('login');
$app->post('/login', LoginController::class . ':postLogin')->setName('login-submit');
$app->get('/logout', LoginAction::class)->setName('logout');
```

Et voici ce que ça donne dans Firefox



Validation du formulaire

Passons maintenant à la soumission du formulaire. Je délègue la récupération des champs et le remplissage du form au form lui-même, ce qui évite de polluer l'action.

```
class LoginForm {

 public $login = '';
```

```
public $password = '';

public static function createForm(array $postParams = []):LoginForm {
 $form = new LoginForm();
 $form->login = $postParams['loginn'] ?? '';
 $form->password = $postParams['password'] ?? '';
 return $form;
}
```

J'ajoute une dépendance à Validation pour valider les champs. J'ajoute aussi son pendant spécialisé pour les emails.

```
composer require respect/validation
composer require egulias/email-validator
```

En guise de validation, je renvoie simplement le nom des champs en erreur. J'aurais aussi pu renvoyer des statuts et des codes d'erreurs.

```
public function validate():array {
 $errors = [];

 if(!v::notEmpty()->validate($this->login)) {
  $errors[] = 'login';
 }

 if(!v::notEmpty()->validate($this->password)) {
  $errors[] = 'password';
 }

 return $errors;
}
```

Si le résultat de la validation est vide, c'est que tout va bien. Sinon, l'utilisateur devra corriger sa saisie.

```
public function postLogin(Request $request, Response $response, $args): Response {
 $loginForm = LoginForm::createForm((array)$request->getParsedBody());
 $errors = $loginForm->validate();

 if($errors) {
  ...
 }
```

On verra plus tard quoi faire en cas d'erreur.

Authentification

Pour l'authentification, je prends quelques raccourcis pour ne pas surcharger encore plus cet article. Idéalement, il faudrait *hasher* le mot de passe et vérifier la correspondance avec *password_verify*. Et ce serait mieux de séparer la recherche. Enfin, ce serait bien d'avoir un bean *User* au lieu d'utiliser le retour de la base directement.

```
public function postLogin(Request $request, Response $response, $args): Response {
 ...

 $query = 'SELECT * FROM user WHERE login = :login';
 $stmt = $this->pdo->prepare($query);
 $queryParams = [
  'login' => $loginForm->login,
 ];
 $stmt->execute($queryParams);
 $u = $stmt->fetch(PDO::FETCH_OBJ);

 if($u == null) {
  ...
 }

 if ($u->password !== $loginForm->password) {
  ...
 }

 $_SESSION['user'] = $u;
 return $this->responder->redirectToRoute($request, $response, 'root');
}
```

Note : conformément à *RGPD*, il ne faudra pas oublier de protéger les données, par exemple en chiffrant les informations personnelles.

À la fin, si tout s'est bien passé, j'enregistre l'utilisateur en session et je redirige vers la page d'accueil. Et c'est dans le responder que ça se passe.

```
class Responder {
 ...

 public function redirectToRoute(Request $request, Response $response, string $routeName, array $route
Params = [], int $httpStatusCode = 302): Response {

  $routeParser = RouteContext::fromRequest($request)->getRouteParser();
  $url = $routeParser->urlFor($routeName, $routeParams);
  return $response->withStatus($httpStatusCode)->withHeader('Location', $url);
 }
```

Flash et erreurs

En cas d'erreur, j'enregistre un message flash en session, avec le détail des erreurs et une copie du form pour le réafficher plus tard, puis je redirige sur la page du formulaire. Je dois donc écrire un middleware pour gérer les données flash.

```
class FlashMiddleware implements Middleware {

 public function process(Request $request, RequestHandler $handler): Response {
  $flash = $_SESSION['flash']; // isset
  unset($_SESSION['flash']);
  return $handler->handle($request->withAttribute('flash', $flash));;
 }
}
```

L'idée est de déplacer les données flash de la session vers la requête. Il y a plus élégant, mais ça fera l'affaire.
Le code de *postLogin* pour ajouter des données en flash est relativement simple.

```
public function postLogin(Request $request, Response $response, $args): Response {
 $loginForm = LoginForm::createForm($request);
 $errors = $loginForm->validate();

 if($errors) {
  $_SESSION['flash'] = [
```

```
    'errors' => $errors,
    'form' => $loginForm,
   ];
   return $this->responder->redirectToRoute($request, $response, 'login');
  }

  ...

  if($u == null || $u->password !== $loginForm->password) {
   $_SESSION['flash'] = [
    'errors' => ['Incorrect login and/or password'],
    'form' => $loginForm,
   ];
   return $this->responder->redirectToRoute($request, $response, 'login');
  }
  ...
```

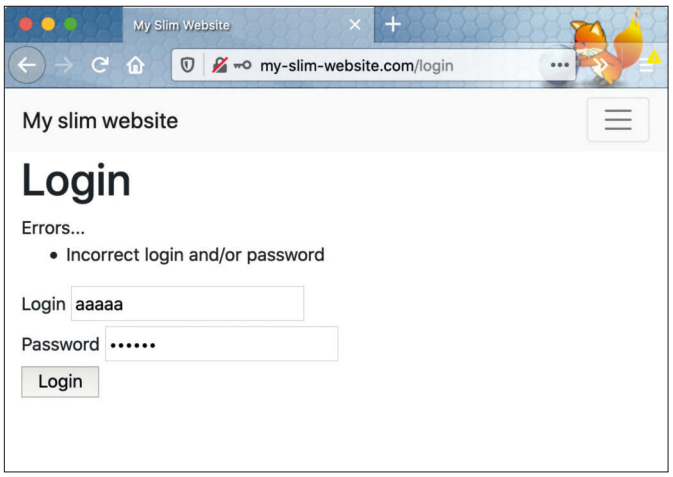
On voit qu'il y a largement matière à factoriser et à automatiser, ce que je vous recommande de faire dans une vraie app.
Au niveau de *getLogin*, qui permet d'afficher le formulaire, il suffit de récupérer le contenu que j'avais mis dans les attributs de la requête depuis le middleware.

```
public function getLogin(Request $request, Response $response, $args): Response {

 $loginForm = null;
 $errors = [];

 $flash = $request->getAttribute('flash');
 if($flash != null) {
  $loginForm = $flash['form'];
  $errors = $flash['errors'];
 }

 if($loginForm == null) {
  $loginForm = new LoginForm();
 }

 $params = [
  'form' => $loginForm,
  'errors' => $errors,
 ];

 return $this->responder->render($response, 'login.twig', $params);
```

Pour l'affichage, c'est un simple parcours dans le template.

```
{% if errors %}
 <div>
  Errors...
  <ul>
   {% for error in errors %}
    <li>{{ error }}</li>
   {% endfor %}
  </ul>
 </div>
{% endif %}
```

Voici ce que ça donne dans Firefox en cas d'erreur.



Middleware d'auth

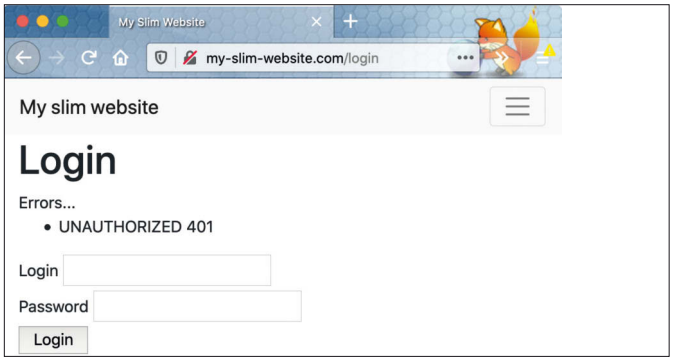
Je souhaite obliger les utilisateurs à se connecter pour accéder à certaines pages. Pour cela, il suffit d'ajouter un middleware dédié sur les routes concernées.

```
$app->get('/hello/{name}', HelloAction::class . ':hello')->setName('hello')
 ->add(AuthMiddleware::class);
```

Avec tout ce qu'on a fait avant, écrire le middleware est une formalité.

```
class AuthMiddleware implements Middleware {

 private $responder;

 public function __construct(Responder $responder) {
  $this->responder = $responder;
 }

 public function process(Request $request, RequestHandler $handler): Response {
  $user = $_SESSION['user']; // isset
  if($user == null) {
   $_SESSION['flash'] = [
    'errors' => ['UNAUTHORIZED 401'],
   ];

   $response = $this->responder->createResponse();
   return $this->responder->redirectToRoute($request, $response, 'login');
  }

  return $handler->handle($request);
 }
}
```

Voici ce que ça donne dans Firefox quand on essaie d'accéder à une page protégée.



Après le login, les utilisateurs sont redirigés vers la page d'accueil, mais ce serait mieux d'aller directement vers la page initialement demandée. Au niveau du middleware, il suffit de récupérer la route depuis le contexte et de la stocker temporairement en session.

```
$routeContext = RouteContext::fromRequest($request);
$route = $routeContext->getRoute();
$routeName = $route->getName();
$routeArgs = $route->getArguments();

$_SESSION['flash'] = [
 'errors' => ['UNAUTHORIZED 401'],
];
$_SESSION['route'] = [
 'name' => $routeName,
 'args' => $routeArgs,
];

$response = $this->responder->createResponse();
return $this->responder->redirectToRoute($request, $response, 'login');
```

Dans l'action, il ne faut pas oublier de nettoyer la session.

```
public function postLogin(Request $request, Response $response, $args): Response {
 ...

 $rName = 'root';
 $rArgs = [];

 If(isset($_SESSION['route'])) {
  $rName = $_SESSION['route']['name'];
  $rArgs = $_SESSION['route']['args'];
  unset($_SESSION['route']);
 }

 return $this->responder->redirectToRoute($request, $response, $rName, $rArgs);
}
```

CSRF

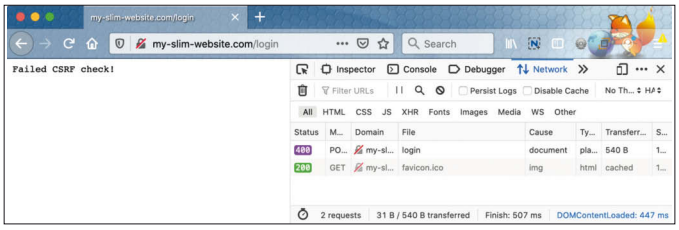
La protection CSRF (Cross-site request forgery), à ne pas confondre avec CORS, est un indispensable pour tout site Web. Slim 4 dispose d'un middleware dédié qu'on peut ajouter avec Composer.

```
composer require slim/csrf
```

Pour l'utiliser, il suffit de l'ajouter au conteneur et dans la liste des middlewares, comme on a déjà fait plusieurs fois.

```
$container->set(Guard::class, function () use ($responseFactory) {
 return new Guard($responseFactory);
});
$app->add(Guard::class);
```

On peut ajouter ce *middleware* pour toutes les routes, de manière globale, ou pour des routes spécifiques. Pour la plupart des cas, l'ajout global sera plus simple.



L'affichage est un peu brutal… Je préférerais indiquer et loguer mes propres erreurs. Pour cela, je peux créer un *handler* dédié.

```
$container->set(Guard::class, function () use ($responseFactory, $container) {
 $guard = new Guard($responseFactory);
 $guard->setFailureHandler($container->get(MyCsrfHandler::class));
 return $guard;
});
```

Quand le CSRF n'est pas validé (ie. tentative de fraude), je veux sortir directement en erreur, sans même passer par les autres middlewares. Dans l'app d'exemple, c'est relativement simple.

```
class MyCsrfHandler {

 private $responder;
 private $logger;

 public function __construct(Responder $responder, Logger $logger) {
  $this->responder = $responder;
  $this->logger = $logger;
 }

 public function __invoke(Request $request): Response {
  $this->logger->error('Error: CSRF bla bla');

  $httpStatusCode = 400;
  $genericErrorMessage = 'CSRF lorem ipsum';

  $response = $this->responder->createResponse();
  return $this->responder->render(
    $response,
    '/http_error.twig',
    ['httpStatusCode' => $httpStatusCode, 'genericErrorMessage' => $genericErrorMessage],
    $httpStatusCode);
```

L'erreur dans Firefox est plus belle. Ici, ce n'est pas tant le fait que ce soit joli qui m'intéresse mais surtout qu'on aie le contrôle.
On aimerait aussi que ça fonctionne dans le cas nominal, càd quand il n'y a pas de tentative de fraude. C'est relativement simple, il suffit d'injecter *csrf* dans l'action et d'envoyer les champs au template.

```
public function getLogin(Request $request, Response $response, $args): Response {
 ...

 $csrfNameKey = $this->csrf->getTokenNameKey();
 $csrfValueKey = $this->csrf->getTokenValueKey();
 $csrfName = $this->csrf->getTokenName();
 $csrfValue = $this->csrf->getTokenValue();
```

```
$csrfValues = [
 'keys' => [
  'name' => $csrfNameKey,
  'value' => $csrfValueKey
 ],
 'name' => $csrfName,
 'value' => $csrfValue,
];

$params = [
 'form' => $loginForm,
 'errors' => $errors,
 'csrf' => $csrfValues,
];

...
```

Dans le template, il suffit d'ajouter des champs cachés.

```
<input type="hidden" name="{{ csrf.keys.name }}" value="{{ csrf.name }}">
<input type="hidden" name="{{ csrf.keys.value }}" value="{{ csrf.value }}">
```

Le principe de CSRF est changer de valeur à chaque requête. Mais ce n'est pas toujours pratique : web services, client lourd, multi onglets, etc. Un code valable pour toute la durée est suffisant pour la plupart des cas. Cela peut se paramétrer directement lors de la construction.

```
$container->set(Guard::class, function () use ($responseFactory, $container) {
 $prefix = 'csrf';
 $storage = null;
 $failureHandler = $container->get(MyCsrfHandler::class);
 $persistentTokenMode = true;
 return new Guard($responseFactory, $prefix, $storage, $failureHandler, 200, 16, $persistentTokenMode);
});
```

Note :*on peut aussi envoyer directement les valeurs dans les variables globales de twig.*

Réorganisation

Arrivé à ce point, j'aimerais vous proposer une dernière petite réorganisation. Cela va me permettre de centraliser la configuration (hors routes et middlewares), m'éviter de mélanger containeur et middleware, et m'éviter de définir des fonctions avec une dépendance vers *$app* (avec *use*), comme dans *templates.php*.
Pour cela, je crée *container.php* dans *config*.
Code complet sur programmez.com & github

Le code d'*index.php* se simplifie d'autant.

```
$containerBuilder = new ContainerBuilder();
$containerBuilder->addDefinitions(__DIR__ . '/../config/container.php');
$container = $containerBuilder->build();

$app = $container->get(App::class);

// Middlewares
(require __DIR__ . '/../config/middlewares.php')($app);

// Routes
(require __DIR__ . '/../config/routes.php')($app);

// Run
$app->run();
```

Et pour vraiment bien faire, ça aurait été encore mieux de le mettre dans un fichier *boot.php* et de n'avoir plus qu'un unique appel dans *index.php*.

Conclusions

Dans ce tuto, on est allé beaucoup plus loin que la simple présentation de Slim 4, et on a également laissé quelques trous à combler dans une vraie application. Il y a notamment quelques petits soucis de sécurité. On pourrait ajouter des helpers et des DAO. Et quelques tests seraient les bienvenus.
Cela nous a toutefois permis de constater que l'utilisation d'un routeur, associé à un ensemble de middleware est très puissant mais bien moins complexe que ça ne le semble au premier abord. L'intégration de Slim 4 est un peu lourde au démarrage du projet mais redevient légère dès que les éléments principaux sont gérés.
Après y avoir goûté, c'est difficile de s'en passer, y compris sur des petits projets. Reste à placer le curseur, en fonction des besoins et de l'équipe, pour s'orienter éventuellement sur un framework plus complet.
Merci à *Daniel Opitz* pour la relecture de cet article.
À titre indicatif, voici à quoi ressemble le fichier *composer.json* à la fin du tuto.

```
{
 "name": "My Slim Website",
 "description": "Very simple example of a website using Slim 4.",
 "require": {
  "slim/slim": "4.*",
  "slim/psr7": "^1.0",
  "slim/twig-view": "^3.1",
  "slim/csrf": "^1.0",
  "php-di/php-di": "^6.1",
  "monolog/monolog": "^2.0",
  "respect/validation": "^2.0",
  "egulias/email-validator": "^2.1"
 },
 "autoload": {
  "psr-4": {
   "App\\": "src/"
  }
 }
}
```

Quelques liens

Slim : http://www.slimframework.com/
GitHub : https://github.com/slimphp/Slim
Documentation : http://www.slimframework.com/docs/v4/
PSR : https://www.php-fig.org/psr/
Twig : https://twig.symfony.com/
Twig-view: https://github.com/slimphp/Twig-View

SQLI DIGITAL EXPERIENCE

Gabriel Euzet
Expert Technique SQLI
Diplômé d'une école d'ingénieur en CFAO (Conception et Fabrication Assistée par Ordi), Robotique et Automatique, Gabriel intègre très vite une SS2I parisienne comme développeur C. Découvrant ainsi un goût pour le développement, il s'initie au PHP. Il ne tarde pas à rejoindre la ville rose pour intégrer Aston (Groupe SQLI) et s'essayer au JAVA. Vrai autodidacte dans l'âme, Gabriel aime bricoler l'électronique et l'IoT quand il ne ride pas sur sa moto.

GraphQL : promesse magique ou nuage de fumée ?

Courant 2019, nous avons assisté à un buzz important autour de la technologie GaphQL dans le milieu du développement. On parlait beaucoup de GraphQL comme d'une solution révolutionnaire. Il y a eu une présentation de son utilisation par un expert en développement « front ». Comme tout buzz c'était forcément trop beau pour être vrai. L'engouement était exagéré. Il fallait apaiser l'atmosphère et remettre les choses à leur place. Il manquait aussi le côté « implémentation » la partie « back ».

J'ai donc voulu étudier la technologie pour vérifier cette présomption avec plusieurs objectifs en ligne de mire :

- Présenter la vision « back » ;
- Avoir la capacité de détecter les cas d'usages propices ;
- Rassurer quant au choix de cette technologie ;
- Aider à sécuriser les projets.

GraphQL : l'alternative à l'API REST

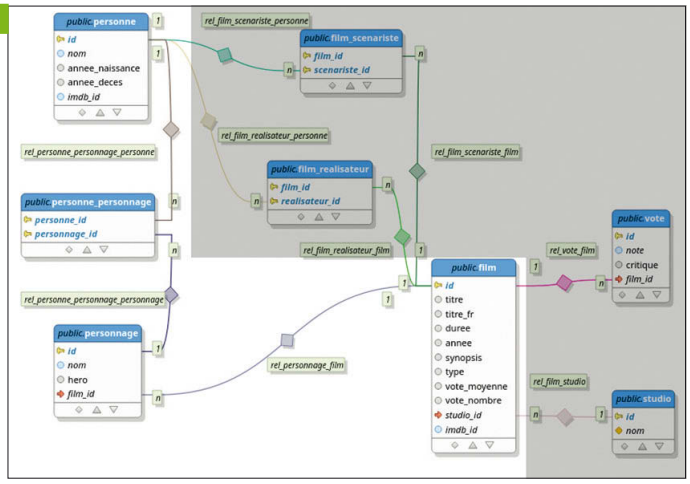
L'API REST est considérée comme trop verbeuse dans certains cas. GraphQL est un langage de requêtes créé par Facebook en 2012 pour minimiser la quantité d'appels à ces API. C'est une technologie dont le but est d'améliorer les performances de la communication entre le front et le back. Cela permet d'alléger les infrastructures nécessaires, et également d'en diminuer le coût. Comme présenté par la Fondation GraphQL, https://graphql.org/learn/thinking-in-graphs/, la puissance du graphe réside dans sa ressemblance à notre façon de penser, de concevoir notre environnement. Comment aller jusqu'au bout de la logique orientée graphe ? Pourquoi ne pas combiner GraphQL avec une base de données graphe ?

Un POC pour mettre à l'épreuve

Le matériel

Pour les facilités qu'elle apporte, j'ai utilisé une machine virtuelle VirtualBox. Elle est constituée de 6 cœurs à 2,6GHz et 16Go de mémoire. L'OS installé est Linux Mint 19.3, et les SGBD choisis sont PosgreSQL 11.7 (première partie du POC avec une base relationnelle) et Neo4j 3.5.14 (seconde partie avec une base graphe). Ces choix sont subjectifs. PostgreSQL parce que j'ai une attirance et un plaisir à travailler avec, et quant à Neo4j, il s'agit du leader du marché en matière de base de données graphes, il est à l'origine du langage de requêtage Cypher dérivé en OpenCypher et utilisé par d'autres acteurs. Un premier projet avec ce produit m'a envoûté ! Alors, continuons… L'IDE est Spring Tool Suite 4. Les requêtes aux API REST et GraphQL sont effectuées avec Postman.

1



La base de données

Pour cette étude le cas étudié est la base de films d'Imdb, car volumineuse et donc propice aux tests de performance. Elle contient 2 millions de films/séries TV, 2 millions de personnages et 9 millions de personnes associées et est récupérable à l'adresse https://www.imdb.com/interfaces/.

Pour intégrer ces données voici le modèle que j'ai conçu : 1

Des index sont ajoutés pour les propriétés impliquées dans la recherche (les titres de films). Je me suis focalisé sur les relations entre les films et les acteurs : zone grise 1

Principes des API GraphQL et REST

Prenons l'exemple d'action : lister les acteurs d'un film donné (dont nous connaissons l'identifiant).

Actions à mener avec une solution REST :

- Récupération du film via un GET à l'URL /api/films/{id du film} ;
- Récupération des personnages via un GET à l'URL /api/films/{id du film}/personnages ;
- Récupération des acteurs via autant d'appels GET que de personnages à l'URL /api/personnages/{id du personnage}/acteurs.

Actions à mener avec une solution GraphQL :

Récupération de toute la grappe avec un seul appel contenant la requête :

```
{
 Film(id:{id du film}) {
  titre
  titreFr
  annee
  personnages {
   nom
   acteurs {
    nom
   }
  }
 }
}
```

Effectivement, d'un point de vue « réseau » GraphQL est vainqueur, et ce, sur 2 points :

- En nombre de requêtes : plus il y a de personnages plus y a d'appels client-serveur avec REST

- En volumétrie : GraphQL permet de lister les données souhaitées là où REST enverra la totalité des propriétés de chaque objet (même celles que nous ne voulons pas afficher)

Préparation du POC : architecture du back

JHipster a été utilisé pour créer une base applicative complète. Voici le visuel de JDL Studio où nous construisons le modèle de données : 2
JHipster ne possède pas de plugin pour générer le code GraphQL. Nous allons donc procéder manuellement. Commençons par embarquer les librairies nécessaires. Il faut ajouter au pom.xml 2 propriétés et quelques dépendances :

```
<graphql.version>5.10.0</graphql.version>
<kotlin.version>1.3.10</kotlin.version>
```

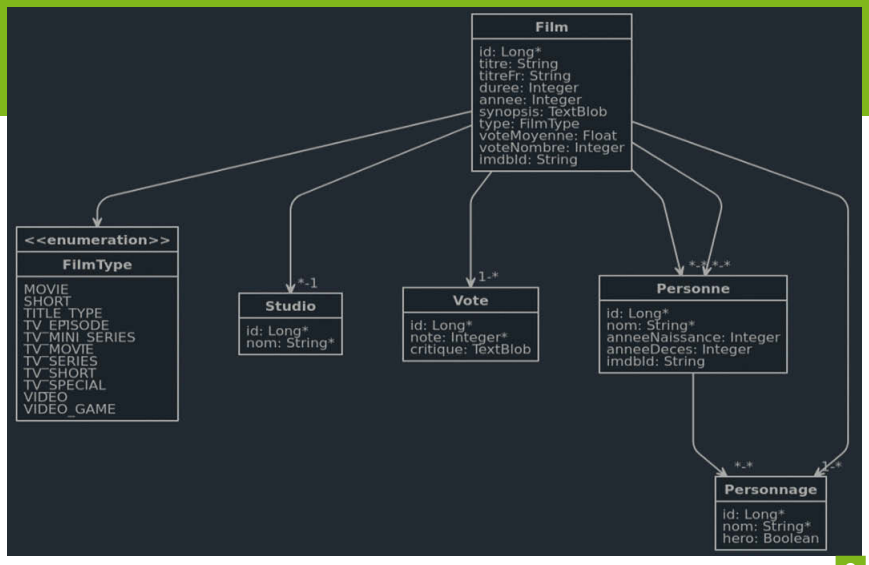
```
<dependency>
  <groupId>com.graphql-java-kickstart</groupId>
  <artifactId>graphql-spring-boot-starter</artifactId>
  <version>${graphql.version}</version>
</dependency>
<!-- to embed GraphiQL tool -->
<dependency>
  <groupId>com.graphql-java-kickstart</groupId>
  <artifactId>graphiql-spring-boot-starter</artifactId>
  <version>${graphql.version}</version>
  <scope>runtime</scope>
</dependency>
<!-- to embed Altair tool -->
<dependency>
  <groupId>com.graphql-java-kickstart</groupId>
  <artifactId>altair-spring-boot-starter</artifactId>
  <version>${graphql.version}</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>com.graphql-java-kickstart</groupId>
  <artifactId>playground-spring-boot-starter</artifactId>
  <version>${graphql.version}</version>
  <scope>runtime</scope>
</dependency>
```

Un peu de configuration dans Spring Boot, ajouter à application.yml :

```
graphql:
 servlet:
  mapping: /graphql
  enabled: true
```

Ceci active la servlet (pour le endpoint GraphQL) et limite la recherche des fichiers de configurations au répertoire /src/main/resources/graphql. Par défaut tous les fichiers *.graphqls seront interprétés.
Décrivons le modèle à travers le fichier entites.graphqls dont voici l'extrait qui nous intéressera :

```
type Film {
  id: ID!
  titre: String
  titreFr: String
  duree: Int
  annee: Int
```



Modèle de données utilisé pour piloter JHipster (format JDL) 2

```
  synopsis: String
  type: FilmType
  voteMoyenne: Float
  voteNombre: Int
  imdbId: String
  votes: [Vote]
  personnages: [Personnage]
  studio: Studio
  realisateurs: [Personne]
  scenaristes: [Personne]
}

type Personnage {
  id: ID!
  nom: String!
  hero: Boolean
  film: Film!
  acteurs: [Personne!]!
}

type Personne {
  id: ID!
  nom: String!
  anneeNaissance: Int
  anneeDeces: Int
  imdbId: String
  personnages: [Personnage]
  filmRealises: [Film]
  filmScenarises: [Film]
}
```

La première requête

Commençons modestement par récupérer les données d'un film. Mais avant d'y arriver il faut expliquer à notre moteur quelles seront les « root queries » qu'il devra servir. Nous créons ainsi le fichier queries.graphqls :

```
type Query {

  filmById(id: ID!): Film

  filmsByTitreFr(titreFr: String!): [Film]

}
```

Ces requêtes sont le point de départ, les racines, des grappes d'objets que nous allons demander au serveur.
A cette description doit correspondre l'implémentation. Il s'agit du « query resolver », il doit implémenter l'interface GraphQLQueryResolver :

Code complet sur programmez.com & github

Je vous passe les détails de la DAO FilmRepository… Mais nous retrouvons bien les méthodes déclarées précédemment.
Lançons-nous, envoyons la requête :

```
{
 filmById(id: 1518508) {
  titre
 }
}
```

Et voilà :

```
{
 "data": {
  "filmById": {
   "titre": "Angel Has Fallen"
  }
 }
}
```

GraphQL permet de choisir finement les données à récupérer :

Requête	Réponse
{ filmById(id: 1518508) { titre titreFr annee } }	{ "data": { "filmById": { "titre": "Angel Has Fallen" "titreFr": "La chute du président" "annee": 2019 } } }

Un premier niveau de récursion

Maintenant, si nous essayons de récupérer les personnages comme ceci…

```
{
 filmById(id: 1518508) {
  titre
  titreFr
  annee
  personnages {
   nom
  }
 }
}
```

Nous obtenons cette trace :

```
[...]
--DEBUG graphql.execution.ExecutionStrategy : '3752384b-894b-4130-9a19-807ca23970d0' fetching field '/filmById/annee' using data fetcher 'com.coxautodev.graphql.tools.MethodFieldResolverDataFetcher'...
--DEBUG graphql.execution.ExecutionStrategy : '3752384b-894b-4130-9a19-807ca23970d0' field '/filmById/annee' fetch returned 'java.lang.Integer'
--DEBUG graphql.execution.ExecutionStrategy : '3752384b-894b-4130-9a19-807ca23970d0' completing field '/filmById/annee'...
--DEBUG graphql.execution.ExecutionStrategy : '3752384b-894b-4130-9a19-807ca23970d0' fetching field '/filmById/personnages' using data fetcher 'com.coxautodev.graphql.tools.MethodFieldResolverDataFetcher'...
--DEBUG graphql.execution.ExecutionStrategy : '3752384b-894b-4130-9a19-807ca23970d0' field '/filmById/personnages' fetch returned 'org.hibernate.collection.internal.PersistentSet'
--DEBUG graphql.execution.ExecutionStrategy : '3752384b-894b-4130-9a19-807ca23970d0' completing field '/filmById/personnages'...
--ERROR graphql.GraphQL : Execution '3752384b-894b-4130-9a19-807ca23970d0' threw exception when executing : query :
[...]
Caused by: org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: com.sqli.tlse.ct.graphql.domain.Film.personnages, could not initialize proxy - no Session
```

La récupération de l'année du film se passe bien, mais pour les personnages il y a un couac !
Le moteur GraphQL tente l'utilisation du getter associé à la propriété, c'est bien une collection Hibernate qui est renvoyée. Cependant la session Hibernate est fermée puisque nous sommes en dehors du DAO. Le parcours de la liste provoque donc une exception.

Remarque : remonter la session à un niveau plus haut n'est pas la solution ! D'une part, l'architecture de GraphQL n'a pas été pensée pour, cela obligerait à aller modifier des classes du moteur. D'autre part cela aurait généré des requêtes en mode « lazy » et amené au même résultat en volume de requêtes que les appels successifs de resolvers comme nous allons le découvrir.

Le principe de récursion de GraphQL se base sur la notion de resolver. Dans notre cas le resolver n'a pas été trouvé alors le moteur a tenté d'utiliser l'accesseur. Nous sommes dans un film et désirons connaître les personnages de ce film. Nous devons donc créer un resolver pour l'entité "Film" contenant une méthode dont le nom est le même que la propriété désirée et qui prend en paramètre le film :

```
@Component
public class FilmResolver implements GraphQLResolver<Film> {
    @Autowired
    private PersonnageRepository personnageRepository;

    public Set<Personnage> personnages(Film film) {
        return personnageRepository.findByFilmId(film.getId());
    }
}
```

Si nous rejouons la requête, les journaux sont plus sympathiques :

```
--DEBUG graphql.execution.ExecutionStrategy : '29388273-8606-41d2-adba-79298edc3722' fetching field '/filmById/personnages' using data fetcher 'com.coxautodev.graphql.tools.MethodFieldResolverDataFetcher'...
-Hibernate: select personnage0_.id as id1_9_, personnage0_.film_id as film_id4_9_, personnage0_.hero as hero2_9_, personnage0_.nom as nom3_9_ from personnage personnage0_ left outer join film film1_ on personnage0_.film_id=film1_.id where film1_.id=?
Hibernate: select film0_.id as id1_0_0_, film0_.annee as annee2_0_0_, film0_.duree as duree3_0_0_, film0_.imdb_id as imdb_id4_0_0_, film0_.studio_id as studio_11_0_0_, film0_.synopsis as synopsis5_0_0_, film0_.titre as titre6_0_0_, film0_.titre_fr as titre_fr7_0_0_, film0_.type as type8_0_0_, film0_.vote_moyenne as vote_moy9_0_0_, film0_.vote_nombre as vote_no10_0_0_, studio1_.id as id1_12_1_, studio1_.nom as nom2_12_1_ from film film0_ left outer join studio studio1_ on film0_.studio_id=
```

```
studio1_.id where film0_.id=?
-DEBUG graphql.execution.ExecutionStrategy : '29388273-8606-41d2-adba-79298edc3722' field '/filmById/personnages' fetch returned 'java.util.LinkedHashSet'
--DEBUG graphql.execution.ExecutionStrategy : '29388273-8606-41d2-adba-79298edc3722' completing field '/filmById/personnages'...
--DEBUG graphql.execution.ExecutionStrategy : '29388273-8606-41d2-adba-79298edc3722' fetching field '/filmById/personnages[0]/nom' using data fetcher 'com.coxautodev.graphql.tools.MethodFieldResolverDataFetcher'...
--DEBUG graphql.execution.ExecutionStrategy : '29388273-8606-41d2-adba-79298edc3722' field '/filmById/personnages[0]/nom' fetch returned 'java.lang.String'
--DEBUG graphql.execution.ExecutionStrategy : '29388273-8606-41d2-adba-79298edc3722' completing field '/filmById/personnages[0]/nom'...
--DEBUG graphql.execution.ExecutionStrategy : '29388273-8606-41d2-adba-79298edc3722' fetching field '/filmById/personnages[1]/nom' using data fetcher 'com.coxautodev.graphql.tools.MethodFieldResolverDataFetcher'...
--DEBUG graphql.execution.ExecutionStrategy : '29388273-8606-41d2-adba-79298edc3722' field '/filmById/personnages[1]/nom' fetch returned 'java.lang.String'
--DEBUG graphql.execution.ExecutionStrategy : '29388273-8606-41d2-adba-79298edc3722' completing field '/filmById/personnages[1]/nom'...
--DEBUG graphql.GraphQL : Execution '29388273-8606-41d2-adba-79298edc3722' completed with zero errors
```

Deux requêtes Hibernate apparaissent. La première correspond aux personnages (celle que nous attendions) et la seconde au lien "ManyToOne" défini dans l'entité Personnage vers Film.
La réponse correspond bien à notre demande :

```
{
 "data": {
  "filmById": {
   "titre": "Angel Has Fallen",
   "titreFr": "La chute du président",
   "annee": 2019,
   "personnages": [
    {
     "nom": "Allan Trumbull"
    },
    {
     "nom": "Mike Banning"
    }
   ]
  }
 }
}
```

Dans le vif du sujet

Performances

Il est temps de pousser la machinerie pour évaluer ses limites. Pour cela nous allons « creuser » la base de données sur 4 niveaux : films → personnages → acteurs → personnages → films. Le cas est peu vraisemblable, mais ce n'est pas l'objectif.
Voici donc la requête :

```
{
 filmsByTitreFr(titreFr: "Alice au pays des merveilles") {
  id
  titre
  annee
  personnages {
   nom
   acteurs {
    nom
    personnages {
     nom
     film {
      titre
      titreFr
      annee
     }
    }
   }
  }
 }
}
```

La réponse « prettyfied » dépasse les 10 000 lignes, nous passerons donc de la copier… En revanche, il est intéressant d'aller voir du côté des requêtes dans les journaux. Il y a 1459 requêtes au total réparties comme suit :

- 1 *select* sur les films avec clause sur 'titre_fr'. Il s'agit de la *root query* ;
- 11 *select* sur les personnages avec clause sur l'identifiant du film. Il s'agit de la récupération du premier niveau ;
- 34 *select* sur les personnes avec clause sur l'identifiant du personnage. Il s'agit de la récupération du deuxième niveau ;
- 34 *select* sur les personnages avec clause sur l'identifiant de la personne. Il s'agit de la récupération du troisième niveau ;
- 1 379 *select* sur les films avec clause sur l'identifiant du film. Il s'agit de la récupération du quatrième niveau avec quelques films liés aux premiers personnages (soient 11) comme vu précédemment.

Le tout prend en moyenne **4,3 secondes** sur un échantillon de 10 appels. Souvenez-vous du bénéfice qu'apporte GraphQL par rapport à REST. Les multiples appels du dernier ont été réduits en un seul appel GraphQL. Cependant, ils ont été répercutés côté serveur au niveau des resolvers et donc des requêtes vers la base de données.
Afin d'avoir une référence en matière de performance codons, un endpoint REST spécifique qui retourne les mêmes données, mais dont l'implémentation back consiste en une seule requête vers de la base de données. Voici la requête JPQL que nous exécutons :

```
select distinct f1
from Film f1
left join fetch f1.personnages p1
left join fetch p1.acteurs a
left join fetch a.personnages p2
left join fetch p2.film f2
where
f1.titreFr = :titreFr
```

Avec une application « classique », il est difficile de faire plus simple et plus efficace. Les temps d'exécution s'en trouvent bien sûr améliorés. Pour le même échantillon de 10 appels, nous atteignons maintenant une moyenne de **1,9 secondes**.

Première conclusion

Nous avons comparé les performances de GraphQL à un développement spécifique. Une implémentation de type RESTful a été volontairement évincée puisque nous pouvons estimer intuitive-

ment qu'elle ne pouvait pas faire mieux (le nombre d'appels REST aurait été équivalent au nombre d'appels de resolvers GraphQL et donc au même nombre de requêtes en base de données, on devine que la somme des échanges réseau à ajouter ne feront qu'augmenter le temps de réponse global). Voici une matrice comparative des 3 technologies étudiées :

	Avantages	Inconvénients
GraphQL	• Souplesse pour le développement du client • Minimum d'appels réseau • Volume de données optimal	• Multiplication des requêtes base de données
RESTful	• Souplesse pour le développement du client	• Multiplication des requêtes base de données
REST spécifique	• Minimum d'appels réseau • Performance back	• Rigidité de la solution • Aucune réutilisabilité

Pour résumer, le bilan est relativement positif. La solution GraphQL offre plus de souplesse dans l'évolutivité des requêtes sans impacter le back tout en minimisant l'utilisation du réseau. En revanche l'utilisation de la base de données n'est pas amoindrie. La situation du back n'est pas désespérée, plusieurs techniques peuvent nous aider…

Optimisations

La situation du back n'est pas désespérée. Plusieurs techniques peuvent nous aider.

Ajout de cache

Une fois le cache configuré dans notre POC les performances s'en trouvent grandement améliorées. La moyenne du temps d'exécution passe à **90 ms** (échantillon de 10 appels). Le test a été fait en mode extrême. L'ajout de cache couvre toutes les entités au point qu'après une première exécution l'ensemble des données s'y retrouve enregistré. La première requête prend entre 7 et 8 secondes. Les requêtes suivantes trouvent toutes leur réponse dans le cache. Dans la vraie vie, cela ne se passe pas comme ça, mais on peut tout de même estimer que cette solution sera très utile.

Utilisation d'un « data loader »

Le principe d'un data loader est de capturer des requêtes sur un court laps de temps, de les réduire en une seule, de récupérer les données pour cette unique requête et de les travailler pour renvoyer les réponses adéquates aux requêtes capturées. Là aussi l'idée est de réduire les impacts sur la base de données.

Paginer

Ce principe est universel. Lorsque beaucoup de données doivent être récupérées, il est de bonne pratique de les saucissonner.

Activer la compression

Il s'agit là de configurer le serveur et le client pour échanger les données au format gzip. Nous diminuons le volume de données qui transite dans le réseau. Les échanges base de données ne sont pas impactés.

Conclusion

GraphQL est une technologie à prendre en considération. En effet, elle amène une souplesse d'utilisation et réduit les flux réseau (quantité et volume), tout ceci pour faciliter le développement et la maintenance d'une application « front ». La souplesse se trouve aussi côté « back ». Comme on l'a vu, pour chaque entité à récupérer, le moteur utilise un resolver. Rien n'empêche au contraire d'avoir des resolvers de types différents. Dans le cadre du POC, pourquoi ne pas aller chercher un complément d'information sur les acteurs auprès de l'API de Wikipédia ? Il peut tout aussi bien s'agir d'un annuaire LDAP, d'une API REST, de web-services SOAP, de fichiers plats… Lors du choix de cette technologie, il faudra tout de même prêter attention à la taille des données retournées, et ce sur 3 dimensions : profondeur de la grappe d'objets, taille d'un objet, nombre d'éléments. Des solutions existent pour gérer ces cas.
Petit bémol, le point de la sécurité n'a pas été abordé. Il est assez vaste et nécessite une étude à part entière. Vous avez pu voir les impacts côté back d'une unique et simple requête. Il faudra sûrement implémenter des solutions anti-DoS. Mais ce n'est qu'un point de vigilance parmi une multitude…

Base de données graphe

GraphQL repose sur la notion de graphe. En effet on peut voir l'enchaînement des resolvers comme le parcours d'un graphe, d'où la question : que se passe-t-il avec un backend supporté par une base graphe ?
Je me suis alors tourné vers Neo4j que nous connaissons déjà chez SQLI. Quelle chance, il y a un plugin GraphQL ! Et, du fait que nous sommes en présence d'une application et d'un langage « graphe », il serait assez logique qu'à 1 re-quête GraphQL le plugin fasse correspondre 1 requête Cipher (langage de requêtage de Neo4j). Nous éliminerions alors le gros inconvénient rencontré avec la base relationnelle. Précision : du fait du plugin, nous n'avons pas besoin de backend Java pour aiguiller les requêtes.

Présentation rapide

Citons Wikipedia :

> *" Par définition, une base de données orientée graphe correspond à un système de stockage capable de fournir une adjacence entre éléments voisins : chaque voisin d'une entité est accessible grâce à un pointeur physique. C'est une base de données orientée objet adaptée à l'exploitation des structures de données de type graphe ou dérivée, comme des arbres."*

Une autre manière « simpliste » de le dire est : une base graphe privilégie les relations.

Mise en place du plugin GraphQL

De la même manière que pour l'implémentation Java, il faut fournir le schéma représentant les relations entre les entités. Pour notre exemple : **Code complet sur programmez.com & github**
Profitons qu'il existe une commande pour vérifier : 3
Nous constatons que les relations entre nœuds sont correctement définies.

Illustration

Une fois la base des films chargée, partons à la recherche des films joués par les acteurs du film d'identifiant 1518508, « Angel Has Fallen » via la requête cipher :

```
MATCH (f:Film{id:1518508})<-[:FIGURE_DANS]-(pa:Personnage)<-[:INTERPRETE]-
(pe:Personne)-[:INTERPRETE]->(pa2:Personnage)-[:FIGURE_DANS]->(f2:Film)
RETURN f,pa,pe,pa2,f2
```

Le résultat est présenté sous forme de graphe : 4
Analysons le résultat :

- Le point de recherche est cerclé de bleu ;
- De là partent les deux personnages, cerclés de vert ;
- À ces personnages sont liés les acteurs, les points rouges ;
- Pour chaque acteur nous voyons les personnages qu'ils ont joués dans leur carrière, les points verts ;
- Et, finalement, pour chaque personnage est présenté le film dans lequel il figure, les points beiges.

Notons que les trois films reliés aux deux « galaxies » entourent chacun de nos acteurs. Cela signifie qu'ils ont participé tous les deux à ces trois films. Effectivement, le film que nous avons recherché fait partie d'une trilogie : « Olympus/London/Angel Has Fallen », respectivement sortis en 2013, 2016 et 2019. Cet exemple simple laisse entrevoir la puissance de cette architecture.

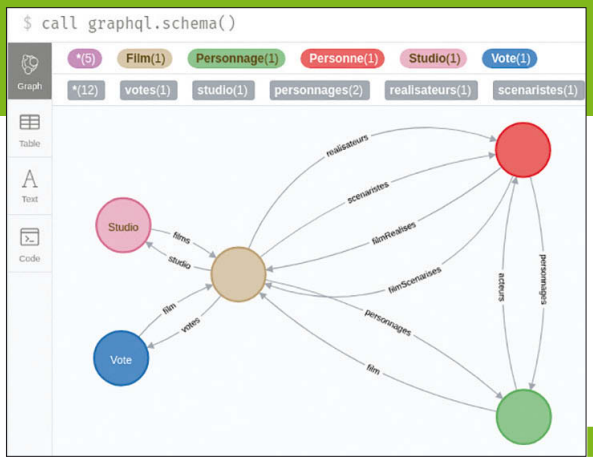
Performances

Plus besoin de s'encombrer des étapes d'explications, lançons le test de performance sur la « grosse » requête. La moyenne du temps d'exécution est de **1,6 seconde**. Tout ça sans avoir personnalisé ou optimisé quoi que ce soit.
Un petit tour dans les journaux d'exécution nous révèle qu'il n'y a eu qu'une seule requête exécutée par le moteur :

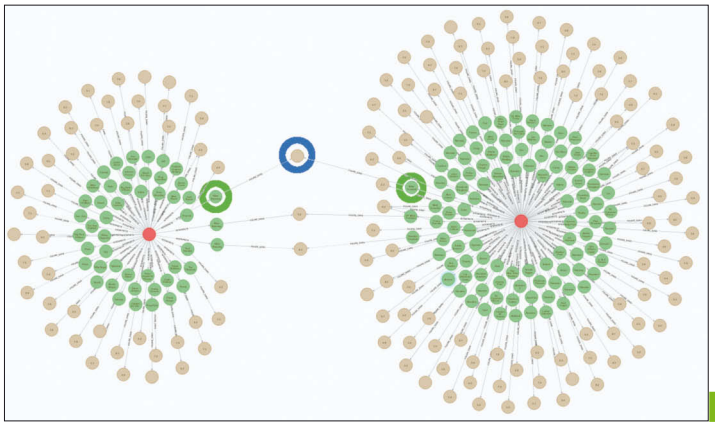
```
MATCH (`film`:`Film`)
WHERE `film`.`titreFr` = "Alice au pays des merveilles"
RETURN graphql.labels(`film`) AS `_labels`,
`film`.`id` AS `id`,
`film`.`titre` AS `titre`,
`film`.`annee` AS `annee`,
[ (`film`)<-[:`FIGURE_DANS`]-(`film_personnages`:`Personnage`)
   | `film_personnages` {`_labels` : graphql.labels(`film_personnages`), .`nom`, `acteurs` : [
   (`film_personnages`)<-[:`INTERPRETE`]-(`film_personnages_acteurs`:`Personne`)
   | `film_personnages_acteurs` {`_labels` : graphql.labels(`film_personnages_acteurs`),
.`nom`, `personnages` : [
      (`film_personnages_acteurs`)-[:`INTERPRETE`]->(`film_personnages_acteurs_
personnages`:`Personnage`)
      | `film_personnages_acteurs_personnages` {`_labels` : graphql.labels(`film_
personnages_acteurs_personnages`), .`nom`, `film` : head([
         (`film_personnages_acteurs_personnages`)-[:`FIGURE_DANS`]->(`film_
personnages_acteurs_personnages_film`:`Film`)
         | `film_personnages_acteurs_personnages_film` {`_labels` : graphql.labels
(`film_personnages_acteurs_personnages_film`), .`titre`, .`titreFr`, .`annee`}
         ])}
      ]}
   ]}
] AS `personnages`
```

Un peu complexe au premier abord en effet. Il s'agit de code généré et comme je ne connais pas la mécanique qui l'a pondu il restera brut de fonderie. En revanche, si nous avions écrit la requête Cypher nous-mêmes nous aurions écrit quelque chose de plus lisible :

```
MATCH (f:Film {titreFr:"Alice au pays des merveilles"})<-[:FIGURE_DANS]-(pa:Personnage)
<-[:INTERPRETE]-(pe:Personne)-[:INTERPRETE]->(pa2:Personnage)-[:FIGURE_DANS]->
(f2:Film)
RETURN f,pa,pe,pa2,f2
```



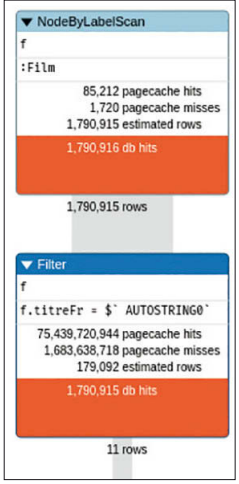
3


4

Cette requête retourne toutes les informations pour chaque entité. J'ai essayé de ne retourner que les attributs spécifiés jusque-là, mais les performances n'ont pas changé. Nous tournons en moyenne à **1,3 seconde**, toujours sur un échantillon de 10 appels.

Amélioration

Comme expliqué plus haut, ces tests ont été réalisés sans optimisation de la base. Cependant un petit « profiling » de la requête pointe du doigt un point noir : 5



5

Le nombre de hits (« db hits ») ainsi que le nombre d'enregistrements transmis à l'étape suivante (« rows ») sont les indicateurs importants. Le moteur a parcouru toute la base de données afin de trouver les films vérifiant le titre donné.
Créons l'index adéquat :

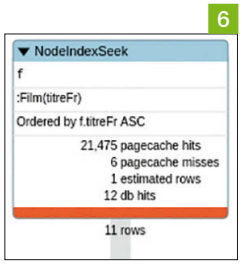
```
CREATE INDEX ON :Film(titreFr);
```

La nouvelle analyse est encourageante : 6
On retrouve là une problématique rencontrée régulièrement en base de données relationnelle. Aucune surprise, les statistiques confirment la justesse de la solution, les temps de réponse descendent à 89 ms pour la requête GraphQL et 51 ms pour la requête Cypher.

Conclusion finale

Ici encore une requête spécifique est plus rapide qu'une requête générée. Il ne faut pas oublier que notre exemple est un peu exagéré. GraphQL associé à une base graphe semble être un couple bien assorti. Nous devrions donc lui don-ner sa chance ! Une solution full stack existe, il s'agit de GRANDstack (https://grandstack.io/) : la prochaine étape ?

Remerciements : *à mes collègues relecteurs qui ont suscité quelques éclaircissements et à Nicolas Rouyer de chez Neo4j pour son aide et ses précisions.*



6

Microsoft® Most Valuable Professional
equisoft BUSINESS-DRIVEN INNOVATION

Anthony Giretti
MVP, MCSD
Developer, Blogger, Speaker
anthony.giretti@gmail.com
http://anthonygiretti.com

gRPC-web avec ASP.NET Core 3.1 et Angular 9 : comment ça marche ?

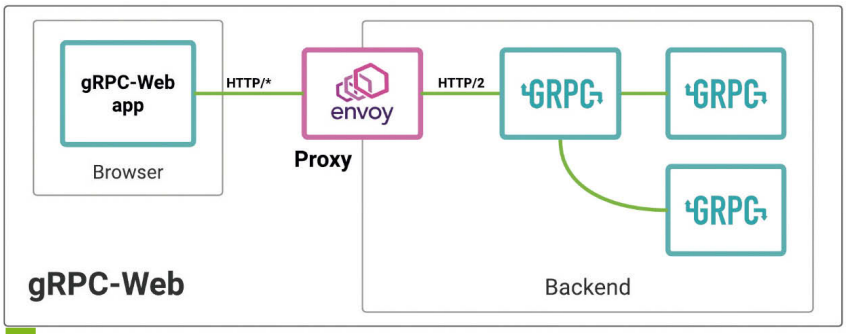
gRPC est un framework RPC (Remote procedure call) et non pas un framework Microsoft. Il utilise le protocole HTTP2 pour le transport des données et Protocol Buffers comme format de sérialisation également développé par Google donc un descripteur d'interface tout comme le WSDL pour SOAP si on veut faire une analogie, ce qui le rend ultra performant par rapport aux API REST. Cependant les avantages n'étaient en grande partie disponibles que pour les développeurs d'applications mobiles et de backend, tandis que les développeurs frontaux ont dû continuer à s'appuyer sur les interfaces JSON REST comme principal moyen d'échange d'informations. Avec la sortie de gRPC-Web, gRPC est en passe de devenir un ajout précieux dans la boîte à outils des développeurs frontaux.

Dans cet article, je vais explorer son fonctionnement dans **ASP.NET Core 3.1** avec un front end de type Angular et son déploiement dans Azure App Services.

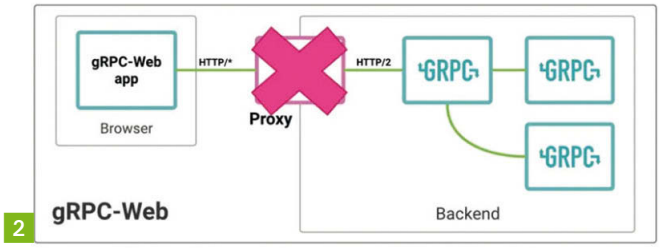
Les spécifications de gRPC-web

Il est actuellement impossible d'implémenter gRPC avec HTTP2 dans le navigateur. Il n'y a aucun moyen de forcer l'utilisation de HTTP/2, et même s'il y en avait, les trames HTTP/2 brutes sont inaccessibles dans les navigateurs. La spécification gRPC-Web commence du point de vue de la spécification HTTP/2, puis définit les différences. Il s'agit notamment :

- Prise en charge de HTTP / 1.1 et HTTP / 2 ;
- Envoi de trailers (métadonnées) gRPC à la toute fin des corps de demande / réponse comme indiqué par un nouveau bit dans l'en-tête du message gRPC ;
- Les données sont transportées en base64 contrairement à gRPC en binaire (Content-Type "application/grpc-web" ou "application/grpc-web-text") ;
- Un proxy obligatoire pour la traduction entre les requêtes Web gRPC et les réponses HTTP / 2 gRPC (voir figure 1).



1



2

gRPC-web dans ASP.NET Core 3.1

En janvier 2020, Microsoft a annoncé une prise en charge expérimentale de gRPC-web avec ASP.NET Core. Compatible avec les navigateurs (comme les applications Angular, React et plus de SPA), Blazor WebAssembly et même Xamarin, cette implémentation apporte une fonctionnalité importante : l'inutilité d'utiliser un proxy tel que Envoy. : 2

Installation des packages nécessaires :

```
Install-Package Grpc.AspNetCore.Web -Version 2.28.0-pre2
```

Configurer l'application :

Tout comme un API REST, gRPC-web nécessite la configuration de CORS (Cross-origin resource sharing).

Les origines doivent être définies avec nos domaines connus (tels que localhost et notre nom de domaine officiel sur Internet).

Les méthodes autorisées doivent être POST et en fonction des OPTIONS du navigateur aussi. Tous les en-têtes de demande doivent être autorisés. Les en-têtes de réponse (en-têtes exposés) doivent autoriser Grpc-Status et Grpc-Message, voici un exemple de configuration dans le Startup.cs :

```
services.AddCors(o =>
{
    o.AddPolicy("MyPolicy", builder =>
    {
        builder.AllowAnyOrigin();
        builder.AllowAnyMethod();
        builder.AllowAnyHeader();
        builder.WithExposedHeaders("Grpc-Status", "Grpc-Message");
    });
});
```

Ensuite l'application a besoin de savoir s'il faut activer ou pas gRPC-web sur tous les endpoints de manière globale comme suit :

```
services.AddGrpcWeb(o => o.GrpcWebEnabled = true);
```

Ou bien de manière individuelle conne suit (ainsi que l'activation des règles CORS) :

```
app.UseEndpoints(endpoints =>
{
        endpoints.MapGrpcService<CountryGrpcService>().RequireCors("MyPolicy").En
ableGrpcWeb();

        endpoints.MapGet("/", async context =>
        {
         await context.Response.WriteAsync("Communication with gRPC endpoints must be
made through a gRPC client. To learn how to create a client, visit: https://go.microsoft.com/fwlink/
?linkid=2086909");
        });
});
```

Enfin, afin de rendre opérant gRPC-web le middleware suivant a besoin d'être ajouté avant le middleware « UseEndpoints » vu précédemment.

```
app.UseGrpcWeb();
```

Ce middleware permet simplement de choisir la bonne sérialisation à fournir au client (Base64 ou binaire) en vérifiant le Content-type, en effet si ASP.NET Core détecte les Content-type suivant, alors le résultat de sortie sera en base64 :

- application/grpc-web compatible avec les services unaires ;
- application/grpc-web-text compatible avec les services de streaming unaires et côté serveur.

Et c'est tout ! En effet un service gRPC peut être consommé à la fois en tant que service gRPC livrant du binaire ou en tant que service gRPC-web livrant du base64.

gRPC-web dans Azure App Services

Contrairement à gRPC, gRPC-web dans ASP.NET Core 3 est pris en charge par Azure App Services.

La chose cependant que vous devez savoir est que Azure App Services ne prenant pas en charge HTTP/2, vous devez donc déployer votre application avec HTTP/1 en tant que paramètre de protocole de points de terminaison par défaut de Kestrel dans le fichier appsettings.json comme suit :

```
{
 "AllowedHosts": "*",
 "Kestrel": {
  "EndpointDefaults": {
   "Protocols": "Http1"
  }
 }
}
```

Si vous conservez HTTP/2 dans vos paramètres, cela ne fonctionnera que sur Windows App Services car celui-ci s'exécute avec un module nommé ASPNetCoreModuleV2 qui garantit que l'application fonctionne correctement sous Windows (transforme les requêtes HTTP/2 en HTTP/1). Sur Linux App Services, le docker en arrière échouera au démarrage :

gRPC-web et Angular 9

J'ai choisi de faire une démo avec Angular 9 car le langage utilise est Typescript et je recommande de travailler avec Typescript pour le développement Front-end. La librairie Web gRPC-improbable est une implémentation pour Typescript (pas seulement) et je vais utiliser celle-ci pour créer une application SPA avec Angular 9.

Installation de protoc

Protoc est le compilateur de fichiers protobuf, permettant de générer les clients et les entités nécessaires a l'implémentation de vos services gRPC-web.

Nous devons installer le compilateur protobuf (protoc) ici : https://github.com/protocolbuffers/protobuf/releases et choisir la version adéquate pour Windows, cette démo étant réalisée sous Windows :

Ne sélectionnez pas la version Javascript pour Windows, mais la version générique, car nous utiliserons Typescript, et nous aurons besoin d'installer un plugin pour cela :

Ensuite, je recommande fortement d'installer le protocole dans votre PATH Windows. Contrairement à ASP.NET Core dont la compilation des fichiers protobuf se fait automatiquement lors de la compilation de votre projet dans Visual Studio, ici nous devons compiler nos fichiers protobuf nous-même en ligne de commande.

Installation du plugin typescript pour protoc et des autres modules node requis

Étant donné que le **protoc** ne prend pas en charge TypeScript par défaut, nous avons besoin d'un plugin qui permet à **protoc** de générer des services TypeScript et un client à partir d'un fichier .proto et ce plugin est nommé **ts-protocol-gen**, installez-le avec la commande suivante :

```
npm install --save-dev ts-protoc-gen
```

Nous devons également installer la bibliothèque gRPC-web d'Improbable pour consommer des client gRPC-web et le package google-protobuf (+ ses types @types/google-protobuf) pour la sérialisation / désérialisation des entités (nommés **message** dans le monde gRPC) :

```
npm install --save google-protobuf
npm install --save-dev @types/google-protobuf
npm install --save @improbable-eng/grpc-web
```

Générer le client et les entités (message)

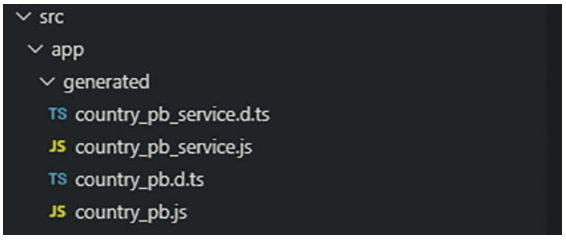
Une fois tout installé on va pouvoir générer notre client et entités a partir par exemple de ce fichier protobuf qui décrit le service **CountryService** exposant la méthode **GetAll** prenant un paramètre **EmptyRequest** et retournant la réponse **CountryReply.** A noter qu'avec le langage protobuf, il est obligatoire de passer des paramètres, même si la méthode n'en a pas, d'où le paramètre vide de toute propriétés **EmptyRequest** :

```
syntax = "proto3";

service CountryService {
  rpc GetAll(EmptyRequest) returns (CountriesReply) {}
}

message EmptyRequest {
}

message CountryReply {
    int32 Id = 1;
   string Name = 2;
   string Description = 3;
}

message CountriesReply {
    repeated CountryReply Countries = 1;
}
```

La commande à exécuter contient quatre instructions :

```
protoc
--plugin=protoc-gen-ts="{ABSOLUTEPATH}\node_modules\.bin\protoc-gen-ts.cmd"
--js_out="import_style=commonjs,binary:src/app/generated"
--ts_out="service=grpc-web:src/app/generated"
src/app/protos/{YOURPROTOFILENAME}.proto
```

--plugin : indique le chemin absolu du plugin **protoc-gen-ts,** le chemin absolu est obligatoire sous **Windows**, cela ne fonctionnera pas avec un chemin relatif contrairement à **macOS ou Linux**.
--js_out : indique avec quel module Javascript les clients et les entités seront générés et dans quel répertoire ils seront déposés. A noter que les clients et les entités sont générés en **Javascript**, et ce sont leurs fichiers de définition respectifs qui sont ensuite générés en **Typescript {servicename}.d.ts**
Voici le résultat attendu :



Nous pouvons écrire maintenant un composant **Angular** consommant un service unaire tel que décrit précédemment **(GetAll).** Pour rappel un service est un service qui envoie une requête consomme la réponse et ferme la connexion ensuite (un aller-retour comme une requête http classique), je n'ai pas ici fait d'exemple de services type streaming.

Code complet sur programmez.com & github

C'est plus verbeux qu'un appel HTTP classique (API REST) comme vous pouvez le voir.
message.toObject () as CountriesReply.AsObject; est le processus de désérialisation.
Voici à quoi ressemble le message **countriesReply** :

```
export class CountriesReply extends jspb.Message {
 clearCountriesList(): void;
 getCountriesList(): Array<CountryReply>;
 setCountriesList(value: Array<CountryReply>): void;
 addCountries(value?: CountryReply, index?: number): CountryReply;

 serializeBinary(): Uint8Array;
 toObject(includeInstance?: boolean): CountriesReply.AsObject;
 static toObject(includeInstance: boolean, msg: CountriesReply): CountriesReply.AsObject;
 static extensions: {[key: number]: jspb.ExtensionFieldInfo<jspb.Message>};
 static extensionsBinary: {[key: number]: jspb.ExtensionFieldBinaryInfo<jspb.Message>};
 static serializeBinaryToWriter(message: CountriesReply, writer: jspb.BinaryWriter): void;
 static deserializeBinary(bytes: Uint8Array): CountriesReply;
 static deserializeBinaryFromReader(message: CountriesReply, reader: jspb.BinaryReader): Co
untriesReply;
}

export namespace CountriesReply {
 export type AsObject = {
  countriesList: Array<CountryReply.AsObject>,
 }
}
```

Démo

Vous pouvez retrouver le code source de cet exemple sur mon dépôt Github : https://github.com/AnthonyGiretti/angular8-grpc-aspnetcore3-1-demo
Ce dernier contient le fichier .proto également si vous souhaitez vous amuser a générer vos clients et entités gRPC-web vous-même.
Vous pouvez naviguer à cette url pour tester le résultat final dans un navigateur également :
https://anthonygiretti.blob.core.windows.net/grpcwebdemo/index.html

Conclusion

Cet article vous a, je l'espère, aidé à appréhender **gRPC-web** dans **ASP.NET Core 3.1** avec un SPA type **Angular**. **gRPC et gRPC-web** sont plus performants qu'une API REST avec **JSON** tandis que cette dernière est beaucoup plus simple à implémenter. Je vous ai donc donné l'opportunité de tester une nouvelle manière de développer des micro services autrement qu'avec des API REST, ce qui vous donne désormais un nouveau choix dans vos spécifications techniques.
Attention toutefois, **gRPC** et **gRPC-web** restent encore immatures dans l'écosystème Microsoft, la preuve l'implémentation de **gRPC-web** dans **ASP.NET Core 3.1** est encore en preview à ce jour, je vous invite donc à la tester sans plus attendre et à vous faire votre propre opinion.

Christophe Pruvost (Oracle)
Senior Cloud Solution Architect
Depuis environ 20 ans chez Oracle, Christophe a joué avec toutes les solutions techniques Database & Middleware & Cloud et son dernier joujou est Oracle Blockchain Platform.

Ghassan Salem (Oracle)
Ingénieur Informaticien, Expert base de données chez Oracle depuis 1995

Transparent Application Continuity (TAC) :

comment une application peut ne perdre aucune transaction avec une base Oracle ?

Les bases de données Oracle sont utilisées comme le socle de persistance d'applications diverses, variées, et souvent critiques. Une application critique se doit bien évidemment de gérer correctement les différents problèmes que l'architecture peut rencontrer : problème réseau, arrêt d'un serveur (planifié ou non), arrêt de la base de données, etc.

Depuis maintenant plusieurs versions, Oracle a introduit des outils et des fonctionnalités dans son produit phare, la base de données relationnelles, pour aider les développeurs à bien gérer les différents événements qui pourraient faire baisser le taux de disponibilité d'une application. TAC constitue l'une de ces fonctionnalités majeures intervenant dans le cadre d'un cluster de bases de données (Real Application Cluster – RAC), c'est-à-dire une architecture où une base de données fonctionne sur plusieurs serveurs, donc avec plusieurs instances, mais avec un seul et même stockage partagé. Dans cet article, nous montrerons comment configurer une base Oracle pour offrir TAC aux applications, et décrirons le code Java permettant à l'application de profiter de TAC.

Retour sur TAC

Application Continuity (AC) est une technologie éprouvée, introduite dans la version 12.1 du SGBDR Oracle, qui est à cheval entre le SGBDR et ses clients (pilotes, pools de connexions). Lorsqu'elle est activée, AC consiste à enregistrer, dans la mémoire du pilote, tous les appels à la base de données effectués au cours d'une unité de travail d'application (généralement une transaction) et à relire ces appels pour les envoyer vers l'une des instances du RAC qui est toujours disponible alors qu'une autre vient par exemple de crasher. Lors de la relecture, si le résultat est correct, AC est réussi et le contrôle est rendu à l'application pour continuer sans aucune exception levée par le code de l'application, comme si rien ne s'était passé. Dans le cas contraire une exception est renvoyée au code de l'application comme cela le serait sans AC.

Transparent Application Continuity (TAC), qui a été introduit dans la base de données Oracle version 18c et les pilotes et pools correspondants, est une amélioration d'Application Continuity (AC). Avec TAC, la plupart des paramètres AC laissés aux développeurs sont pris en charge de manière totalement transparente. Par exemple :

- TAC possède un mécanisme automatique capable de restaurer tout un ensemble d'états concernant la session en cours (NLS_*, CLIENT_INFO, MODULE, ACTION, CLIENT_ID, ECONTEXT_*).
- Si vous n'utilisez pas les pools de connexions Oracle (Universal Connection Pool – UCP, Weblogic Active GridLink) vous n'avez plus besoin d'encadrer une unité de travail pour protéger vos transactions avec le code conn.beginRequest() et conn.endRequest() pour que cela fonctionne.
- Vous n'avez plus besoin non plus d'implémenter un Callback qui se déclenchera au moment où la connexion est basculée lorsque vous utilisez certaines fonctions qui pourraient renvoyer un résultat différent suite au basculement (ex : SYSDATE, SYSTIMESTAMP, SYS_GUID et sequence.NEXTVAL).

Enfin, si le développeur a implémenté un état complexe à gérer et non pris en charge automatiquement, il a toujours la possibilité de le gérer lui-même lors du basculement en implémentant un "JDBC Connection Initialization Callback" ou un "UCP Connection Labelling Callback" qui se déclenchera au moment où la connexion change d'instance. Malgré tout, il existe quelques cas où le développeur doit désactiver ce mécanisme pour éviter des effets de bord non maîtrisés. On trouve dans ces cas tous les appels externes effectués depuis la DB (FTP, email, etc..). Désactiver le mécanisme est très simple comme on le voit ci-dessous.

```
if (connection instanceof oracle.jdbc.replay.ReplayableConnection)
{
   (( oracle.jdbc.replay.ReplayableConnection)connection).disableReplay();|
}
```

Oracle Real Application Clusters (RAC) : configuration de service pour TAC.

Dans un environnement RAC, les applications se connectent à la base de données (soit directement, soit via des serveurs d'applications) grâce à ce qu'on appelle un service. Le service permet de découpler la connexion à une base des serveurs associés. L'application se connectera donc sur le serveur disponible offrant ce service. En plus de la haute disponibilité, ceci permet une meilleure répartition de la charge entre les différents serveurs.

TAC & RAC

Chaque service dans une base RAC possède plusieurs propriétés permettant de gérer son comportement vis-à-vis des incidents et définissant la répartition de charge entre les serveurs. Nous indiquerons ici les quelques propriétés essentielles pour TAC :

Preferred instances : sur quelles instances ce service est disponible, par exemple, DB02021,DB02022. Ceci indique que le service est disponible sur 2 machines.

Available instances : en cas d'arrêt de l'une des preferred instances, le service peut être rendu disponible sur l'une des available instances.

Failover Type : en cas d'arrêt d'un serveur, comment basculer sur un autre. Pour TAC, il faut mettre la valeur 'AUTO'.

Failover retries : combien de fois il faut essayer de basculer une connexion avant de retourner une erreur à l'application. Pour TAC, la valeur conseillée est de 60.

Failover delay : attente entre deux essais de reconnexion, en secondes. Mettre à 5.

```
PoolDataSource dataSource = PoolDataSourceFactory.getPoolDataSource();
dataSource.setConnectionFactoryClassName("oracle.jdbc.replay.OracleDataSourceImpl");
dataSource.setUser(username);
dataSource.setPassword(password);
dataSource.setURL(url);
dataSource.setFastConnectionFailoverEnabled(true);
dataSource.setValidateConnectionOnBorrow(true);
dataSource.setMinPoolSize(6);
dataSource.setMaxPoolSize(12);
```

1

```
Node Down, Public Network Down
Instance Down, Instance Up
Service Down, Service Up, Service member Down, Service member Up
Database Down, Database Up
```

2

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.807 seconds

	CON_ID	TOTAL_REQUESTS	TOTAL_CALLS	TOTAL_PROTECTED	PROTECTED
1	3	133802	197387	197405	100.009119141584805483643806329697497809

4

Failover restore : indique s'il faut remettre en place, pour les connexions, l'environnement des sessions. Mettre à Auto pour TAC.

Retention : Le nombre de secondes durant lequel le résultat d'un commit est conservé. Pour TAC, la valeur est de 86400 secondes (1 journée).

Replay Initiation Time : après combien de secondes on renonce à rejouer les transactions en cours au moment de l'arrivée de l'incident (300 secondes).

Notification : faut-il automatiquement notifier les applications de l'incident ? Mettre à True pour TAC.

Commit_outcome : suivre ou non le résultat des commits. Mettre à True pour TAC.

Drain_timeout : le temps à attendre avant de couper les connexions, en cas de basculement initié manuellement. Une bonne valeur semble être 300 secondes..

La commande 'srvctl modify service' peut être utilisée pour ajuster les paramètres, si ce ne sont pas les bonnes valeurs à la création du service :

```
$ srvctl add service -db mydb -service GOLD –preferred inst1 -available serv2 -failover
_restore AUTO -failoverretry 1 -failoverdelay 3 - commit_outcome TRUE -failovertype
 AUTO -replay_init_time 600 -retention 86400 -notification TRUE -drain_timeout 300 -
stopoption IMMEDIATE
```

Serveur Java (Tomcat ou autre) : la dataSource TAC à utiliser.

Quelques prérequis :

- Il vous faut utiliser l'un des JDK suivant : JDK8 (JDK8u163+), JDK9, JDK10 or JDK11.
- Il faut récupérer le dernier driver JDBC avec tous les jar l'accompagnant depuis Maven Central (aujourd'hui c'est le driver 19.6.0.0) afin de les mettre dans le classpath de votre application : ojdbc8.jar or ojdbc10.jar (dépendant de votre JDK), ucp.jar, oraclepki.jar, osdt_core.jar, osdt_cert.jar, ons.jar, simplefan.jar; xdb.jar ;xmlparserv2.jar;orai18n.jar.

Au niveau code :

- La dataSource doit provenir de UCP et donc du package oracle.ucp.jdbc.*
- La Factory class vient de oracle.jdbc.replay.*
- L'url doit suivre la syntaxe suivante (ex avec 2 instances) :

jdbc:oracle:thin:@(DESCRIPTION=(CONNECT_TIMEOUT=120)(RETRY_COUNT=20)(RETRY_DELAY=3)(TRANSPORT_CONNECT_TIMEOUT=3)(ADDRESS_LIST=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=129.146.**.***)(PORT=1521)))(ADDRESS_LIST=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=129.146.**.***)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=soe_tac.subxxxxx.vcnadwc.oraclevcn.com)))" 1

```
totalRequests:                          2967
totalCompletedRequests:                 2962
totalCalls:                             33848
totalProtectedCalls:                    30644
totalCallsAffectedByOutages:            1
totalCallsTriggeringReplay:             1
totalCallsAffectedByOutagesDuringReplay: 0
successfulReplayCount:                  1
failedReplayCount:                      0
replayDisablingCount:                   948
totalReplayAttempts:                    1
currentRequestSize:                     0
```

3

On peut remarquer ici qu'en plus on dispose de Fast Connection Failover (FCF) qui permet au pool de connexions de s'adapter instantanément aux évènements de la base remontés par Fast Application Notification (FAN). Voici la liste des évènements possibles : 2

Par exemple :

- Sur un évènement « Instance Down », le pool de connexions supprime instantanément les connexions qui pointaient sur cette instance.
- Sur un évènement « Instance up », le pool de connexions se rééquilibre en supprimant des connexions vers les autres instances et en créant de nouvelles connexions vers l'instance qui a redémarré.

Astuces :

On peut récupérer l'instance sur laquelle pointe la connexion avec le code suivant :

```
instancename = ((oracle.jdbc.internal.OracleConnection)conn).getServerSessionInfo().get
Property("INSTANCE_NAME");
```

Au niveau du code Java on peut récupérer des statistiques sur le replay avec la syntaxe suivante :

```
ReplayStatistics replayStatistics = ((ReplayableConnection)connection).getReplayStatistics
(ReplayableConnection.StatisticsReportType.FOR_ALL_CONNECTIONS);
```

3

Au niveau de la Database on peut aussi récupérer des statistiques sur le replay avec le code suivant :

```
set lines 85
col Service_name format a30 trunc heading "Service"
break on con_id skip1
col Total_requests format 999,999,9999 heading "Requests"
col Total_calls format 9,999,9999 heading "Calls in requests" col Total_protected format
 9,999,9999 heading "Calls Protected" col Protected format 999.9 heading "Protected %"
select con_id, total_requests, total_calls,total_protected,total_protected*100/NULLIF
(total_calls,0) as Protected from(
select * from
(select s.con_id, s.name, s.value
FROM GV$CON_SYSSTAT s, GV$STATNAME n WHERE s.inst_id = n.inst_id
AND s.statistic# = n.statistic#
AND s.value != 0 )
pivot(
 sum(value)
 for name in ('cumulative begin requests' as total_requests, 'cumulative end requests' as
Total_end_requests, 'cumulative user calls in requests' as Total_calls,
'cumulative user calls protected by Application Continuity' as total_protected)
))
order by con_id;
```

4

Ressources supplémentaires

La video démo sur youtube : https://youtu.be/tKpljU7hKJ0

Le document qui retrace toutes les recommandations sur TAC : https://www.oracle.com/technetwork/database/options/clustering/applicationcontinuity/adb-continuousavailability-5169724.pdf

JOSEPH Karthik
Senior Manager chez Magellan Consulting, JOSEPH Karthik est architecte entreprise et architecte Big Data qui accompagne ses clients dans la transformation digitale et la gestion des assets Data. Dans son temps libre, il aime se prendre la tête avec Scala, Haskell, etc.

Magellan Consulting

Programmation fonctionnelle : plus simple qu'on ne le pense

Lorsque j'ai découvert la programmation fonctionnelle, il y a plus de trois ans, j'étais un peu perdu avec le jargon mathématique un brin ésotérique qui lui est souvent associé (monoïde, monade, isomorphisme, etc.). À première vue, ça me semblait surtout anecdotique ou une sorte de troll (c'est toujours un sujet de troll avec mes collègues…).
Mais depuis, les choses ont beaucoup changé et je reste fasciné par l'élégance et l'efficience apportées par cet ovni qu'est la programmation fonctionnelle (ou FP – Functional Programming). J'espère qu'à travers cet article, vous percevrez et comprendrez mieux les avantages de ce paradigme.
La FP n'a rien de nouveau. Apparue dans les années 30, à travers les théories du lambda-calcul et de LISP, les idées et notions qui soutiennent cette dernière ont été intégrées dans bon nombre de langages de programmation utilisés aujourd'hui tels que JavaScript, Python, PHP, Java, etc. Oubliés ou relégués au cercle académique, ces concepts refont surface aujourd'hui via des langages comme Scala, Elixir, Haskell, F#, Clojure, pour ne mentionner qu'eux. Il est regrettable qu'une querelle de chapelle se soit installée entre la programmation orientée objet (OOP) et la programmation fonctionnelle… Alors qu'elles sont plutôt complémentaires.

Qu'est-ce donc que la programmation fonctionnelle ?

Il s'agit d'un paradigme de programmation où ce sont les fonctions (et non les objets) qui sont au cœur des considérations. De manière schématique, dans la FP, on exprime ce qu'on l'on souhaite obtenir au lieu de la façon dont on doit procéder pour obtenir ce que l'on cherche (ce qu'on fait habituellement en OOP via la programmation impérative).
Par exemple, pour calculer la somme d'une liste de nombres entiers, dans la FP, on décrira plutôt une structure récursive (plus précisément un « reduce » ou un « fold »), au lieu de définir une boucle avec un incrément de manière impérative. La FP pousse à réfléchir et à résoudre les problèmes autrement. Et avec un peu d'habitude, ce mode de raisonnement vous deviendra peut-être beaucoup plus naturel et fluide. 1

D'ailleurs, on n'est pas obligé de comprendre les concepts et la terminologie mathématique qui se cachent derrière.
Voici de manière très schématique les « besoins » qu'on peut avoir et comment ces derniers se déclinent en termes de solutions et terminologies mathématiques correspondantes :

Besoins	Formulation mathématique	Scala | Haskell
Composer	Curryfication, fonctions d'ordre supérieur	Composition et manipulation de fonctions
Itérer	Récursions, compréhensions	Structures récursives, etc.
Combiner et agréger	Monoïde	fold| fold
Manipuler des effets de bords	Foncteur	map | fmap
Enchaîner des opérations avec des effets de bords	Monade	flatMap | >>=

Il serait beaucoup trop long de rentrer dans le détail de chaque ligne ; mais on peut regarder de plus près un ou deux éléments à titre d'exemples afin d'illustrer les concepts qui se cachent dans la FP.
Une des notions qui fait la force de la FP, c'est la composition. C'est un peu comme des briques de Lego. Chaque pièce de Lego est unitaire et atomique. Par ailleurs elles sont fabriquées de telle manière pour qu'elles puissent s'imbriquer l'une dans l'autre, quel que soit le contexte. On peut soit fabriquer une maison ou une voiture, mais ce sont toujours les mêmes pièces qu'on réutilise. Et plus fort encore, un ensemble de pièces devient à son tour réutilisable par conception (on peut commencer par construire des chambres, puis des étages et finalement un appartement). Tout s'emboîte ! Dans la FP, les fonctions sont comme des pièces de Lego et la curryfication (à partir du mathématicien Haskell Curry) permet de créer de nouvelles fonctions et de les composer. La curryfication permet de

1

Java

```
public static int somme(List<Integer> uneListe) {
  int somme = 0 ;
  for (int i : uneListe) {
     somme += i ;
  }
  return somme ;
}
```

Haskell

```
somme :: [Int] -> Int
somme = foldl (+) 0
```

Explication : *La 1ère ligne donne la signature de la fonction, c'est-à-dire ce que la fonction va prendre en entrée (dans notre cas une liste d'entiers) et donner en sortie (un entier). Dans la 2ème ligne, la fonction foldl est un des outils dans le trousseau du programmeur fonctionnel. Il permet d'appliquer une fonction sur une liste d'éléments et de la réduire. Il prend 3 arguments : une fonction (dans notre cas, il s'agit de l'opération d'addition +), un accumulateur et une liste d'éléments. La fonction va appliquer de manière récursive la fonction donnée sur chaque élément de la liste et en conservant le résultat courant dans l'accumulateur. Il est intéressant de noter que le dernier argument n'est pas déclaré – il est implicite.*

transformer une fonction qui a plusieurs arguments en une fonction qui en a qu'un seul.

Penons l'exemple suivant d'une fonction qui ajoute deux entiers :

ajouter2Entiers x y = x + y

Cette fonction peut être vue comme une fonction à un seul argument (qui est y par exemple). On peut donc grâce à la currification créer une nouvelle fonction ajouter5 qui ajoute 5 à un nombre.

ajouter5 = ajouter2Entiers 5

ajouter5 10 → La fonction va nous retourner 15

On peut aussi chaîner des opérations facilement :

ajoutEnChaine = ajouter5 ajouter2 (cette nouvelle fonction va ajouter 5 et 2 à un nombre passé en argument.)

Le principe de currification peut sembler simple, mais il est extrêmement puissant et utile, car il permet de composer des fonctions existantes et d'en créer des nouvelles

Un autre objet puissant, mais qui peut paraître obscur, ce sont les monades. Les monades ne sont ni plus ni moins qu'une stratégie pour gérer les effets de bords et les exceptions de manière robuste et « propre » (on va essayer de conserver la pureté de nos fonctions).

Elles ont trois propriétés fondamentales qui les rendent utiles :

- L'isolation ou le cloisonnement. À travers les monades, vous pouvez séparer les ensembles de votre application qui sont purs des ensembles qui ont des effets de bords. Par exemple, si vous avez besoin de faire un appel API vers un service distant, vous n'avez aucune garantie que vous allez obtenir une réponse et ceci peut générer des erreurs. En mettant le résultat de l'appel API dans une monade, vous pouvez séparer la partie qui peut être source d'erreurs de la partie qui ne l'est pas (par exemple, l'exploitation d'une réponse valide dans notre cas).
- La flexibilité, car elles forcent la gestion des différents états dans un endroit unique.
- La modularité, car elles facilitent la composition et la création de logiques plus complexes à partir de briques élémentaires.

En gérant de manière explicite les effets de bords, on accroît la capacité à composer des fonctions avec des effets de bords. Une exception devient un état « nominal » de l'application qui va forcer sa gestion (avant que l'exception ne se transforme en erreur).

Quelques monades connues (en terminologie Haskell, mais vous pouvez trouver très facilement les équivalents dans les autres langages) :

- *Maybe* (qui a deux valeurs possibles *Nothing* et *Just x*), utiliser lorsqu'une opération n'est pas sûre de renvoyer un résultat,
- la monade *Error*, utilisée pour gérer des exceptions,
- la monade *IO* pour gérer les actions liées aux entrées / sorties du système.

Un des petits plus liés à une monade est qu'elle rend le code beaucoup plus lisible et clair. Par exemple, quand on code un traitement qui est fait d'étapes successives dépendant de la bonne exécution des tâches précédentes (par exemple lorsqu'on a des callbacks), on se retrouve à avoir quelque chose comme ceci (qui devient vite illisible) :

```
faireAction =
    if ( tache1 == success ) {
        if ( tache2 == success {
            … (on a notre « pyramid of Doom »)
        }
    }
return resultat
```

Avec les monades,

```
faireAction = do
    x1 <- tache1
    x2 <- tache2
    …
    return xn
```

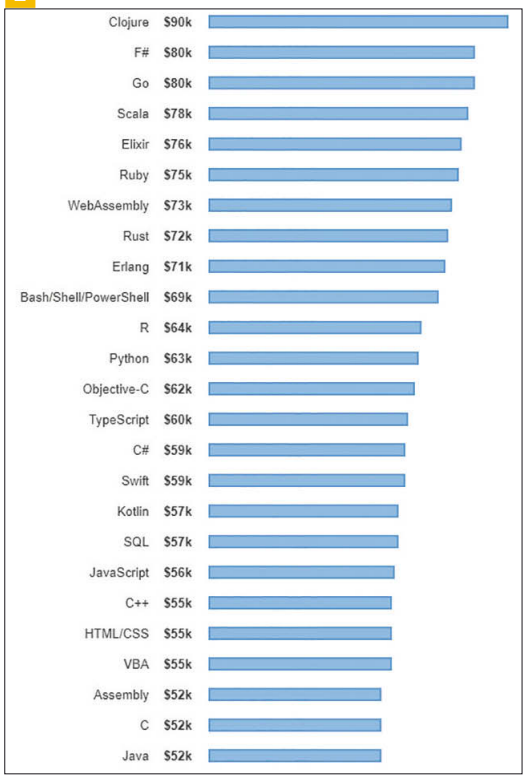
De manière générale, on travaille sur des définitions et les comportements associés. Les fonctions sont des sortes de « boîtes noires » génériques dotées d'un comportement appelé « pur » (reproductible et stable sans effets de bords) avec des entrées et des sorties bien définies (fortement typées). Ces dernières permettent d'avoir un meilleur contrôle sur les opérations (gain en lisibilité et exactitude du code, etc.). La FP est donc plus pertinente lorsqu'on travaille avec un ensemble de données « statiques », mais qui sont manipulées par beaucoup d'opérations. Cependant, lorsqu'on manie un grand nombre de types de données avec des structures variables, mais peu d'opérations, la modularité offerte par les langages OOP devient plus intéressante. Chaque paradigme possède ses propres avantages et inconvénients, et c'est au développeur de choisir le meilleur outil pour répondre à son besoin.

Les partis pris de la FP engendrent de nombreux avantages, comme une plus grande facilité pour paralléliser les opérations, du code qui est plus facile à comprendre, une réduction des bugs, la facilitation des tests… C'est donc un outil de choix pour traiter des problématiques data. D'ailleurs, lorsque les développeurs travaillent avec des outils comme Spark, ils travaillent dans un écosystème fortement typé fonctionnel ! L'explosion des besoins autour de la valorisation de la donnée et de son traitement va tirer les langages fonctionnels de l'ombre. À ce jour, c'est surtout Scala (qui permet d'associer à la fois la FP et OOP) qui jouit d'une forte popularité contrairement aux langages comme Haskell ou Elixir. D'ailleurs, en regardant les sondages portant sur les langages de programmation et les rémunérations associées, on constate que 4 des 5 langages qui ont les rémunérations les plus élevées sont des langages de programmation fonctionnelle. 2

Source : https://insights.stackoverflow.com/survey/2019#top-paying-technologies

Mais au fur à mesure que les librairies s'enrichissent, on peut s'attendre à ce que les langages fonctionnels jouent un rôle beaucoup plus structurant dans les SI de demain.

2



Sallah KOKAINA
Ingénieur en informatique et auteur du livre « Software Craftsmanship : L'art du code et de l'agilité technique en entreprise » (Editions ENI 2019). Sur ces 12 dernières années, il a tenu différents rôles (Développeur, CTO, Qualiticien, Manager Technique…) au sein de différentes structures et y a mis en place des solutions basées notamment sur Scala et Java.

Scala : le dossier in a nutshell en comparaison à la JDK 14

Scala est un langage hybride, à la fois fonctionnel et orienté objet, qui se base sur la JVM. À l'aube de sa troisième version, prévue d'ici fin 2020, il regroupe un ensemble de concepts qui lui donnent une longueur d'avance sur les autres langages, notamment Java 14 (Mars 2020).

Scala est un langage multi-paradigme moderne, au typage statique fort, conçu pour exprimer les patrons de conception communs de façon concise, élégante et type-safe. Il supporte une approche dite hybride combinant tant de la programmation dite fonctionnelle que de la programmation dite orientée-objet. Scala tire ses origines de l'anglais Scalable, qui définit la capacité d'un système à s'étendre pour répondre à des montées de charge ou à des besoins en évolution de syntaxe. Et dans le premier cas, en étant par exemple capable de paralléliser des traitements sur plusieurs machines. Martin Odersky l'aborde pendant l'OSCON Java 2011, en expliquant l'avènement depuis 2005 de limitations en puissance de calcul des micro-processeurs, avec une industrie du hardware qui compense ce déficit par l'augmentation du nombre de cœurs CPU par machine. Là où par le passé on cherchait à gagner en puissance de calcul verticale, Scala vient comme un langage JVM alternatif, pensé pour supporter nativement la recherche de puissance de calcul de façon horizontale : soit sur plusieurs cœurs, ou sur un réseau de machines. Dans ce mode, les perspectives en puissance CPU deviennent infinies, plaçant Scala dans l'ère du temps, et comme une alternative pertinente pour répondre aux nouveaux enjeux de l'industrie Digitale : haute disponibilité, faible latence, traitement de flux de données, architecture multi-tenant.

Il est pensé pour permettre l'extensibilité, et veiller dès la rédaction des briques de code à avoir une syntaxe favorable au parallélisme, tout en aidant à créer une cohérence des résultats et de l'état applicatif. Cela passe principalement par l'utilisation des fonctions comme first-class citizen du langage. Contrairement à Java où le first-class citizen est l'Objet, il est possible dans Scala, depuis les premières versions, de définir des variables comme étant des fonctions, ou de passer des fonctions en paramètres d'autres fonctions. On parle ainsi de High-order function. C'est là où la magie s'opère ; en passant par des fonctions pures, on s'ouvre la possibilité de déléguer des parties de programme à d'autres machines ou cœurs CPU qui n'ont pas besoin d'avoir accès à l'état applicatif père pour réaliser le traitement qui leur est confié. Spark* et Akka** sont des frameworks qui en démontrent le potentiel.

Les concepts clés en code

Scala vient avec de nombreux concepts. Nous allons en illustrer quelques-uns avec du code. On va comparer certaines caractéristiques avec un équivalent en Java 14. On s'accordera sur quelques notations :

- //res: … : résultat d'une expression ;
- [//version: …](//version) : version du langage utilisée pour l'exemple ;
- //note: … : commentaires.

Surcharge d'opérateurs, Classes, Objets et Traits

Dans Scala, les opérateurs arithmétiques fonctionnent comme en Java, Javascript ou C++. La petite particularité de Scala c'est que ces opérateurs (+ / * -) sont en réalité des méthodes.

a + b est en réalité un raccourci pour a.+(b), b étant l'argument de la méthode + définie dans le type de l'instance a. Dans Scala, il n'y a pas d'opérateurs, tout est fonction ! Du coup, il est possible de redéfinir des opérateurs pour des objets, et ainsi appliquer des opérations arithmétiques sur les objets non-natifs, telles que les collections (List, Set, Map, …). Voici quelques exemples qui illustrent à quel niveau la surcharge d'opérateur permet d'écrire du code plus naturel :

```
//version: scala 2.13
//note: combinaison de listes
val combination = List(1,2,3) ++ List(2,1,3)
// res: List(1,2,3,2,1,3)

//version: java 14
//note: un equivalent fonctionnel en java 14 utilisant des streams
Stream.of(List.of(1, 2, 3),List.of(2, 1, 3))
    .flatMap(Collection::stream)
            .collect(Collectors.toList())
// res: List(1,2,3,2,1,3)

//version: scala 2.13
//note: autres exemples
val set = Set(1, 2, 3) + 4
// res: Set(1, 2, 3, 4)
val map = (Map(1 -> "Hello", 2 -> ", ") + (3 -> "Scala")).values.mkString
// res: Hello, Scala
val sq = 1 +: Seq(2, 3) :+ 4
// res: List(1, 2, 3, 4)
```

La surcharge d'opérateurs s'implémente tel qu'illustré dans l'exemple suivant pour la classe Approx:

```
//version: scala 2.13

trait ApproxUnion {
 def |(that: Approx): Approx
 def asString(approx: Approx): String = s"${approx.x}"
}

class Approx(val x: Double) extends ApproxUnion {
 def +(that: Approx): Approx = new Approx(this.x + that.x)
 override def |(that: Approx): Approx = this + that
 def unary_~ = Math.floor(x)
```

```
 def unary_! = 0.0
 override def toString: String = asString(this)
}

object Approx {
 def apply(x: Double): Approx = new Approx(x)
 def unapply(x: Doube): Some[Double] = Some(x)

 def main(args: Array[String]): Unit = {
  println(Approx(1.1) + Approx(3.1))  //res: 4.2
  println(~Approx(1.1) + ~Approx(3.1)) //res: 4.0
  println(~Approx(1.1) + !Approx(3.1)) //res: 1.0
  println(Approx(1.1) | Approx(3.1))  //res: 4.2
 }
}
```

Quelques éléments intéressants à relever :

1 • La création de la classe **Approx**, ne nécessite pas la déclaration d'un constructeur. Par défaut, le compilateur de Scala crée automatiquement les méthodes nécessaires, incluant : getters, setters ainsi que le constructeur nécessaire. Dans ce cas précis, il n'est pas possible d'invoquer le constructeur new Approx(). Le compilateur s'attend à ce qu'un paramètre de type Double soit renseigné : new Approx(double).

2 • Un trait est équivalent à une interface en Java. Il est possible de fournir une implémentation par défaut aux méthodes comme illustré par la fonction asString.

3 • La surcharge des opérateurs + et | enrichit la syntaxe en permettant d'utiliser une notation mathématique et plus naturelle pour additionner les valeurs doubles des instances Approx.

4 • Les opérateurs unaires ou préfixes ~, !, +, - peuvent être définis en déclarant respectivement les méthodes unary_~, unary_!, unary_+ et unary_-. Ce qui dans l'exemple est reflété par la simulation de la négation (!) et de l'approximation (~).

5 • On peut créer pour une classe, à l'aide du mot clé object, un objet dit compagnon, qui vient avec quelques particularités. Ce dernier est équivalent en Java aux classes statiques, il ne peut être ni instancié ni paramétré avec des types génériques. Toutefois, il vient avec quelques spécificités :

6 • La première étant la possibilité de définir les méthodes apply, un syntaxic suggar qui permet de faire appel au constructeur d'une classes sans recours au mot clé new ;

La méthode unapply est utilisée par le pattern matching et sert d'extracteur. Des méthodes comme flatMap y ont recours ;

Il doit être déclaré dans le même fichier de la classe et porter le même nom.

Comme en Javascript, il est possible de recourir à l'interpolation des chaînes de caractères, en préfixant celles-ci d'un s. Il existe d'autres préfixes natifs : f, raw. Cerise sur le gâteau, on peut créer des interpolations customisées à l'image du préfixe json dans l'exemple suivant :

```
// version: scala 2.13
import scala.util.parsing.json._

val pattern = "([a-zA-Z]+)".r
val name = "Hello"
val id = "Scala"

// note: on étend AnyVal pour empêcher l'instanciation de JsonHelper au runtime
implicit class JsonHelper(val sc: StringContext) extends AnyVal {
 def json(args: Any*): JSONObject = {
  val keys: Seq[String] = pattern.findAllIn(sc.parts.mkString("")).toList
  val values: Seq[Any] = args.toList
  JSONObject((keys zip values).toMap)
 }
}

json"{ name: $name, id: $id }"
//res: JSONObject = {"name" : "Hello", "id" : "Scala"}
```

Ces quelques éléments montrent à quel point Scala favorise la création d'un code à l'expression riche, extensible et naturelle.

Pattern Matching

Scala dispose d'un mécanisme de switch assez puissant appelé pattern matching. Ça peut identifier des instances de classes, des expressions régulières, faire des évaluations logiques ou encore des évaluations avancées vis-à-vis de la composition d'une liste.

Dans cet exemple, on voit comment le pattern matching permet d'identifier une instance d'objet :

```
//version: scala 2.13
val result = objet match {
    case x: Int => x
    case s: String => s.toInt
    case _: BigInt => Int.MaxValue //un objet de type BigInt
    case _ => 0 //un objet autre que Int, String ou BigInt
}

//version: Java 14
var result = -1;
if(objet instanceof Integer entier) {
  result = entier;
}else if (objet instanceof  String s) {
  result = Integer.parseInt(s);
}else if (objet instanceof BigInteger b){
  result = Integer.MAX_VALUE;
}else {
  result = 0;
}
```

Ici, on pattern-match une liste avec interpolation (s"${var}") des chaînes de caractères :

```
//version: scala 2.13
List("A",2,"B",5) match {
    case x :: Nil => s"${x} - un seul élément"
 case x :: tail => s"${x} - 1er élement, ${tail} - reste de la liste"
 case _ => "liste vide"
}
//res: A - 1er élement, List(2, B, 5) - reste de la liste

//version: Java 14
var list = List.of("A",2,"B",5);
var listInstrospection = "";
if(list instanceof List l && l.size() == 1){
  listInstrospection = String.format( "%s - un seul élément", l.get(0));;
```

```
}else if(list instanceof List l && l.size() > 1){
   listInstrospection = String.format( "%s - 1er élément, %s - reste de la liste", l.get(1),
 l.stream().skip(1).collect(Collectors.toList()));
}else {
   listInstrospection = "liste vide"
}
//res: A - 1er élement, [2, B, 5] - reste de la liste
```

Ceux-ci sont quelques exemples, mais les possibilités sont bien plus larges. Notamment, les traitements de flux avec différents types d'objets ou encore le parsing d'objet Json. Comparé à Java qui vient d'introduire la notion de pattern matching dans sa version 14, Scala a le mérite d'offrir cette capacité depuis bien plus longtemps, permettant jusqu'à ce jour une expression bien plus compacte, concise et lisible.

High-order functions pour les collections : map, flatMap, fold, reduce, scan, zip

En complément, voici quelques méthodes clés qui illustrent à quel point ce langage rend aisée la manipulation des listes.

• **map**: transformer les éléments un à un

```
//version: scala 2.13.2
List(1,2,4).map(x => { if (x < 3) Math.pow(x,2) else x })
// res: List(1.0, 4.0, 4.0)

//version: java 14
List.of(1, 2, 4).stream().map( x -> {
            if(x < 3) return Math.pow(x, 2);
            else return x;
         }).collect(Collectors.toList()) ;
// res: List(1.0, 4.0, 4.0)
```

• flatMap: transformer (map) puis aplatir (flatten)

```
//version: scala 2.13.2
val list = List(1,2,3,4,5)
def g(x:Int) = List(x-1, x, x+1)

list.flatMap(g)
//res: List[Int] = List(0, 1, 2, 1, 2, 3, 2, 3, 4, 3, 4, 5, 4, 5, 6)

//note: équivalent à
list.flatMap(x => g(x)) ou list.flatMap(_ => g(_))
//res: List[Int] = List(0, 1, 2, 1, 2, 3, 2, 3, 4, 3, 4, 5, 4, 5, 6)

//note: ou encore
val transformWithGx = list.map(x => g(x))
//res: List[List[Int]] = List(List(0, 1, 2), List(1, 2, 3), List(2, 3, 4), List(3, 4, 5), List(4, 5, 6))
transformWithGx.flatten
//res: List[Int] = List(0, 1, 2, 1, 2, 3, 2, 3, 4, 3, 4, 5, 4, 5, 6)
fold, reduce & scan
//version: scala 2.13.2

val numbers = List(1, 2, 3, 4, 5)
numbers.fold(10)(_+_) // res: Int = 25
numbers.fold(0)(_+_)  // res: Int = 15
numbers.reduce(_+_)  // res: Int = 15
numbers.scan(0)(_+_)  // res: Int = List(0, 1, 3, 6, 10, 15)
numbers.scan(10)(_+_)  // res: Int = List(10, 11, 13, 16, 20, 25)
```

• **numbers.fold(0)(_+_)** utilise la notation wildcard _, équivalente à - numbers.fold(0) { (acc, el) => a + el) } ;

• **reduce**: cumule les éléments en un résultat ;

• **fold****: cumule les éléments en un résultat avec une valeur de départ ;

• **scan****: cumule une collection de résultats intermédiaires en utilisant une valeur de départ.

Ci-dessous, on illustre la méthode **zip** qui permet de fusionner des collections :

```
//version: scala 2.13.2

//1. Instruction 1
val zipped: List[(Int, String)] = List(1,2,3).zip(List("A","B","C","E")) // res: List[(Int, String)] =
 List((1,A), (2,B), (3,C))

//2. Instruction 2: vous voulez une map ? :)
val zippedMap = zipped.toMap
//res: scala.collection.immutable.Map[Int,String] = Map(1 -> A, 2 -> B, 3 -> C)
```

La première ligne crée à partir d'une liste d'Int et d'une liste de String, une liste de Tuple(Int, String). Autrement dit, une liste de binômes. De manière pratique, la seconde instruction permet de convertir la liste en un dictionnaire de type Map avec les entiers comme clé et les chaînes de caractères comme valeurs.

Implicits

Les implicites participent à réduire le nombre de paramètres explicitement passés en paramètres à une fonction, voire de pouvoir exécuter des fonctions sans appel direct. C'est le compilateur qui devinera en cas d'omission, quelle valeur passer à une fonction ou quelle opération invoquer.

```
//version: scala 2.13.2

//note: 1. paramètre implicite
def f(base: Int)(implicit by: Int) = (2 * by) / base
implicit val factor = 3
f(1) // res: Int = 6
f(2) // res: Int = 3

//note: 2. fonction implicite
implicit def doubleToInt(c: Double): Int = c.toInt
val x: Int = 42.0  //rest: Int = 42
```

Respectivement, le compilateur détecte :

1 • **Le paramètre implicite** : pour l'appel de fonction f, que factor peut être affecté au paramètre implicite by de la fonction f.

2 • **La fonction implicite** : on se rappelle bien que Scala est un langage au typage statique fort. Il ne serait pas possible par défaut d'affecter un Double à un objet de type Int. Pourtant la déclaration de la fonction implicite doubleToInt permet au compilateur de créer une conversion implicite.

Autant l'utilisation des implicites peut fournir quelques avantages, autant elle peut rendre la lecture du code difficile et induire en erreur. Toutefois, le compilateur vient avec quelques contrôles qui aident à détecter certains problèmes à la compilation.

```
//version: scala 2.13.2
```

```
def exemple1(implicit x: Int)                  // x est implicite
def exemple2(implicit x: Int, y: Int)          // x and y sont implicites
def exemple3(x: Int, implicit y: Int)          // erreur de compilation
def exemple4(x: Int)(implicit y: Int)          // seulement y est implicite
def exemple5(implicit x: Int)(y: Int)          // erreur de compilation
def exemple6(implicit x: Int)(implicit y: Int)     // erreur de compilation
```

For-comprehension loop

Passons désormais aux boucles. Scala possède un mécanisme qui ressemble à une gestion de boucles dans les langages classiques (C, Java, Javascript…).

```
//version: scala 2.13.2

for (x <- List(1,2,3)){
 print(x) //note :affichera 123 dans la sortie standard
}
```

Mais en réalité c'est bien plus puissant. En effet, l'expression for est un syntaxic sugar des fonctions map/flatten ou encore flatMap comme l'illustrent les exemples qui suivent :

Code complet sur programmez.com & github

Comme on peut le constater, une for-comprehension loop est plus facile à lire qu'un enchaînement de map et de flatMap. Et permet ainsi de mieux comprendre l'intention du code sous-jacent, d'où son nom.

Immutabilité

L'immutabilité est une propriété de Scala. Les objets et les collections utilisés par défaut ont la propriété d'être "Immutables". Ce qui renforce l'intégrité du traitement qui se réalise au niveau d'une fonction, vu que les objets ne sont pas altérés par celle-ci, ainsi que la capacité d'écrire du code nativement parallélisable. La déclaration d'un attribut avec le mot clé val rendra ce dernier immutable, ainsi il ne sera plus possible d'assigner une nouvelle valeur en cours de route à l'instance associée.

```
//version: scala 2.13.2
val immutable = List(1,2,3)
immutable = List(1) // erreur de compilation

//version: java 14
final var immutableJ = List.of(1,2,3)
```

On pourra noter par ailleurs que les collections sous le package scala.collection, avec le sous-package mutable, contiennent les implémentations mutables, et le sous-package immutable, contiennent les implémentations non mutables.

Code complet sur programmez.com & github

- La liste ml est étendue avec deux nouveaux entiers, 6 et 9 ;
- Il n'est pas possible d'ajouter un nouvel élément à une liste immutable, même en la déclarant en tant qu'attribut variable ;
- Bien qu'il semble que notre instance iml ait muté en List(1, 3, 5), il s'agit en réalité d'une copie de l'instance à laquelle on a intégré l'élément 5.

Currying

Passons à un autre concept intéressant, celui du currying. Utilisé notamment pour la création de fonctions de haut-niveau, cette notation permet de convertir une fonction qui prend deux arguments en une fonction qui n'en prend qu'un seul. Au final, la fonction retourne une autre fonction qui consomme le second argument. Voyons un exemple :

```
//version: scala 2.13.2

//note: fonction d'addition
def add(x: Int, y: Int) => x + y

//note: version sans currying
def addF(x: Int) => (y: Int) => x + y
addF(1)(2) //res: 3

//note: version currying
def addC(x: Int)(y: Int) => x + y
addC(1)(2) //res: 3

//note: high-order function
def addS(x: Int)(f: Int => Int) => f(x)
addS(1){_ + 2}
```

L'intérêt du currying peut être discutable. Néanmoins, il peut s'avérer utile quand on veut manipuler des Génériques, et ainsi mieux capter les inférences de types. C'est là un concept avancé mais bon à avoir en tête.

Generics

Les Génériques en Scala ressemblent à ce qu'on peut voir en Java ou C++. Toutefois les capacités du langage vont bien au-delà. Rien de mieux que quelques exemples pour développer l'idée.

```
//version: scala 2.13.2
class Pair[T,S](val first: T, val second: S)
new Pair(12, 4)
// res: Pair[Int,Int]
new Pair("toto", 4)
// res: Pair[String,Int]

//version: java 14
record PairRecord<T, R>(T t, R r) { }
var pair = new PairRecord(1, "120");
// res: PairRecord[t=1, r=120]
```

Nous venons de déclarer une classe Pair avec deux attributs de types génériques T et S. Ce qui est intéressant à remarquer, c'est que Scala arrive à inférer les types à l'appel du constructeur.
Les génériques peuvent être aussi utilisés au niveau des fonctions.

```
//version: scala 2.13.2
def middleOf[T](a: Array[T]) = a(a.length /2)
middleOf(Array(1, 2, "z", 4, "a"))
// res: z

//note: on peut contraindre le type à l'appel
middleOf[Int](Array(1, 2, "z", 4, "a"))
// res: erreur de compilation
```

Dans l'exemple précédent, le paramètre T n'est pas contraint, on peut au final passer n'importe quel élément héritant de Any dans la liste. On ouvre ainsi la porte à ce que la fonction middleOf soit utilisée à mauvais escient. Heureusement, il est possible de définir des contraintes et borner le type de paramètres supportés par la fonction.

```
def middleOf[T <: Option[Int]](a: Array[T]): Int = a(a.length /2).getOrElse(-1)
val t = middleOf(Array(Some(1), Some(6), Some(3)))
//res: 6
val t = middleOf(Array(Some(1), None, Some(3)))
//res: -1
```

Ici, on a déclaré que le type T doit être un sous-type à Option[Int]. Ainsi, il n'est désormais possible de passer que des objets héritant d'optionnels d'entiers. L'avantage c'est qu'on peut ainsi mettre plus de logique spécifique aux entiers et aux optionnels dans l'implémentation de la méthode middleOf comme l'illustre l'appel de la fonction getOrElse. L'opérateur >: permet de réaliser la déclaration inverse, en l'occurrence T >: R reviendrait à contraindre le type R à être un sous-type de T.

Mais on peut aller plus loin avec les principes de covariance C[+T] et de contre-variance C[-T]. En déclarant un paramètre avec le signe +, on indique au compilateur que le type C suit les mêmes règles hiérarchiques que le type T. Ainsi, si T est un sous-type de P alors C[T] est un sous-type de C[P]. On pourra alors passer à une méthode déclarant recevoir un paramètre de type C[P] avec un paramètre de type C[T]. L'effet inverse est obtenu avec le signe - créant ainsi ce qu'on appelle une possibilité de contre-variance.

Ceci donne un premier tour d'horizon des capacités d'expression sur types génériques. Pour aller plus loin, il est intéressant pour les curieux de jeter un œil sur les contraintes de types (T =:= U, T <:< U, T <%< U), la conversion implicite via l'opérateur <% ou encore le recours au principe de 'Context Bounds'.

Au-delà de ces aspects qui montrent quelques forces du langage, ce dernier vient avec d'autres atouts qui contribuent à en faire une technologie d'entreprise.

Pour en lister quelques-uns :

ScalaTest : une librairie de tests unitaires puissante qui permet de tester tant du code Scala, Java ou encore Javascript (Scala.js) ;

Sbt : qui signifie Scala Built Tool, un outil de build interactif et open-source. Il peut aussi être utilisé pour construire des projets Java ;

L'interopérabilité : il est possible d'exécuter du code Java dans du code Scala et vice-versa. L'interopérabilité avec Javascript est possible via Scala.js ;

Un large écosystème de frameworks : PlayFramework (web), Akka (calcul distribué), Breeze (data science / machine learning), Spark (big data streaming), Lagom (microservices), Spray (api rest) , Slick (dsl base de données), ...

Dotty : c'est les prequels de la version 3 de Scala qui va permettre une meilleure gestion de la rétro-compatibilité, l'extension de méthodes, le paramétrage de traits et bien plus encore...

Toutefois, bien que Scala vienne avec de nombreux avantages, certains peuvent se transformer en faiblesse. Pour tirer bon parti de ce langage, il est nécessaire d'être vigilant pour ne pas tomber dans certains pièges : rédaction de code illisible en abusant d'expressions implicites ou encore en créant des opérateurs obscurs que l'on peut retrouver dans certaines librairies : <*> starship operator, >=> fish operator. Un autre point de vigilance, c'est la rétrocompatibilité. Contrairement à Java, Scala n'est pas retro-compatible. Et suivant la version de Scala, il faudra aligner une JDK adaptée. Toutefois, la dernière version en date 2.13.2, peut s'exécuter sur une JDK 8. 1

Scala est un langage précurseur qui donne le **La** dans le monde de la JVM depuis plusieurs années. Il est de plus en plus privilégié dans l'industrie logicielle pour le traitement de flux de données et répondre aux problématiques de haute réactivité. Un langage à suivre avec intérêt ! Ce qu'on peut souligner c'est que les récentes releases de Java intègrent de plus en plus des patrons de programmation et des fonctionnalités initialement introduites au niveau de la JVM par Scala. Par exemple, le pattern matching est désormais supporté dans la version 14 de Java. Si ce n'est pas encore fait et que voulez faire un saut vers le futur, alors n'attendez plus, essayez Scala !

*spark (https://en.wikipedia.org/wiki/Apache_Spark)

**akka (https://en.wikipedia.org/wiki/Akka_(toolkit))

***conférence : (https://www.youtube.com/watch?v=3jg1AheF4n0)

****reactive manifesto: https://www.reactivemanifesto.org/

ET BIEN PLUS ENCORE

Q : Finalement, Scala est-il un bon compromis fonctionnel / OO par rapport à du pur C++ / Java / C# ou par rapport à du fonctionnel pur comme Haskell? Peut-on le comparer à F# côté .Net ?

R : Oui c'est un excellent compromis fonctionnel / OO, en gros le meilleurs des deux mondes. F# et Scala ne sont pas vraiment comparables, ce n'est pas le même paradigme de programmation. F# ne supporte pas la programmation orientée objet à proprement parler. Alors que Scala est à la fois un langage fonctionnel et orienté objet, combinant ainsi le meilleur des deux paradigmes.

Q : Y a-t-il des choses que l'on fait en Scala et pas en Java?

R : Oui beaucoup, par exemple la surcharge d'opérateur, l'utilisation d'arguments wild card (_) ou encore la réalisation d'opérations implicites.

Q : Comment se passe la gestion des éléments obsolètes? Quid de la migration d'un code scala ancien vers le nouveau?

R : Ce ne sera qu'à partir de la version 3, avec TASTy, qu'il y aura un début de backward compatibility entre les versions futures de Scala. Autrement, à ce jour, la backward compatibility n'est supportée que sur les releases mineures (2.x ou 1.x), ainsi il n'est pas de facto possible d'exécuter un code écrit en Scala 1.x sur un runtime Scala 2.x.

Q : Peut-on mixer les langages, wrapper du Scala dans Java par exemple ?

R : Oui c'est possible, mais c'est plus naturel dans le sens inverse, et ainsi intégrer du code Java dans un programme écrit en Scala. Par ailleurs, Scala est aussi bien interopérable avec du Javascript.

Q : Tous les IDE / outils supportent-ils Scala ?

R : Intellij et ScalaIDE basé sur Eclipse sont les principaux IDE Scala.

1 Version compatibility table

JDK version	Minimum Scala versions	Recommended Scala versions
13, 14	2.13.2, 2.12.11	2.13.2, 2.12.11
12	2.13.1, 2.12.9	2.13.2, 2.12.11
11	2.13.0, 2.12.4, 2.11.12	2.13.2, 2.12.11, 2.11.12
8	2.13.0, 2.12.0, 2.11.0, 2.10.2	2.13.2, 2.12.11, 2.11.12, 2.10.7
6, 7	2.11.0, 2.10.0	2.11.12, 2.10.7

Stéphane Philippart
Architecte logiciel et DevOps chez
Harmonie Mutuelle

Pipeline as code avec Jenkins

En ces périodes de DevOps à outrance, il apparaît une chose commune à toutes celles et ceux qui s'y essaient : la nécessité d'automatiser et de fluidifier le processus de build, de livraison et de déploiement. C'est là qu'interviennent la construction continue (Continuous Integration), la livraison continue (Continuous Delivery) et le déploiement continu (Continuous Deployment). En résumé ce que l'on appelle le CI / CD.

1

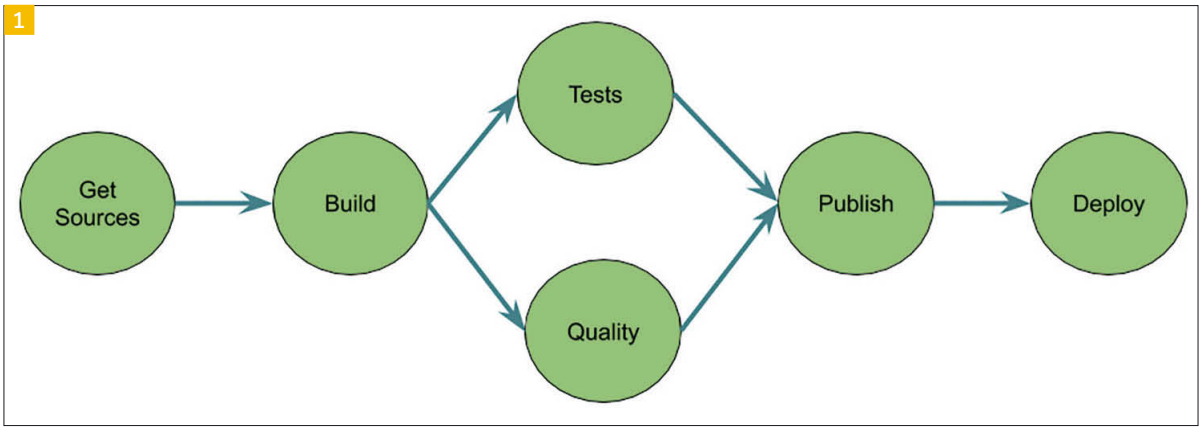


Figure 1: Un pipeline classique de construction d'application Java

On croise de plus en plus la notion de *Pipelines* (comment concevoir et représenter les processus précédemment cités) qui ne sont au final qu'un enchaînement de tâches séquentielles ou parallèles permettant d'atteindre un but, à savoir la construction d'un livrable et son déploiement de manière la plus automatisée possible. 1

Le but de cet article va être de parcourir ces notions afin de comprendre (un peu) à quoi cela sert mais surtout comment bien le faire.

Outillages

On se doute bien que l'on peut construire tout cela manuellement, en script bash par exemple, mais que cela risque de vite prendre du temps et ne pas être très simple à maintenir ! Il existe pléthore d'outils dédiés plus ou moins à cela : le plus ou moins venant du fait que certains outils, dont ce n'est pas le cœur de métier historique, finissent par le proposer afin de permettre aux utilisateurs de bénéficier d'une « suite » logicielle complète pour la fabrication de leur application (entre nous cela permet aussi de garder l'utilisateur bien au chaud dans son giron sans qu'il aille voir ailleurs).

Citons quelques-uns de ces outils :

- Jenkins: https://jenkins.io/
- Travis CI: https://travis-ci.org/
- Circle CI: https://circleci.com
- GitLab CI: https://docs.gitlab.com/ee/ci/
- GitHub Actions : https://github.com/features/actions
- ...

Il y en a plein d'autres, mais l'idée est juste d'illustrer que ce n'est pas le choix qui manque ! Les deux derniers (GitLab et GitHub) sont un peu particuliers car ce ne sont pas *que* des outils de CI/CD mais bien plus : je vous laisse aller voir cela par vos propres moyens. Intéressons-nous à Jenkins, sujet de cet article.

Jenkins

Deux versions existent, même si la version 1 n'est plus beaucoup utilisée (à part dans du legacy), c'est avec elle que beaucoup ont commencé (dont moi). Je ne vais pas m'appesantir dessus car ce n'est pas cette version que nous allons utiliser.

La version 2 de Jenkins est celle qui nous intéresse, non pas que les autres soient mauvaises (j'utilise à titre personnel d'autres outils gratuits), mais je ne les utilise pas au quotidien dans mon travail.

Dans sa première mouture Jenkins permettait de définir un *job* qui était un ensemble de *stages* : tâche Maven, appel Sonar, Bash... Toute la configuration se faisait par le biais de l'interface graphique. Pratique, mais pas très industrialisable ni facile à généraliser : une modification transverse et c'était sur l'ensemble des jobs qu'il fallait repasser. Jenkins 2, sorti en 2016, a apporté une grosse nouveauté : la généralisation du Pipeline As Code pour définir ses jobs. L'autre grosse nouveauté c'est que l'on peut désormais « embarquer » le pipeline directement dans l'arborescence projet sans devoir créer le job dans l'interface de Jenkins : le pipeline se définit par du code dynamiquement chargé et exécuté par le moteur de Jenkins.

Pipeline As Code

Jenkins est écrit en Java et permet d'exécuter des pipelines écrits en Groovy (je pense qu'il serait possible de le faire en Java aussi mais ce n'est clairement pas comme cela que ça a été pensé).

Pour développer son pipeline[1] il y a deux possibilités : *Scripted Pipeline[2]* ou *Declarative Pipeline[3]*. Les deux sont basées sur Groovy et le DSL (Domain Specific Language) proposé par Jenkins. Le *Scripted Pipeline* est la première façon qui a vu le jour pour développer ses pipelines, le *Declarative Pipeline* est plus récent et permet, principalement, de simplifier le développement des pipelines en n'utilisant que du DSL qui fait plus penser à de la config as code.

Au final nous allons essentiellement conserver la philosophie des *Declaratives Pipelines* : ne pas mettre de code Groovy dans nos Jenkinsfiles, réserver cela à des classes utilitaires ou la définition de nos propres steps.

En avant pour le développement de notre pipeline !

Jenkinsfile

Ça y est, on a les notions de base. Maintenant il faut nous lancer et pour cela il va falloir que l'on code notre premier pipeline : par convention on code le pipeline dans un fichier se nommant *Jenkinsfile* mais au final, peu importe le nom tant que c'est un script groovy !
Un *Jenkinsfile* en mode *Declarative pipeline* :

```
pipeline {
  agent any

  stages {
    // Première étape de mon pipeline
    stage('Build') {
      steps {
        // Appel à des steps Jenkins
        echo 'Building..'
        sh 'mvn clean package'
      }
    }
    // Deuxième étape de mon pipeline
    stage('Test') {
      steps {
        echo 'Testing..'
      }
    }
    // Troisième étape de mon pipeline
    stage('Deploy') {
      steps {
        echo 'Deploying....'
      }
    }
  }
}
```

L'idée n'est pas d'expliquer dans le détail comment développer un pipeline (cela viendra dans la suite de l'article) mais simplement d'illustrer comment cela se déroule.
Un pipeline est une suite d'étapes (*stages*) qui comportent plus au moins de sous-étapes (*steps*), elles-mêmes pouvant regrouper plusieurs commandes (par exemple *echo*).
En standard Jenkins fournit déjà beaucoup de steps[4]. Les différents plugins que l'on rajoute dans Jenkins mettent aussi souvent des steps à disposition. Malgré tout, il est possible que dans le contexte d'une entreprise, il soit nécessaire de faire une action particulière ou d'utiliser un outil qui ne possède pas de plugin Jenkins.
De manière assez naturelle, on a tendance à utiliser la step *sh* qui permet d'exécuter n'importe quelle commande bash (y compris d'appeler un script) comme on l'aurait fait avec Jenkins 1. Cela fonctionne mais je trouve que l'on tombe dans le travers boîte noire, car une fois que l'on passe la main à un script, il est plus compliqué d'interagir avec les éléments qui le composent et surtout cela fait deux référentiels de code à maintenir pour notre pipeline (le *Jenkinsfile* et le script bash).
C'est pourquoi on va devoir faire nos propres « steps » pour répondre à des problématiques d'entreprise et sortir du pipeline *hello world*, (au final ce ne sont pas des vraies steps mais ils s'utilisent de la même façon). Ceci en codant du début à la fin notre pipeline dans une librairie afin de n'avoir à coder que le minimum dans le *Jenkinsfile*.
Le gain est double :

- On factorise du code et donc les modifications sont plus simples ;
- On contextualise à notre société ce qui doit l'être.

Cette librairie s'appelle dans Jenkins une *SharedLib*.

Les sharedlib

Comme je l'ai déjà indiqué, une fois que l'on a commencé à coder des pipelines dans nos *Jenkinsfiles*, on a fait un grand pas par rapport à l'approche *click to config* de Jenkins 1. Mais on se retrouve avec le même problème : toutes les actions identiques entre les projets sont dupliquées et lors d'un changement, par exemple de plugin, on se retrouve à faire une maintenance sur tous les *Jenkinsfiles*.
Il faut donc pouvoir factoriser le code pour pouvoir le réutiliser, c'est là qu'interviennent les sharedlib[5]: une lib qui regroupe des classes / scripts réutilisables dans les *Jenkinsfiles*.
Imaginons que nous voulions reprendre l'exemple précédent sans utiliser la step *sh* mais un développement maison permettant d'appeler du Maven.
Pour cela il va falloir développer une classe utilitaire se chargeant d'appeler des commandes Maven. Le code de la classe *Utilities* :

```
/**
* Classe utilitaire pour un pipeline Jenkins.
*/
class Utilities implements Serializable{
  // Contexte Jenkins, permet d'accéder aux steps fournies par Jenkins
  Script steps

  Utilities(Script steps) {
    this.steps = steps
  }

  /**
  * Simple méthode pour exécuter des commandes Maven.
  * @param args Maven arguments.
  */
  void mvn(String args) {
    // Appel de la commande Maven grâce à la step sh de Jenkins
    steps.sh "mvn ${args}"
  }
}
```

En dehors de l'utilité même de cette classe utilitaire ce qu'il faut retenir :

- La classe implémente *Serializable* : c'est une obligation pour une classe utilisée dans un pipeline Jenkins car celui-ci doit pouvoir « sauvegarder » à tous moments l'état du pipeline. C'est notamment induit par le fait que Jenkins utilise le CPS (Continuation Passing Style). Il s'agit d'un principe assez complexe et obscur mais il faut être au courant que ça existe car il n'est pas rare d'avoir une exception du genre : *java.io.NotSerializableException*,

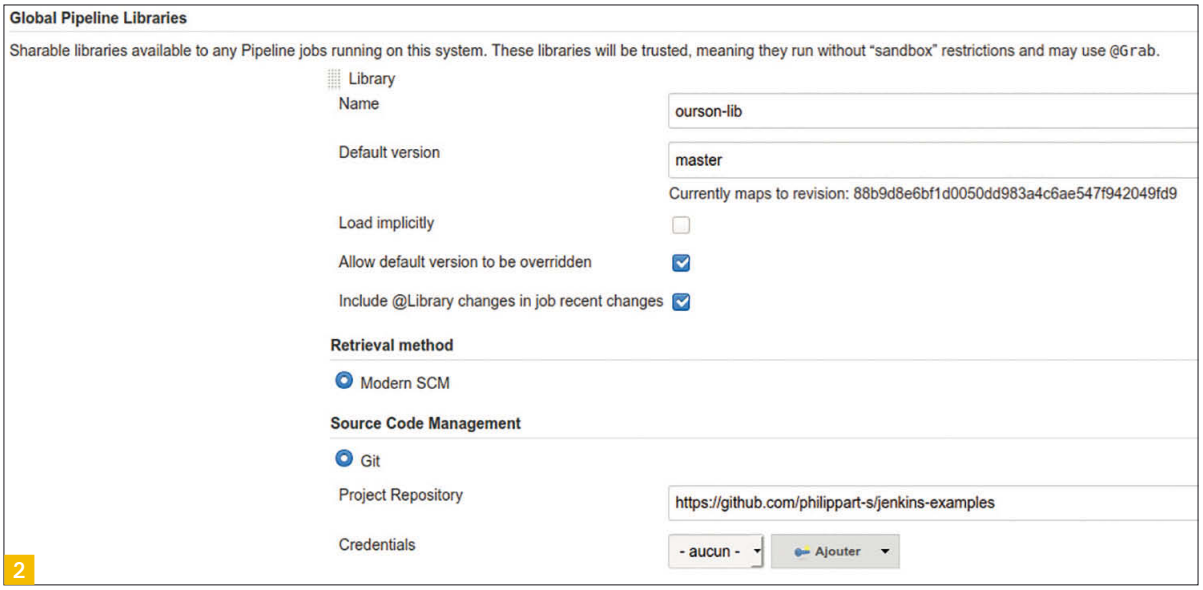


Figure 2: Référencement d'une sharedlib dans Jenkins

et, dans ce cas, la doc de Jenkins a une section dédiée[6] au fonctionnement du CPS et comment s'en sortir dans le code.

- La classe possède un constructeur avec un paramètre (habituellement appelé *steps*) permettant de passer le contexte Jenkins et notamment permettant l'utilisation des steps déjà présentes dans Jenkins
- C'est donc grâce à cela que l'on peut utiliser la step *sh* proposée par Jenkins dans notre code.

Il ne reste « plus » qu'à mettre à disposition la classe pour n'importe quel pipeline défini dans Jenkins. Cela se fait simplement en mettant le source dans un repository Git ; il faut que le projet respecte la bonne arborescence[7] (la classe doit se trouver dans un répertoire *src*).

Ensuite il faut la déclarer dans Jenkins en indiquant l'endroit où elle se trouve : 2

Ensuite il ne reste plus qu'à la déclarer grâce à l'instruction *@Library* dans le code du *JenkinsFile*.

Le code de notre pipeline devient donc :

```
// Accès à la librairie déclaré dans la configuration de Jenkins
@Library('ourson-lib') _

// Import pour pouvoir accéder à la classe utilitaire
import fr.ourson.utils.Utilities

// Le this permet de passer à la classe utilitaire le contexte de Jenkins
def utils = new Utilities(this)

pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        echo 'Building..'
        script {
          // Utilisation de la classe utilitaire, nécessite une
          // balise script car c'est du code!!
          utils.mvn 'clean package'
        }
      }
    }
    stage('Test') {
      steps {
        echo 'Testing..'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying....'
      }
    }
  }
}
```

Quelques explications :

- La sharedlib se charge grâce à *@Library('ourson-lib') _*, le nom étant celui donné lors de la déclaration dans Jenkins ;
- Il est nécessaire dans notre exemple d'utiliser la step *script* car nous essayons d'exécuter du script (en fait du code groovy) dans un pipeline de type *Declarative Pipeline*. Pour éviter cela, il faut utiliser une step prédéfinie ou en faire une nous-même (on va voir que l'on arrive à quelque chose de similaire un peu plus loin).

Comment simplifier l'utilisation de notre shared lib ?

Passer en mode Scripted Pipeline

C'est moins joli mais c'est plus simple d'appeler directement du code exposé par une sharedlib :

Code complet sur programmez.com & github

Cela ressemble beaucoup à notre code précédent donc la plus-value est plutôt faible.

Passer par la définition de steps custom

L'idée d'une custom step[8] est d'avoir la facilité d'écriture d'un *Declarative Pipeline* sans le *boilerplate code* qui est nécessaire pour appeler du code groovy.

Pour cela il va falloir rajouter un script (et non une classe) dans le répertoire *vars* de notre projet sharedlib. Les scripts dans ce répertoire ne contiennent qu'une seule méthode (*call*) et accessible

comme une step. Le nom du script est important car cela va devenir le nom de la step dans le pipeline, dans notre exemple le script se nomme *myMaven.groovy*, la step ne nommera donc **myMaven** !
Le code de notre script :
Code complet sur programmez.com & github

Et enfin le code de notre pipeline :
Code complet sur programmez.com & github

Simple non ?
Cela commence à ressembler à quelque chose de sympathique, plus d'import, et une utilisation qui ressemble à une step classique.

Pour aller plus loin

Branches

Ce qui est bien en déportant la sharedlib dans un repository Git, c'est que l'on va pouvoir utiliser le mécanisme de branches pour tester une fonctionnalité sans impacter l'ensemble des pipelines qui dépendent de la sharedlib :
Code complet sur programmez.com & github

Notez le *@feature/test* qui permet de cibler une branche qui s'appelle *feature/test* dans mon repository Git.

Tout le pipeline as code

Jusqu'à présent, les différents exemples de code montrent qu'il faut tout de même définir une partie du pipeline dans le *Jenkinsfile*. Pas grand-chose mais tout de même assez pour que cela apporte quelques problèmes :

- Un changement de paradigme d'entreprise sur les pipelines (par exemple ajout systématique d'un stage dédié à la performance) implique la modification de tous les *Jenkinsfiles* ;
- Il est simple pour un projet d'alléger son pipeline, voire de le rendre instable en le modifiant de manière erronée.

Pour pallier cela, il est possible de pousser encore plus loin le principe de la factorisation de code : concevoir l'ensemble du pipeline dans la sharedlib et ne l'exposer que sous forme de step dans un *Jenkinsfile*.
C'est une sorte de compilation de tout ce qui a été présenté jusqu'ici :

- On reprend la classe utilitaire ;
- On reprend le principe de step maison... mais amélioré !

Le code de la step, stocké dans *myMavenStep.groovy* devient :

```
import fr.ourson.utils.Utilities

/**
* Définition de la step custom permettant de simplifier le Jenkinsfile
* @param config Paramètres à passer à la custom step
*/
def call(Map config) {

  Utilities util = new Utilities(this)

  node() {
    stage('Build') {
      echo 'Building..'
      // Appel à la classe utilitaire en lui passant le paramètre contenu dans la map
      util.mvn config.mvnArgs
    }
    stage('Test') {
      echo 'Testing..'
    }
    stage('Deploy') {
      echo 'Deploying....'
    }
  }
}
```

Quelques explications :

- La map en paramètre permet de passer n'importe quel paramètre nommé à notre step ;
- On voit ici qu'en fait on code un *Scripted Pipeline* ;
- Le paramètre passé dans le *Jenkinsfile* est récupéré dans la map.

Enfin l'utilisation dans notre *Jenkinsfile* :

```
@Library('ourson-lib') _

// Appel directement de la step sans devoir définir un pipeline
myMavenStep mvnArgs: 'clean compile'
```

Plutôt concis non ? :)
Et comme indiqué, le paramètre *mvnArgs* est celui que l'on récupère dans la map de notre script.

Conclusion

J'espère avoir réussi à illustrer ce que sont les pipelines dans Jenkins et comment on peut les développer pour ne plus avoir à le faire via l'interface graphique.
À ce stade je pense que l'on a plutôt réussi ce que l'on voulait non ?

- Avoir du pipeline as code ;
- Avoir du code centralisé pour aider la maintenance ;
- Pouvoir coder des steps maison pour adapter nos pipelines à nos outils.

Je n'ai pas parlé de Blue Ocean[9] qui est un projet fortement poussé par la communauté. Il offre notamment la particularité de permettre de concevoir de manière plus ou moins graphique ses pipelines. Ce n'est pas forcément ce que je recherche, donc je ne l'utilise pas pour cela. Il faut néanmoins noter qu'il offre une visualisation d'exécution beaucoup plus jolie (ça, ce n'est pas compliqué…) et pratique que l'interface standard, notamment dans le cas de pipeline avec des jobs parallèles.
Pour être complet, il ne reste plus qu'à industrialiser ce mode de développement afin de pouvoir dépasser le mode scripting et permettre de développer de manière standard (avec un IDE et une compilation locale). Il faut ensuite ajouter des tests pour ne pas avoir à attendre que le code soit exécuté sur l'instance Jenkins pour vérifier que tout fonctionne (ou pas...), tout ceci viendra dans de futurs articles.

[1] : https://jenkins.io/doc/book/pipeline/
[2] : https://jenkins.io/doc/book/pipeline/syntax/#scripted-pipeline
[3] : https://jenkins.io/doc/book/pipeline/syntax/#declarative-pipeline
[4] : https://jenkins.io/doc/pipeline/steps/
[5] : https://jenkins.io/doc/book/pipeline/shared-libraries/
[6] : https://wiki.jenkins.io/display/JENKINS/Pipeline+CPS+method+mismatches
[7] : https://jenkins.io/doc/book/pipeline/shared-libraries/#directory-structure
[8] : https://jenkins.io/doc/book/pipeline/shared-libraries/#defining-custom-steps
[9] : https://jenkins.io/projects/blueocean/

Laetitia Avrot
Experte PostgreSQL, Lætitia est co-fondatrice du mouvement Postgres Women et ancienne membre du comité du code de conduite de la communauté PostgreSQL. Elle est également trésorière de PostgreSQL Europe. Elle contribue au projet PostgreSQL de différentes manières (patchs, mentor pour Google Code-in Contest, speaker dans des conférences, bénévolat dans les événements…).

La réplication physique avec PostgreSQL en pratique

Dans des articles précédents du magazine Programmez, j'avais développé différentes notions (réplication, reines, princesses, ouvrières…). Dans cet article, nous allons mettre tout ça en pratique avec la création d'un cluster reine-princesse.

Pourquoi pas sous Windows ?

PostgreSQL a été écrit au départ pour Linux. L'adaptation du code Postgres vers Windows 64 bits a été finalisé en version 9.0, c'est-à-dire en septembre 2010. Le mot important de la phrase précédante est "adaptation". De la même manière que je préfère lire un texte dans sa version originale, je conseille d'utiliser Postgres sous Linux car il a été écrit pour Linux. Quelle que soit la qualité du portage, cela ne reste qu'un portage.

De plus, plus de 99% des bases de données Postgres tournent sur un OS Linux. La probabilité de rencontrer un bug inconnu est donc supérieure sous Windows. En utilisant Linux, vous avez moins de risque d'être dans une configuration particulière.

Enfin, la plupart des administrateurs de bases de données sont plus à l'aise sur des machines Linux que Windows car le métier d'administrateur de bases de données nécessite des connaissances techniques sur les systèmes d'exploitation et pourquoi s'embêter à monter en compétences sur un OS utilisé dans moins de 1% des cas ?

Reines, princesses, ouvrières

Dans le monde de Postgres, on n'utilise pas les termes maître/esclave qui sont des termes peu flatteurs pour l'humanité et qui surtout sont une analogie fausse, ce qu'on essaye à tout prix d'éviter en pédagogie.

Une autre dénomination est apparue, basée sur une comparaison avec le monde des abeilles. Un abeille peut avoir trois rôles :

- Reine ;
- Princesse ;
- Ouvrière.

La reine est le nœud primaire, celui qui sert les requêtes de lecture et d'écriture. La princesse n'est là que pour prendre la place de la reine si la reine meurt. Elle ne sert pas de requête. L'ouvrière sert les requêtes de lecture uniquement.

Debian-based ou Red Hat based ?

La communauté propose différents packages. Les plus utilisés sont ceux pour les distributions basées sur Debian et ceux pour les distributions basées sur Red Hat. Nous allons fournir les étapes pour les deux types de systèmes.

Je vais utiliser une Ubuntu Server Bionic Beaver et une Centos 7 pour mes démonstrations. Elles auront lieu sur des machines virtuelles créées par Vagrant dont voici le Vagrantfile :

```
HOSTS = [
 { :type => "queen", :name => "amidala", :ip_private => "10.0.0.2", :ip_public =>
 "192.168.0.2", :ssh_port => "2221" },
 { :type => "princess", :name => "leia", :ip_private => "10.0.0.3", :ip_public =>
 "192.168.0.3", :ssh_port => "2222" }
]

Vagrant.configure("2") do |config|
 HOSTS.each do |host|
  config.vm.box = "ubuntu/bionic64"
  config.vm.define host[:name] do |machine|
   machine.vm.hostname = host[:name]
   machine.vm.network "private_network", ip: host[:ip_private]
   machine.vm.network "public_network", ip: host[:ip_public]
  end
 end
end
```

Pour Centos 7, il suffit de remplacer "ubuntu/bionic64" par "centos/7". Vous pouvez adapter les IP publiques et privées pour correspondre à votre propre réseau.

Toutes les commandes seront données pour l'utilisateur root. Si un autre utilisateur doit être utilisé, cela sera précisé.

Installer le repository du PGDG

Même si les distributions Linux proposent des versions de Postgres, je conseille toujours de récupérer celles du dépôt officiel du projet PostgreSQL. Cette étape est à effectuer sur la reine et sur la princesse.

Ubuntu

Sous Ubuntu, cela donnera :

```
echo "deb http://apt.postgresql.org/pub/repos/apt/ bionic-pgdg main" > /etc/apt/sources.list.d/pgdg.list
```

Puis, on va récupérer la clé du dépôt et mettre à jour la liste des dépôts :

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
apt-get update
```

Vous pouvez maintenant installer le serveur PostgreSQL (la version 12 est la dernière version majeure) :

```
apt-get install postgresql-12 -y
```

Les paquets Debian créent automatiquement un cluster main que nous allons utiliser, l'étape suivante n'est donc valable que pour Centos.

CentOS

Comme sous Ubuntu, on va commencer par installer le repository du PGDG (PostGreSQL Developer Group):

```
yum install https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
yum install postgresql12
yum install postgresql12-server
```

Le paquetage postgresql12 est le paquetage client de PostgreSQL. Le paquetage postgresql12-server contient tout ce qui est nécessaire pour faire tourner une instance Postgres.

Créer la reine

Nous allons procéder aux différentes étapes très simples permettant de créer la reine. Conformément à ce qui est indiqué dans le Vagrantfile, ce serveur sera nommé "amidala".

Initialiser un cluster

Cette étape est faite automatiquement si vous êtes sous une distributions basée sur Debian. Il n'est donc pas nécessaire d'effectuer cette étape si vous êtes sous Debian ou un de ses amis.
Sous CentOS, il faut créer l'instance ainsi :

```
/usr/pgsql-12/bin/postgresql-12-setup initdb
```

Configurer l'instance

Sous Ubuntu, vcous trouverez les fichiers de configurations sous /etc/postgresql/<version>/<cluster>/. Dans notre cas, on trouvera donc le fichier postgresql.conf sous /etc/postgresql/12/main/.
Sous CentOS, vous trouverez les fichiers de configuration dans le répertoire du cluster. Le cluster est créé par défaut dans le répertoire /var/lib/pgsql/12/data.
Par défaut, Postgres ne permet que des connexions locales à l'instance Postgres, comme nous souhaitons y accéder depuis l'extérieur, il va falloir autoriser les connexions sur toutes les interfaces réseau de la machine et non pas juste en local.

```
listen_addresses='*'
```

Il va falloir aussi modifier les paramètres d'archivage. Ce n'est pas forcément necessaire, mais c'est fortement recommandé. C'est une sécurité qui permet de s'assurer de la disponibilité des WALs ce qui peut permettre d'éviter de refaire un backup ou de reconstruire totalement une princesse.

```
archive_mode=on
archive_command='...'
```

L'archive_command peut être n'importe quelle commande qui copie les fichiers. Il est préférable d'utiliser la commande de votre outil de sauvegarde (si possible) ou sinon d'utiliser la commande rsync plutôt que scp ou cp qui ne sont pas atomiques et pourraient entraîner une corruption des logs archivés.
Assurez-vous que le répertoire de destination existe et que le user postgres ait les droits d'écriture dessus.

Voici un exemple d'archive_command :

```
archive_command = 'rsync %p /backups/main/%f'
```

Ensuite, il va falloir permettre à la princesse de se connecter à la reine. Pour ce faire, nous allons modifier le fichier pg_hba.conf (dans le même repertoire que postgresql.conf). Il suffit de rajouter cette ligne en fin de section sur la réplication :

```
host    replication    repuser    192.168.0.3/32    md5
```

(Le chiffrement scram-sha-256 serait meilleur, mais ce n'est pas le sujet de cet article.)

(Re)Démarrer le cluster

Si vous êtes sous Ubuntu, l'instance a certainement été déjà démarrée après sa création lors de l'installation des paquets. Il faut donc plutôt la redémarrer :

```
pg_ctlcluster 12 main restart
```

Sous CentOS, il va falloir utiliser Systemd pour activer le service au démarrage et démarrer l'instance :

```
systemctl enable postgresql-12
systemctl start postgresql-12
```

Créer l'utilisateur bdd repuser

Nous allons créer l'utilisateur repuser. Le plus simple pour ce faire est de se connecter à postgres avec psql :

```
sudo -u postgres psql
postgres=# create user repuser with replication;
CREATE ROLE
postgres=# \password repuser
Enter new password:
Enter it again:
postgres=# \q
```

Créer la princesse

Nous allons créer la princesse en clonant la reine avec l'outil pg_basebackup intégré à Postgres. L'option -R (pour réplication) de l'outil permet de prévenir que le backup sera utilisé pour de la réplication.
Normalement, vous avez déjà installé PostgreSQL sur cette machine (voir l'étape 1).

Nettoyage (Debian-based uniquement)

Il va falloir nettoyer d'abord le répertoire /var/lib/postgresql/12/main qui comporte une instance toute neuve, mais inutile.

```
pg_ctlcluster 12 main stop
rm -rf /var/lib/postgresql/12/main/*
```

Fichier de mot de passe

Puis, nous allons créer un fichier de mot de passe pour éviter qu'un mot de passe soit demandé systématiquement.

```
su - postgres
echo "192.168.0.2:5432:*:repuser:password" > $HOME/.pgpass
chmod 0600 $HOME/.pgpass
```

Il est possible de tester une connexion de type réplication ainsi :

```
su - postgres
psql "replication=true" -c "IDENTIFY_SYSTEM;" -U repuser -h 192.168.0.2
```

Préparer le répertoire (CentOS uniquement)

Nous allons créer le répertoire pour pouvoir y cloner l'instance.

```
mkdir -p /var/lib/pgsql/12/data
chown postgres:postgres /var/lib/pgsql/12/data
```

Cloner la reine sous Ubuntu

C'est le moment de forger notre belle commande pg_basebackup pour cloner notre instance reine et créer une princesse. Sous CentOS, n'oubliez pas de remplacer le répertoire de destination par /var/lib/pgsql/12/data.

```
su - postgres
pg_basebackup -h 192.168.0.2 -U repuser -R -P -D /var/lib/postgresql/12/main
```

-R permet d'être en mode réplication -P permet d'afficher une barre de progression -D permet de fournir le répertoire de destination

Recopie des fichiers de configuration (Ubuntu seulement)

Comme sous Ubuntu les fichiers de configuration ne sont pas dans le répertoire du cluster, ils ne sont pas sauvegardés en même temps que le cluster. On pourrait recopier le fichier de la reine (recommandé), mais comme nous n'avions modifié que 3 paramètres sur la reine, nous allons les reprendre sur la princesse.
Les fichiers de configuration sont sous /etc/postgresql/12/main. Voici les paramètres à modifier sur postgresql.conf:

```
listen_addresses='*'
archive_mode=on
archive_command='...'
```

Configuration de la réplication

Si on regarde le répertoire /var/lib/postgresql/12/main, on remarque qu'en plus des fichiers existants, pg_basebackup a créé deux fichiers: standby.signal et postgresql.auto.conf. Le fichier standby.signal permet de dire à l'instance qu'elle sera une princesse et non une reine. Le fichier postgresql.auto.conf comporte le paramètre primary_conninfo qui permet de definir comment la princesse va se connecter à la reine.

```
cat /var/lib/postgresql/12/main/postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo = 'user=repuser passfile=''/var/lib/postgresql/.pgpass'' host=192.
168.0.2 port=5432 sslmode=prefer sslcompression=0 gssencmode=prefer krbsrvname
=postgres target_session_attrs=any'
```

Il ne reste plus qu'à ajouter deux paramètres dans postgresql.conf

```
promote_trigger_file = '/tmp/mastergone'
restore_command= 'rsync /backups/main/%f %p'
```

Démarrer la princesse

Il n'y a plus qu'à démarrer la princesse et s'assurer que la réplication fonctionne.
Voici la commande Ubuntu pour démarrer :

```
pg_ctlcluster 12 main start
```

Et la commande CentOS :

```
systemctl start postgresql-12
```

Il ne reste plus qu'à tester que notre princesse est bien une princesse :

```
sudo -u postgres psql
postgres=# select pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 t
(1 row)
```

Pour finaliser la configuration, je conseille de recopier les paramètres de réplication sur la reine et de modifier pg_hba.conf sur la princesse pour permettre une connexion depuis la reine. Ainsi la comfiguration sera prête en cas de bascule.

Maxime Ellerbach
16 ans, lycéen
https://github.com/Maximellerbach

Courses virtuelles de voitures autonomes sur simulateur !

A cause du Covid19, les courses physiques de voitures autonomes auxquelles je participais régulièrement ont toutes été annulées. Un coureur aux Etats Unis a donc remis au goût du jour un simulateur pour organiser des courses même à distance !

Un peu plus sur le simulateur

Le simulateur a été créé par Tawn Kramer (https://github.com/tawnkramer/sdsandbox) à partir de Unity (https://unity.com/), une plateforme de développement de jeu 2D ou 3D. Il avait déjà développé le simulateur bien avant la Covid19, mais s'est remis à son développement pour qu'il puisse accueillir des courses virtuelles. Ces courses sont organisées par DIY Robocar (https://www.meetup.com/fr-FR/DIYRobocars/events/270833511/; https://diyrobocars.com/).

Unity est programmable en C# en utilisant UnityEngine. Les scripts peuvent donner vies aux objets, changer leur apparence ou bien automatiser des tâches comme la création d'un circuit !

Comment fonctionne le simulateur ?

Le simulateur est construit pour être un serveur auquel les concurrents se connectent. L'intelligence artificielle qui pilote la voiture est donc désolidarisée du simulateur et peut tourner sur un ordinateur distant.

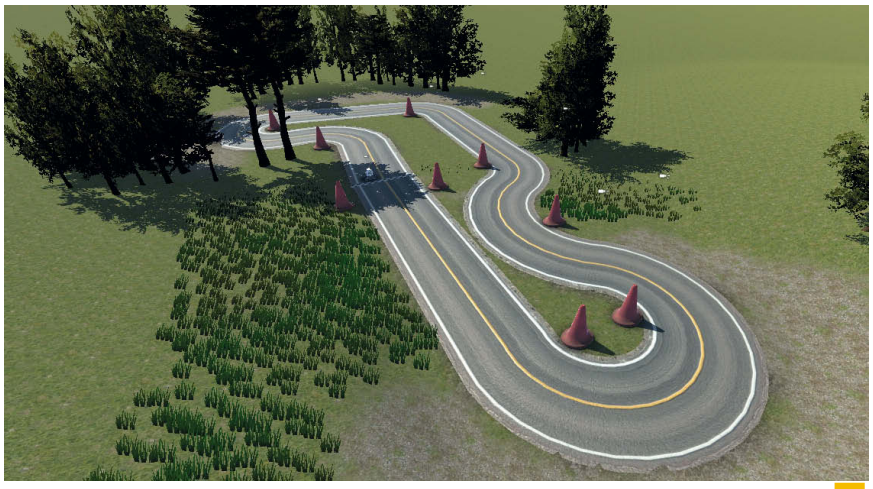
Le serveur envoie à une fréquence de 20hz à chaque client les informations suivantes : vitesse, position XYZ dans l'environnement, image de la caméra placée sur la voiture, la distance par rapport au centre de la route et un booléen si la voiture a touché une entité (autres voitures ou murs).

Un exemple de paquet reçu : {'msg_type': 'telemetry', 'steering_angle': 0.007519087, 'throttle': 0.9546723, 'speed': 9.646547, 'hit': 'none', 'pos_x': 51.35168, 'pos_y': 5.306193, 'pos_z': 24.27408, 'time': 110.0058, 'cte': 2.487985}

Ces informations permettent d'implémenter à peu près n'importe quel algorithme, que ce soit prise de décision à partir de l'image (mon approche), apprentissage par renforcement ou bien planification de trajectoire.

Le client doit renvoyer des paquets contenant l'angle de direction des roues, la proportion de la pédale d'accélération et de freinage.

Un exemple de paquet renvoyé : {'msg_type': 'control', 'steering': '0.3497206935886237', 'throttle': '0.6509060152362044', 'brake': '0.0'}



1

Qu'est-ce que ça change ?

Premièrement, avec le simulateur, toutes les voitures ont les mêmes propriétés, seul l'algorithme utilisé compte ! Plus de problèmes de batteries, de faux contacts ou autres ! Un facteur un peu moins cool : le délai avec le serveur ! En effet, les courses étant hébergées aux Etats Unis, on doit s'attendre à avoir un peu de délai si l'on n'a pas d'instance hébergée pas loin du serveur. Pour mon cas c'est 170ms, ça peut être un réel handicap si l'on ne prévoit pas de compenser ce délai !

Pour ce qui est de la récupération et labellisation d'image, rien ne change à part la provenance de l'image. Il est notamment possible de personnaliser sa caméra (FOV, dimension, inclinaison, le type d'encodage, distorsion...) en envoyant un paquet de configuration. Par exemple : {"msg_type" : "cam_config", "fov" : "70", "fish_eye_x" : "0.0", "fish_eye_y" : "0.0", "img_w" : "255", "img_h" : "255", "img_d" : "3", "img_enc" : "JPG", "offset_x" : "0.0", "offset_y" : "1.7", "offset_z" : "1.0", "rot_x" : "40.0"}

Ça nous permet donc d'avoir une flexibilité pour convenir au mieux aux participants et se rapprocher le plus de la caméra à laquelle ils sont habitués.

Côté code, j'ai dû réécrire ma boucle de prédiction pour recevoir les paquets et en renvoyer au serveur. Sinon aucun changement du côté de mes modèles et de leurs entraînements ! A noter que virtuellement, il est plus simple de récupérer des données n'ayant pas de contraintes de batteries et aucun risque de casser la voiture !

Et le dernier point, c'est quand même la bonne nouvelle. Et c'est aussi ce qui est intéressant… Que ce soit environnement réel ou virtuel, au final, il n'y a pas tant de différence et passer de l'un à l'autre n'est pas si compliqué.

Une fois que l'on a bien pris connaissance du fonctionnement du simulateur, on peut en étudier un peu plus en profondeur le code, bien évidemment crayon à la main pour noter quelques idées.

Création de circuit automatisé

A première vue, l'objectif de créer un circuit automatiquement me paraissait compliqué ! J'ai donc d'abord fait un circuit à la main et je me suis alors armé du logiciel Unity. Sans trop essayer de comprendre l'utilité de chaque objet ou contrôleur, après quelques heures mon circuit était fini ! 1

Pour moi, n'ayant jamais fait de Unity, c'était un grand accomplissement !

Pendant que je construisais le circuit, je me suis rendu compte que les seules choses qui changeaient d'un circuit à l'autre étaient : la route et le décor. Tous les contrôleurs et interfaces peuvent rester inchangés d'un circuit à l'autre.

Afin de créer une route dans le simulateur, il existe une fonction qui transforme une suite de points X, Y, Z en maille (mesh en anglais). Une texture de route est ensuite appliquée sur cette maille.

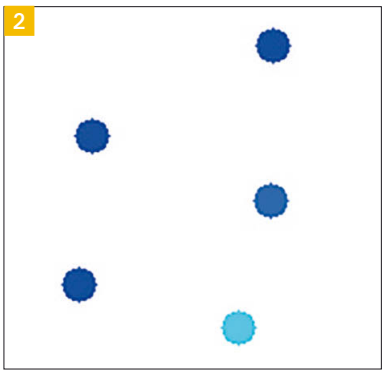
J'ai donc exploité cette fonctionnalité pour créer une route à partir de coordonnées récoltées en conduisant la voiture, et ça marchait ! J'avais un circuit fonctionnel !

Le problème avec cette méthode c'est que je devais quand même conduire la voiture et puis... Il manquait de volume ! Le monde étant plat, à moins de créer des collines, le circuit créé à partir de cette conduite était plat.

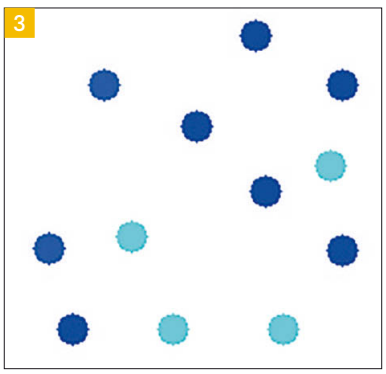
Je me suis alors mis en quête de générer un circuit grâce à de l'aléatoire (et un peu de contraintes bien évidemment). J'utiliserais le langage Python pour profiter du module Scipy qui me sera très utile pour la construction du circuit ! Je sauvegarderai ensuite les points obtenus dans un .txt que chargera le simulateur.

Pour ce faire, je génère une vingtaine de points aléatoirement dont les valeurs sont comprises entre 0 et 200 pour l'axe X et Y ; Entre 0 et 20 pour l'axe de Z. C'est la première contrainte imposée, celle de la taille de l'environnement que je veux. Ensuite, je garde uniquement les points constituant l'extérieur. Cela me donne une première forme, on y enlève les points qui sont trop proches les uns des autres et on obtient quelque chose de ce genre : 2

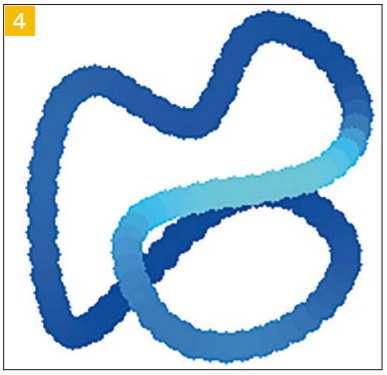
On a réduit le nombre de points d'une vingtaine à seulement 5. Ici, la couleur des points représente la hauteur du point (plus il est clair plus il est haut). A partir de ces 5 points, on rajoute 2/3 points entre



2



3



4

chaque segment entre les points. On va imposer une distance minimale et maximale entre ces points. On va aussi rajouter des contraintes comme l'angle maximal que va former un virage, pour ma part, j'ai choisi 90 dégrés. On peut aussi imaginer une contrainte de pente pour éviter que la hauteur ne varie de trop.
Toutes ces contraintes vont former un processus d'optimisation qui va être répété plusieurs fois jusqu'à ce que les points ne bougent plus.
Voici les points de construction une fois l'optimisation terminée : 3

Ces points vont ensuite être reliés les uns aux grâce à la fonction « scipy.interpolate.splprep » (en savoir plus : https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.splprep.html).

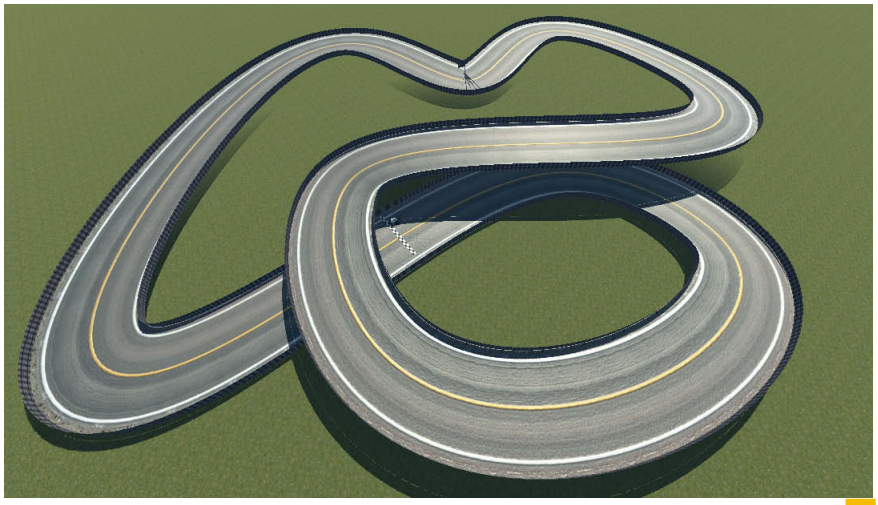
```
tck, u = interpolate.splprep([x, y, z], s=0, per=True)
xs, ys, zs = interpolate.splev(np.linspace(0, 1, N_POINTS), tck)
```

La première ligne calcule une courbe passant par tous les points [x, z, y]. Cette « courbe » est en fait l'association de plusieurs courbes mises bout à bout sous forme d'une fonction.
J'ai utilisé le paramètre « per » de la fonction pour que la courbe soit calculée de manière périodique (le début correspond aussi à la fin).
La deuxième ligne créé un nombre N de points s'inscrivant dans cette courbe en calculant l'image de N valeurs entre 0 (début de la courbe) et 1 (fin de la courbe).
En affichant ces points, on obtient ceci : 4
Il nous reste à choisir l'emplacement de la ligne de départ, sauvegarder nos nouveaux points dans un .txt et charger le circuit dans le simulateur !
En Python, rien de plus simple :

```
circuit_coords = open(save_file, 'w')
for pt in pts:
circuit_coords.write(str(pt[0])+','+str(pt[1])+','+str(pt[-1])+'\n')
circuit_coords.close()
```

On ouvre notre fichier en mode écriture puis on y sauvegarde chaque point tel que l'on doit retrouver un point par ligne avec les éléments x, y et z séparés d'une virgule. Une fois fini, on ferme le fichier.
Côté C#, j'ajoute donc la fonction qui va charger mes nouvelles coordonnées enregistrées en .txt :

```
TextAsset bindata = Resources.Load("track_points") as TextAsset;
string[] lines = bindata.text.Split('\n');

List<Vector3> points = new List<Vector3>();
Vector3 np = Vector3.zero;

foreach (string line in lines)
{
  string[] tokens = line.Split(',');

  if (tokens.Length != 3)
  continue;

  np.x = float.Parse(tokens[0], CultureInfo.InvariantCulture.NumberFormat);
  np.y = float.Parse(tokens[1], CultureInfo.InvariantCulture.NumberFormat);
  np.z = float.Parse(tokens[2], CultureInfo.InvariantCulture.NumberFormat);

  points.Add(np);
}
```



5

On vient charger le fichier « track_points » placé dans le dossier Ressources de Unity. Ensuite, on va parcourir cette liste de string et la diviser en utilisant la fonction « Split » qui va me renvoyer 3 éléments de type string que je convertis ensuite en flottant. Notez qu'utiliser l'invariant de Culture pour les virgules permet de ne pas avoir de problèmes avec les virgules françaises et anglaises qui sont différentes ! Enfin on ajoute à notre liste le point créé à partir de ces 3 éléments extraits. Ces points sont ensuite retournés et utilisés pour la construction des mailles du circuit.
Voici une vue de haut du circuit généré : 5
Pour le moment, je n'ai pas trouvé de solution au problème de génération de décor, peut-être pour une prochaine fois :-) ? A noter aussi que j'ai rajouté des barrières de part et d'autre de la route pour éviter que la voiture ne puisse sortir du circuit et donc tricher !
Vous pouvez retrouver le code pour la génération de circuit à cette adresse : https://github.com/Maximellerbach/Virtual_Racing/blob/master/track_creation/random_points.py
Ainsi que les modifications apportées au simulateur ici : https://github.com/Maximellerbach/sdsandbox

Conclusion

Le simulateur ouvre plein de possibilités tout en restant proche de la réalité. Il permet d'éviter tout problème mécanique et de se concentrer sur la partie programmation. Pour ma part, j'ai pu perfectionner mes modèles en testant beaucoup d'architectures différentes. En moins de 10 minutes je pouvais à la fois entraîner mes réseaux de neurones, et à la fois les tester sur des circuits complexes. Ça peut être un bon moyen de se rendre compte des points faibles et points forts du modèle. Mais il faut avouer qu'il manque le côté « bricolo » des courses physiques !

Pascale Bailly
Au sein des Labs d'Orange, évoluant entre marketeurs, ingénieurs et technologies informatiques, je développe les méthodes d'innovation par les business models et les algorithmes. Ces méthodes poussent vers des écosystèmes d'affaires aux impacts sociétaux et environnementaux vertueux. Démystifier, publier des articles et faire des conférences dans nos Ecoles sont une suite logique pour transmettre les clés des devenirs numériques… Et il IA un max de travail !

Algorithmes génétiques pour clusteriser un nuage

Ma petite nièce m'a demandé : « pourquoi le chameau a-t-il 2 bosses et le dromadaire une seule alors qu'ils font le même travail dans les mêmes déserts ? » J'ai répondu : parce que c'est préférable de… heu… ». En effet, quelle optimisation dame Nature cache-t-elle derrière cette différence ? Y a-t-il forcément unicité de l'optimum ? Avant de lui répondre j'ai décidé de tester des algorithmes génétiques pour l'optimisation de clusters ; trouve-t-on deux clusterisations différentes qui ont la même note de performance ? Nos clusters sont-ils vraiment optimaux ? Quels sont nos critères de jugement ?

Une problématique classique en informatique est l'optimisation, l'art de trouver le ou les extremums d'une fonction, dans un espace de solutions donné. On peut y trouver beaucoup d'applications, y compris dans le métier de design de business models, voir Programmez! N°212, Pascale Bailly, Co-innovation, co-clustering and Co.

Certains espaces de solutions sont si vastes, dans des dimensions si grandes, qu'il est parfois difficile, voire impossible de trouver la solution optimale en un temps limité. On a alors tendance à utiliser des méthodes de résolution dites « **heuristiques** » parmi lesquelles figurent les algorithmes génétiques.

Introduction au fonctionnement des algorithmes génétiques

Ces algorithmes fonctionnent sur imitation du bio-mimétisme et de la théorie de l'évolution. Pour simplifier notre exposé, nous résumons ces théories ainsi.

Nous connaissons l'exemple de maillots de natation qui imitent la peau de requin pour favoriser le glissement dans l'eau. Le domaine logiciel n'est pas en reste avec les algorithmes génétiques qui miment l'évolution des espèces vivantes, car toutes les espèces sont « optimisées » pour faire face au mieux à l'environnement qui les entoure, et survivre. De la même manière, en faisant évoluer une population d'individus informatiques, génération après génération, on parvient à un optimum. Dans notre univers fictif ultra-simplifié, les individus sont des chromosomes qui vivent sur une planète où la nourriture est limitée ; ils se reproduisent un peu et sont quelquefois frappés par un rayon cosmique qui provoque une mutation. Nous allons voir dans ce dossier le fonctionnement général des algorithmes génétiques, quelques astuces dans leur conception puis une application ultra simple, le calcul de pi. Nous finirons par une exploration de leur application aux problèmes de clusterisation d'un nuage d'idées issues d'un travail en entreprise.

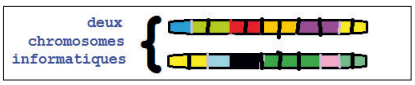
Le fonctionnement d'un Algorithme Génétique

Le fonctionnement est expliqué ci-dessous :

- **Initialisation** d'une population d'individus (solutions possibles)
- **Sélection** des individus les plus aptes (meilleures solutions)
- **Croisement** des individus sélectionnés (nouvelles solutions)
- **Mutation** de certains individus (pour ne pas rester coincé sur un optimum local)
- *Répétition* des étapes 2-4 jusqu'à avoir une convergence vers une **solution satisfaisante**.

Définition d'un individu :

Chaque individu (aussi appelé « chromosome ») représente une solution possible au problème donné, sous la forme d'une liste d'éléments.



1 *Représentation de deux chromosomes informatiques.*

Chaque élément de cette liste est appelé « allèle ». Le cas le plus simple est celui où les allèles sont binaires, mais on peut également utiliser des entiers, ou même des réels ou des listes, mais plus le nombre de possibilités pour un allèle est grand, plus l'espace des solutions à explorer sera grand, il est donc conseillé de privilégier la simplicité.

Initialisation de la population d'individus

Taille de l'ensemble de départ

La taille minimum d'une population de départ n'est pas forcément immense, certains auteurs recommandent certaines formules logarithmiques en fonction du nombre d'allèles et du nombre de modalités des allèles.

Génération des individus

Il existe principalement deux possibilités pour générer une population d'individus :

- La générer aléatoirement. Ce cas peut donner des résultats trop localisés, mais plus simple à mettre en œuvre ; **pour une situation générale si on n'a pas d'informations fiables sur le problème**.
- Utiliser des solutions déjà proches de l'optimum. Ce cas permet une convergence plus rapide, mais nécessite une connaissance préalable du problème. Cela peut converger rapidement vers un optimum local au détriment de la solution globale ; **seulement si on a une information fiable sur le problème, pour accélérer la convergence**.

Les deux solutions ont leurs qualités et leurs défauts, il est possible de sélectionner en fonction du problème, ou de générer une partie de la population avec l'une et le reste avec l'autre, c'est ce que je fais quand les données s'y prêtent.

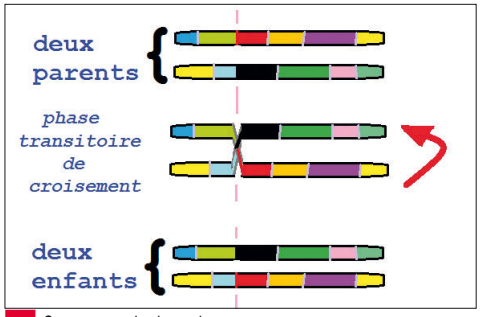
Sélection des individus. La fonction de fitness

La sélection vise à ne conserver que les individus les plus adaptés au problème donné. On appelle fonction de fitness, (ou fonction de performance), la fonction qui détermine la capacité de survie liée

à chaque individu, son optimisation pour notre problème. Pour effectuer la sélection, on ne conservera que les individus ayant obtenu le score le plus haut à cette évaluation.

Le croisement des individus

Après avoir sélectionné les meilleurs individus, on va faire des croisements (en :*crossover*) entre ces individus afin de faire croître la population, et faire émerger de nouvelles solutions. On espère ainsi augmenter la biodiversité.



2 *Croisement de deux chromosomes.*

C'est la méthode la plus simple, elle consiste à choisir aléatoirement une position dans la liste, et à sélectionner les allèles situés avant ce point dans le premier chromosome et ceux après ce point dans le deuxième. De cette manière, on génère un nouveau chromosome 'enfant' sans altérer l'ordre des allèles. En assemblant les deux autres morceaux restants, on génère un deuxième 'enfant'.

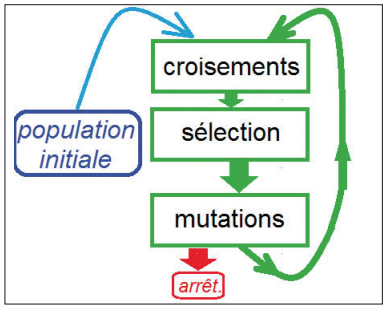
Notons qu'il existe des méthodes de croisement plus complexes, on pourra en trouver sur le web [https://en.wikipedia.org/wiki/Chromosomal_crossover].

La mutation des individus, choix de la mutation

Une boucle de sélections et de croisements risque de faire tendre les individus vers un optimum local. Cet optimum peut aussi être global, mais rien ne nous en assure, donc pour éviter ce genre d'ennui, on ajoute une étape de mutations, destinée à ajouter des valeurs différentes à certains allèles de nos individus, et ainsi augmenter la diversité. Dans notre univers fictif nous imaginons qu'un rayon cosmique frappe un chromosome sur un certain allèle et modifie sa valeur de manière aléatoire. C'est la mutation la plus simple possible. On peut en imaginer beaucoup d'autres que nous inventerons, si besoin est, au cours de nos exemples.

Algorithme complet

Nous avons maintenant en main tous les outils nécessaires à l'élaboration d'un algorithme génétique. L'algorithme se présente sous la forme d'une boucle des générations qui fera des sélections, croisements et mutations jusqu'à stabilisation des valeurs.



3 *Algorithme complet*

Remarque. On trouvera sur le web une multitude de variantes concernant les croisements de chromosomes et les mutations imaginables. Sur les tutoriels l'exemple classique d'application, le ***« Hello, word »*** du génétique, est le ***problème du voyageur de commerce***. Un voyageur de commerce doit visiter les 20 principales villes de France et revenir au point de départ en un minimum de kilomètres. Il y a plus de 10 puissance 16 circuits possibles, donc une explosion combinatoire, donc les chercheurs tentent un algorithme génétique.

Dans cet article je préfère vous offrir une autre illustration didactique plus accessible que ce voyageur, le nombre pi.

Exemple d'utilisation pour le calcul de pi

Afin d'illustrer concrètement la mise en œuvre des mécanismes génétiques je vous propose un exemple que l'on peut facilement suivre. Supposons que l'on souhaite calculer le nombre pi en ayant complètement oublié les algorithmes mathématiques enseignés au lycée et à la fac. Il faudra trouver environ 3,141592653589…etc., mais faisons semblant de l'ignorer pour le moment. On se doute simplement que le résultat sera compris entre 3 et 4. En revanche on sait un peu programmer en Python, sans connaître les subtilités du codage des réels flottants ni leur limite de précision (1/10 puissance 16 ? ou 32 ?). Le critère de vérification que j'ai choisi est fondé sur le fait que la tangente de pi/4 (=45°) est 1, donc je cherche l'angle dont la tangente se rapproche au mieux de 1 ; et je fais confiance à python.math pour calculer la tangente approximative d'un angle selon une formule mathématique que je n'ai pas explorée, d'ailleurs étrangère à cet article.

Note historique. La formule de Madhava–Leibniz pour ceux qui souhaiteraient calculer pi différemment. Cette formule présente l'avantage d'être très simple, elle permet d'approximer pi par des additions et soustractions alternées : pi=4 − 4/3 + 4/5 − 4/7 + 4/9 −etc. Elle est facile à programmer en python ou sur un tableur. Voici le début : 4 ; 2,6666666666666 ; 3,46666666666667 ; 2,8952380952381 ; 3,33968253968254 ;… Son explication ? Elle découle du fait que tan(45°)=tan(pi/4)=1. La formule est le développement en série de arctan(t) quand t=1. Son inconvénient est une lenteur de convergence insupportable. En une centaine d'itérations on obtient l'approximation médiocre 3,131 et en 10000 itérations 3.14149. On doit pouvoir mieux faire.

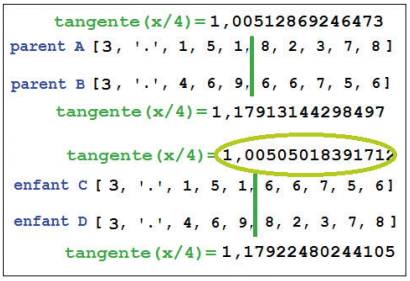
Modélisation

Il faut trouver un nombre réel, appelé flottant en informatique. Un chromosome sera un nombre décimal avec beaucoup de chiffres après la virgule, par exemple **3.32669802017349** ; mais, pour faciliter les manipulations génétiques des chiffres isolés, j'ai décidé de le coder sous forme de liste de caractères en Python, ce qui donne le chromosome : **["3",".","3","2","6","6","9","8","0","2","0","1","7","3","4","9"].**

Croisement

Ce point ne pose pas de problème. J'ai opté pour une reproduction avec deux parents hermaphrodites et avec un seul point de

croisement. On choisit dans la liste une position de croisement, on coupe les chromosomes des parents, on croise et on recolle, obtenant ainsi deux enfants. La figure 4 représente un exemple, en simplifiant les individus avec seulement huit chiffres après la virgule. On voit que l'enfant C est meilleur que les trois autres individus car la tangente de son quart est plus proche de 1, voilà donc un bon croisement aléatoire.



4 *Calcul de pi : création de deux 'enfants'.*

Dans l'algorithme expérimental j'ai adopté 32 chiffres après la virgule, c'est trop luxueux, mais je suis censée ignorer la précision des flottants en Python, quoi qu'il en soit, ce n'est qu'un exemple didactique dans lequel je fais l'impasse sur nombre de subtilités.

D'autre part, nous devons faire un nouveau choix. Lors d'un croisement, que conserver et que jeter ? J'ai opté pour la **survie des deux parents et des deux enfants**, laissons-leur une faible chance de mutation miraculeuse lors d'étapes ultérieures, ou de croisements efficaces.

La fonction d'évaluation d'un chromosome et sa vitesse

Rappelons que le mot anglais *fitness* désigne l'adaptation génétique des espèces à leur milieu naturel. Ici un candidat à pi est bien adapté, présente une bonne *fitness*, si la tangente de son quart est proche de 1.

C'est simple en théorie. Le chromosome est donné sous forme de liste de caractères, on le transforme en un vrai nombre proche de pi, on divise ce nombre par 4 dont on calcule la tangente, on la retranche de 1 (1=tangente du vrai pi/4). Cet écart est la *fitness*, plus elle est proche de 0, meilleur est le chromosome.

Mais regardons la pratique... Le calcul de la tangente en Python est fourni par la fonction math.tan(x). Le développeur n'est pas censé savoir comment python/math calcule cette tangente. On peut supposer que Python procède en calculant une série telle que tan(x) = x + x**3/3 + 2*x**5/15 + 17*x**7/315 + ... + etc. Ou bien une autre méthode plus rapide. Mais quoi qu'il en soit le modeste calcul de la fitness d'un chromosome cache peut-être une durée insoupçonnée qui, effectuée des milliers de fois, pénalisera l'algorithme génétique. Bien sûr, mais nous n'y pouvons rien. Dans les algorithmes génétiques c'est sans conteste le calcul de dizaines de milliers de fitness qui est chronophage. Donc en pratique tâchons d'imaginer une fonction de fitness rapide et pertinente quand c'est possible. Le code Python. Considérons un chromosome codé sous forme de liste de caractères.

formatstrs *== ['3', '.', '7', '4', '5', '4', '0', '9', '5', '6', '5', '6', '8', '6', '1', '2', '6', '5', '1', '4', '7', '1', '9', '2', '1', '9', '6', '8', '6', '4', '0', '0', '2', '9'] <class 'list'>*

Code Python de calcul d'une fitness.

```
# formatstrs est un chromosome
CIBLE=1
chaine=""
for it in formatstrs:
  chaine=chaine+it
  pass # on a concaténé ces caractères
nombre = float( chaine)# conversion en réel
tangente = math.tan(nombre/4)
fitness = abs( tangente -CIBLE )
print(" formatstrs  ==", formatstrs ,type(formatstrs))
print("\n chaine ==", chaine ,type(chaine))
print(" nombre ==", nombre ,type(nombre) )
print(" fitness==", fitness )
```

et on obtient par exemple

formatstrs *== ['3', '.', '7', '4', '5', '4', '0', '9', '5', '6', '5', '6', '8', '6', '1', '2', '6', '5', '1', '4', '7', '1', '9', '2', '1', '9', '6', '8', '6', '4', '0', '0', '2', '9'] <class 'list'>*
chaine *== 3.7454095656861265147192196864002 9 <class 'str'>*
nombre *== 3.7454095656861266 <class 'float'>*
fitness == 0,358800390842144

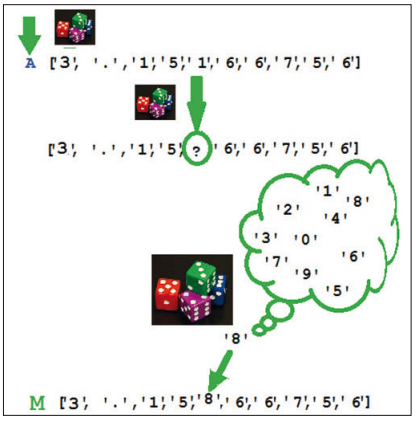
Pour en revenir à la figure 4 du croisement, vous voyez que l'enfant C est meilleur que ses parents et meilleur que son frère/sœur (hermaphrodite). Mais le cas contraire peut se produire, les deux 'enfants' peuvent être pires que les deux 'parents'. Est-ce grave ? Non puisqu'un enfant peu performant, lors de la génération suivante, peut effectuer un croisement ou une mutation particulièrement bénéfique ; ce qui signifie que deux parents médiocres peuvent avoir deux enfants pires et éventuellement un petit-enfant génial, faisons confiance aux hasards de la génétique et soyons patients.

Sélection naturelle

La nature étant impitoyable, notre planète imaginaire ne peut nourrir que 133 individus, pour ne pas trop alourdir les calculs. La phase de sélection ne conserve donc que les 133 chromosomes qui ont la meilleure fitness. Ceci fera sans doute disparaître quelques parents et enfants aux mauvais scores.

Mutations

Avant de passer à la génération suivante, effectuons quelques (N_MUT) mutations. Ma mutation basique fait appel à trois tirages au sort : On choisit un des chromosomes de la population courante qui est frappé par un rayon cosmique. On choisit aléatoirement un allèle du chromosome. On le remplace par un caractère tiré au sort entre « 0 » et « 9 ».



5 *Calcul de pi : une mutation.*

Dans cet exemple le 1 de la décimale n3 a été remplacé par 8 ; ce qui fait que le chromosome s'est éloigné de 3,14, tant pis c'est la dure règle des jeux de hasard. Mais bien entendu une autre mutation aurait pu nous rapprocher de la bonne valeur, tant mieux. Même dans ce cas, il resterait l'espoir que ce mauvais mutant M, lors de la génération suivante, crée par croisement un enfant performant.

Algorithme global

Il comporte quelques constantes de pilotage ou méta-paramètres. CIBLE =1, tangente de x/4 à atteindre. N_INIT = 64 est le nombre initial de chromosomes aléatoires. N_SELEC= 133, nombre maximal d'individus que la planète peut nourrir.
N_MUT = int((5/100)*N_SELEC) # nombre de mutations ^ chaque génération. N_GENER = 10000, nombre maximal de générations parce que l'obtention du résultat n'est pas urgente ici.
Dans le cadre de ce petit exercice, les valeurs précises de ces constantes sont sans importance. Nous avons codé une population sous un format de dictionnaire, les clés étant des noms et les valeurs étant les chromosomes codés en liste de caractères.
Le programme écrit en pseudo-code est structuré selon le schéma du début, vous pourrez coder chaque subroutine à tête reposée :

```
import math
fixation des méta-paramètres CIBLE, N_INIT, N_SELEC, N_MUT, N_GENER.
popul_initiale = init_la_popul( N_INIT )
# la fonction init_la_popul effectue N_INIT tirages au sort de chromosomes
# de format liste de chiffres telle que ['3', '.', '7', '4', '5', '4', '0', '9', '5', '6',
 '5', '6', '8', '6', '1', '2', '6', '5', '1', '4', '7', '1', '9', '2', '1', '9', '6', '8', '6', '4',
 '0', '0', '2', '9'] <class 'list'>
"""FORMAT : une population est un dict nom_du_chromosome: liste_de_chiffres ;
 le nom d'un chromosome est tiré au sort lors de sa naissance"""
populcourante =popul_initiale
for gener in range ( N_GENER ):
  calculer les fitness des individus de populcourante
  # la fitness est l'écart entre la CIBLE et la tangente de l'angle/4
  les classer par ordre croissant
  # plus la fitness est faible, mieux c'est
  populselctionnee = ne garder que les N_SELEC premiers
  # fin de la sélection
  """  mutations à additionner """
  nouv_mutants = effectuer_des_mutations( N_MUT , populselctionnee )
  populcourante = additionner_2_dicts( populselctionnee , nouv_mutants )
  """  supprimer les sosies, simple précaution"""
  populcourante = supprimer_les_sosies( populcourante )
  """ calcul du champion et test d'arrêt"""
  championcourant =extraire_champion(populcourante)
  if test_d_arret(championcourant,populcourante) == True :
    break
  pass #bouclage sur la génération gener
##***##
"""  retour au programme principal pour enregistrement sur fichier ou pour affichage
 du pseudo-Pi championcourant."""
```

On remarquera que la taille de la population dépasse N_SELEC= 133 parce que j'ai volontairement introduit les **mutations»** après **«la limitation de population»**. Ceci dans le but d'augmenter la diversité ; en gros j'opte pour une augmentation de la lenteur d'une génération en espérant un gain en nombre de générations.

Astuces spécifiques au problème

Un *run* typique a donné le champion == [3, '.', 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 4, 1, 1], dont la tangente du quart est 0.9999999999621588, donc 10 bonnes décimales, mais en réclamant plus de 1000 générations. Coup de malchance ou pas, c'est lent, il faudrait sans doute améliorer. Pour ce calcul de pi, je ne m'attarderai pas sur les améliorations imaginables de mutations, nous verrons ce point lors des clusterisations.

Questions de rapidité

Notre motivation initiale était de trouver le résultat plus rapidement que l'examen exhaustif des milliards de nombres commençant par 3. La méthode génétique procède par approximations successives génération après génération. Chaque approximation étant le chromosome champion de sa génération. Dans toute méthode d'approximations successives se posent inévitablement deux questions : ne pas être coincé dans un sous-optimum local et aller vite. La sortie du coinçage est censée être assurée par la biodiversité des mutations aléatoires ingénieuses.
Reste le souci de la vitesse de convergence vers l'optimum et nous avons réussi avec environ 1000 générations d'environ 133 individus. C'est satisfaisant.
Certes, mais on pourra nous objecter qu'il existe un algorithme beaucoup plus rapide, celui de Ramanujan.

Note : la série de Ramanujan (1887-1920).
Il s'agit d'une approximation de 1/pi par la série dont les termes u(k) sont :
u(k)=(2*racine(2)/9801)/(((4*k) !*(1103 +26390*k))/((k!)**4*396**(4*k))).
Cette série invraisemblable converge à une vitesse ahurissante, voyez sur un tableur LibreOffice :

k	approx pi de Ramanujan	constante pi du tableur
0	3,14159273001331	3,14159265358979
1	3,14159265358979	3,14159265358979
2	3,14159265358979	3,14159265358979

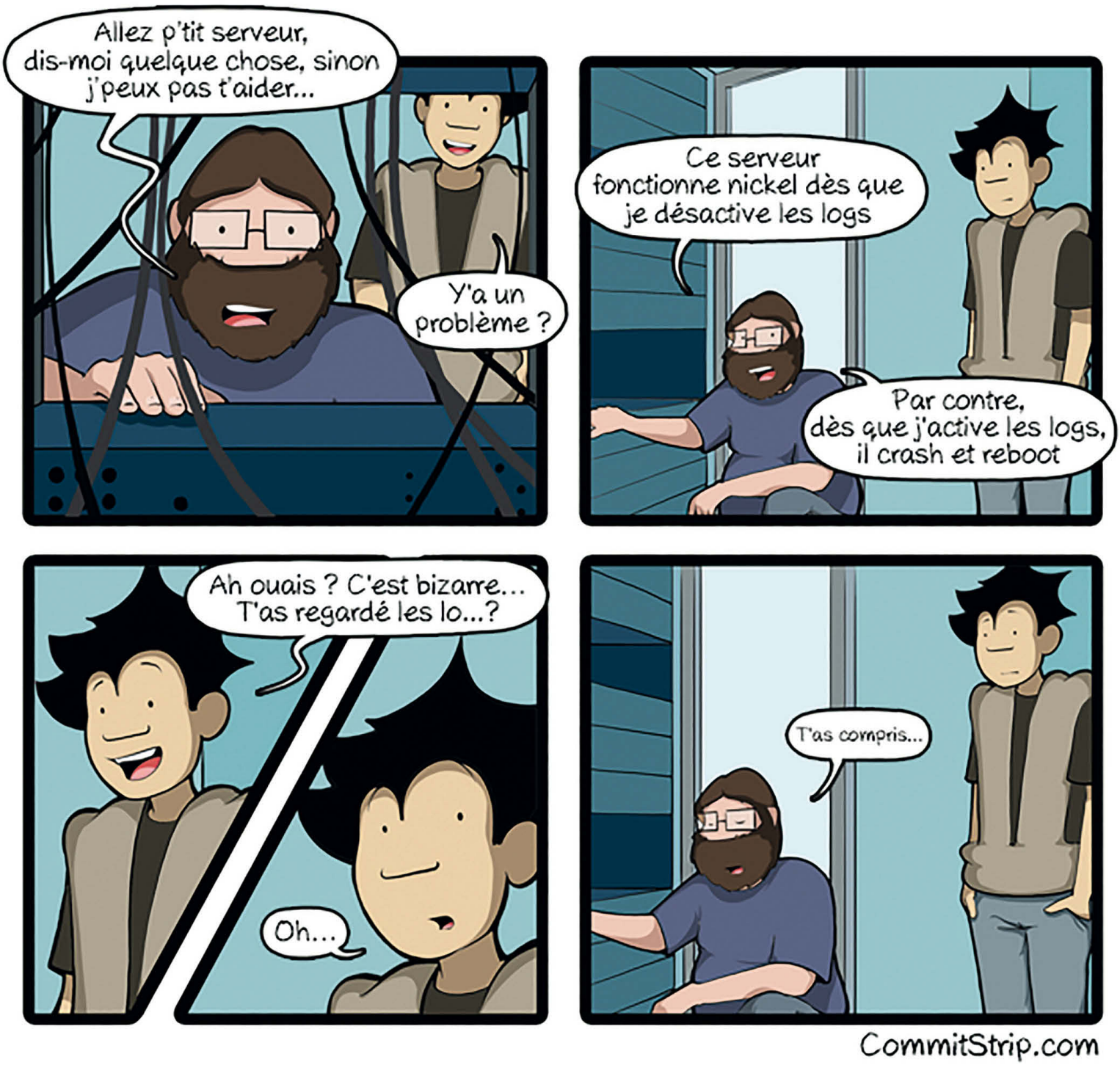
Dès le troisième terme l'approximation dépasse la précision du tableur. Personne ne peut concurrencer Ramanujan, l'algorithme génétique est KO, mais rappelons qu'il ne s'agit que d'un exercice didactique.

Bilan de pi

Le math.pi natif de Python vaut 3.141592653589793 ; notre algorithme a trouvé [3, '.', 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 4, 1, 1] ; il y a 10 décimales correctes. c'est satisfaisant et on pourrait faire mieux en augmentant les générations.

Dans Programmez! 243, nous aborderons l'algorithme génétique pour une clusterisation et les bonnes pratiques.

Parler le serveur



Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication : François Tonic
Rédacteurs en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Participation à ce numéro : ZDNet, Epitech
Les experts techniques : L. Saget-Lethias, F. Franchin, E. Martineau, A. Caussignac, A. Roman, C. Pruvost, G. Salem, T. Leriche, D. Opitz, G. Euzet, A. Giretti, S. Kokaina, S. Philippart, L. Avrot, M. Ellerbach, P. Bailly
Photo de couverture : © François Tonic - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@programmez.com.
Imprimeur : SIB Imprimerie
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34

Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, août 2020

Les parutions du magazine de la culture mathématique

Tangente 195 et *Tangente* HS 75

Deux numéros passionnants disponibles dès la rentrée

Procurez-vous les numéros chez votre marchand de journaux ou sur

www.infinimath.com/librairie

(ou mieux, abonnez-vous)

En ligne sur

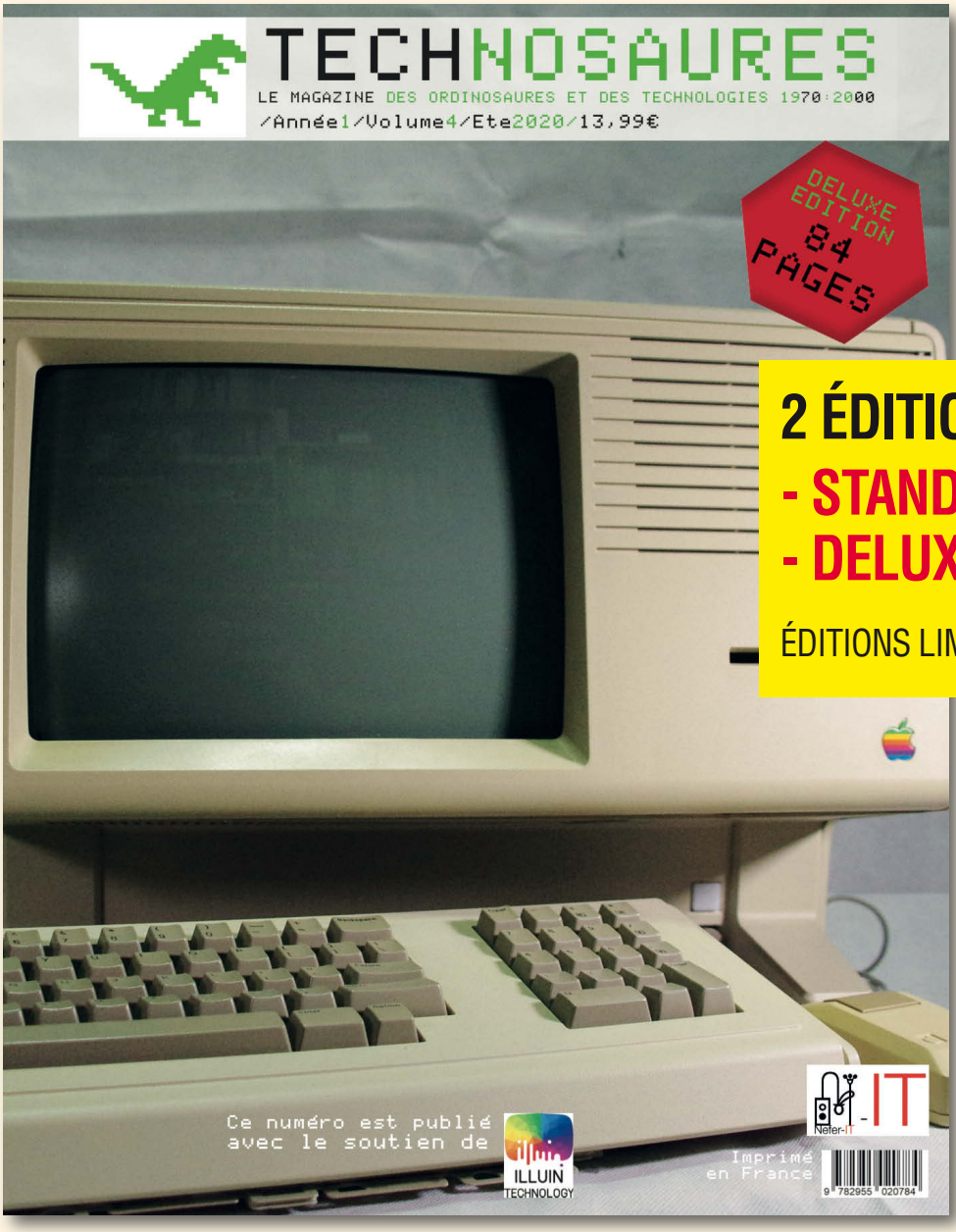
tangente-mag.com

(intégralement pour les abonnés)

Parution en septembre

NUMÉRO EXCEPTIONNEL

100 % APPLE LISA



2 ÉDITIONS :

- STANDARD *52 PAGES*

- DELUXE *84 PAGES*

ÉDITIONS LIMITÉES

Commandez directement sur **www.programmez.com**

Standard édition : **8,99 €** *

Deluxe édition : **13,99 €** *

** Frais de port : 1,01 €*